

---

# GNU/Linux

## Programmer's Manual

### Maintainers:

Alejandro Colomar <alx@kernel.org> 2020 - present (5.09 - HEAD)  
Michael Kerrisk <mtk.manpages@gmail.com> 2004 - 2021 (2.00 - 5.13)  
Andries Brouwer <aeb@cwi.nl> 1995 - 2004 (1.6 - 1.70)  
Rik Faith 1993 - 1995 (1.0 - 1.5)

**NAME**

intro – introduction to commands

**DESCRIPTION**

This section describes publicly accessible commands in alphabetic order.

The name of a particular machine at the head of the page means that the command lives there and not necessarily elsewhere. ‘Local’ means the same, without being specific about where.

**SEE ALSO**

Section (7) for databases.

Section (8) for ‘hidden’ commands for booting, maintenance, etc.

Section (9) for commands that involve the Teletype 5620 terminal.

*How to get started*, in the Introduction.

**DIAGNOSTICS**

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of ‘normal’ termination) one supplied by the program; see [exit\(2\)](#). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously ‘exit code’, ‘exit status’ or ‘return code’, and is described only where special conventions are involved.

**NAME**

2500 – BVH2500 videotape recorder

**SYNOPSIS**

**2500** [ **-lq** ]

**DESCRIPTION**

*2500* is an interpreter of commands to control a SONY BVH2500 1-inch video recorder, whose inputs and outputs have already been set up. The options are

**-l** Create a log file; useful in pursuit of bugs.

**-q** Suppress the initial status report.

Most of the commands require an intimate knowledge of the equipment. The simpler commands are described below; see the **help** command for a complete list. Times are given as **[[hrs.]min.]sec.fr** where there are 30 frames per second. The commands are

**cue** *t* Move the tape to time *t*.

**help** Produce a list of all commands.

**loop** *t0 t1 fps*

Play from *t0* through *t1* and back again at *fps* frames per second.

**play** Start playing the tape from the current frame.

**snap** *n* When in still record mode, record the current input onto the next *n* frames. A missing *n* is taken to be 1.

**status** Print some status information. The command **status status** prints all available status information.

**still** *t* Go into still record mode and cue to time *t*. The command returns before the tape transport is done; usually it must be followed by

**still mode on|off**

Turn still mode on or off.

**stop** Stop the tape transport.

**view** *t0 t1* Play from *t0* through *t1*. **wait** Wait for the previous tape transport command to finish.

**!** Interpret the rest of the line as a *sh(1)* command.

**#** Comment. Ignore the rest of the line.

**EXAMPLES**

Assuming you have already set up the video switch to feed the BVH2500, the following script will record (or rerecord) a movie starting at 2 minutes.

```
still 2.0.0
wait
!generate an image
snap 1
# repeat the last two lines as necessary
still mode off
stop
```

**BUGS**

The BVH2500 will misbehave if the pause between **snaps** (in still record mode) is too long, or if you record for many hours on end. The latter problem can be avoided by using scripts that run for 2 or 3 hours and sleeping for 10 minutes between scripts with the tape transport off.

Commands in the help list are (incorrectly) capitalized.

**NAME**

300, 300s – handle special functions of DASI 300 and 300s terminals

**SYNOPSIS**

**300** [ +12 ] [ -n ] [ -dt,l,c ]

**300s** [ +12 ] [ -n ] [ -dt,l,c ]

**DESCRIPTION**

*300* supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. *300* can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | 300
```

**WARNING:** if your terminal has a PLOT switch, make sure it is turned *on* before *300* is used.

The behavior of *300* can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

- +12** permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.
- n** controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, *nroff*(1) half-lines could be made to act as quarter-lines by using **-2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **-3** alone, having set the PITCH switch to 12-pitch.
- dt,l,c** controls delay factors. The default setting is **-d3,90,30**. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/passwd**. Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The **-d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file **/etc/passwd** may be printed using **-d3,30,5**. The value **-d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

*300* can be used with the *nroff* **-s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

```
nroff -T300 files ... and nroff files ... | 300
nroff -T300-12 files ... and nroff files ... | 300 +12
```

The use of *300* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *300* may produce better-aligned output.

The *neqn*(1) names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(7).

**SEE ALSO**

[450\(1\)](#), [eqn\(1\)](#), [graph\(1G\)](#), [msg\(1\)](#), [sty\(1\)](#), [tabs\(1\)](#), [tbl\(1\)](#), [tplot\(1G\)](#), [troff\(1\)](#), [greek\(7\)](#).

**BUGS**

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

**NAME**

450 – handle special functions of the DASI 450 terminal

**SYNOPSIS**

**450**

**DESCRIPTION**

*450* supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as [300\(1\)](#). *450* can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | 450
```

WARNING: make sure that the PLOT switch on your terminal is ON before *450* is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

*450* can be used with the *nroff(1)* `-s` flag or `.rd` requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the use of *450* can be eliminated in favor of one of the following:

```
nroff -T450 files ...
```

or

```
nroff -T450-12 files ...
```

The use of *450* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *450* may produce better-aligned output.

The *neqn(1)* names of, and resulting output for, the Greek and special characters supported by *450* are shown in [greek\(7\)](#).

**SEE ALSO**

[300\(1\)](#), [eqn\(1\)](#), [graph\(1G\)](#), [mesg\(1\)](#), [stty\(1\)](#), [tabs\(1\)](#), [tbl\(1\)](#), [tplot\(1G\)](#), [troff\(1\)](#), [greek\(7\)](#).

**BUGS**

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

**NAME**

acro – find acronyms in a text file

**SYNOPSIS**

**acro** [ **-flags** ] [ **-ver** ] file ...

**DESCRIPTION**

*Acro* searches for acronyms in a text file. It prints each sentence containing an acronym. *Acro* also prints a frequency count of all acronyms used in the text.

*Acro* skips lines that begin with a dot, "."; so text files that contain standard *nroff*(1) and *mm*(1) macros are acceptable input.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**USES**

*Acro* can be used to locate acronyms in a text. The user can then check to see that an acronym is fully defined when it is first used.

**FILES**

/tmp/\$\$\*                      temporary files

**SEE ALSO**

*nroff*(1), *mm*(1).

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

adb – debugger

**SYNOPSIS**

**adb** [ *option ...* ] [ *objfil* [ *corfil* ] ]

**DESCRIPTION**

*Adb* is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

*Objfil* is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is

Requests to *adb* are read from the standard input and responses are to the standard output. Quit signals are ignored; interrupts cause return to the next *adb* command. The options are

**-w** Create *objfil* and *corfil* if they don't exist; open them for writing as well as reading.

**-I***path*

Directory in which to look for relative pathnames in \$< and \$<< commands.

In general requests to *adb* have the following form. Multiple requests on one line must be separated by

[*address*] [, *count*] [*command*]

If *address* is present then the current position, called 'dot', is set to *address*. Initially dot is set to 0. In general commands are repeated *count* times. Dot advances between repetitions. The default *count* is 1. *Address* and *count* are expressions.

Some formats, data sizes, and command details have different behavior on different systems. See the MACHINE DEPENDENCIES attachment for details.

**Expressions**

Expressions are computed with sufficient precision to address the largest possible file; generally this means a long integer. On the VAX, expressions are 32 bits; on the Cray, 64 bits.

- .
  - +
  - ^
  - "
- The value of dot.  
 The value of dot incremented by the current increment.  
 The value of dot decremented by the current increment.  
 The last *address* typed.

*integer*

A number in the *default radix*; see the \$d command. Regardless of the default, the prefixes and (zero oh) force interpretation in octal radix; the prefixes and force interpretation in decimal radix; the prefixes and force interpretation in hexadecimal radix. Thus and all represent sixteen.

*integer.fraction*

A floating point number.

'*cccc*'

The ASCII value of one or more characters. may be used to escape a

<*name*

The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables named by single letters or digits. The register names are those printed by the \$r command.

*symbol*

A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*.

*routine.name*

The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*; if *routine* is omitted, the active procedure is assumed.



*(exp)* The value of the expression *exp*.

#### Monadic operators

*\*exp* The contents of the location addressed by *exp* in *corfil*.

*@exp* The contents of the location addressed by *exp* in *objfil*.

*-exp* Integer negation.

*~exp* Bitwise complement.

*%exp* If *exp* is used as an address, it is in register space; see 'Addresses'.

*Dyadic operators* are left associative and are less binding than monadic operators.

*e1+e2* Integer addition.

*e1-e2* Integer subtraction.

*e1\*e2* Integer multiplication.

*e1%e2*  
Integer division.

*e1&e2*  
Bitwise conjunction.

*e1|e2* Bitwise disjunction.

*e1#e2* *E1* rounded up to the next multiple of *e2*.

### Commands

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands *and* may be followed by see 'Addresses' for further details.)

*?f* Locations starting at *address* in *objfil* are printed according to the format *f*.

*/f* Locations starting at *address* in *corfil* are printed according to the format *f*.

*=f* The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. If no format is given then the last format is used.

Most format letters fetch some data, print it, and advance (a local copy of) dot by the number of bytes fetched. The total number of bytes in a format becomes the *current increment*. 'Long integers' are full words, the size of an expression item: e.g. 4 bytes on the VAX, 8 bytes on the Cray. 'Short integers' are some useful shorter size: 2 byte short integers on the VAX, 2 byte parcels on the Cray.

<b>r</b>	Print short integer in the current default radix.
<b>R</b>	Print long integer in the current default radix.
<b>o</b>	Print short integer in octal.
<b>O</b>	Print long integer in octal.
<b>q</b>	Print short in signed octal.
<b>Q</b>	Print long in signed octal.
<b>d</b>	Print short in decimal.
<b>D</b>	Print long in decimal.
<b>x</b>	Print short in hexadecimal.
<b>X</b>	Print long in hexadecimal.
<b>u</b>	Print short in unsigned decimal.
<b>U</b>	Print long in unsigned decimal.
<b>f</b>	Print as a floating point number.
<b>F</b>	Print double-precision floating point.
<b>b</b>	Print the addressed byte in octal.
<b>c</b>	Print the addressed character.

- C** Print the addressed character. Control characters are printed in the form `^X` and the delete character is printed as `^?`.
- s** Print the addressed characters until a zero character is reached. Advance dot by the length of the string, including the zero terminator.
- S** Print a string using the `^X` escape convention (see **C** above).
- Y** Print a long integer in date format (see [ctime\(3\)](#)).
- i** Print as machine instructions. This style of printing causes variables `0`, `(1, ...)` to be set to the offset parts of the first (second, ...) operand of the instruction.
- a** Print the value of dot in symbolic form. Dot is unaffected.
- p** Print the addressed value in symbolic form. Dot is advanced by the size of a machine address (4 bytes on the VAX, 8 bytes on the Cray).
- t** When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop. Dot is unaffected.
- n** Print a newline. Dot is unaffected.
- "..."** Print the enclosed string. Dot is unaffected.
- ^** Dot is decremented by the current increment. Nothing is printed.
- +** Dot is incremented by 1. Nothing is printed.
- Dot is decremented by 1. Nothing is printed.

newline

Update dot by the current increment. Repeat the previous command with a *count* of 1.

**[?/]l value mask**

Words starting at dot are masked with *mask* and compared with *value* until a match is found. If **l** is used, the match is for a short integer; **L** matches longs. If no match is found then dot is unchanged; otherwise dot is set to the matched location. If *mask* is omitted then `-1` is used.

**[?/]w value ...**

Write the short *value* into the addressed location. If the command is **W**, write a long. Option **-w** must be in effect.

**[?/]m b e f [?]**

New values for *(b, e, f)* in the first map entry are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. The address type (instruction or data) is unchanged in any case. If the `or` is followed by then the second segment of the mapping is changed. If the list is terminated by `or` then the file (*objfil* or *corfil* respectively) is used for subsequent requests. For example, `will` cause to refer to *objfil*.

**>name**

Dot is assigned to the variable or register named.

**!** A shell is called to read the rest of the line following `'!`.

**\$modifier**

Miscellaneous commands. The available *modifiers* are:

- <f** Read commands from the file *f*. If *f* cannot be found, try `/usr/lib/adb/f`. If this command is executed in a file, further commands in the file are not seen. If *f* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable **9** before the first command in *f* is executed.
- <<f** Similar to `<` except it can be used in a file of commands without causing the file to be closed. Variable **9** is saved during the execution of this command, and restored when it completes. There is a (small) limit to the number of `<<` files that can be open at once.
- >f** Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.
- ?** Print process id, the signal which caused stopping or termination, as well as the registers. This is the default if *modifier* is omitted.
- r** Print the general registers and the instruction addressed by **pc**. Dot is set to **pc**.
- R** Like **\$r**, but include miscellaneous registers such as the kernel stack pointer.

- b** Print all breakpoints and their associated counts and commands.
- c** C stack backtrace. If *address* is given then it is taken as the address of the current frame; otherwise, the current C frame pointer is used. If **C** is used then the names and (long) values of all parameters, automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.
- a** Set the maximum number of arguments printed by **\$c** or **\$C** to *address*. The default is 20.
- d** Set the default radix to *address* and report the new value. *Address* is interpreted in the (old) current radix; never changes the default radix. To make decimal the default radix, use A radix of zero (the initial default) is a special case; input with a leading zero is octal, that with a leading sharp-sign is hexadecimal, other numbers are decimal. When the default radix is zero, the default output radix is appropriate to the machine: hexadecimal on the VAX, octal on the Cray.
- e** The names and values of all external variables are printed.
- w** Set the page width for output to *address* (default 80).
- s** Set the limit for symbol matches to *address* (default 255).
- q** Exit from *adb*.
- v** Print all non zero variables in the current radix.
- m** Print the address maps.
- k** Simulate kernel memory management.
- p** Simulate per-process memory management.  
**\$k** and **\$p** are used for system debugging. Their details vary with machine and operating system.

#### *:modifier*

Manage a subprocess. Available modifiers are:

- bc** Set breakpoint at *address*. The breakpoint is executed *count*-1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command is omitted or sets dot to zero then the breakpoint causes a stop.
- d** Delete breakpoint at *address*.
- r** Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are enabled on entry to the subprocess.
- cs** The subprocess is continued. If *s* is omitted or nonzero, the subprocess is sent the signal that caused it to stop; if 0 is specified, no signal is sent. Breakpoints and single-stepping don't count as signals. Breakpoint skipping is the same as for **r**.
- ss** As for **c** except that the subprocess is single stepped *count* times. If a signal is sent, it is received before the first instruction is executed. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
- k** The current subprocess, if any, is terminated.

#### Variables

*Adb* provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

**0, 1, ...**

The offset parts of the first, second, ... operands of the last instruction printed. Meaningless if the operand was a register.

**9** The count on the last **\$<** or **\$<<** command.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a core image then these values are set from *objfil*.

**b** The base address of the data segment.

<b>d</b>	The data segment size.
<b>e</b>	The entry point.
<b>m</b>	The ‘magic’ number ( <i>a.out</i> (5)).
<b>s</b>	The stack segment size.
<b>t</b>	The text segment size.

### Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by one or more quadruples (*t, b, e, f*), each mapping addresses of type *t* (instruction, data, user block) in the range *b* through *e* to the part of the file beginning at address *f*. An address *a* of type *t* is mapped to a file address by finding a quadruple of type *t*, for which  $b \leq a < e$ ; the file address is *address*+*f*-*b*. As a special case, if an instruction space address is not found, a second search is made for the same address in data space.

Typically, the text segment of a program is mapped as instruction space, the data and bss segments as data space. If *objfil* is an *a.out*, or if *corfil* is a core image or process file, maps are set accordingly. Otherwise, a single ‘data space’ map is set up, with *b* and *f* set to zero, and *e* set to a huge number; thus the entire file can be examined without address translation.

The **?** and **/** commands attempt to examine instruction and data space respectively. **?\*** tries for data space (in *objfil*); **/\*** accesses instruction space (in *corfil*).

Registers in process and core images are a special case; they live in a special ‘register’ address space. The contents of register 0 are located at address **%0**; register 1 at **%4** (if registers are 4 bytes long); and so on. **%** addresses are mapped to the registers for the ‘current frame,’ set by local variable references, and reset to the outermost frame (the ‘real’ registers) whenever a process runs or a stack trace is requested.

Simulated memory management translations (the **\$k** and **\$p** commands) are done before the mapping described above.

### FILES

parameter files

### SEE ALSO

*cin*(1), *pi*(9) *nm*(1), *proc*(4), *a.out*(5), *bigcore*(1)

J. F. Maranzano and S. R. Bourne, ‘A Tutorial Introduction to ADB’ in Bell Laboratories, *UNIX Programmer’s Manual*, Volume 2, Holt, Rinehart and Winston (1984)

### DIAGNOSTICS

‘Adb’ when there is no current command or format. Exit status is 0, unless last command failed or returned nonzero status.

### BUGS

Either the explanation or the implementation of register variables is too complex and arcane.

### MACHINE DEPENDENCIES

#### PDP-11

Short integers (printed by **r** format) are 2 bytes; long integers (printed by **R** format) are 4 bytes. Addresses printed by **a** format are 2 bytes.

Register variables match the hardware in the obvious way: **r0** is at address **%0**, **r1** at **%2**, and so on.

The default output radix is octal.

**\$k** and **\$p** are unimplemented.

#### VAX

Short integers are 2 bytes, long integers are 4 bytes, addresses are 4 bytes.

Register variables match the hardware in the obvious way: **r0** is at address **%0**, **r1** at **%4**, and so on.

The default output radix is hexadecimal.

**\$k** sets the system base register pointer to *address*. System space addresses are thereafter mapped according to the system page table starting at that physical address. An *address* of zero turns off mapping.

**\$p** sets the process control block pointer to *address*; user space addresses are thereafter translated according to the user page tables described by the PCB. Kernel mapping must already be in effect. *Address*

may be a physical address (that of the PCB) or the system space virtual address of a page table entry pointing to the PCB (the number stored in *p\_addr*). If *address* is zero, user mapping is turned off; addresses less than 0x80000000 will be treated as physical addresses.

The command will initialize registers and mapping from a kernel crash dump.

### **Cray**

Short integers are 2 bytes; long integers are 8 bytes. Addresses are 8 bytes.

Registers are funny, and yet to be described.

The default output radix is octal.

**\$k** and **\$p** are unimplemented.

**NAME**

altran – language for algebraic manipulation

**SYNOPSIS**

**altran** [ *option* ]

**DESCRIPTION**

*Altran* compiles the language described in the reference via Fortran as an intermediate language. The *files* may have names ending in **.al**, for Altran source, **.f**, for Fortan source, or **.o** for binary object files. The output normally includes a **.f** file for each **.al** file, a **.o** file for each old or new **.f** file, an executable file **a.out**, and a listing file whose name ends in **.list** for each **.al** file.

All the *options* of [f77\(1\)](#) and [ld\(1\)](#) are accepted with the same meanings.

The executable **a.out** file accepts two parameters. One is of the form **p=n** where *n* is the number of decimal digits of precision to be used for long integers,  $18 \leq n \leq 900$ , default 18. The other parameter is of the form **w=n** where *n* is the workspace size in thousands of words,  $10 \leq n \leq 800$ , default 50.

**FILES**

/usr/lib/altran the compiler proper  
/usr/lib/libal.a library  
fort.[789] intermediate files  
\*.al  
\*.f  
\*.o  
\*.list

**SEE ALSO**

[f77\(1\)](#), [ld\(1\)](#)

W. S Brown, *ALTRAN User's Manual*, Bell Laboratories, Murray Hill, NJ, 1977

**BUGS**

Run-time output is voluminous.

**NAME**

apl – an apl interpreter

**SYNOPSIS**

**apl**

**DESCRIPTION**

*Apl* is an APL interpreter. All of the operators are exactly as in `apl\360`. Overstrikes are often required, and they work (use `ctrl-h`).

Function definition is not what you would expect. Functions are loaded from files. The first line of the file is the function header, as you would expect it but with no `del`. The rest of the file is the lines of the function. Lines are numbered, but there are no square brackets with line numbers. If you say `)READ FILE` it will load the function in that file. If you say `)EX FILE` it will put you in the editor to change that file. Upon exit, it will read the file in as though by `)READ`.

All of the usual operators are available, including `domino`. Also available are monadic `encode` and `epsilon`.

The following *apl* system commands are available.

**)ASCII**

changes terminal to accept and print ASCII characters and operators; this is the default. If you are stuck in APL mode on an ASCII terminal, “” is ‘)’ and lowercase letters map to uppercase.

**)APL**

changes terminal to accept and print APL characters. Erase is set to  $\Omega$  and kill is set to  $\alpha$ .

**)DIGITS n**

sets the number of digits displayed to *n*, from 1 to 19.

**)FUZZ n**

sets the fuzz to *n*.

**)ORIGIN n**

sets the origin to *n*, which should be 1 or 0.

**)WIDTH n**

sets *apl*'s idea of your terminal's carriage width.

**)ERASE n**

gets rid of function or variable named *n*.

**)SAVE n**

saves all variables and functions (workspace) in file named *n*. Workspaces are sensitive to changes in *apl*.

**)LOAD n**

gets the workspace in file *n* (which must have been `)SAVE'd`.)

**)COPY n**

like `)LOAD` but variables and functions are not erased. Things in the loaded file take precedence over stuff already in.

**)CLEAR**

clears the workspace.

**)DROP n**

deletes file *n* in your directory, which need not be saved from *apl*.

**)CONTINUE**

exits and saves workspace in file *continue* which is loaded next time you run *apl*.

**)OFF**

exits *apl*.

**)READ n**

reads in a function from file *n*. The first line is the header, with no `del`'s. The full APL360 header is accepted. All other lines in the file are lines in the function. Lines are implicitly

numbered, and transfers are as usual. There are no labels.

- )EDIT *n*  
runs the editor *ed(1)* on file *n*, and then )READ's the file when you leave the editor.
- )EX *n*  
runs the editor *ex(1)* on file *n*, and then )READ's the file when you leave the editor.
- )VI *n*  
runs the editor *vi(1)* on file *n*, and then )READ's the file when you leave the editor.
- )LIB  
lists out all of the files in the current directory.
- )FNS  
lists out all current functions.
- )VARS  
lists out all current variables.
- )DEBUG  
toggles a debugging switch, which can produce vast amounts of hopelessly cryptic output.

## FILES

apl\_ws – temporary workspace file  
continue – continue workspace

## AUTHORS

Ken Thompson, Ross Harvey, Douglas Lanam

## BUGS

This program has not been extensively used or tested.



## ASCII CHAR MNEMONICS

&	$\wedge$	and	#	$\times$	times
-	-	minus	+	+	add
<	<	less than	>	>	greater than
=	=	equal to	,	,	comma
%	$\div$	divide	*	*	exponential (power)
!	!	factorial and combinations	?	?	deal
.le	$\leq$	less than or equal	.ge	$\geq$	greater than or equal
.ne	$\neq$	not equal	.om	$\Omega$	omega (not used)
.ep	$\epsilon$	epsilon	.rh	$\rho$	shape (rho)
.nt	$\neg$	not (also $\sim$ )	.tk	$\uparrow$	take (also $\sim$ )
.dr	$\downarrow$	drop	.it	$\iota$	iota
.ci	$\circ$	circular function	.al	$\alpha$	alpha (not used)
.cl	$\lceil$	maximum (ceiling)	.fl	$\lfloor$	minimum (floor)
.dl	$\Delta$	del (not used)	.de	$\nabla$	upside down del
.jt	$\circ$	small circle (null)	.qd	$\square$	quad
.ss	$\subset$	right U (not used)	.sc	$\supset$	left U (not used)
.si	$\cap$	Down U	.su	$\cup$	U (not used)
.[ $\wedge$	$\nabla$	upside-down del	.bv	$\parallel$	decode (base)
.rp	$\parallel$	encode (rep)	.br		residue (mod)
.sp	$\leftarrow$	assignment (also ' _')	.go	$\rightarrow$	goto
.or	$\vee$	or	.nn	$\Lambda$	nand
.nr	$\tilde{\vee}$	nor	.lg	$\otimes$	log
.rv	$\phi$	reversal	.tr	$\phi$	transpose
.rb		reverse bar	.cb	;	comma bar ( not used)
.sb	/	slash bar	.bb	$\backslash$	backslash bar
.gu	$\Delta$	grade up	.gd	$\nabla$	grade down
.qq	$\square$	quote quad	.dm	$\boxtimes$	domino
.lm	$\hat{\wedge}$	lamp	.ib	$\parallel$	I-beam
.ex		execute (not used)	.fr		format(not used)
.di		diamond (not used)	.ot		out (not used)
.ld	$\Delta$	locked del (not used)	._a	A	alias for 'A'
._b	B	alias for 'B'	._c	C	alias for 'C'
._d	D	alias for 'D'	._e	E	alias for 'E'
._f	F	alias for 'F'	._g	G	alias for 'G'
._h	H	alias for 'H'	._i	I	alias for 'I'
._j	J	alias for 'J'	._k	K	alias for 'K'
._l	L	alias for 'L'	._m	M	alias for 'M'
._n	N	alias for 'N'	._o	O	alias for 'O'
._p	P	alias for 'P'	._q	Q	alias for 'Q'
._r	R	alias for 'R'	._s	S	alias for 'S'
._t	T	alias for 'T'	._u	U	alias for 'U'
._v	V	alias for 'V'	._w	W	alias for 'W'
._x	X	alias for 'X'	._y	Y	alias for 'Y'
._z	Z	alias for 'Z'			

**NAME**

apply, pick – repeatedly apply a command; select arguments

**SYNOPSIS**

**apply** [ **-ac** ] [ **-n** ] *command* *arg* ...

**pick** [ *arg* ... ]

**DESCRIPTION**

*Apply* runs the named *command* on each argument *arg* in turn. Normally arguments are chosen singly; the optional number *n* specifies the number of arguments to be passed to *command*. If *n* is zero, *command* is run without arguments once for each *arg*. Character sequences of the form *%d* in *command*, where *d* is a digit from 1 to 9, are replaced by the *d*th following unused *arg*. If any such sequences occur, *n* is ignored, and the number of arguments passed to *command* is the maximum value of *d* in *command*. The character may be changed by the **-a** option.

*Pick* writes each argument to the standard error and reads a reply. If the reply is the argument is echoed to the standard output; if the reply is *pick* exits without reading any more arguments; there is no output for any other response. If there are no arguments, lines of the standard input are taken instead.

**EXAMPLES**

apply echo \*

Time-consuming way to do

apply -2 cmp a1 b1 a2 b2

Compare the ‘a’ files to the ‘b’ files.

wc -l ‘pick \*.ch’

Interactively select ‘.c’ and ‘.h’ files and count the lines in each.

apply "wc -l %1" ‘pick \*.ch’

Same, but use a separate process to count each file.

**SEE ALSO**

[sh\(1\)](#)

**BUGS**

There is no way to pass a literal if is *apply*’s argument expansion character.

**NAME**

apsend – send troff output to phototypesetter

**SYNOPSIS**

**apsend** [ *options* ] [ *file ...* ]

**DESCRIPTION**

*Apsend* sends [troff\(1\)](#) output from the named files or from the standard input to the Murray Hill computer center for high-quality typesetting.

The options, which need only be spelled far enough to be unique, are

**account=xx**

comp center account number (default from password file)

**bin=xx**

comp center bin number (default from password file)

**mailto=xx**

mailing instructions, up to 28 characters, instead of comp center bin

**comment=xx**

up to 30 characters, for file entry (default value is *file* or

**device=imagen**

Print on laser printer instead of phototypesetter.

**FILES**

record of apsend activity

**SEE ALSO**

[troff\(1\)](#), [lp\(1\)](#), [font\(5\)](#)

**BUGS**

Do not be misled by the historical name of this program. The correct *troff* device selection is **-Tpost**, which is fortunately the default.

**NAME**

`ar`, `ranlib` – archive and library maintainer

**SYNOPSIS**

`ar key [ posname ] afile [ file ... ]`

`ranlib archive ...`

**DESCRIPTION**

*Ar* maintains groups of files combined into a single archive file, *afile*. If it is not to be modified, the archive may be read from standard input, indicated by the name `.`. The main use of a *ar* is to create and update library files for the loader *ld*(1). It can be used, though, for any similar purpose.

*Key* is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcl**. The *files* are constituents of the archive *afile*. The meanings of the *key* characters are:

- d** Delete *files* from the archive file.
- r** Replace *files* in the archive file. Optional modifiers are
  - u** Only replace files with modified dates later than that of the archive.
  - a** Place new files after *posname* in the archive rather than at the end.
  - b** or **i** Place new files before *posname* in the archive.
- q** Quick. Append *files* to the end of the archive without checking for duplicates. Avoids quadratic behavior in
- t** List a table of contents of the archive. If names are given, only those files are listed.
- p** Print the named files in the archive.
- m** Move the named files to the end or elsewhere, specified as with
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.
- v** Verbose. Give a file-by-file description of the making of a new archive file from the old archive and the constituent files. With **p**, precede each file with a name. With **t**, give a long listing of all information about the files, somewhat like a listing by *ls*(1), showing
  - mode uid/gid size date name**
- c** Create. Normally *ar* will create a new archive when *afile* does not exist, and give a warning. Option **c** discards any old contents and suppresses the warning.
- l** Local. Normally *ar* places its temporary files in the directory `/tmp`. This option causes them to be placed in the local directory.

*Ranlib* makes a table-of-contents file for each library *archive*. With this table the loader *ld*(1) will extract files as if it were repeatedly invoked until no more subroutines can be found.

**EXAMPLES**

`ar cr lib.a *.o; ranlib lib.a`

Replace the contents of library with the object files in the current directory.

`pcat old.a.z | ar t -`

List the contents of an archived and compressed collection of old files; see *pack*(1).

**FILES**

temporaries

**SEE ALSO**

*ld*(1), *ar*(5)

**BUGS**

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

*Ld*(1) warnings that a library is newer than its table of contents happen when a library is copied.

**NAME**

as – assembler

**SYNOPSIS**

as [ *option ...* ] [ *name ...* ]

**DESCRIPTION**

As assembles the named files, or the standard input if no file name is specified. The options are:

- dn** Specifies the number of bytes *n* (1, 2, or 4) to be assembled for offsets which involve forward or external references, and which have sizes unspecified in the assembly language. Default is **-d4**.
- L** Save defined labels that begin with which are normally discarded to save space in the resultant symbol table. The compilers generate such temporary labels.
- V** Use virtual memory for intermediate storage, rather than a temporary file.
- W** Do not complain about errors.
- J** Use long branches to resolve jumps when byte-displacement branches are insufficient. This must be used when a compiler-generated assembly contains branches of more than 32K bytes.
- R** Make initialized data segments read-only, by concatenating them to the text segments. This obviates the need to run editor scripts on assembly code to make initialized data read-only and shared.
- t** Specifies a directory to receive the temporary file, other than the default
- oobj** Place output in file *obj*. Default is

All undefined symbols in the assembly are treated as global.

**FILES**

default temporary file

default object file

**SEE ALSO**

[ld\(1\)](#), [nm\(1\)](#), [adb\(1\)](#), [pi\(9\)](#) [a.out\(5\)](#)

J. F. Reiser and R. R. Henry 'Assembler Reference Manual', *Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version*, 1980, Volume 2C (Berkeley)

**BUGS**

**-J** should be eliminated; the assembler should automatically choose among byte, word and long branches.

**NAME**

as80 assembler for the 8080 and Z80 microprocessors

**SYNOPSIS**

**as80** [ *lhzi* ] name ...

**DESCRIPTION**

*as80* assembles the concatenation of the named files. The output of the assembly is left on the file **80.out**. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

The -l option causes as80 to produce a listing on the standard output.

The -h option causes as80 to produce the output listing in hex. Octal is default.

Register names:	a,b,c,d,e,h,l,af,bc,de,hl,ix,iy,sp
Condition codes:	nz,z,nc,c,po,pe,p,m
Pseudo operations:	.globl,.text.textorg,.data,.dataorg .bss,.bssorg,.byte,.word,.list,

**FILES**

as80 the assembler  
"80.out"  
OPCODES the external instruction set

**DIAGNOSTICS**

When an input file cannot be read, its name and a "can't open" diagnostic is produced and assembly ceases. Whenever syntactic or semantic errors are encountered, a single-character diagnostic is produced. The possible diagnostics are:

[	byte constant error
(	Parentheses error
"	String not terminated properly
E	Illegal expression
R	Illegal register usage
G	Garbage (unknown) character
M	Multiply defined symbol
P	'.' different in pass 1 and 2
T	A 16 bit expression has been truncated to an 8 bit value
U	Undefined symbol
X	Syntax error

**BUGS**

If .list 1 is ever encountered, a listing will start to come out whether or not -l was selected.

**NAME**

asa – interpret ASA control characters

**SYNOPSIS**

**asa** [ *file* ]

**DESCRIPTION**

*Asa* takes files which were written with ASA carriage control characters, usually by FORTRAN programs, converts them to a form suitable for printing on a terminal, line printer, and so on, and writes the results on the standard output.

The control characters handled are:

‘ ‘	single space
‘0’	double space
‘-’	triple space
‘+’	overprint the previous line
‘1’	start a new page

If no file names are given, the standard input is used.

Each input file given starts a new page. A skip to a new page on the first line of the first input file is ignored.

**NAME**

ascii – interpret ASCII characters

**SYNOPSIS**

**ascii** [ **-oxdbn** ] [ **-nct** ] [ **-e** ] [ *text* ]

**DESCRIPTION**

*Ascii* prints the ASCII values corresponding to characters and *vice versa*. The values are interpreted in a settable numeric base; **-o** specifies octal (the default), **-d** decimal, **-x** hexadecimal, and **-bn** base *n*.

With no arguments, *ascii* reproduces in the specified base. Characters of *text* are converted to their ASCII values, one per line. If, however, the first *text* argument is a valid number in the specified base, conversion goes the opposite way. Control characters are printed as they appear in Other options are:

- n** Force numeric output.
- c** Force character output.
- t** Convert from numbers to running text; do not interpret control characters or insert newlines.
- e** Interpret remaining arguments as *text*.

**EXAMPLES**

Print the

ASCII table base 10.

Print the octal value of 'p'.

Show which character is octal 160.

**SEE ALSO**

[ascii\(6\)](#)



**NAME**

at – execute commands at a later time

**SYNOPSIS**

at [ -r ] *time* [ *day* ] [ *file* ]

at -l

**DESCRIPTION**

At squirrels away a copy of the named *file* (standard input default) to be used as input to [sh\(1\)](#) at a specified later time. A *cd* command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the *userid* and *groupid* of the creator of the copy.

The *time* is 1 to 4 digits, with an optional following or for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional *day* is either a month name followed by a day number, or a day of the week; if the word follows, invocation is moved seven days further off. Names of months and days may be recognizably truncated. A year number, spelled out in full, may follow the month.

The options are

- r Remove the specified activity.
- l List all activities scheduled for this user.

At programs are executed by periodic execution of from [cron\(8\)](#). The granularity of *at* depends upon how often *atrun* is executed.

The standard output and standard error files are lost unless redirected.

**EXAMPLES**

```
at 0800 dec 24
echo ho ho ho | mail claus
at -r 'at -l'
    Remove a scheduled activity.
```

**FILES**

*/usr/spool/at/yy.ddd.hhmm.\**  
activity for year, day, hour

last *hhmm*  
activities in progress

**SEE ALSO**

[calendar\(1\)](#), [pwd\(1\)](#), [sleep\(1\)](#), [cron\(8\)](#)

**BUGS**

Due to the granularity of the execution of *atrun*, there may be bugs in scheduling things almost exactly 24 hours into the future.

**NAME**

awk – pattern-directed scanning and processing language

**SYNOPSIS**

```
awk [ -F fs ] [ -v var=value ] ... [ -f ] prog [ file ... ]
```

**DESCRIPTION**

*Awk* scans each input *file* for lines that match any of a set of patterns in *prog*, which may appear as one literal argument or as one or more file names each preceded by **-f**. With each pattern may be associated an action to be performed when a line of a *file* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement in order; the associated action is performed for each matched pattern. The file name means the standard input. Any *file* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. Option **-v** designates an assignment to be done before *prog* is executed.

An input line is made up of fields separated by white space, or by regular expression **FS**. The fields are denoted **\$1**, **\$2**, ..., while **\$0** refers to the entire line.

A pattern-action statement has the form

```
pattern { action }
```

A missing { *action* } means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

An action is a sequence of statements. A statement can be one of the following:

```
if( expression ) statement [ else statement ]
while( expression ) statement
for( expression ; expression ; expression ) statement
for( var in array ) statement
do statement while( expression )
break
continue
{ [ statement ... ] }
expression           # commonly var = expression
print [ expression-list ] [ > expression ]
printf format [ , expression-list ] [ > expression ]
return [ expression ]
next                 # skip remaining patterns on this input line
delete array[ expression ] # delete an array element
exit [ expression ]     # exit immediately; status is expression
```

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for **\$0**. String constants are quoted " ", with the usual C escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators + - \* / % ^ (exponentiation), and concatenation (indicated by white space). The operators ! ++ -- += -= \*= /= %= ^= > >= < <= == != && || ?: are also available in expressions. Variables may be scalars, array elements (denoted *x*[*i*]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as [*i,j,k*] are permitted; the constituents are concatenated, separated by the value of **SUBSEP**.

The **print** statement prints its arguments on the standard output (or on a file if >*file* or >>*file* is present or on a pipe if |*cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The **printf** statement formats its expression list according to the format (see [printf\(3\)](#)). The built-in function **close(*expr*)** closes the file or pipe *expr*.

The mathematical functions **exp**, **log**, **sqrt**, **sin**, **cos**, and **atan2** are built in. Other built-in functions:

**length**

the length of its argument taken as a string, or of **\$0** if no argument.

- rand** random number on (0,1)
- srand** sets seed for **rand** and returns the previous seed.
- int** truncates to an integer value
- substr**(*s, m, n*)  
the *n*-character substring of *s* that begins at position *m* counted from 1.
- index**(*s, t*)  
the position in *s* where the string *t* occurs, or 0 if it does not.
- match**(*s, r*)  
the position in *s* where the regular expression *r* occurs, or 0 if it does not. The variables **RSTART** and **RLENGTH** are set to the position and length of the matched string.
- split**(*s, a, fs*)  
splits the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator **FS** if *fs* is not given.
- sub**(*r, t, s*)  
substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, **\$0** is used.
- gsub** same as **sub** except that all occurrences of the regular expression are replaced; **sub** and **gsub** return the number of replacements.
- sprintf**(*fmt, expr, ...*)  
the string resulting from formatting *expr ...* according to the *printf*(3) format *fmt*
- system**(*cmd*)  
executes *cmd* and returns its exit status

The “function” **getline** sets **\$0** to the next input record from the current input file; **getline <file** sets **\$0** to the next record from *file*. **getline x** sets variable *x* instead. Finally, *cmd* | **getline** pipes the output of *cmd* into **getline**; each call of **getline** returns the next line of output from *cmd*. In all cases, **getline** returns 1 for a successful input, 0 for end of file, and -1 for an error.

Patterns are arbitrary Boolean combinations (with **!** || **&&**) of regular expressions and relational expressions. Regular expressions are as in *egrep*; see *gre*(1). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators **~** and **!~**. */rel/* is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

*expression matchop regular-expression*  
*expression relop expression*  
*expression in array-name*  
*(expr,expr,...) in array-name*

where a *relop* is any of the six relational operators in C, and a *matchop* is either **~** (matches) or **!~** (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns **BEGIN** and **END** may be used to capture control before the first input line is read and after the last. **BEGIN** and **END** do not combine with other patterns.

Variable names with special meanings:

- FS** regular expression used to separate fields; also settable by option **-F fs**.
- NF** number of fields in the current record
- NR** ordinal number of the current record

**FNR** ordinal number of the current record in the current file

**FILENAME**

the name of the current input file

**RS** input record separator (default newline)

**OFS** output field separator (default blank)

**ORS** output record separator (default newline)

**OFMT**

output format for numbers (default **%.6g**)

**SUBSEP**

separates multiple subscripts (default 034)

**ARGC**

argument count, assignable

**ARGV**

argument array, assignable; non-null members are taken as filenames

**ENVIRON**

array of environment variables; subscripts are names.

Functions may be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global. Thus local variables may be created by providing excess parameters in the function definition.

**EXAMPLES**

```
length > 72
```

Print lines longer than 72 characters.

```
{ print $2, $1 }
```

Print first two fields in opposite order.

```
BEGIN { FS = ",[ \t]*|[ \t]+"
```

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs.

```
{ s += $1 }
```

```
END { print "sum is", s, " average is", s/NR }
```

Add up first column, print sum and average.

```
/start/, /stop/
```

Print all lines between start/stop pairs.

```
BEGIN { # Simulate echo(1)
```

```
for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
```

```
printf "\n"
```

```
exit }
```

**SEE ALSO**

[gre\(1\)](#), [lex\(1\)](#), [sed\(1\)](#)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

**BUGS**

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate " " to it.

The scope rules for variables in functions are a botch; the syntax is worse.

**NAME**

backup – backup and recover files

**SYNOPSIS**

**backup recover** [ *option ...* ] *file ...*  
**backup grep** [ *option ...* ] *pattern ...*  
**backup fetch** [ *option ...* ] [ *file ...* ]  
**backup stats** [ *option ...* ]  
**backup backup** [ *file ...* ]  
**backup munge**  
**backup mount** [ *option ...* ] *mountpt*

**DESCRIPTION**

The *backup* programs save and restore archival copies of files in an optical disk store on a central system (see *backup(5)*). Backup occurs automatically daily (see *backup(8)*) and upon specific request via *backup backup*. *Backup grep* shows backup copy names for specific files, and *backup fetch* restores data from specific backup copies. *Backup recover* is a combination of these two; it fetches the most recent copy. All the *backup* programs describe their options when presented with a bad option such as *-?*.

*Backup recover* retrieves *files* by name. The names should be full pathnames rooted at */n/*; if not, *backup* tries to guess names that begin with */n/*. Directories should be recovered before their contents. Regular files that are linked together will stay linked if they are recovered together. The options for *recover* are:

- o dir** The argument is restored as an entry in the directory *dir*.
- v** Verbose (enforced).
- F** Restore directories as files containing a null-terminated list of element names.
- r** Recursively recover any subdirectories.
- d** Create any missing intermediate directories.
- Dold=new**  
Replace the prefix *old* of the original filename with *new* to form the new output filename.
- m** The names are backup copy names, as determined from *backup grep*, not original filenames.
- fdevice**  
Use *device* rather than */dev/worm0* for the WORM. *Device* may be on another machine: *machine!device*. An initial **w** implies a WORM device; a **j** implies a jukebox. A numeric *device* means */dev/wormdevice*.
- e** Cause the *worm fetch* server on the backup system to terminate gracefully.
- i** Append *.n* to the output name for each file where *n* is an increasing integer. This is useful for recovering multiple copies of the same file.

A diagnostic like **need disk backup2a** means you need to mount the A side of the cartridge labeled **backup2**.

*Backup grep* searches for names of backed up files that match the strings *patterns*. If the pattern is a literal (no **-e**) that looks like a filename, it reports the filename catenated with *//* and the time of the most recent backup copy. If the pattern is a literal that looks like the output under option **-d**, it reports the name of the corresponding backup copy. The options are:

- d** Print file change times (**ctime**, see *stat(2)*) as integers rather than as dates.
- e** Interpret *patterns* as regular expressions given in the notation of *regex(3)*. Warning: this option can execute extremely slowly; it is almost always better to use *gre(1)* on the backup machine; see *backup(5)*.
- a** Print all names in the database.
- V** Treat *pattern* as a literal filename and list all versions of the file.
- <n** Only list entries with a date less than or equal to *n*. If *n* is not a simple integer date, it is interpreted as by *timec(3)*.
- >n** Only list entries with a date greater than or equal to *n*.
- D** Print the most recent entry for every file name starting with *pattern*, taking into account any cut-off date, but turning off option **-e**.

*Backup fetch* takes from its arguments or from standard input backup copy names as reported by *backup grep* (such as **v2345/987**) and restores the corresponding files. It accepts the same options as *backup recover* except **-m**; **-v** is really optional. Irrelevant prefixes are stripped from backup copy names. Thus the output of the *backup grep* command can be used directly.

*Backup stats* provides statistics about the files backed up. By default, it looks for all systems and all users and gives a grand total. The options are

**-i** Give information per system or user rather than a total.

**-s** *systems*

**-u** *users*

With option **-i**, restrict the total to the systems or users named in comma-separated lists. The name expands to all systems or all users.

**-d** Print average number of files and bytes for the last 1 day, 7 days and 30 days.

*Backup backup* backs up files. If no file names are given, they are taken from standard input. File names are interpreted as in *backup recover*. The files are safely on the backup system when the command exits but will normally take a day to get into the backup database.

*Backup munge* causes the backup system to process any received files. When this terminates (assuming no errors), the files have been put onto backup media and have been absorbed into the database.

*Backup mount* is an experimental way to access backed up files. The specified part of the backup files (set by **-Droot** or **/** by default) is mounted at *mountpt*. There is one option

**-d** *date*

Make the mounted hierarchy reflect the state at the given date. The mounting can be reversed with *umount*; see [mount\(8\)](#).

## EXAMPLES

```
backup stats -i -s '*'
```

Get totals for all systems.

```
backup fetch `backup grep -d `backup grep -d /n/bowell/etc/passwd`
```

What *backup recover* does for you.

```
backup recover /n/coma/usr/rob/fortunes
```

```
cd /n/coma/usr/rob; backup recover fortunes
```

Two ways to get the latest available copy of **/n/coma/usr/rob/fortunes**.

```
backup grep -V /n/coma/usr/rob/fortunes
```

List all available copies of **/n/coma/usr/rob/fortunes** with their dates.

```
backup recover -m -o /tmp /n/wild/usr/backup/v/v919/678
```

```
backup recover -m -o /tmp v919/678
```

Two ways to recover a specific backup copy and place the result in **/tmp**. **/n/wild/usr/backup/v/v919/678** is the name of the backup copy; the file will be restored to its home machine, not to **wild**.

```
backup grep -V /n/coma/usr/rob/fortunes | backup fetch -i -o .
```

Recover all the versions of the fortunes file into **fortunes.1**, **fortunes.2**, ... in the current directory.

## FILES

home of all datafiles and executables (on client machines)

## SEE ALSO

[worm\(8\)](#), [backup\(5\)](#), [backup\(8\)](#)

## BUGS

Recovery via symbolic links may not work; use the non-linked pathname.

**NAME**

badge – print Bell Labs badge

**SYNOPSIS**

**badge** *string1 string2 [ picture.ps ]*

**DESCRIPTION**

*Badge* is an ASCII-to-PostScript converter that frames its arguments in a pleasant, colorful badge, suitable for laminating. File *picture.ps* is assumed to contain 24-bit color encapsulated PostScript. If it is omitted or doesn't exist, a blank box will appear; **goofy** and **donald** (see FILES) are available for the camera-shy.

**EXAMPLES**

```
badge "DONALD F" DUCK /usr/games/ps/donald | lp -dpeacock
```

```
badge "P J" "WEINBERGER" /usr/games/ps/goofy | lp -dpeacock
```

**BUGS**

Very long names (over 1.75 inches in 14 point type) are not accounted for.

Unlike a similar badge provided by security, the logo adheres to corporate standards.

**FILES**

/usr/games/ps/donald

/usr/games/ps/goofy

/usr/games/ps/logo      A corporate logo.

**NAME**

basename, dirname – strip filename affixes

**SYNOPSIS**

**basename** *string* [ *suffix* ]

**dirname** *string*

**DESCRIPTION**

These functions split off useful parts of a pathname; they are typically used inside substitution marks ‘ ‘ in shell scripts.

*Basename* deletes any prefix ending in and the *suffix*, if present in *string*, from *string*, and prints the result on the standard output.

*Dirname* places on standard output the name of the directory in which a file named *string* would nominally be found. The calculation is syntactic and independent of the contents of the file system.

**EXAMPLES**

```
cc $1 -o 'basename $1 .c'
```

Compile into where is or

```
cc $1 -o 'dirname $1'/'basename $1 .c'
```

Compile into

**SEE ALSO**

[sh\(1\)](#)



**NAME**

basic, bas, bite – basic language interpreters

**SYNOPSIS**

**/usr/bin/lcl/basic**

**/usr/bin/lcl/bas**

**/usr/bin/lcl/bite**

**DESCRIPTION**

Of these three completely different Basic interpreters, *basic* is the biggest, and unfortunately the best. Caveat emptor.

**NAME**

bc – arbitrary-precision arithmetic language

**SYNOPSIS**

**bc** [ **-c** ] [ **-l** ] [ *file ...* ]

**DESCRIPTION**

*Bc* is an interactive processor for a language that resembles C but provides arithmetic on numbers of arbitrary length with up to 100 digits right of the decimal point. It takes input from any files given, then reads the standard input. The **-l** argument stands for the name of an arbitrary precision math library. The following syntax for *bc* programs is like that of C; *L* means letter **a-z**, *E* means expression, *S* means statement.

## Lexical

comments are enclosed in */\* \*/*

newlines end statements

## Names

simple variables: *L*

array elements: *L[E]*

The words **ibase**, **obase**, and **scale**

## Other operands

arbitrarily long numbers with optional sign and decimal point.

*(E)*

**sqrt***(E)*

**length***(E)*

number of significant decimal digits

**scale***(E)*

number of digits right of decimal point

*L(E,...,E)*

## Operators

**+** **-** **\*** **/** **%** **^** (**%** is remainder; **^** is power)

**++** **--** (prefix and postfix; apply to names)

**==** **<=** **>=** **!=** **<** **>**

**=** **+=** **-=** **\*=** **/=** **%=** **^=**

## Statements

*E*

**{** *S* ; ... ; *S* **}**

**if** (*E*) *S*

**while** (*E*) *S*

**for** (*E* ; *E* ; *E*) *S*

null statement

**break**

**quit**

"text"

## Function definitions

**define** *L* (*L* , ... , *L*) {

**auto** *L* , ... , *L*

*S* ; ... ; *S*

**return** (*E*)

}

Functions in

- l** math library
- s(x)** sine
- c(x)** cosine
- e(x)** exponential
- l(x)** log
- a(x)** arctangent
- j(n,x)** Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Text in quotes, which may include newlines, is also printed. Either semicolons or newlines may separate statements. Assignment to **scale** influences the number of digits to be retained on arithmetic operations in the manner of [dc\(1\)](#). Assignments to **ibase** or **obase** set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. Automatic variables are pushed down during function calls. In a declaration of an array as a function argument or automatic variable empty square brackets must follow the array name.

*Bc* is actually a preprocessor for [dc\(1\)](#), which it invokes automatically, unless the **-c** (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

## EXAMPLES

Define a function to compute an approximate value of the exponential. Use it to print 10 values. (The exponential function in the library gives better answers.)

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; 1==1; i++){
        a = a*x
        b = b*i
        c = a/b
        if(c == 0) return(s)
        s = s+c
    }
}
for(i=1; i<=10; i++) e(i)
```

## FILES

mathematical library

## SEE ALSO

[dc\(1\)](#), [hoc\(1\)](#)

## BUGS

No or operators.

A statement must have all three

A is interpreted when read, not when executed.

**NAME**

**bcp** – reformat black-and-white picture files

**SYNOPSIS**

**bcp** [ *option ...* ] [ *file1* [ *file2* ] ]

**DESCRIPTION**

*Bcp* copies black-and-white (B&W) image *file1* to *file2*, optionally changing the file format and transforming the image. If *file1* is a directory name, then every leaf of its file tree is processed in turn; and, in this case, if *file2* also is specified, it is made the root directory of an isomorphic tree of output files. If *file2* is not specified, all output is catenated to stdout.

*Bcp* can copy among all the B&W *picfile(5)* formats, and some others. The default output format is **TYPE=ccitt-g4**. Image transformations include trimming, translation, scaling, and rotation, performed in that order (not in argument order).

Input files in *picfile(5)* format must begin with an ASCII **TYPE=type** header line. *Types* supported both as input and output are:

- dump** One byte/pixel. **NCHAN=1** is required. On input, the grey pixel values are thresholded to B&W; see option **-T**. On output, black becomes 0 and white 255.
- bitmap** One bit/pixel. Essentially Sun rasterfile format, with a *picfile(5)* header replacing the Sun binary header.
- ccitt-g4** CCITT Group 4 FAX encoding, strongly compressive on printed text. Also, **ccitt-g31** (Group 3, 1-dim) and **ccitt-g32** (Group 3, 2-dim; see **-k**).

Other supported *types* are:

- binary** One bit/pixel encoding; obsolescent, but needed for old image archives. Both input and output.
- rle** Fast run-length encoding; obsolescent, but needed for old image archives. Input or output, but not both.
- pico** Same as **dump**. Input only.
- cdf** ‘Compound document format’, used in AT&T FAX Connection product. Input only. Only the first of multiple pages is read.

Other formats not using a **TYPE=type** header, are: *bitfile(9)* format; tiff 5.0 format; PostScript bitmap format (output only); and Sun rasterfile format. Input tiff files may be encoded using the Group 3 or Group 4 schemes, LZW algorithm, modified Huffman encoding, Apple PackBits, or uncompressed. Sun rasterfiles may be encoded using no compression, or the byte-length encoding scheme.

The options are:

- B[io]** Read/write *bitfile(9)* format (no **TYPE=type** header).
- Fc** Write tiff format, compression scheme *c*, where *c* is *g3* or *g31* (1-dim Group 3), *g32* (2-dim Group 3), *g4* (Group 4), *L* (LZW compression), *P* (Apple PackBits), or *N* (no compression).
- M** Write **TYPE=bitmap** format.
- P** Write Postscript format.
- R<sub>x,y</sub>** Force output resolutions to *x,y* (pixels/inch). If *y* is missing, it is taken to be the same as *x*. Overrides **-x<sub>x,y</sub>**. Requires a **RES=x y** line in the header (but, see **-Z**).
- R=** Force the output resolution to be equal to the greater of the input resolutions.
- S** Write Sun rasterfile format (standard encoding).
- T<sub>t</sub>** Threshold. When reading **TYPE=dump**, assign black to grey levels less than *t*, and white to others. Default: **-T128**.
- Z<sub>x,y</sub>** Force input **RES=x y**.

- b** Write **TYPE=binary** format.
- g4**
- 4** Write **TYPE=ccitt-g4** format. Similarly, **-g31** or **-31** and **-g32** or **-32**.
- kn** Set the 'k' for **ccitt-g32** encoding on output (default **-k4**).
- o.x,y** Offset (translate) the image by *x,y* pixels. The width and height of the picture are not changed.
- p** Write **TYPE=dump NCHAN=1** format. Map black to 0, white to 255.
- r** Write **TYPE=rle** format.
- tl** Rotate the image to bring the left edge of the page to the top. Set top-left corner of the rotated image at the top-left corner of the image.
- td** Rotate the image *d* degrees counterclockwise about its center. *d* is a real number.
- wl,t,r,b** Specify window (trim the image): *l,t* is the left-top corner and *r,b* the right-bottom corner measured in pixels. If the new margins are outside the original picture, the new area is set to white. An argument given as leaves the edge unchanged.
- x.x,y** Expand/contract (scale) the image, by real factors *x* and *y*. If *y* is missing, *y* is taken to be the same as *x*. May be overridden by **-R.x,y**. Requires a **RES=x y** line in the header (but, see **-Z**).

**SEE ALSO**

[cscan\(1\)](#), [imscan\(1\)](#), [ocr\(1\)](#), [pico\(1\)](#), [picfile\(5\)](#)

CCITT facsimile coding standards Rec. T.4(1988) and T.6(1988).

**BUGS**

Concatenated pages are supported, but only if each new page has a complete header.

Scaling is accomplished by naive replication/deletion of pixels.

Rotation by small angles exhibits aliasing effects, and is slow.

Rotations **-tr** and **-tb** are unfinished.

CCITT FAX 'uncompressed' (or, 'transparent') mode is not implemented.

Postscript output is useful only for small images.

**WINDOW=l t r b** where *l* or *t* is non-zero may not be handled correctly for every combination of file types.

tiff LZW compression may not be working properly (input and output).

**TYPE=rle** can't be both input and output.

Should be merged with T. Duff's *pcp*.

**NAME**

`bdiff` – big diff

**SYNOPSIS**

`bdiff` file1 file2 [n] [-s]

**DESCRIPTION**

*Bdiff* is used in a manner analogous to *diff(1)* to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is `-`, the standard input is read. The optional `-s` (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*). If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

**FILES**

`/tmp/bd????`

**SEE ALSO**

*diff(1)*.

**DIAGNOSTICS**

Use *help(1)* for explanations.

**NAME**

*bigcore*, *coreid* – permit big core images, identify source of image

**SYNOPSIS**

***bigcore*** *command*

***coreid*** [ *file* ]

**DESCRIPTION**

*Bigcore* calls the named *command* with no restriction on the size of core images. By default, no core image will be written when a program aborts if it would be larger than a megabyte.

*Coreid* reads the file or the specified core image *file*, and prints on the standard output the argument list of the program that produced the core image.

**FILES****SEE ALSO**

[core\(5\)](#)

**BUGS**

*Coreid* only works for core images from C and Fortran programs.

The arguments shown are those at the time of the dump, not at invocation of the program.

**NAME**

bison – GNU Project parser generator (yacc replacement)

**SYNOPSIS**

**bison** [ **-dvy** ] file

**DESCRIPTION**

*Bison* is a parser generator in the style of [yacc\(1\)](#). It should be upwardly compatible with input files designed for *yacc*.

Input files should follow the *yacc* convention of ending in “.y”. Unlike *yacc*, the generated files do not have fixed names, but instead use the prefix of the input file. For instance, a grammar description file named **parse.y** would produce the generated parser in a file named **parse.tab.c**, instead of *yacc*’s **y.tab.c**.

*Bison* takes three optional flags.

- d**     Produce a **.tab.h** file, similar to *yacc*’s **y.tab.h** file.
- v**     Be verbose. Analogous to the same flag for *yacc*.
- y**     Use fixed output file names. I.e., force the output to be in files **y.tab.c**, **y.tab.h**, and so on. This is for full *yacc* compatibility.

**FILES**

/usr/lib/bison.simple	simple parser
/usr/lib/bison.hairy	complicated parser

**SEE ALSO**

[yacc\(1\)](#)

**DIAGNOSTICS**

“Self explanatory.”



**NAME**

bitship – convert file to or from visible representation

**SYNOPSIS**

**bitship** [ **-a** | **-b** ]

**DESCRIPTION**

*Bitship -a* pipes an arbitrary file into a visible ASCII-95 representation. *Bitship -b* performs the inverse transformation. If you are sending a file to someone for the first time, you should probably include a copy of the source code.

**FILES**

/usr/src/cmd/bitship.c

**DIAGNOSTICS**

"Usage:..." in case of an error in the command line.

**BUGS**

There is no error correction. Illegal characters in a "visible" file produce garbage.

**NAME**

bmd08v – Analysis of Variance

**SYNOPSIS**

**bmd08v**

**DESCRIPTION**

*Bmd08v* performs analysis of variance for any hierarchical design with equal cell sizes. This includes the nested, partially nested and partially crossed, and fully crossed designs. Separate analyses may be performed on several dependent variables simultaneously. *Bmd08v* takes its input from the standard input and writes its results on the standard output. All bmd control cards must have the identifying field in upper case (eg. PROBLM, INDEX, FINISH, etc.). One important departure from previous versions concerns the variable format card – it is no longer necessary! If you specify 0 for the number of variable format cards, the data will be assumed to be in "free format" – items separated by blanks or commas.

**EXAMPLES**

In the first example, *data* contains the bmd control cards and the input data. Output is directed to the standard output.

```
bmd08v <data
```

In the second example, *prefix* contains the initial bmd control cards, *data* contains the input data, and *finish* contains the FINISH card (many users prefer not to contaminate their data files). Output is directed to file **output**.

```
cat prefix data finish | bmd08v >output
```

**FILES**

```
/tmp/1?????    temporary file  
/tmp/2?????    temporary file
```

**SEE ALSO**

BMD User's Guide, /usr/doc/bmd08v

**BUGS**

Temporary files are not always scratched.

**NAME**

`bprint` – expression profiler

**SYNOPSIS**

**bprint** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*bprint* produces on the standard output a listing of the programs compiled by *lcc* with the **-b** option. Executing an **a.out** so compiled appends profiling data to **prof.out**. The first token of each expression in the listing is preceded by the number of times it was executed enclosed in angle brackets as determined from the data in **prof.out**. *bprint* interprets the following options.

- c**      Compress the **prof.out** file, which otherwise grows with every execution of **a.out**.
- b**      Print an annotated listing as described above.
- n**      Include line numbers in the listing.
- f**      Print only the number of invocations of each function. A second **-f** summarizes call sites instead of callers.
- Idir**   specifies additional directories in which to seek files given in **prof.out** that do not begin with `'/'`.

If any file names are given, only the requested data for those files are printed in the order presented. If no options are given, **-b** is assumed.

**FILES**

<code>prof.out</code>	profiling data
<code>/usr/lib/bbexit.o</code>	creates <b>prof.out</b> when <b>a.out</b> exits

**SEE ALSO**

*lcc*(1), *prof*(1)

**BUGS**

Macros and comments can confuse *bprint* because it uses post-expansion source coordinates to annotate pre-expansion source files. If *bprint* sees that it's about to print a statement count *inside* a number or identifier, it moves the count to just *before* the token.

Can't cope with an ill-formed **prof.out**.

**NAME**

`bundle` – collect files for distribution

**SYNOPSIS**

`bundle file ...`

**DESCRIPTION**

*Bundle* writes on its standard output a shell script for *sh*(1) that, when executed, will recreate the original *files*. Its main use is for distributing small numbers of text files by *mail*(1).

Although less refined than standard archives from *ar*(1), *cpio*(1), or *tar*(1), a *bundle* file is self-documenting and complete; little preparation is required on the receiving machine.

**EXAMPLES**

```
bundle makefile *.ch | mail elsewhere!mark
```

Send a makefile to Mark together with related and files. Upon receiving the mail, Mark may save the file sans postmark, say in **gift/horse**, then do

```
cd gift; sh horse; make
```

**SEE ALSO**

*ar*(1), *cpio*(1), *tar*(1), *mail*(1)

**BUGS**

*Bundle* will not create directories and is unsatisfactory for non-ASCII files.  
Beware of gift horses.

**NAME**

byteyears – time-space product for file residency

**SYNOPSIS**

**byteyears** [ **-a** ] [ *file* ]

**DESCRIPTION**

*Byteyears* reports the product of the age of each *file* in years and the length in bytes. Files for which this number is large may be reasonable candidates for deletion. If the file is a directory, *byteyears* reports (recursively) on everything in that directory. If no arguments are given, the current directory is assumed. If the **-a** option is given, the time since last access is used instead of the time since last modification.

Each line of output contains the number of byte years (rounded to the nearest integer), the size of the file in bytes, the time last modified, and the name of the file.

**EXAMPLES**

```
byteyears | sort -r | sed 10q
```

List the ten leading candidates in the current directory.

**NAME**

CC, cfront – C++ compiler

**SYNOPSIS**

CC [ *option ...* ] *file ...*

**cfront** [ *option ...* ] *file ...*

**DESCRIPTION**

CC compiles and links C++ programs in the manner of [cc\(1\)](#). It handles source files with names ending in assembler files in and object files in Various passes of the compiler can be substituted via environment variables listed under 'FILES'. Options include those of [cc\(1\)](#) except **-B** and **-t**, those of [ld\(1\)](#), those of [cfront](#), and in addition

**-F** Run only the macro preprocessor [cpp\(8\)](#) and [cfront](#) on the named **.c** files, and send the result to the standard output.

**-.suffix**

Instead of the standard output, place **-E** and **-F** output in files whose name is that of the source with **.suffix** substituted for

[Cfront](#) reads C++ code (without preprocessing) from the standard input and writes equivalent C code on the standard output. The options are

**+d** Don't expand inline functions.

**+xfile** Take size and alignment information from *file* for cross compiling.

**+e0**

**+e1** Make external declarations (**+e0**) or definitions (**+e1**) for virtual function tables. These tables may appear as static data in every compilation; the options are intended to save redundant space.

**+a0** Produce classic C output (default).

**+a1** Produce ANSI C output. If this option is used with [CC](#), then an ANSI C compiler such as [lcc](#) must be specified in environment variable **ccC**.

**+fname**

Use *name* to identify the source file in diagnostics.

**+L** Produce ANSI standard **#line** directives instead of **#number**.

**FILES**

**cppC=/lib/cpp**

C preprocessor

**cfrontC=/usr/bin/cfront**

C++ translator

**ccC=/bin/cc**

C compiler

**munchC=/usr/lib/munch**

linker postprocessor for static initialization

C++ library

standard directory for C++  
files

Other files as in [cc](#)

**SEE ALSO**

[cc\(1\)](#), [ld\(1\)](#)

B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1986

B. Stroustrup, *C++ Reference Manual*, AT&T Bell Laboratories, May 1989

**NAME**

calendar – reminder service

**SYNOPSIS**

**calendar** [ - ] [ *n* ]

**DESCRIPTION**

*Calendar* consults calendar files and prints out lines that contain today's date or any date up through the *n*th working day hence (*n*=1 by default). Most American-style month-day dates such as **Aug. 19**, **august 19**, **8/19**, etc., are recognized, but not The symbol denotes every month as in \* 19 or \*/19. A year may follow the day, as in **August 19 86**, **8/19/86**, or **Aug. 19, 1986**.

By default, the program consults the file in directory **\$HOME** (see [sh\(1\)](#)), or in the current directory if a home directory is not known. Other calendar files to be consulted may be specified by calendar lines in one of the forms

```
#include file
#include machine!file
```

where *file* is the name of some other calendar and *machine* is the name of a machine or service accessible via [con\(1\)](#).

When the optional argument is present, *calendar* reminds all users of their calendar engagements by [mail\(1\)](#). Normally this happens daily in the wee hours under control of [cron\(8\)](#). Calendars not in home directories, or recipients not registered as users, may be registered for reminder service by placing lines of the form *calendarfile mailname* in file

**EXAMPLES**

```
#include /usr/pub/btlcalendar
#include /n/coma/usr/pub/btlcalendar
#include mh/astro/coma.calendar!/usr/pub/btlcalendar
```

Ways to subscribe to a public calendar by (1) users of (2) users elsewhere who have [netfs\(8\)](#) access to **coma**, and (3) users elsewhere without *netfs* access.

**FILES****SEE ALSO**

[at\(1\)](#)

**BUGS**

**#includes** do not nest.

The mail reminder service doesn't work when it finds fewer than two calendars.

Your calendar must be public information for you to get reminder service.

Holidays are what the program says they are.

**NAME**

can, bcan, dcan, tcan, xcan – interface to Canon laser-printer spooler

**SYNOPSIS**

**can** [ *option ...* ] [ *file ...* ]

**bcan** [ *option ...* ] [ *file ...* ]

**dcan** [ *option ...* ] [ *file ...* ]

**tcan** [ *option ...* ] [ *file ...* ]

**xcan** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

These commands print *files* (standard input by default) on Canon laser printers. Four commands, all special cases of the generic *xcan*, handle particular kinds of data files:

*can* ASCII text

*bcan* bitmap images created by *blitblt(9)*

*dcan* output from *troff(1)*

*tcan* output for a Tektronix 4014 terminal, as produced by *plot(1)*

The destination printer is determined in the following ways, listed in order of decreasing precedence.

option **-d** *dest*  
 environment variable  
 printer named in file

Printers at the mother site are:

**3** 3rd floor, end of 9S corridor (seki)  
**4** 4th floor, stair 8 (swift)  
**8** 3rd floor, stair 8 (tukey)  
**9** 4th floor, stair 9 (wild)  
**j** jones room (jones)  
**u** unix room (panther)  
*/name* printer attached to machine with Datakit destination *name*

Options:

**-d** *dest*

Select the destination printer.

**-f** *font* Set the font (default for *can*; see *font(7)*)

**-L** (landscape) Rotate *bcan* pages 90 degrees.

**-l** *n* Set number of lines per page for *can* (default 66).

**-m** *n* Set *bcan* magnification (default 2).

**-n** Spool only, input has already been formatted by a remote *xcan*.

**-o** *list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *n-m* means pages *n* through *m*; a range *-b* means from the beginning to page *n*; a range *n-* means from page *n* to the end. **-o** implies **-r**.

**-r** print pages in reverse order (default for *can* and *dcan*).

**-sb** make *xcan* expect *bcan* input; **-sb** implies defaults of **-x176** and **-y96**.

**-sc** make *xcan* expect *can* input; **-sc** does not imply **-r**.

**-sd** make *xcan* expect *dcan* input; **-sd** does not imply **-r**.

**-st** make *xcan* expect *tcan* input.

**-t** *n* *tcan* scale factor is  $(n/100)/(n\%100)$ . The default is 813, i.e., 13 tekpoints become 8 dots on the laser printer.



- u** *user* set the name which appears on the banner page; default is login name.
- x** *n* set the horizontal offset of the print image, measured in dots (default 48). There are 240 dots to the inch.
- y** *n* set the vertical offset of the print image (default 0), except in *tcan*, where this option specifies *n* extra tekpoints vertically.

**FILES**

default destination  
font directory  
spool directory

**SEE ALSO**

[pr\(1\)](#), [lpr\(1\)](#), [blitblt\(9\)](#) [plot\(1\)](#), [font\(7\)](#)

**BUGS**

The 'landscape' option is supported only by *bcan*; **-o** and **-r** are supported only by *can* and *dcan*. There ought to be a way to determine the service class from the input data.

**NAME**

cat – catenate and print

**SYNOPSIS**

cat [ *file ...* ]

**DESCRIPTION**

*Cat* reads each *file* in sequence and writes it on the standard output. Thus

```
cat file
```

prints a file and

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no *file* is given, or if the argument is encountered, *cat* reads from the standard input. Output is buffered in blocks matching the input.

**SEE ALSO**

[p\(1\)](#), [pr\(1\)](#), [cp\(1\)](#)

**BUGS**

Beware of `>` and `>>` which destroy input files before reading them.

**NAME**

`cb` – C program beautifier

**SYNOPSIS**

`cb` [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

`Cb` reads C programs either from the named *files* or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. The options are:

**-s** Place newlines as in Kernighan and Ritchie. (Original newlines are preserved by default.)

**-j** Join split lines.

**-l *leng***  
Split lines that are longer than *leng*, 120 by default.

**SEE ALSO**

[pr\(1\)](#), [troff\(1\)](#), [lp\(1\)](#), [font\(6\)](#)

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1988.

**BUGS**

Punctuation hidden in preprocessor statements causes `cb` to make indentation errors.

**NAME**

cbt – btree utilities

**SYNOPSIS**

```
cbt creat name ...
cbt report name ...
cbt cat [ -R ] name ...
cbt squash [ -o dest ] name
cbt build [ -R ] name
cbt add [ -R ] name
cbt delete [ -R ] name
cbt look [ -R ] name
```

**DESCRIPTION**

A B-tree *name* corresponds to a pair of files named *name.T* and *name.F*. *Name.T* contains an access tree, and *name.F* contains the data.

The version of the command creates empty B-trees.

The version of the command scans each B-tree and reports how many records it contains.

The version of the command scans the B-tree in key-sort order, writing on its standard output. With no option, writes each key followed by a tab, followed by the record, followed by a new-line. If option **-R** (raw) is present, each key-record pair has the format

```
struct {
    short keylen;
    char key[keylen];
    short reclen;
    char rec[reclen];
};
```

Keys and records are not null-terminated and consecutive key-record pairs are not separated by new-lines. Keys may be no longer than 255 bytes.

The version of the command compresses the access tree to minimal size. Option **-o** names the squashed database *dest*, leaving the original database unaltered.

The version of the command reads a sorted list of keys and records from the standard input and fills the file with them. Input is in the form produced by the corresponding option.

The (**delete**, version of the command inserts (removes, looks up) records. Input (and output of is in the form produced by the corresponding option. The records may be unsorted. In newline-separated input, only the keys need be present.

**EXAMPLES**

```
sort -t<tab> +0 -1 inputfile | cbt build btreefile
<tab> denotes a tab character
```

**FILES**

temporaries for  
**squash**

**SEE ALSO**

[cbt\(3\)](#)

**NAME**

cc, lcc – C compilers

**SYNOPSIS**

**cc** [ *option ...* ] *file ...*

**lcc** [ *option ...* ] *file ...*

**DESCRIPTION**

*Cc* compiles the classic C language; *lcc* compiles ANSI. They are otherwise similar. In the absence of options, any named source *files* are compiled into object files and then linked, together with any named object *files*, into a single executable file named *Compilation* normally has four phases: preprocessing of # directives, compilation to assembly language, assembly, and linking. Suffixes of *file* names determine which phases they participate in:

- .c** C source to be preprocessed and compiled. Object code for this file is finally placed in a correspondingly named file, except when exactly one file is being compiled and linked.
- .i** C source to be compiled without preprocessing; # directives are ignored by *cc*, forbidden by *lcc*.
- .s** Assembler source to be assembled, producing a file.
- .o** A preexisting object file to be linked.

Both compilers accept options of *ld(1)*, the most common of which are **-o** (to substitute a name for **a.out**) and **-l** (to link from a library), and in addition

- c** Suppress the linking phase, producing **.o** files but no **a.out**.
- g** Produce additional symbol table information for debuggers such as *pi(9)*
- O** Invoke an object-code improver; superfluous in *lcc*.
- w** Suppress warning diagnostics. In *lcc*, **#pragma ref variable** supplies a dummy reference to suppress an unused-variable diagnostic.
- p** Arrange for the compiler to produce code which counts the number of times each routine is called; also, if linking takes place, replace the standard startup routine by one which arranges to gather profiling data for later examination by *prof(1)*.
- pg** Like **-p** but for *gprof* instead of *prof(1)*.
- S** Compile the named C programs, and leave the assembler-language output in **.s** files.
- E** Run the preprocessor on the named C programs, and send the result to the standard output.
- C** Prevent the preprocessor from eliding comments.
- Dname=def**
- Dname** Define the *name* to the preprocessor, as if by *If* no definition is given, the name is defined as *Lcc* predefines a few symbols on most machines; option **-v** exposes them.
- Uname** Remove any initial definition of *name*.
- Idir** files whose names do not begin with *are* always sought first in the directory of the *file* argument, then in directories named in **-I** options, then in directories on a standard list.

These options are peculiar to *cc*:

- P** Run the preprocessor on each file. Produce no line numbers. Place results in files.
- R** Cause *as(1)* to make initialized variables shared and read-only.
- Bstring** Find substitute compiler passes in the files named *string* with the suffixes *cpp*, *com* and *c2*. If *string* is empty, use a standard backup version.
- t[p012]** Find only the designated compiler passes in the files whose names are constructed by a **-B** option. In the absence of a **-B** option, the *string* is taken to be

These options are peculiar to *lcc*:

- N** Do not search standard directories for include files. Omit non-ANSI language extensions.
- A** Warn about calls to functions without prototypes.
- b** produce code that writes an expression-level profile into *prof.out*. *bprint(1)* produces an annotated listing, and **-Wf-a** uses the profile to improve register assignments.
- dn** Generate jump tables for switches with density at least *n*, a floating-point constant between zero and one, 0.5 by default.
- P** Write declarations for all defined globals on standard error.
- n** Produce code that reports and aborts upon dereferencing a zero pointer.
- M** Run only the preprocessor to generate *make(1)* dependencies on the standard output.
- t** Produce code to print trace messages at function entry and exit.
- Wpopt**  
Pass preprocessor option *opt* to the (Gnu) preprocessor. For example, **-Wp-T** allows ANSI tri-graph sequences.
- Waopt, -Wlopt, -Wfopt**  
Pass option *opt* to the assembler (*as(1)*), loader (*ld(1)*), or compiler proper.
- Bstr** Use the compiler *strrc* instead of the default version. *Str* usually ends with a slash.
- v** Report compiler steps (and some version numbers) as they are executed. A second **-v** causes steps to be reported but not executed.

*Lcc* supports **asm(string)**. The given string constant is copied to the generated assembly language output with occurrences of *%name* replaced by the address or register for identifier *name* if it is visible. Otherwise, *%name* is simply copied to the output. Wide-character literals are treated as plain char literals; ints and long ints are the same size, as are doubles and long doubles.

## EXAMPLES

```
lcc -N -I/usr/include/libc file.c
```

Use local include files instead of ANSI standard ones, which lack most functions of Section 2 of this manual, and often disagree (especially about **const**) with those in Section 3. See *intro(3)*.

## FILES

Different machines use different file names, so this list is only representative. *Lcc* option **-v** exposes the correct names.

linked output

temporary

preprocessor,

*cpp(8)*

ANSI preprocessor

*cc* compiler proper

optional optimizer for

*cc*

assembler,

*as(1)*

*lcc* compiler proper

runtime startoff

startoff for profiling

standard library, see

*intro(3)*

directory for  
  *cc* files

directory for  
  ANSI standard files

directory for local  
  *lcc* include files

## SEE ALSO

[lint\(1\)](#), [ld\(1\)](#), [strip\(1\)](#), [nm\(1\)](#), [prof\(1\)](#), [bprint\(1\)](#), [cin\(1\)](#), [adb\(1\)](#), [pi\(9\)](#) *c++(1)*

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd Ed., Prentice-Hall, 1988

## BUGS

*Cc* cannot handle the flag of *ld*.

*Lcc* currently uses the pre-ANSI library.

## MACHINE DEPENDENCIES

### VAX

**-pg** is unimplemented.

*Cc* and *lcc* use incompatible bit-field layouts and structure return conventions.

### MIPS

*Lcc* does not implement **-p** or **-pg**, and its **-g** supports breakpoints but not the examination of variables.

*Cc* and *lcc* use incompatible bit-field layouts.

### Sun

*Lcc* options **-Bdynamic** and **-Bstatic** give the binding strategy; see [ld\(1\)](#).

*Cc* and *lcc* use incompatible bit-field layouts and structure return conventions.

**NAME**

charge, charges – change (show) charges for share system usage calculations

**SYNOPSIS**

**charge** [-flags]  
**charges** [-flags]

**DESCRIPTION**

*Charge* will change the costs associated with the usage calculations for the *Share Scheduler*. The flag **-C**, if used, should be mentioned first to cause *charge* to read the existing values, instead of working on default values. The alternate name is used to show the charges currently in operation, for which the default assumes **-C**, unless the flag **-** is used. The optional flags are as follows:-

- Causes *charges* to show the default settings for the charging parameters.
- C** Causes *charge* to read in the current values, which it will use instead of the defaults. **Must be first flag if used.**
- Dh1,h2** Will set the decay rate for process priorities with normal *nice* so that they will decay to half their initial value in *h1* seconds, and set the decay rate for process priorities with maximum *nice* so that they will decay to half their initial value in *h2* seconds.
- Ehalf-life** Will set the decay rate for users' process *rates* so that they will decay to half their initial value in *half-life* seconds.
- F flags** Sets various global scheduling flags — see [share\(5\)](#) for details. *Flags* are assumed to be in octal.
- Gmaxgroups** Sets the maximum depth for the scheduling tree.
- Khalf-life[s]** Will set the decay rate for users' usages so that they will decay to half their initial value in *half-life* hours. If the *half-life* is followed by the character *s*, then the number will be interpreted as seconds.
- Nmaxnormu** Upper bound on *normalised usage* used in process priority calculations. The number can be added to a running process's priority every clock tick, so it should be small enough not to overrun the value *maxupri* in too short a time interval (ie: it should be less than  $(maxupri * (1 - pri\_decay)) / HZ$ , see the output of *charges -v* for the low priority value of *pri\_decay*.)
- Pmaxpri** Absolute upper bound for a process's priority. (Something less than the largest non-negative integer.)
- Qmaxupri** Upper bound for normal processes' priorities. *Idle* processes run with priorities in the range  $maxupri < pri < maxpri$ .
- Rdelta** Sets the *run-rate* for the share scheduler in seconds.
- Smaxusers** Sets the maximum number of users and groups that can be active. Note that this cannot exceed the maximum configured in the kernel.
- Umaxusage** Upper bound for "reasonable" usages. Users with usages larger than this are grouped together and given a *normalised usage* which prevents them from interfering with "normal" users.
- Xmaxushare** If the LIMSHARE scheduling flag is on, then this parameter limits the maximum effective share an individual user can have to *maxushare* times their allocated share.
- Ymingshare** If the ADJGROUPS scheduling flag is on, and any group is getting less than *mingshare* times its allocated share, the costs incurred by the group members will be adjusted down to compensate. (Does not affect the long-term charges.)
- bbio** The charge for a disk block I/O operation.
- mclick** The charge for a *memory tick*.
- ssyscall** The charge for a system call.



- ttick**           The charge for a CPU tick.
- v**                Show scheduling feed-back parameters (*charges* only).
- ytio**            The charge for a stream I/O operation. (This is really dependent on the number of kernel buffer operations, so a *write(1)* will cost the same as a *write(64)* to an ordinary stream, or a *write(1024)* to a pipe.)
- percent*         The percentage change to apply to all the charges.

## EXAMPLES

- charge 10**        will change the costs to 10% of the default.
- charges**         show the current charges.
- charges -**        show the default settings.

## SEE ALSO

- /usr/include/sys/charges.h*   The default values in the kernel.
- /usr/include/sys/share.h*    Definition of charges structure.
- [share\(5\)](#)            A description of the *Share Scheduler*.

## BUGS

- The *percent* flag will also affect any new constants, so bias them accordingly.
- Charges* works out the current charging percentage by using the difference between the default cost for “ticks” and the current setting.
- The defaults are hardly ever relevant.

**NAME**

*touch*, *chdate* – set modification or access date of a file

**SYNOPSIS**

**touch** [ **-c** ] *file* ...

**chdate** [ **-am** ] *date file* ...

**DESCRIPTION**

*Touch* attempts to set the modification time of the *files* to the current time. If a *file* does not exist, it will be created unless option **-c** is present.

*Chdate* sets the access and modification times of *files*. The *date* comprises two or more arguments: a month (3 letters or more), a day number, an optional time in hour:min[:sec] form, and an optional year. A missing year means a time in the last 12 months. The options are

**-a**      Change the access time only.

**-m**      Change the modification time only.

*Chdate* knows how to carry between fields of a date. Only a file's owner or the super-user can change its date.

**EXAMPLES**

```
chdate jul 4 12:00 1976 independence
```

```
chdate jul -3 1976 independence      # backdate one week
```

**SEE ALSO**

[ls\(1\)](#), [utime](#) in [chmod\(2\)](#), [stat\(2\)](#), [timec\(3\)](#), [chmod\(1\)](#), [chown\(8\)](#)

**DIAGNOSTICS**

*Chdate* returns the number of files on which the date could not be changed.

**BUGS**

*Touch* will not touch directories.

The first *file* name for *chdate* cannot begin with a digit.

**NAME**

checknr – check nroff/troff files

**SYNOPSIS**

**checknr** [ **-s** ] [ **-f** ] [ **-a.x1.y1.x2.y2. ... .xn.yn** ] [ **-c.x1.x2.x3 ... .xn** ] *file* ...

**DESCRIPTION**

*Checknr* checks a list of *nroff*(1) or *troff*(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. Delimiters checked are:

- (1) Font changes using `\fx ... \fP`.
- (2) Size changes using `\sx ... \s0`.
- (3) Macros that come in open ... close forms, for example, the `.TS` and `.TE` macros which must always come in pairs.

*Checknr* knows about the *ms*(7) and *me*(7) macro packages.

Additional pairs of macros can be added to the list using the `-a` option. This must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair `.BS` and `.ES`, use `-a.BS.ES`

The `-c` option defines commands which would otherwise be complained about as undefined.

The `-f` option requests *checknr* to ignore `\f` font changes.

The `-s` option requests *checknr* to ignore `\s` size changes.

*Checknr* is intended to be used on documents that are prepared with *checknr* in mind, much the same as *lint*. It expects a certain document writing style for `\f` and `\s` commands, in that each `\fx` must be terminated with `\fP` and each `\sx` must be terminated with `\s0`. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from *checknr*. Since it is probably better to use the `\fP` and `\s0` forms anyway, you should think of this as a contribution to your document preparation style.

**SEE ALSO**

*nroff*(1), *troff*(1), *ms*(7), *me*(7), *checkeqn*(1)

**DIAGNOSTICS**

Complaints about unmatched delimiters.

Complaints about unrecognized commands.

Various complaints about the syntax of commands.

**AUTHOR**

Mark Horton

**BUGS**

There is no way to define a 1 character macro name using `-a`

**NAME**

chmod – change mode

**SYNOPSIS**

**chmod** *mode file ...*

**DESCRIPTION**

The mode of each named file is changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes. (Modes that contain a 1000 bit are incompatible with other modes that have any bits among 7000.)

4000	set user ID on execution
3000	set exclusive access mode (1 writer or <i>n</i> readers)
2000	set group ID on execution
1000	set synchronized access mode (1 writer and <i>n</i> readers)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission [op permission ...]*

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**. If *who* is omitted, the default is **a**.

*Op* can be + to add *permission* to the file's mode, - to take away *permission* and = to assign *permission* absolutely (all other bits will be reset).

*Permission* is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group id) **e** (set exclusive access mode) and **y** (set synchronized access mode). Letters **u**, **g** or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g**.

Only the owner of a file (or the super-user) may change its mode.

Synchronized access guards against inconsistent updates by preventing concurrent opens for writing. Exclusive access guards against inconsistent views by preventing concurrent opens if one is for writing.

**EXAMPLES**

chmod o-w file  
Deny write permission to others.

chmod +x file  
Make file executable.

**SEE ALSO**

[ls\(1\)](#), [chmod\(2\)](#), [stat\(2\)](#), [chdate\(1\)](#), [chown\(8\)](#)

**NAME**

chunk – segment text into phrase units

**SYNOPSIS**

**chunk** [ **-flags** ] [ **-ver** ] [file ...]

**DESCRIPTION**

*Chunk* segments text into phrase units by beginning a new line at the end of each unit it identifies.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**USES**

This program is useful for reading research, and it can be used to reformat text files for easier editing. Additionally, editors may find it easier to edit a text whose microstructure is typographically clear.

**BUGS**

Since *chunk* runs *deroff(1)* on the input text, formatting commands (and thus paragraph and heading structures) are lost in the output.

*Chunk* will think unfamiliar abbreviations are the end of a sentence, and hence the end of the chunk.

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

cin – C interpreter

**SYNOPSIS**

**cin** [ *option* ... ] [ *file* ... ] [ *-- arg* ... ]

**DESCRIPTION**

*Cin* interprets a C program comprising the *file* arguments as in [cc\(1\)](#). The special name signifies standard input. When called with no *file* arguments, *cin* defaults to **-i**. Arguments *arg* are passed to the interpreted program as **cin\_argv[1]**, **cin\_argv[2]**, ... and **cin\_argc**.

Options **-Dname=def**, **-Dname**, **-Lname**, **-Uname**, **-Iname**, **-lx**, **-oname**, and **-uname** are as in *cc*. Options **-O** and **-g** are ignored. Other options are:

**-Oname**

Pass *name* to the interpreted program as **cin\_argv[0]**.

**-Cname**

Use *name* as the interpreter startup file (by default, the file `The interpreter startup file` is ignored by using **-C/dev/null**).

**-Fi:oe**

Use file descriptors *i,o,e* as standard input, standard output, and standard error for the interpreter, as distinct from the interpreted code (by default, 0, 1, and 2).

**-S**

Enable interactive mode after run-time errors.

**-Vname:n**

Declare function *name* to have a variable number of arguments, the first *n* of which are to be type-checked.

**-c**

Parse files but do not execute the program.

**-i**

Enable interactive interpretation. C statements are evaluated immediately using local and global variables. Non-void values are printed. Function, variable, and C pre-processor declarations are stored in the current ‘view’. A function definition must include its return type. Declarations and statements can appear in any order and identifiers can be redeclared.

**-r**

Prohibit multiply-declared global variables.

**-s**

Do not catch signals using [signal\(2\)](#) (by default, *cin* catches **SIGBUS**, **SIGEMT**, **SIGFPE**, **SIGILL**, **SIGINT**, **SIGIOT**, **SIGSEGV**, **SIGSYS**, **SIGTRAP**).

**-v**

Print the user and system times associated with loading files and executing the program, as in [time\(1\)](#).

**+option**

Turn off the specified *option*.

The functions and variables listed below are predefined in *cin* and *libcin.a*. Their arguments are typed according to these conventions:

```
char *func, *message, *mod, *name, *ref, *string;
enode *code;
int level, line;
long (*after)();
unsigned long addr;
void (*before)(), (*routine)();
```

This first group of functions and variables are the most commonly used. To use them load and include

**int cin\_break(name, line, string)**

If *string* is **(char \*)0**, place a **cin\_system** in file *name* before line number *line*. Otherwise, place the C-language statement specified by *string* in file *name* before line number *line*. Thus, *string* is read and evaluated within the prevailing context of the function without stopping execution. Return 0 if line number *line* cannot be found in file *name*.

**int cin\_dump**(*name*)

Create an and place it in the file *name*. Return 0 if *name* cannot be created.

**void cin\_info**(*string*)

Where *string* is:

**breakpoint**

Print *cin*'s **cin\_breaks** and **cin\_stopins**.

**log**

Print *cin*'s interactive mode log file name.

**memory**

Print *cin*'s memory usage.

**spy**

Print *cin*'s spies.

**symboltable**

Print *cin*'s symbol table.

**undefined**

Print *cin*'s undefined variables.

**usage**

Print *cin*'s usage message.

**version**

Print *cin*'s version number.

**view**

List the available views. The current view is starred (\*).

**where**

Print a trace of function calls.

**wrapper**

Print *cin*'s wrappers.

**int cin\_load**(*string*)

Evaluate *string* as invocation arguments of *cin*. Return 0 if arguments are not valid.

**int cin\_log**(*name*)

*mv*(1) the interactive mode session log to file *name*. Return 0 if *name* cannot be *mved*.

**void cin\_make**(*string*)

If *string* is **(char \*)0**, then re-load any file that is out of date. Otherwise, run **\$MAKE**, passing *string* as arguments, and scan its standard output for lines that begin with pound sign (#). Characters following the pound sign are executed as a C-language statement.

**int cin\_pp**(*func*, *level*)

Print the C-language for the function *func*. Where *level* is:

**0**

Print declaration.

**1**

Print declaration and body.

Return 0 if *func* cannot be found.

**void cin\_quit**()

Flush output and exit *cin*. If all else fails, use *\_exit*(2).

**int cin\_reset**()

Preserve function definitions, zero bss variables, and restore initialized data variables to their original values. Return 0 if state cannot be reset.

**void cin\_return**()

Return from a call to **cin\_system**.

**void cin\_run**(*string*)

Set **cin\_argv**[1], **cin\_argv**[2], ... and **cin\_argc** from *string* and then **main**(**cin\_argc**, **cin\_argv**).

**int cin\_spy**(*func*, *name*, *mod*, *ref*)

Trace variable references and modifications. If *mod* is not **(char \*)0**, place the C-language statement specified by *mod* after the variable *name* is modified in the function *func*. If *ref* is not **(char \*)0**, place the C-language statement specified by *ref* before the variable *name* is

referenced in the function *func*. If function value *func* is **(char \*)0**, search all functions. Return 0 if *name* cannot be found in *func*.

**void cin\_step()**

Step over function calls to the next C-language statement in the current or previous function.

**void cin\_stepin()**

Step into any function to the next C-language statement.

**void cin\_stepout()**

Step out of the current function back to the next C-language statement in the previous function.

**int cin\_stopin(*func*, *string*)**

If *string* is **(char \*)0**, place a **cin\_system** before the first executable line in the function *func*, either in the current view or wherever *func* can be found. Otherwise, place the C-language statement specified by *string* before the first executable line in the function *func*, either in the current view or wherever it can be found. Thus, *string* is read and evaluated within the prevailing context of the function without stopping execution. Return 0 if function *func* cannot be found.

**char \*cin\_sync(*string*)**

Where *string* is:

**filename**

Return the non-interactive C source file name being executed.

**lineno**

Return the non-interactive C source line number being executed.

**void cin\_system()**

Start a **cin\_read-cin\_eval-cin\_print** loop.

**int cin\_unbreak(*name*, *line*, *string*)**

Clear a **cin\_break** or a **cin\_stopin** in file *name* before line number *line* with string *string*. Return 0 if **cin\_break** or **cin\_stopin** cannot be cleared.

**int cin\_unload(*name*)**

Unload the object file *name*. Return 0 if *name* cannot be unloaded.

**int cin\_unspy(*func*, *name*, *mod*, *ref*)**

Remove the C-language statement specified by *mod* after the variable *name* is modified, and the C-language statement specified by *ref* before the variable *name* is referenced in the function *func*. Return 0 if spy cannot be cleared.

**int cin\_unstopin(*func*, *string*)**

Clear a **cin\_stopin** or **cin\_break** at the first executable line in the function *func* with string *string* in either the current view or wherever *func* can be found. Return 0 if the **cin\_stopin** or **cin\_break** cannot be cleared.

**int cin\_unwrapper(*func*, *before*, *after*)**

Remove the call of function *before* before the function *func* is called. Remove the call of function *after* after the function *func* is called. Return 0 if *func* cannot be found.

**int cin\_view(*name*)**

Change the current view to *name*. Return 0 if the view cannot be found.

**int cin\_wrapper(*func*, *before*, *after*)**

If *before* is not **(void (\*)())0**, call the function *before* with the arguments of function *func* before the function *func* is called. If *after* is not **(long (\*)())0**, call the function *after* with the argument of the return value of function *func* after the function *func* is called. The return value of function *after* is substituted for the return value of function *func*. Return 0 if *name* cannot be found either in the current view or any other view.

**extern int cin\_argc**

The number of elements passed to the interpreted program.



**extern char \*\*cin\_argv**

An array of the arguments passed to the interpreted program.

**extern char \*cin\_filename**

The current C source file name being executed.

**extern int cin\_level**

The number of nested calls to **cin\_system**.

**extern char \*cin\_libpath**

A colon (:)-separated list of libraries to search for undefined routines (by default, the libraries specified on the command line and

**extern int cin\_lineno**

The current C source line number being executed.

**extern char \*cin\_prompt**

The interactive mode prompt (by default, the string “**cin>**”).

These are some of the less frequently used functions and variables in *cin*. They are primarily used by *cin* library or language developers.

**enode \*cin\_compile(code)**

Analyze *code* and return an optimized program. Return **(enode \*)0** if *code* cannot be compiled.

**int cin\_epp(func)**

Print the **enodes** for function *func*. Return 0 if *func* cannot be found.

**enode \*cin\_eprint(code)**

Print the *code* as **enodes**. Return the argument.

**char \*cin\_error\_code\_set(message, string)**

Where *message* is:

**dynamic error**

When *cin* detects a divide by zero, a modulus by zero, a null pointer access, or an abnormal signal execute the C-language statement specified by *string*.

**undefined function**

When *cin* detects an undefined function execute the C-language statement specified by *string*.

**undefined symbol**

When *cin* detects an undefined symbol execute the C-language statement specified by *string*.

If *string* is **(char \*)0**, execute **cin\_system()**. Return the old *string* for *message*.

**enode \*cin\_eval(code)**

Execute the *code* as if it were present in the program where **cin\_eval** is called. Return the resulting program.

**ident \*cin\_find\_ident(name)**

If *name* is not **(char \*)0**, return the identifier for the variable *name* either in the current view or wherever *name* can be found. If *name* is **(char \*)0**, return the identifier for the previous non-**(char \*)0** value of the variable *name* in the next view where *name* can be found. Return **(ident \*)0** if *name* cannot be found.

**struct nlist \*cin\_find\_nlist(addr)**

Return the loader symbol table entry for the external address *addr*. Return **(struct nlist \*)0** if *addr* cannot be found.

**void (\*cin\_info\_set())(string, routine)**

Inform **cin\_info** that it should call *routine* when it is passed *string*. Return the old *routine* for *string*.

**char \*cin\_ltof(*line*)**

Return the function name at line number *line* in the current view. Return (**char \***)0 if a function cannot be found for *line*.

**void cin\_pop(*level*)**

Replace *level* interpreted function calls from the stack with **cin\_system**.

**enode \*cin\_print(*code*, *level*)**

Print the *code* as C-language code. Where *level* is:

- 0**      Print declaration.
- 1**      Print declaration and body.

Return the *code* argument.

**enode \*cin\_read(*string*)**

Read *string* and return a program. Return (**enode \***)0 if *string* cannot be parsed.

**char \*cin\_slashify(*string*)**

Return a pointer to storage obtained from *malloc(3)* and there create a character array from *string* by translating backspace, form feed, newline, carriage return, horizontal tab, vertical tab, backslash, single quote, and double quote into **\b**, **\f**, **\n**, **\r**, **\t**, **\v**, **\\**, **\'**, and **\"** respectively. Other non-printable characters are translated into *ddd* octal notation.

**extern int cin\_err\_fd**

*Cin*'s standard error file descriptor.

**extern int cin\_in\_fd**

*Cin*'s standard input file descriptor.

**extern int cin\_out\_fd**

*Cin*'s standard output file descriptor.

**extern stackelem \*cin\_stack**

The trace of function calls.

**extern view \*cin\_views**

The list of available views.

**EXAMPLES**

The world's shortest "Hello world" program.

```
cin> printf("Hello world\n");
Hello world
(int)12
```

Setting breakpoints and tracing function calls.

```
$ cin -lcin
cin> int f(x) { return x <= 1 ? 1 : x * f(x - 1); }
extern int f();
cin> cin_stopin("f", (char*)0);
/tmp/cin006795: 1: f: set breakpoint: (char *)0
(int)1
cin> f(2);
/tmp/int006795: 1: f: stopped execution:
cin> cin_return();
/tmp/int006795: 1: f: stopped execution:
cin> (void)cin_info("where");
/tmp/cin006795: 5: cin_system: info: where: ()
/tmp/cin006795: 1: f: info: where: (x = (int)1)
/tmp/cin006795: 1: f: info: where: (x = (int)2)
/tmp/cin006795: 3: cin_system: info: where: ()
cin> x;
(int)1
cin> cin_return();
```

```
(int)2  
cin> cin_quit();  
$
```

**FILES**

default interactive log  
various function and variable declarations  
various predefined functions  
interpreter startup file  
other files as in [cc\(1\)](#)

**SEE ALSO**

*Cin User Manual*  
*Cin Reference Manual*  
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978  
[lint\(1\)](#), [ar\(1\)](#), [cc\(1\)](#)

**BUGS**

The addresses of *etext*, *edata*, and *end* are not meaningful with incrementally loaded code.

**NAME**

`cite` – process citations in a document

**SYNOPSIS**

`cite [ -s ] [ -u ] [ files ]`

**DESCRIPTION**

*Cite* is a [troff\(1\)](#) preprocessor for forward and backward references. It copies the *files* or the standard input to standard output, observing lines of the forms

```
.CD "key" "definition"  
.CU "key"text
```

Each **.CD** line is remembered. A later **.CU** with the same *key* will be replaced by the *definition* for that key; the *text* will be copied verbatim. If no definition is yet available for a **.CU** reference, the key will be replaced by **ZZ**.

At the location of each **.CD** command, *cite* causes *troff* to send the line (with macro substitutions) to the standard error file. The resulting definitions may be included at the beginning of another *cite-troff* run, effectively eliminating forward references. Unfortunately, the definition file may contain obsolete definitions (included from a previous run) plus other *troff* messages. These should be deleted using option **-s**.

The options are

- s** Place only the latest definitions on the standard output; shunt non-*cite* data to standard error.
- u** Place on standard error a list of undefined or unused citations.

**EXAMPLES**

```
cite file.defs file | troff -ms 2>temp.defs >/dev/null  
cite -s temp.defs >file.defs  
cite file.defs file | troff -ms 2>temp.defs | lp  
Run cite-troff to collect updated definitions in temp.defs.  
Eliminate old definitions, putting updates back in file.defs.  
Run cite-troff again, using latest definitions.
```

**SEE ALSO**

[troff\(1\)](#)

**NAME**

clear – clear terminal screen

**SYNOPSIS**

**clear**

**DESCRIPTION**

*Clear* clears the screen of the terminal on its standard output. It depends upon the environment variable **TERM** to know how to do it.

**FILES****SEE ALSO**

*term*(7)

**NAME**

cmp – compare two files

**SYNOPSIS**

**cmp** [ **-lsL** ] *file1 file2* [ *offset1* [ *offset2* ] ]

**DESCRIPTION**

The two files are compared. If the contents differ a diagnostic results, otherwise there is no output.

The options are:

- I**      Print the byte number (decimal) and the differing bytes (octal) for each difference.
- s**      Print nothing for differing files, but set the return code.
- L**      Print the line number of the first differing byte.

If offsets are given, comparison starts at the designated byte position of the corresponding file. Offsets that begin with **0x** are hexadecimal; with **0**, octal; with anything else, decimal.

**SEE ALSO**

*diff(1)*, *comm(1)*

**DIAGNOSTICS**

*Cmp* reports 'EOF' and identifies the file if one file is short. It reports the number of the first disagreeing byte if contents differ. The return code is 0 for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

**NAME**

col, 2, 3, 4, 5, 6, mc, fold, expand – column alignment

**SYNOPSIS**

**col** [ **-bfx** ]

**2** [ *file* ]

**mc** [ - ] [ *-N* ] [ **-t** ] [ *file ...* ]

**fold** [ *-N* ] [ *file ...* ]

**expand** [ *-stops* ] [ *file ...* ]

**DESCRIPTION**

These programs rearrange files for appearance's sake. All read the standard input and write the standard output. Some optionally read from files instead.

*Col* overlays lines to expunge reverse line feeds (ESC-7) and half line feeds (ESC-9 and ESC-8) as produced by *nroff* for .2C in *ms*(6) and for *tbl*(1). It normally emits only full line feeds; option **-f** (fine) allows half line feeds too. Option **-b** removes backspaces, printing just one of each pile of overstruck characters. *Col* normally converts white space to tabs; option **-x** overrides this feature. Other escaped characters and non-printing characters, except for SO and SI, are ignored.

*Col* should not be used for printing on an HP ThinkJet printer with *think* (*thinkblt*(9.1)), which performs the *col* function itself.

Commands *2*, *3*, *4*, *5*, *6* convert their input to 2-, 3-, 4-, 5-, or 6-column form, with consecutive input lines arranged across each row.

*Fold* inserts newlines after each *N* characters (default *n*=80, or *mux*(9) window size) of long lines.

*Mc* splits the input into as many columns as will fit in *N* print positions (default *N*=80). Under option - each input line ending in a colon is printed separately (see example). On output, multiple spaces are converted to tabs; this is suppressed by option **-t**.

*Expand* replaces tabs by spaces. The optional *stops* argument is a comma-separated list of tab stops, counted from 0; default is every 8 columns.

**EXAMPLES**

```
tbl file | nroff -ms | col | hp
```

Format some tables for printing on typewriters; use *col* to remove reverse line feeds and *hp* (see *ul*(1)) to do underlining, etc., on an HP terminal.

```
ls directory1 directory2 | mc -
```

List files in multiple columns, separated by directory.

**SEE ALSO**

*pr*(1)

**BUGS**

*Col* can't back up more than 128 lines or handle more than 800 characters per line, and understands (013) as reverse line feed.

**NAME**

comm – select or reject lines common to two sorted files

**SYNOPSIS**

**comm** [ **-123** ] *file1 file2*

**DESCRIPTION**

*Comm* reads *file1* and *file2*, which are ordered in ASCII collating sequence, and produces a three column output: lines only in *file1*; lines only in *file2*; and lines in both files. The filename means the standard input.

Flag or suppresses printing of the corresponding column.

**EXAMPLES**

```
comm -12 file1 file2
```

Print lines common to two sorted files.

```
deroff -w /usr/lib/upas/names.last | tr a-z A-Z | sort -u >temp
```

```
spell temp | comm -13 - temp
```

Print names that are known both to [mail\(1\)](#) and [spell\(1\)](#)

**SEE ALSO**

[sort\(1\)](#), [cmp\(1\)](#), [diff\(1\)](#), [uniq\(1\)](#), [join\(1\)](#)



**NAME**

`con`, `rx` – remote login and execution

**SYNOPSIS**

`con` [ **-l** ] *machine*

`rx` [ **-n** ] *machine* [ *command-word* ... ]

`/usr/bin/m/machine` [ *command-word* ... ]

**DESCRIPTION**

`Con` connects to the computer whose network address is *machine* and logs in if possible. Standard input and output go to the local machine. Option **-l** prevents automatic login; a normal login dialog ensues.

The quit signal (control-`\`) is a local escape. It prompts with the local machine name and `>>`. Legitimate responses to the prompt are

**i**                   Send a quit [sic] signal to the remote machine.  
**q**, **x**, or **.**       Exit.  
**b**                   Send a break.  
**!command**       Execute *command* locally.

`Rx` executes one shell command on the remote machine as if logged in there, but with local standard input and output. Unquoted shell metacharacters in the command are interpreted locally, quoted ones remotely. The assignment **REXEC=1** appears in the remote environment. With no arguments, `rx` just diagnoses availability. Option **-n** ignores sporadic end-of-file indications on a sick network.

Network addresses for both `con` and `rx` have the form *network!host* or simply *host*. Supported networks are (Datakit) and (TCP/IP, usually Ethernet).

Directory contains machine names as commands: `/usr/bin/m/machine` with no argument runs an appropriate flavor of `con` for the named *machine*. If given arguments, `/usr/bin/m/machine` runs `rx` with those arguments. If is in the `sh(1)` search path, the names become commands for navigating the local cluster.

**EXAMPLES**

`rx overthere cat file1 >file2`

Copy remote *file1* to local *file2*.

`rx overthere cat file1 ">file2"`

Copy remote *file1* to remote *file2*.

`eqn paper | rx pipe troff -ms | rx arend lp`

Parallel processing: do each stage of a pipeline on a different machine.

**FILES**

authentication

servers

**SEE ALSO**

[push\(1\)](#), [dcon\(1\)](#), [cu\(1\)](#), [dkmgr\(8\)](#), [svcmgr\(8\)](#), [tcpmgr\(8\)](#), [ipc\(3\)](#)

D. L. Presotto, ‘Interprocess Communication in the Eighth Edition UNIX System’, this manual, Volume 2

**BUGS**

The remote standard error and standard output are combined and go inseparably to the local standard output.

Under `rx`, a program that should behave specially towards terminals may not: `sh(1)` will not prompt, `vi(1)` will not manage the screen, etc. `Nrx` (see [dcon\(1\)](#)) avoids this trouble, but has others of its own.

`Con` and `rx` may not guess the right kind of connection. In case of trouble, try the programs in [dcon\(1\)](#).

The names in are conventions, not actual network addresses.

**NAME**

cospan, psr – coordination-specification analyzer and pretty-printer

**SYNOPSIS**

**cospan** [ *option ...* ] *file*

**psr** [ *option ...* ] *file ...*

**DESCRIPTION**

*Cospan* analyzes the behavior of coordinating systems. Three types of input *file* are distinguished by suffix:

- .sr** The normal case. The file contains S/R specifications as described in the reference, possibly including *cpp(8)* commands, to be compiled into
- .c** C code, which is compiled and linked into
- .an** executable analysis program.

The options are

- Dname=value**
- Dname**
- Uname**
- Idirectory**  
Same as in *cc(1)*.
- v** Produce verbose syntax error messages.
- p** Suppress file-name/line-number information for embedded C code.
- i** Produce an implementation version of the C code.
- m** Produce a merged version of the C code.
- n** Compile no transition checks (except deadlocks). By default, the analysis gives a warning on the first stability violation and aborts on non-semi-deterministic resolutions.
- b** Use C built-in (machine-dependent) integer division operations. By default, an S/R integer division  $i/j$  results in the greatest integer not higher than the mathematical quotient, and the remainder operation  $i \bmod j$  yields a result in the range  $0 \dots j-1$ .
- Copt** Pass option *-opt* to the C compiler.
- hsize** Set the state hash table size to the next prime after *size* ; default is 32693.
- Hsize**  
Similar to **-hsize**, except that states which produce hash collisions are ignored.
- tsecs** Abort analysis after the specified number of seconds.
- Vs** Produce verbose analysis output messages. The string *s*, by default specifies message types: *advice*, *warning*, *error*, or *list*.
- r** Restart previously aborted analysis. Recovery is possible in cases of hangups, interrupts, software termination signals (due to a kill command), timer alarms, no-space conditions, and aborts due to **-c** or **-L** requests.
- d** Abort on deadlocks. By default, the analysis gives a warning on the first deadlock and reports the number of deadlocks in the analysis summary.
- s** Abort on stability failures.
- l** List analysis on standard output.
- T** Time each translation and execution step.
- Lnumber,number**  
List analysis, reporting states in the given range, and abort after searching the upper bound.
- cnumber**  
Check each back-edge in the component identified by the given *number* and abort analysis.

**CC**=*name*

Use an alternate C compiler; default is **CC=cc**.

The order of the arguments is arbitrary, and several options may be combined to a single argument, provided that option values are terminated by white space. Options can be preset by defining the environment variable **COSPANOPT**.

*Psr* is a pretty-printer for S/R specifications. It places *troff*(1) or *nroff* output on the standard output.

The options, which may be reset between *files*, are:

- d** Show current date in page footer.
- m** show file modification time in footer (default).
- nN** Number every *N*th line; default is **-n0**, no numbering.
- sN** Set type size to *N* points, vertical spacing to *N*/60 inch, and tab stops every *N*/20 inch.
- wN** Set the page width to *N* (in *troff* notation).
- fF** Use the *troff* font *F* and its italic, bold, and bold-italic counterparts. Known fonts are Bembo, CW, Euro, Futura, H, Hcond, Memphis, Optima, PA, R.
- .request**  
Issue a *troff* request before printing the next *file*. Multiple requests may be given.
- Tname**  
As in *troff*. Applies to all *files*. If *name* is omitted, *troff* input is written on standard output.

*Psr* sets the escape character to BEL. The \ character is copied without interpretation, to allow printing of embedded C code. The macro **.SO** may be used to include *troff* text that uses the standard escape character.

The strings **DT**, **L**, and **R** contain today's date, the left-hand, and the right-hand side of the page header, respectively.

## EXAMPLES

**COSPANOPT=-TlImyincludedir cospan myfile.sr**  
equivalent to **cospan -T -l -Imyincludedir myfile.sr**.

**psr -ll6.5i -lt6.5i myfile.sr**  
equivalent to **psr -w6.5i myfile.sr**

## FILES

recovery data  
error track  
list output (  
    **-L** option)  
merging data  
temporary file  
S/R compiler  
S/R verbose compiler  
header file  
implementation header file  
analysis object library  
*troff* preprocessor  
*troff* macros

## SEE ALSO

Z. Har'El and R. P. Kurshan, *COSPAN User's Guide*, 11211-871009-21TM, AT&T Bell Laboratories.  
[spin\(1\)](#), [d202\(1\)](#)

**NAME**

courier – remote procedure call compiler

**SYNOPSIS**

**courier** [ -x ] *specfile*

**DESCRIPTION**

*Courier* compiles the Mesa-like specification language associated with the Courier remote procedure call protocol.

**FILES**

prog.cr            Courier specification file for *prog*.

The following files are generated by courier from the above:

prog.h            definitions and typedefs  
prog\_stubs.c      mappings between C and Courier  
prog\_server.c    server routines  
prog\_client.c    client routines

**BUGS**

Note that program names are restricted to 5 characters to keep generated filenames within the 14 character limit.

**SEE ALSO**

Eric C. Cooper, 'Writing Distributed Programs with Courier'  
'Courier: The Remote Procedure Call Protocol,' Xerox System Integration Standard 038112, December 1981.

**NAME**

cp, mv, ln, reccp – copy, move, or link files

**SYNOPSIS**

**cp** [ **-z** ] *file1 file2*

**cp** [ **-z** ] *file ... directory*

**mv** [ **-f** ] *file1 file2*

**mv** [ **-f** ] *file ... directory*

**ln** [ **-s** ] *file1 file2*

**ln** [ **-s** ] *file ... directory*

**reccp** [ **-z** ] *file1 file2*

**reccp** [ **-z** ] *file ... directory*

**DESCRIPTION**

In the first form of each command, *file2* is any name except an existing directory. In the second form the command copies, moves, or links one or more *files* into a *directory* under their original filenames, as if by a sequence of commands in the first form. Thus is equivalent to

*Cp* copies the contents of plain *file1* to *file2*. The mode and owner of *file2* are preserved if it already exists; the mode of *file1* is used otherwise.

*Mv* moves *file1* to *file2*. If the two files are in the same file system, the name *file1* is simply changed to *file2*; if they are in different file systems, *file1* is copied and then removed. If *file2* already exists, it is removed before *file1* is moved. In this case the mode of *file2* is reported if it is not writable and the standard input is a terminal. Respond (and newline) to permit removal.

*Ln* links plain *file1* and *file2*. *File2* becomes an alternate name for, and is otherwise identical to, *file1*. *File2* must be in the same file system as *file1* and must not already exist.

*Reccp* copies plain files like *cp*, but copies directories and their contents recursively. It attempts to duplicate linkage and dates. When run by the super-user, it preserves ownership and copies device files as device files.

The options are:

**-z** Preserve ‘holes’; see [lseek\(2\)](#).

**-f** Forcibly remove *file2* without asking.

**-s** Make symbolic links: record the (arbitrary) name *file1* in *file2*. Except in special cases, such as [rm\(1\)](#) and *lstat* (see [stat\(2\)](#)), subsequent references to *file2* are treated as references to *file1*. See [link\(2\)](#) for details.

**EXAMPLES**

**mkdir /usr1/ken; cp /usr/ken/\* /usr1/ken**

Place in **/usr1/ken** copies of all files from **/usr/ken**.

**reccp /usr/ken /usr1**

**mkdir /usr1/ken; reccp /usr/ken/\* /usr1/ken**

Two ways to duplicate in **/usr1/ken** the whole file hierarchy from **/usr/ken**.

**SEE ALSO**

[cat\(1\)](#), [link\(2\)](#), [stat\(2\)](#), [push\(1\)](#), [uucp\(1\)](#), [rcp\(1\)](#), [cpio\(1\)](#)

**DIAGNOSTICS**

*Cp*, *mv*, and *reccp* refuse to copy or move files onto themselves or directories into themselves.

**BUGS**

*Mv* to a different file system is imperfect: if *file1* is a plain file links to it are broken; if it is a directory, nothing happens.

**NAME**

`cpio` – copy file archives in and out

**SYNOPSIS**

`cpio -o` [ *acBv* ]

`cpio -i` [ *BcdmrtuvfsSb6* ] [ *pattern ...* ]

`cpio -p` [ *adlmrv* ] *directory*

**DESCRIPTION**

**Cpio -o** (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

**Cpio -i** (copy in) extracts files from the standard input which is assumed to be the product of a previous **cpio -o**. Only files with names that match the *patterns* are selected. *Patterns* are given in the name-generating notation of *sh(1)*; the default is \* (all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. File ownership is preserved if possible.

**Cpio -p** (pass) reads from the standard input a list of path names of files to copy into the destination *directory*.

The options are:

- a**     Reset access times of input files after they have been copied.
- B**     Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from
- d**     *Directories* are to be created as needed.
- c**     Write header information in ASCII character form for portability.
- r**     Interactively rename files. An empty name (newline only) causes a file to be skipped.
- t**     Print a table of contents of the input. No files are created.
- u**     Copy unconditionally (normally an older file will not overwrite a newer file).
- v**     (Verbose) List file names; **-vt** looks like **ls -l**.
- l**     Whenever possible, link files rather than copying them. Usable only with the **-p** option.
- m**     Retain previous file modification time. This option is ineffective on directories that are being copied.
- f**     Copy in all files except those in *patterns*.
- s**     Swap bytes. Use only with the **-i** option.
- S**     Swap halfwords. Use only with the **-i** option.
- b**     Swap both bytes and halfwords. Use only with the **-i** option.
- 6**     Process an old (i.e., UNIX Sixth Edition format) file. Only useful with **-i** (copy in).

**EXAMPLES**

```
ls | cpio -oc >/dev/mt1
    Copy the contents of the current directory to a tape.

mkdir newdir
cd olddir
find . -print | cpio -pd ../newdir
    Reproduce a directory hierarchy; newdir must exist.
```

**SEE ALSO**

*ar(1)*, *bundle(1)*, *tar(1)*, *find(1)*, *cpio(5)*, *cp(1)*

**BUGS**

- Path names are restricted to 128 characters.
- Does not know about symbolic links.
- If there are too many unique linked files, linking information is lost.
- Only the super-user can copy special files.
- The archive size is reported in archaic ‘blocks’ of 512 bytes.

**NAME**

`cray` – run job remotely on cray-xmp

**SYNOPSIS**

`cray` [ *options* ] [ *jcl* [ [ + ] *file* ] ]

`c1sts`

**DESCRIPTION**

*Cray* submits the named files to the MHCC Cray. A plus sign stands for a Cray end-of-file, which separates *jcl*, source code, and data files. A file *jcl* that doesn't exist in the current directory is searched for in the directory specified by the shell environment variable CRAYJCL.

- o*ofile***            Send the job output to *ofile*.
- p*pages***            If the *jcl* file doesn't begin with a job line, *cray* supplies one. In that case, the option **-p*n*** specifies the maximum number of pages (actually, 45-line blocks) that can be output; the default is 100.
- s*n***                 Run the job at service grade *n*. (The default is fastest is long runs should be at
- t*seconds***           If the *jcl* file doesn't begin with a job line, specifies the maximum running time; the default is 15.

*C1sts* gives a status report on the *cray*.

**EXAMPLES**

Run a Fortran program and subroutine, reading from param on unit 5 and writing on unit 6,

```
cray -o output ft + main.f sub.f + param
```

Run a Fortran program and subroutine, reading from param on unit 5 and writing on unit 6,

**BUGS**

Because of a problem with the standard input, the **-o** option is required for remote execution via Datakit.

**THE FIRST TIME**

An incredible amount of busywork is required the first time you use this command. Get an account on mhuxa by filing an application at the computer center accounting office, and if possible get password aging turned off. Set up your on your home machine and in the comp center so that remote execution in either direction gets a silent login. (Otherwise, you may get a 'Bad magic number' message when trying to execute a push.) (Try copying the mhuxa file Set up mail forwarding from mhuxa to your home machine. (See Now, from your home machine, execute

to be sure everything is set up properly. The first time you try this, you will be asked to login; this legalizes remote execution from your home machine onto the comp center machine. For the reverse direction, your user id should be added to on your home machine. Export from your on your home machine. By editing a copy of in your bin, you can arrange for special action to be taken when your output arrives; the default is mail notification. Now you should be ready to run; try

If you don't get some job output back from the *cray* within a few minutes, something is wrong.

**NAME**

crypt, encrypt, decrypt – encode/decode

**SYNOPSIS**

**/usr/games/crypt** [ *password* ]

**/usr/games/encrypt** [ **-p** ] [ *password* ]

**/usr/games/decrypt** [ **-p** ] [ *password* ]

**DESCRIPTION**

These commands read from the standard input and write on the standard output. The *password* is an enciphering key. If no password is given, one is demanded from the terminal; echoing is turned off while it is being typed in. *Crypt* uses a relatively simple, fast method (rotor machine) for both enciphering and deciphering. *Encrypt* and *decrypt* use a more robust, slower method (DES). Files enciphered by *crypt* are not intelligible to *encrypt/decrypt*, and vice versa.

It is prudent to supply the key from the terminal, not from the command line, and to pick a reasonably obscure and long key (6 letters for *crypt* and much longer for *encrypt*).

Under option **-p** *encrypt* enciphers into printing characters, which can be sent by [mail\(1\)](#). *Decrypt* can distinguish ciphertext from clear: it will work on a full mail message, headers and all.

**FILES**

for typed key

**SEE ALSO**

[ed\(1\)](#), [makekey\(8\)](#)

J. A. Reeds and P. J. Weinberger, 'File Security and the Unix Crypt Command,' *AT&T Bell Laboratories Technical Journal*, **63** (1984) 1673-1684

**BUGS**

*Crypt* is breakable by knowledgeable cryptanalysts. Its only practical use is for mildly private data transmission. *Encrypt/decrypt* gives strong protection for transmission over untrusted channels between trusted machines.

It is unwise to count on encryption of any sort for safe storage of documents.



**NAME**

`cscan` – scan documents on canon scanner

**SYNOPSIS**

`cscan` [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Cscan* acquires black-and-white images using the Canon 9030 Laser Copier flatbed document scanner located in the Graphics Lab. Each *file* receives one image; multiple *files* are scanned as fast as possible (every 12s, after a startup delay). If no *file* is specified, one image is written to the standard output.

Each image is compressed using the CCITT FAX Group 4 standard and supplied with a header as described in [picfile\(5\)](#). The default page format is 3456 by 4400 pixels (*i.e.* letter-size 8.64"×11.0" at 400 pixels/inch digitizing resolution).

The options are:

- f***x,y*    Format is *x* by *y* pixels. *X* is truncated to a multiple of 32.
- fL**      Double-letter format **-f4416,6800** or 11"×17" at 400 pixels/inch (the largest possible).
- s***s*      Sleep between scans for an extra *s* seconds.
- v**        Verbose: announce each *file* as it arrives.

**SEE ALSO**

[bcp\(1\)](#), [ocr\(1\)](#), [picfile\(5\)](#)

**BUGS**

Although only 10s are required to acquire and compress each image, it can take 40s for the first file to appear, due largely to networking latencies involving [rcp\(1\)](#).

*Cscan* is unkillable: scanning continues and the files keep coming.

The automatic document feeder is not supported.

The resolution recorded in the header is always 400 pixels/inch, even though it is possible manually to zoom to other resolutions.

**NAME**

`cs`h – a shell (command interpreter) with C-like syntax

**SYNOPSIS**

`cs`h [ `-cefinstvVxX` ] [ `arg ...` ]

**DESCRIPTION**

*Csh* is a first implementation of a command language interpreter incorporating a history mechanism (see **History Substitutions**) job control facilities (see **Jobs**) and a C-like syntax. So as to be able to use its job control facilities, users of *csh* must (and automatically) use the new tty driver summarized in *newtty*(4) and fully described in *tty*(4). This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See *stty*(1) for details on setting options in the new tty driver.

An instance of *csh* begins by executing commands from the file `.cshrc` in the *home* directory of the invoker. If this is a login shell then it also executes commands from the file `.login` there. It is typical for users on crt's to put the command `"stty crt"` in their *.login* file, and to also invoke *tset*(1) there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with `%`. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file `.logout` in the users home directory.

**Lexical structure**

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters `&` `|` `;` `<` `>` `(` `)` form separate words. If doubled in `&&`, `| |`, `<<` or `>>` these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with `\`. A newline preceded by a `\` is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, `"`, `'` or `""`, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of `"` or `""` characters a newline preceded by a `\` gives a true newline character.

When the shell's input is not a terminal, the character `#` introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by `\` and in quotations using `"`, `'`, and `""`.

**Commands**

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by `|` characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by `;`, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an `&`.

Any of the above may be placed in `(` `)` to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with `| |` or `&&` indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions*.)

**Jobs**

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started asynchronously with `&`, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the jobs which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key `^Z` (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and

print another prompt. You can then manipulate the state of this job, putting it in the background with the *bg* command, or run some other commands and then eventually bring the job back into the foreground with the foreground command *fg*. A *^Z* takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key *^Y* which does not generate a STOP signal until a program attempts to *read(2)* it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command “*stty tostop*”. If you set this *tty* option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character ‘%’ introduces a job name. If you wish to refer to job number 1, you can name it as ‘%1’. Just naming a job brings it to the foreground; thus ‘%1’ is a synonym for ‘*fg %1*’, bringing job 1 back into the foreground. Similarly saying ‘%1 &’ resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus ‘%*ex*’ would normally restart a suspended *ex(1)* job, if there were only one suspended job whose name began with the string ‘*ex*’. It is also possible to say ‘%?string’ which specifies a job whose text contains *string*, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a ‘+’ and the previous job with a ‘-’. The abbreviation ‘%+’ refers to the current job and ‘%-’ refers to the previous job. For close analogy with the syntax of the *history* mechanism (described below), ‘%%’ is also a synonym for the current job.

### Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say ‘*notify*’ after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that ‘You have stopped jobs.’ You may use the *jobs* command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

### Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

#### History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character ‘!’ and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.) This ‘!’ may be preceded by an ‘\’ to prevent its special meaning; for convenience, a ‘!’ is passed unchanged when it is followed by a blank, tab, newline, ‘=’ or ‘.’. (History substitutions also occur when an input line begins with ‘*^*’. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the *history* variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the *history* command:

```

 9 write michael
10 ex write.c
11 cat oldwrite.c
```

## 12 diff \*write.c

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an '!' in the prompt string.

With the current event 13 we can refer to previous events by event number '!11', relatively as in '!-2' (referring to the same event), by a prefix of a command word as in '!d' for event 12 or '!wri' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a *redo*.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

0	first (command) word
<i>n</i>	<i>n</i> 'th argument
↑	first argument, i.e. '1'
\$	last argument
%	word matched by (immediately preceding) ?s? search
<i>x</i> - <i>y</i>	range of words
- <i>y</i>	abbreviates '0- <i>y</i> '
*	abbreviates '↑-\$', or nothing if only 1 word in event
<i>x</i> *	abbreviates ' <i>x</i> -\$'
<i>x</i> -	like ' <i>x</i> *' but omitting word '\$'

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '↑', '\$', '\*' '↑-\$' or '%'. After the optional word designator can be placed a sequence of modifiers, each preceded by a ':'. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
e	Remove all but the extension '.xxx' part.
s/l/r/	Substitute <i>l</i> for <i>r</i>
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a 'g' the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\' quotes the delimiter into the *l* and *r* strings. The character '&' in the right hand side is replaced by the text from the left. A '\' quotes '&' also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in '!?s?'. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing '?' in a contextual scan.

A history reference may be given without an event specification, e.g. '!\$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus '!?foo?↑ !\$' gives the first and last arguments from the command matching '?foo?'.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a '↑'. This is equivalent to '!:s↑' providing a convenient shorthand for substitutions on the text of the previous line. Thus '↑Ib↑lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls -ld ~paul' we might do '!{1}a' to do 'ls -ld ~paula', while '!la' would look for a command starting 'la'.

**Quotations with ' and "**

The quotation of strings by “” and “” can be used to prevent all or some of the remaining substitutions. Strings enclosed in “” are prevented any further interpretation. Strings enclosed in “” are yet variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a “” quoted string yield parts of more than one word; “” quoted strings never do.

### Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for ‘ls’ is ‘ls -l’ the command ‘ls /usr’ would map to ‘ls -l /usr’, the argument list here being undisturbed. Similarly if the alias for ‘lookup’ was ‘grep !↑ /etc/passwd’ then ‘lookup bill’ would map to ‘grep bill /etc/passwd’.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can ‘alias print ‘pr \!\* | lpr’ to make a command which *pr*’s its arguments to the line printer.

### Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell’s argument list, and words of this variable’s value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the *-v* command line option.

Other operations treat variables numerically. The ‘@’ command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by ‘\$’ characters. This expansion can be prevented by preceding the ‘\$’ with a ‘\’ except within “”s where it **always** occurs, and within “”s where it **never** occurs. Strings quoted by “” are interpreted later (see *Command substitution* below) so ‘\$’ substitution does not occur there until later, if at all. A ‘\$’ is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in “” or given the ‘:q’ modifier the results of variable substitution may eventually be command and filename substituted. Within “” a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the ‘:q’ modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

\$name

`${name}`

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but `:` modifiers and the other forms given below are not available in this case).

`$name[selector]`

`${name[selector]}`

May be used to select only some of the words from the value of *name*. The selector is subjected to '\$' substitution and may consist of a single number or two numbers separated by a '-'. The first word of a variables value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '\$#name'. The selector '\*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

`$#name`

`${#name}`

Gives the number of words in the variable. This is useful for later use in a '[selector]'.

`$0`

Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

`$number`

`${number}`

Equivalent to '\$argv[number]'.

`$*`

Equivalent to '\$argv[\*]'.

The modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces.

**The current implementation allows only one ':' modifier on each '\$' expansion.**

The following substitutions may not be modified with ':' modifiers.

`$?name`

`${?name}`

Substitutes the string '1' if name is set, '0' if it is not.

`$?0`

Substitutes '1' if the current input filename is know, '0' if it is not.

`$$`

Substitute the (decimal) process number of the (parent) shell.

`$<`

Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

### Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

### Command substitution

Command substitution is indicated by a command enclosed in ```. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within ```s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a

command substitution to yield only part of a word, even if the command outputs a complete line.

### Filename substitution

If a word contains any of the characters ‘\*’, ‘?’, ‘[’ or ‘{’ or begins with the character ‘~’, then that word is a candidate for filename substitution, also known as ‘globbing’. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters ‘\*’, ‘?’ and ‘[’ imply pattern matching, the characters ‘~’ and ‘{’ being more akin to abbreviations.

In matching filenames, the character ‘.’ at the beginning of a filename or immediately following a ‘/’, as well as the character ‘/’ must be matched explicitly. The character ‘\*’ matches any string of characters, including the null string. The character ‘?’ matches any single character. The sequence ‘[...]’ matches any one of the characters enclosed. Within ‘[...]’, a pair of characters separated by ‘-’ matches any character lexically between the two.

The character ‘~’ at the beginning of a filename is used to refer to home directories. Standing alone, i.e. ‘~’ it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and ‘-’ characters the shell searches for a user with that name and substitutes their home directory; thus ‘~ken’ might expand to ‘/usr/ken’ and ‘~ken/chmach’ to ‘/usr/ken/chmach’. If the character ‘~’ is followed by a character other than a letter or ‘/’ or appears not at the beginning of a word, it is left undisturbed.

The metanotation ‘a{b,c,d}e’ is a shorthand for ‘abe ace ade’. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus ‘~source/s1/{oldls,ls}.c’ expands to ‘/usr/source/s1/oldls.c /usr/source/s1/ls.c’ whether or not these files exist without any chance of error if the home directory for ‘source’ is ‘/usr/source’. Similarly ‘./{memo,\*box}’ might expand to ‘./memo ./box ./mbox’. (Note that ‘memo’ was not sorted with the results of matching ‘\*box’.) As a special case ‘{’, ‘}’ and ‘{}’ are passed undisturbed.

### Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< name

Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word

Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting ‘\’, ‘”’, ‘”’ or ‘`’ appears in *word* variable and command substitution is performed on the intervening lines, allowing ‘\’ to quote ‘\$’, ‘\’ and ‘`’. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name

>! name

>& name

>&! name

The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or ‘/dev/null’) or an error results. This helps prevent accidental destruction of files. In this case the ‘!’ forms can be used and suppress this check.

The forms involving ‘&’ route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as ‘<’ input filenames are.

>> name

>>& name

>>! name

>>&! name

Uses file *name* as standard output like '>' but places output at the end of the file. If the variable *no-clobber* is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is **not** modified to be the empty file '/dev/null'; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see **Jobs** above.)

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

### Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit*, *if*, and *while* commands. The following operators are available:

|| && | ↑ & == != =~ !~ <= >= < > << >> + - \* / % ! ~ ( )

Here the precedence increases to the right, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '\*' '/' and '%' being, in groups, at the same level. The '==' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '==' except that the right hand side is a *pattern* (containing, e.g. '\*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '-l name' where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

### Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if-then-else* form of the *if* statement require



that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

### Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

#### **alias**

**alias** name

**alias** name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

#### **alloc**

Shows the amount of dynamic core in use, broken down into used and free core, and address of the last location in the heap. With an argument shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether it is used or free. This is a debugging command and may not work in production versions of the shell; it requires a modified version of the system memory allocator.

#### **bg**

**bg** %job ...

Puts the current or specified jobs into the background, continuing them if they were stopped.

#### **break**

Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

#### **breaksw**

Causes a break from a *switch*, resuming after the *endsw*.

**case** label:

A label in a *switch* statement as discussed below.

#### **cd**

**cd** name

#### **chdir**

**chdir** name

Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './' or './.'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

#### **continue**

Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

#### **default:**

Labels the default case in a *switch* statement. The default should come after all *case* labels.

#### **dirs**

Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

**echo** wordlist

**echo -n** wordlist

The specified words are written to the shells standard output, separated by spaces, and terminated

with a newline unless the **-n** option is specified.

**else**  
**end**  
**endif**  
**endsw**

See the description of the *foreach*, *if*, *switch*, and *while* statements below.

**eval** arg ...

(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using *eval*.

**exec** command

The specified command is executed in place of the current shell.

**exit**

*exit*(*expr*)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

**fg**

**fg** %*job* ...

Brings the current or specified jobs into the foreground, continuing them if they were stopped.

**foreach** name (wordlist)

...

**end**

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

**glob** wordlist

Like *echo* but no '\ ' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

**goto** word

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

**hashstat**

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding *exec*'s). An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/ '.

**history**

**history** *n*

**history** -**r** *n*

Displays the history event list; if *n* is given only the *n* most recent events are printed. The -**r** option reverses the order of printout to be most recent first rather than oldest first.

**if** (*expr*) *command*

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

```

if (expr) then
  ...
else if (expr2) then
  ...
else
  ...
endif

```

If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second *else* are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else*.)

**jobs****jobs -l**

Lists the active jobs; given the **-l** options lists process id's in addition to the normal information.

```
kill %job
```

```
kill -sig %job ...
```

```
kill pid
```

```
kill -sig pid ...
```

```
kill -l
```

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix "SIG"). The signal names are listed by "kill -l". There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

**limit**

```
limit resource
```

```
limit resource maximum-use
```

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

Resources controllable currently include *cputime* (the maximum number of cpu-seconds to be used by each process), *filesize* (the largest single file which can be created), *datasize* (the maximum growth of the data+stack region via *sbrk(2)* beyond the end of the program text), *stacksize* (the maximum size of the automatically-extended stack region), and *coredumpsize* (the size of the largest core dump that will be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is 'k' or 'kilobytes' (1024 bytes); a scale factor of 'm' or 'megabytes' may also be used. For *cputime* the default scaling is 'seconds', while 'm' for minutes or 'h' for hours, or a time of the form 'mm:ss' giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

**login**

Terminate a login shell, replacing it with an instance of */bin/login*. This is one way to log off, included for compatibility with *sh(1)*.

**logout**

Terminate a login shell. Especially useful if *ignoreeof* is set.

**newgrp**

Changes the group identification of the caller; for details see *newgrp(1)*. A new shell is executed by *newgrp* so that the shell state is lost.

**nice**

```
nice +number
```

```
nice command
```

```
nice +number command
```

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number.

The final two forms run *command* at priority 4 and *number* respectively. The super-user may specify negative niceness by using ‘*nice -number ...*’. *Command* is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

**nohup****nohup** command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with ‘&’ are effectively *nohup*’ed.

**notify****notify** %job ...

Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

**onintr****onintr** -**onintr** label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form ‘*onintr -*’ causes all interrupts to be ignored. The final form causes the shell to execute a ‘*goto label*’ when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

**popd****popd** +n

Pops the directory stack, returning to the new top directory. With a argument ‘+*n*’ discards the *n*th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

**pushd****pushd** name**pushd** +n

With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (ala *cd*) and pushes the old current working directory (as in *csd*) onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

**rehash**

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

**repeat** count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

**set****set** name**set** name=word**set** name[index]=word**set** name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index*’th component of *name* to *word*; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

**setenv** name value

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variable USER, TERM, and PATH are automatically imported to and exported from the *cs*h variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

**shift**

**shift** variable

The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

**source** name

The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is **never** placed on the history list.

**stop**

**stop** %job ...

Stops the current or specified job which is executing in the background.

**suspend**

Causes the shell to stop in its tracks, much as if it had been sent a stop signal with **^Z**. This is most often used to stop shells started by *su*(1)

**switch** (string)

**case** str1:

...

**breaksw**

...

**default:**

...

**breaksw**

**endsw**

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '\*', '?' and '['...] may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

**time**

**time** command

With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

**umask**

**umask** value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

**unalias** pattern

All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias \*'. It is not an error for nothing to be *unaliased*.

**unhash**

Use of the internal hash table to speed location of executed programs is disabled.

**unlimit** *resource***unlimit**

Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed.

**unset** *pattern*

All variables whose names match the specified pattern are removed. Thus all variables are removed by ‘unset \*’; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset*.

**unsetenv** *pattern*

Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and [printenv\(1\)](#).

**wait**

All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

**while** (*expr*)

...

**end**

While the specified expression evaluates non-zero, the commands between the *while* and the matching *end* are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

**%job**

Brings the specified job into the foreground.

**%job &**

Continues the specified job in the background.

**@**

@ *name* = *expr*

@ *name*[*index*] = *expr*

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains ‘<’, ‘>’, ‘&’ or ‘|’ then at least this part of the expression must be placed within ‘( )’. The third form assigns the value of *expr* to the *index*’th argument of *name*. Both *name* and its *index*’th component must already exist.

The operators ‘\*’, ‘+’, etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix ‘++’ and ‘--’ operators increment and decrement *name* respectively, i.e. ‘@ i++’.

**Pre-defined and environment variables**

The following variables have special meaning to the shell. Of these, *argv*, *cwd*, *home*, *path*, *prompt*, *shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it. (It could be set once in the *.login* except that commands through *net(1)* would not see the definition.)

**argv**

Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. ‘\$1’ is replaced by ‘\$argv[1]’, etc.

<b>cdpath</b>	Gives a list of alternate directories searched to find subdirectories in <i>chdir</i> commands.
<b>cwd</b>	The full pathname of the current directory.
<b>echo</b>	Set when the <b>-x</b> command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.
<b>history</b>	Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of <i>history</i> may run the shell out of memory. The last executed command is always saved on the history list.
<b>home</b>	The home directory of the invoker, initialized from the environment. The filename expansion of <b>~</b> refers to this variable.
<b>ignoreeof</b>	If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.
<b>mail</b>	The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.  If the first word of the value of <i>mail</i> is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.  If multiple mail files are specified, then the shell says 'New mail in <i>name</i> ' when there is mail in the file <i>name</i> .
<b>noclobber</b>	As described in the section on <i>Input/output</i> , restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that <b>&gt;&gt;&gt;</b> redirections refer to existing files.
<b>noglob</b>	If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.
<b>nonomatch</b>	If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. <b>echo [</b> still gives an error.
<b>notify</b>	If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.
<b>path</b>	Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no <i>path</i> variable then only full path names will execute. The usual search path is <b>.</b> , <b>/bin</b> and <b>/usr/bin</b> , but this may vary from system to system. For the super-user the default search path is <b>/etc</b> , <b>/bin</b> and <b>/usr/bin</b> . A shell which is given neither the <b>-c</b> nor the <b>-t</b> option will normally hash the contents of the directories in the <i>path</i> variable after reading <i>.cshrc</i> , and each time the <i>path</i> variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the <i>rehash</i> or the commands may not be found.
<b>prompt</b>	The string which is printed before each command is read from an interactive terminal input. If a <b>!</b> appears in the string it will be replaced by the current event number unless a preceding <b>\</b> is given. Default is <b>%</b> , or <b>#</b> for the super-user.
<b>shell</b>	The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of <i>Non-builtin Command Execution</i> below.) Initialized to the (system-dependent) home of the shell.
<b>status</b>	The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status <b>1</b> , all other builtin commands set status <b>0</b> .

- time** Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.
- verbose** Set by the `-v` command line option, causes the words of each command to be printed after history substitution.

### Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *exec(2)*. Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a `-c` nor a `-t` option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a `-c` or `-t` argument, and in any case for each directory component of *path* which does not begin with a `'/'`, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus `(cd ; pwd) ; pwd` prints the *home* directory; leaving you where you were (printing this after the home directory), while `cd ; pwd` leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. `'$shell'`). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

### Argument list processing

If argument 0 to the shell is `'-'` then this is a login shell. The flag arguments are interpreted as follows:

- `-c` Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.
- `-e` The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.
- `-f` The shell will start faster, because it will neither search for nor execute commands from the file `'.cshrc'` in the invokers home directory.
- `-i` The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.
- `-n` Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.
- `-s` Command input is taken from the standard input.
- `-t` A single line of input is read and executed. A `'\'` may be used to escape the newline at the end of this line and continue onto another line.
- `-v` Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.
- `-x` Causes the *echo* variable to be set, so that commands are echoed immediately before execution.
- `-V` Causes the *verbose* variable to be set even before `'.cshrc'` is executed.
- `-X` Is to `-x` as `-V` is to `-v`.

After processing of flag arguments if arguments remain but none of the `-c`, `-i`, `-s`, or `-t` options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by `'$0'`. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a 'standard' shell if the first character of a script is not a `'#'`, i.e. if the script does not start with a comment.



Remaining arguments initialize the variable *argv*.

### Signal handling

The shell normally ignores *quit* signals. Jobs running detached (either by ‘&’ or the *bg* or *%... &* commands) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell’s parent. In no case are interrupts allowed when a login shell is reading the file ‘.logout’.

### AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now.

### FILES

~/cshrc	Read at beginning of execution by each shell.
~/login	Read by login shell, after ‘.cshrc’ at login.
~/logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a ‘#’.
/tmp/sh*	Temporary file for ‘<<<’.
/etc/passwd	Source of home directories for ‘~name’.

### LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6’th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

### SEE ALSO

sh(1), newcsh(1), access(2), exec(2), fork(2), killpg(2), pipe(2), sigsys(2), umask(2), vlimit(2), wait(2), jobs(3), sigset(3), tty(4), a.out(5), environ(5), ‘An introduction to the C shell’

### BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (i.e. wrong) as the job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form ‘a ; b ; c’ are also not handled gracefully when stopping is attempted. If you suspend ‘b’, the shell will then immediately execute ‘c’. This is especially noticeable if this expansion results from an *alias*. It suffices to place the sequence of commands in ()’s to force it to a subshell, i.e. ‘( a ; b ; c )’.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by ‘?’, are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with ‘|’, and to be used with ‘&’ and ‘;’ metasyntax.

It should be possible to use the ‘:’ modifiers on the output of command substitutions. All and more than one ‘:’ modifier should be allowed on ‘\$’ substitutions.

**NAME**

ctags – create a tags file

**SYNOPSIS**

**ctags** [ **-BFatuwx** ] name ...

**DESCRIPTION**

*Ctags* makes a tags file for *ex*(1) from the specified C, Pascal and Fortran sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these objects definitions.

If the **-x** flag is given, *ctags* produces a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

Files whose name ends in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

- F** use forward searching patterns (*/.../*) (default).
- B** use backward searching patterns (*?...?*).
- a** append to tags file.
- t** create tags for typedefs.
- w** suppressing warning diagnostics.
- u** causing the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing **.c** removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

**FILES**

tags                    output tags file

**SEE ALSO**

*ex*(1), *vi*(1)

**AUTHOR**

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and **-x**, replacing *cxref*; C typedefs added by Ed Pelegri-Llopart.

**BUGS**

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done in a very simple-minded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

Does not know about **#ifdefs**.

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **-tx** shows only the last line of typedefs.

**NAME**

cu, ct – call out to a terminal or another system

**SYNOPSIS**

**cu** [ **-htn** ] [ **-p** *parity* ] [ **-s** *speed* ] *telno* [ *service-class* ]

**ct** [ option ... ] *phone-number* [ *service-class* ]

**DESCRIPTION**

*Cu* places a data call to a given telephone number and expects a computer to answer. It manages an interactive conversation with possible transfers of text files. *Telno* is the telephone number, consisting of digits with minus signs at appropriate places to indicate delay for second or subsequent dial tones. A telephone number may also be expressed symbolically. A symbolic number is looked up in the files and whose lines look like this:

```
symbolic-number actual-number service-class comment
```

The actual number may be preceded by options such as **-t**. The *comment*, if present, is printed out when the connection is made.

The options are

- n** Print the the called number but do not call it.
- t** Tandem: use DC1/DC3 (control-**S**/control-**Q**) protocol to stop transmission from the remote system when the local terminal buffers are almost full. This argument should only be used if the remote system understands that protocol.
- h** Half-duplex: echo locally the characters that are sent to the remote system.
- s** *speed*  
Set the line speed; means 1200 baud, etc. The default depends on service class.
- p** *parity*  
Set the parity of transmitted characters: **0**, **1**, **e**, **o** mean zero, one, even, odd parity. **0** is the default.

The service class is expressed as in [dialout\(3\)](#). A special class causes the *telno* argument to be taken as the pathname of a terminal line. *Cu* opens the file, sets line speed and other modes, and proceeds as if connected. The default line speed is 9600 baud.

An explicit service class on the command line overrides any specified in a file.

After making the connection, *cu* runs as two processes: the sending process reads the standard input and passes most of it to the remote system; the receiving process reads from the remote system and passes most data to the standard output. Lines beginning with have special meanings.

The sending process interprets:

- ~  
~**EOT** Terminate the conversation.
- ~<*file* Send the contents of *file* to the remote system, as though typed at the terminal.
- ~!  
~!*cmd* Run the command on the local system (via
- ~\$*cmd* Run the command locally and send its output to the remote system.
- ~**b**  
~%**break**  
Send a break (300 ms space).
- ~%**take** *from* [*to*]  
Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* name is used both places.
- ~%**put** *from* [*to*]  
Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* name is used both places.

*~text* send the line *~text*.

WARNING: Using *cu* to reach your home machine from a machine you don't trust can be hazardous to your password.

*Ct* places a telephone call to a remote terminal and allows a user to log in on that terminal in the normal fashion. The terminal must be equipped with an auto-answer modem.

The phone number and service class are as in *cu*. The options are

**-c** *count*

If the number doesn't answer, try *count* times before giving up (default 5).

**-w** *interval*

Space retries *interval* seconds apart (default 60).

**-h** Try to hang up the phone before placing the call. This is useful for a 'call me right back' arrangement.

## FILES

## SEE ALSO

[con\(1\)](#), [ttyld\(4\)](#), [dialout\(3\)](#)

## BUGS

Unless erase and kill characters are the same on the two machines, they will be damaged by **~%put**.

**~%take** uses **~>** at the beginning of line to synchronize transmission. This sequence can cause misfunction if it is received for any other purpose.

**NAME**

cut, paste – rearrange columns of data

**SYNOPSIS**

**cut** **-clist** [ *file* ... ]

**cut** **-flist** [ **-dchar** ] [ *file* ... ]

**paste** [ **-s** ] [ **-dchars** ] *file* ...

**DESCRIPTION**

*Cut* selects fields from each line of the *files* (standard input default). In data base parlance, it projects a relation. The fields can be fixed length, as on a punched card (option **-c**), or be marked with a delimiter character (option **-f**).

The meanings of the options follow. A *list* is an increasing sequence of integers separated by commas, or by **-** to indicate a range, for example

**-clist** The *list* specifies character positions.

**-flist** The *list* specifies field numbers.

**-dchar**

The character is the delimiter for option **-f**. Default is tab.

**-s** Suppress lines with no delimiter characters. Normally such lines pass through untouched under option **-f**.

*Paste* concatenates corresponding lines of the input *files* and places the result on the standard output. The file name refers to the standard input. Lines are glued together with characters taken circularly from the set *chars*. The set may contain the special escape sequences **\n** (newline), **\t** (tab), **\\** (backslash), and **\0** (empty string, not a null character). The options are

**-dchars**

The output separator characters. Default is a tab.

**-s** Paste together lines of one file, treating every line as if it came from a distinct input.

**EXAMPLES**

```
cut -d: -f1,3 /etc/passwd
```

Print map from login names to userids, see [passwd\(5\)](#).

```
NAME='who am i | cut -f1 -d" "'
```

Set to current login name (subtly different from [getuid\(1\)](#)).

```
ls | paste - - - -
```

```
ls | paste -s '-d\t\n' -
```

4-column and 2-column file listing

**SEE ALSO**

[gre\(1\)](#), [awk\(1\)](#), [sed\(1\)](#), [pr\(1\)](#), [column\(1\)](#)

**BUGS**

*Cut* should handle disordered lists under option **-f**.

In default of file names, *paste* should read the the standard input.

**NAME**

*d202*, *tc* – typesetter filters

**SYNOPSIS**

**d202** [ *option ...* ] [ *file ...* ]

**e202** [ *option ...* ] [ *file ...* ]

**tc** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*D202* and its companions print files created by [troff\(1\)](#) on various devices:

*d202* Mergenthaler Linotron 202

*e202* same, with half-tone and extra graphics capability

*tc* Tektronix 4014 display

If no *file* is mentioned, the standard input is printed. The following options are understood.

- b** Report whether the typesetter is busy; do not print.
- olist** Print pages whose numbers are given in the comma-separated *list*. The list contains comma-separated numbers *N* and ranges *N1-N2*. A missing *N1* means the lowest-numbered page, a missing *N2* means the highest.
- sn** Stop after every *n* pages of output. (Default 1 on 4014). Proceed when the ‘RUN’ button is pushed on the typesetter (*d202*) or newline on the terminal (*tc*).
- t** Direct output to the standard output instead of the typesetter. Don’t wait between pages in *tc*.
- w** Wait for typesetter to become free, then print.
- f dir** Take font information from directory *dir* instead of the default.
- ar** Set the aspect ratio to *r* (default *r*=1.5). *Tc* only.

While waiting between pages *tc* accepts *!command* to insert a shell command; *+n* to skip forward *n* pages; *-n* to skip backwards *n* pages; *ar* to set the aspect ratio; and *?* to print the list of available actions.

**FILES**

202 description files

**SEE ALSO**

[lp\(1\)](#), [troff\(1\)](#), [proof\(9\)](#) [apsend\(1\)](#), [font\(5\)](#)

**BUGS**

*E202*, largely a superset of *d202*, should be combined with *d202*.

**NAME**

dag – preprocessor for drawing directed graphs

**SYNOPSIS**

**dag** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Dag* is a *pic*(1) or PostScript preprocessor for laying out directed graphs. It does well on acyclic graphs (dags) and other graphs that can be drawn hierarchically. Graph statements are contained between **.GD** (node ranks increase downward) or **.GR** (rightward) and **.GE**. Edges point in the direction of increasing rank if possible; the other direction is favored for edges within rank. A summary of statements follows.

**edge from** *tail to head0 edge-items, to head1 edge-items, to head2 edge-items...*; Create edges from the tail node to the head node(s). Nodes are created if they do not already exist. Edge-items (described below) and the noise words **edge**, **from**, **to**, and comma are optional. Node names may be quoted to protect blanks or keywords.

**ordered** *tail head0 head1 head2...*; Make edges with heads in given left-to-right order on the same rank. May contain noise words and edge-items.

**path** *node0 node1 node2...*; Make a path of edges. May contain noise words and edge-items.

**backedge** *tail head0 head1 head2...*; Same as edges with opposite node ranking preferred.

**backpath** *node0 node1 node2...*; Make a path of backedges.

**draw nodes** *node-items*; Set properties of subsequently created nodes. Legal *node-items*:

**as** *shape*

Known shapes are **Box**, **Circle**, **Doublecircle**, **Ellipse** (default), **Diamond**, **Square**, **Plaintext**. Other shapes may be specified within braces {} in the output language (e.g. *pic*) or defined; see below.

**label** "*string*"

Label with *string* instead of node name.

**pointsize** *points*

**height***inches*

**color** "*string*"

Hue-saturation-brightness triple; works only with PostScript.

**draw** *odelist node-items*; Set properties of listed nodes.

**draw edges** *edge-items*; Set properties of subsequently created edges. Legal *edge-items*:

**dotted**, **dashed**, **solid**, **invis**

**label** "*string*"

**weight** *n*

High-weight edges are kept short. Default weight 1.

**color** "*string*"

**pointsize** *points*

**minimum rank** *odelist*; Listed nodes must be on the topmost rank (leftmost with **.GR**).

**maximum rank** *odelist*; Bottommost or rightmost rank.

**same rank** *odelist*;

**separate ranks** *inches how* ; Set minimum separation between ranks. The optional *how* is **exactly** or **equally**.

**separate nodes** *inches* ;

The options are

**-O** Place nodes ‘optimally’; practical for graphs of a few dozen nodes.

- Tps** Prepare output for PostScript rather than *pic*.
- Tsimple** Output similar to that of *graphdraw(9)*
- Tcip** Output readable by *cip(9)*
- pwidthxheight,marginwidthxmarginheight** Set PostScript page dimensions; *marginwidth* and/or *marginheight* may be omitted.
- l** Disable automatic loading of the *dag* graphics library.

The introductory **.GD** or **.GR** line may carry optional parameters in the form **.GD width height fill**. *Width* and *height* are maximum values in inches; **fill** causes the graph to be stretched to the full dimensions.

Graphics code written in the output language (*pic* or PostScript) may be embedded between **.PS** and **.PE**. Macros defined with three arguments—label, width, and height—may be used as shape names in node-items.

## EXAMPLES

```
.GD 2 2
a b c;
path a x y;
draw nodes as Box;
a z label "hi" weight 1000;
draw edges dashed;
b x;
same rank b x;
.GE
a b c; path a x y; draw nodes as Box; a z label "hi" weight 1000; draw edges dashed; b x; same rank b x;
```

## FILES

default  
**TOOLS=/usr/lib**

## SEE ALSO

*pic(1)*, *lp(1)*, *graphdraw(9)* *psi(9)* *troff(1)*  
 E. R. Gansner, S. C. North, K. P. Vo, ‘DAG—A Program that Draws Directed Graphs’, this manual, Volume 2

## BUGS

The delimiter **.GD** is nonstandard; it may be called **.GS** in installations where *ped(9)* is not used.  
*Troff* lacks dotted or dashed splines; use PostScript.  
 Edge labels may overlap.



**NAME**

date – print or set the date

**SYNOPSIS**

**date** [ *yymmddhhmm* [ . *ss* ] ]

**DESCRIPTION**

If no argument is given, the current date and time are printed. If an argument is given and the user is the super-user, the current date is set. *yy* is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* is optional and is the seconds. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time. The options are

- u** Set or report GMT rather than local time.
- n** Set or report the date as the number of seconds since the epoch.

**EXAMPLES**

date 10080045

Set the date to Oct 8, 12:45 AM, local time.

**FILES**

to record time-setting

**SEE ALSO**

[utmp\(5\)](#)

**DIAGNOSTICS**

‘No permission’ if a non-super user tries to change the date; ‘bad conversion’ if the date is invalid.

**NAME**

dc – desk calculator

**SYNOPSIS**

dc [ *file* ]

**DESCRIPTION**

*Dc* is an arbitrary precision desk calculator. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number

The value of the number is pushed on the stack. A number is an unbroken string of the digits **0-9A-F**. It may be preceded by an underscore **\_** to input a negative number. Numbers may contain decimal points.

+ - / \* % ^

Add subtract multiply divide remainder or exponentiate the top two values on the stack. The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

**s***x*

**S***x* Pop the top of the stack and store into a register named *x*, where *x* may be any character. Under operation **S** register *x* is treated as a stack and the value is pushed on it.

**l***x*

**L***x* Push the value in register *x* onto the stack. The register *x* is not altered. All registers start with zero value. Under operation **L** register *x* is treated as a stack and its top value is popped onto the main stack.

**d** Duplicate the top value on the stack.

**p** Print the top value on the stack. The top value remains unchanged. **P** interprets the top of the stack as an ASCII string, removes it, and prints it.

**f** Print the values on the stack.

**q**

**Q** Exit the program. If executing a string, the recursion level is popped by two. Under operation **Q** the top value on the stack is popped and the string execution level is popped by that value.

**x** Treat the top element of the stack as a character string and execute it as a string of *dc* commands.

**X** Replace the number on the top of the stack with its scale factor.

[ ... ] Put the bracketed ASCII string on the top of the stack.

<*x*>*x*=*x*

Pop and compare the top two elements of the stack. Register *x* is executed if they obey the stated relation.

**v** Replace the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

**!** Interpret the rest of the line as a UNIX command.

**c** Clear the stack.

**i** The top value on the stack is popped and used as the number base for further input.

**I** Push the input base on the top of the stack.

**o** The top value on the stack is popped and used as the number base for further output. In bases larger than 10, each ‘digit’ prints as a group of decimal digits.

**O** Push the output base on the top of the stack.

- k** Pop the top of the stack, and use that value as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
- z** Push the stack level is pushed onto the stack.
- Z** Replace the number on the top of the stack with its length.
- ?** A line of input is taken from the input source (usually the terminal) and executed.
- ;** Used by *bc* for array operations.

The scale factor set by **k** determines how many digits are kept to the right of the decimal point. If *s* is the current scale factor, *sa* is the scale of the first operand, *sb* is the scale of the second, and *b* is the (integer) second operand, results are truncated to the following scales.

[CB]+,[CB]-  $\max(sa, sb)$   
 [CB]\*  $\min(sa+sb, \max(s, sa, sb))$   
 [CB]/  $s$   
 [CB]% so that dividend = divisor\*quotient + remainder; remainder has sign of dividend  
 [CB]^  $\min(sa \times b, \max(s, sa))$   
 [CB]v  $\max(s, sa)$

## EXAMPLES

```
[1a1+dsa*pla10>y]sy
0sa1
lyx
```

Print the first ten values of *n*!

## SEE ALSO

[bc\(1\)](#), [hoc\(1\)](#)

## DIAGNOSTICS

- x* where *x* is an octal number: an internal error.
- 'Out of headers' for too many numbers being kept around.
- 'Nesting depth' for too many levels of nested execution.

## BUGS

When the input base exceeds 16, there is no notation for digits greater than **F**.

**NAME**

*dcon*, *ndcon*, *rlogin*, *nrx*, *rsh*, *scriptcon* – remote login and execution

**SYNOPSIS**

**dcon** [ *option ...* ] *machine*

**ndcon** *machine*

**rlogin** *machine*

**nrx** *machine* [ *command-word ...* ]

**rsh** [ *option ...* ] *machine* [ *command-word ...* ]

**scriptcon** *machine script*

**DESCRIPTION**

Do not read this page unless you are familiar with *con(1)*.

*Dcon*, *ndcon*, and *rlogin* are analogs (or special cases) of *con(1)* for specific kinds of network connection. They support the same local escape convention with the quit signal.

Similarly, *nrx* and *rsh* are analogs of *rx*.

Network addresses are as in *con(1)*. The default networks for the various commands are

*dcon*, *ndcon*, *nrx*

**dk**

*rlogin*, *rsh*      **tcp**

*Dcon* connects to the remote machine, and attempts automatically to log in under the login id of the invoking user. Option **-l** turns off automatic login; the remote machine will ask for a login id and password.

*Ndcon* behaves like *dcon* but provides a more transparent transport protocol. In particular terminal line disciplines are preserved and it is possible to download into a *mux(9)* window across an *ndcon* connection.

*Rlogin* is like *dcon*, but uses the connection protocol found on Berkeley systems.

*Rx* (see *con(1)*) executes one shell command on the remote machine as if logged in there, with local standard input and output. It uses a connection protocol specific to Research machines.

*Nrx* is to *rx* as *ndcon* is to *dcon*: it runs a command remotely with line discipline preserved.

*Rsh* is to *rx* as *rlogin* is to *dcon*: it runs a command remotely using the Berkeley execution protocol.

*Scriptcon* provides a connection like **dcon -l**, except that the login and other initial protocol are controlled by a *script* file. The first line of the file gives a string (e.g. expected from the remote machine; the second gives the local response, and so on in alternation. Unrecognized data from the remote machine are ignored. Warning: a script that contains a password may compromise the security of the remote system, hence *scriptcon* should be used only for restricted logins.

*Con* tries to connect using the protocol of *ndcon*; if that fails, it tries that of *dcon*, then that of *rlogin*. *Rx* attempts its own style of connection; if that fails, it tries that of *rsh*.

**SEE ALSO**

*con(1)*, *dkmgr(8)*, *svcmgr(8)*, *tcpmgr(8)*, *ipc(3)*

D. L. Presotto, ‘Interprocess Communication in the Eighth Edition UNIX System’, this manual, Volume 2

**BUGS**

If a program run by *nrx* won’t let go, for example by ignoring signals, there is no way of getting out short of hanging up.

There is no error correction or retry in a *scriptcon* script.

**NAME**

dd, ddbuf – convert and copy a file

**SYNOPSIS**

**dd** [ *option=value* ]

**ddbuf** [ **-b** *blocksize* ] [ *file* ]

**DESCRIPTION**

*Dd* copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O. The options are

**if=***file*

Set the input file (standard input by default).

**of=***file*

Set the output file (standard output by default).

**ibs=***n* Set input block size to *n* bytes (default 512).

**obs=***n*

Set output block size (default 512).

**bs=***n* Set both input and output block size, superseding *ibs* and *obs*. If no conversion is specified, preserve the input block size instead of packing short blocks into the output buffer. This is particularly efficient since no in-core copy need be done.

**cbs=***n*

Set conversion buffer size.

**skip=***n*

Skip *n* input records before copying.

**iseek=***n*

Seek *n* records forward on input file before copying.

**files=***n*

Copy and concatenate *n* input files (makes sense only where input is a magnetic tape or similar device).

**oseek=***n*

Seek *n* records from beginning of output file before copying.

**count=***n*

Copy only *n* input records.

**conv=ascii** Convert EBCDIC to ASCII.

**ebcdic**

Convert ASCII to EBCDIC.

**ibm** Like **ebcdic** but with a slightly different character map.

**block** Convert variable length ASCII records to fixed length.

**unblock**

Convert fixed length ASCII records to variable length.

**lcase** Map alphabetics to lower case.

**ucase** Map alphabetics to upper case.

**swab** Swap every pair of bytes.

**noerror**

Do not stop processing on an error.

**sync** Pad every input record to *ibs* bytes.

Where sizes are specified, a number of bytes is expected. A number may end with *o* to specify multiplication by 1024, *s* for 512, or *b* for 2 respectively; a pair of numbers may be separated by *x* to indicate a product. Multiple conversions may be specified in the style:

*lcase* is used only if *conv* or conversion is specified. In the first two cases, *n* characters are copied into the conversion buffer, any specified character mapping is done, trailing blanks are trimmed and new-line is added before

sending the line to the output. In the latter three cases, characters are read into the conversion buffer and blanks are added to make up an output record of size *n*. If *n* is unspecified or zero, the `and` options convert the character set without changing the block structure of the input file; the `and` options become a simple file copy.

*Dblbuf* copies the named *file*, or the standard input if no file is specified, to the standard output. Output is written in blocks matching the input up to the given blocksize, or 32768 bytes if not specified.

*Dblbuf* uses multiple processes to run faster, which is particularly useful in dealing with a device such as a streaming tape drive.

## EXAMPLES

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into an ASCII file.

Note the use of raw magtape to handle arbitrary record sizes.

```
tar cf /dev/stdout /usr | dblbuf >/dev/rmt1
```

Copy the directory to tape on

## SEE ALSO

[cp\(1\)](#), [tar\(1\)](#), [cpio\(1\)](#)

## DIAGNOSTICS

*Dd* reports the number of full + partial input and output blocks handled.

## BUGS

The ASCII/EBCDIC conversion tables for *dd* were taken from the 256-character standard in CACM Nov, 1968. The conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

Options **if** and **of** are verbose equivalents of `<` and `>`.

**NAME**

deroff, demonk, detex, delatex – remove formatting requests

**SYNOPSIS**

**deroff** [ *option ...* ] *file ...*

**demonk** [ *option ...* ] *file ...*

**detex** *file*

**delatex** *file*

**DESCRIPTION**

*Deroff* reads each file in sequence and removes all *nroff* and *troff*(1) requests and non-text arguments, backslash constructions, and constructs of preprocessors such as *eqn*, *pic*, and *tbl*(1). Remaining text is written on the standard output. *Deroff* follows files included by *and* commands; if a file has already been included, a *for* that file is ignored and a *terminates* execution. If no input file is given, *deroff* reads from standard input.

The options are

- w** Output a word list, one ‘word’ (string of letters, digits, and properly embedded ampersands and apostrophes, beginning with a letter) per line. Other characters are skipped. Otherwise, the output follows the original, with the deletions mentioned above.
- i** Ignore *and* requests.
- ms**
- mm** Remove titles, attachments, etc., as well as ordinary *troff* constructs, from *ms*(6) or *mm* documents.
- ml** Same as **-mm**, but remove lists as well.

*Demonk* removes all *monk*(1) commands and then invokes *deroff* to handle both *troff* commands and preprocessor constructs. *Demonk* follows files included by *and* commands as well as *troff* and requests. If no input file is given, *demonk* reads from standard input.

*Demonk* recognizes the following options and passes all options except **-b** and **-d** to *deroff*.

- i** Ignore *monk* and commands as well as *troff* and requests.
- b** Do not output blank lines resulting from the removal of *monk* commands.
- ddir** Use non-standard *monk* database directory *dir*.

*Detex* and *delatex* do for *tex*(1) and *latex*(6) files what **deroff -w** does for *troff* files. *Delatex* largely subsumes *detex*.

**SEE ALSO**

*troff*(1), *monk*(1), *tex*(1), *spell*(1), *wwb*(1)

**BUGS**

These filters are not complete interpreters of *troff* or *tex*. For example, macro definitions containing *cause* chaos in *deroff* when the popular delimiters for *eqn* are in effect.

Text inside macros is emitted at place of definition, not place of call.

**NAME**

dictadd – add phrases to user’s diction or sexist dictionary

**SYNOPSIS**

**dictadd** [ **-flags** ] [ **-ver** ]

**DESCRIPTION**

*Dictadd* adds words and/or phrases to the user’s dictionaries for use by the *wwb(1)*, *proofr(1)*, *dictplus(1)*, *diction(1)*, and *sexist(1)* programs. *Sexist* searches a text for sexist phrases, while the other programs search a text for wordy or misused diction. These programs allow the user to have dictionary files containing additional words and/or phrases for the programs to locate or ignore. *Dictadd* automatically sets up these dictionary files.

*Dictadd* asks users whether they want to add words to *\$HOME/lib/ddict*, *\$HOME/lib/sexdict*, or some other file. The results depend on the user’s response, as shown below.

User’s request	<i>Dictadd</i> ’s action
<i>\$HOME/lib/ddict</i>	adds words and/or phrases to the user’s dictionary, <i>\$HOME/lib/ddict</i> . This file is checked automatically by <i>wwb</i> and <i>proofr</i> and can be specified for use by <i>dictplus</i> and <i>diction</i> . (See <i>diction(1)</i> )
<i>\$HOME/lib/sexdict</i>	adds words and/or phrases to the user’s dictionary, <i>\$HOME/lib/sexdict</i> . This file is checked automatically by <i>sexist</i> .
<i>filename</i>	adds words to <i>filename</i> , to be used with <i>diction</i> , <i>dictplus</i> , or <i>sexist</i> .

In all cases, *dictadd* questions whether the user wants instructions, and prompts with ">" for more words or phrases. If the dictionary is not in existence when *dictadd* is invoked, it is created. If the dictionary already exists, *dictadd* adds to it. To quit, type "q" after the prompt.

Two options give information about the program:

- flags** print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer’s Workbench version number of the command, then exit.

**EXAMPLE**

1. The sequence:

```

dictadd (carriage return)
(program asks if the user wants the words to be added to $HOME/lib/ddict)
y (user responds yes)
(program asks if the user wants instructions)
[~]phrase 1 (carriage return)
[~]phrase 2 (carriage return)
[~]phrase n (carriage return)
    
```

will add phrases to *\$HOME/lib/ddict*. Phrases to be ignored must be preceded by a tilde(~), phrases to be located require no special symbol. When finished, type "q" on a line by itself.

**SEE ALSO**

*diction(1)*, *dictplus(1)*, *proofr(1)*, *sexist(1)*, *suggest(1)*, *wwb(1)*.

**SUPPORT**

**COMPONENT NAME:** Writer’s Workbench  
**APPROVAL AUTHORITY:** Div 452  
**STATUS:** Standard  
**SUPPLIER:** Dept 45271  
**USER INTERFACE:** Stacey Keenan, Dept 45271, PY x3733  
**SUPPORT LEVEL:** Class B - unqualified support other than Div 452



**NAME**

dictplus – automatic combination of diction and suggest

**SYNOPSIS**

**dictplus** [ **-flags** ] [ **-ver** ] [ **-f** *pfile* [ **-n** ] ] [ file ... ]

**DESCRIPTION**

*Dictplus* automatically combines *diction* and *suggest*. Options are:

- f** *pfile*    Use the user's phrase file, *pfile*, in addition to the default file of bad or wordy diction. *Dictadd*(1) can be used to set up this file.
- n**            Locate the phrases in *pfile* instead of the default phrase file. **-n** cannot be used without **-f** *pfile*.

*Dictplus* is one of the programs run under the *proofr*(1) and *wwb*(1) commands.

*Dictadd*(1) adds words and/or phrases that are to be located or ignored by *diction* or *dictplus* to the user's dictionary, *\$HOME/lib/ddict*. *Dictadd* gives instructions on the necessary format for phrases to be located or ignored by *diction* or *dictplus*. *\$HOME/lib/ddict* is only used by *diction* and *dictplus* when it is specified by the **-f** flag. *Proofr*(1) checks *\$HOME/lib/ddict* automatically when it runs *dictplus*.

All programs can take the following two options that give information on the programs:

- flags**            print the command synopsis line (see above) showing command flags and options, then exit.
- ver**            print the Writer's Workbench version number of the command, then exit.

**EXAMPLES**

1. The command:

**dictplus -f patfile filename**

will print sentences from *filename* that contain bad or wordy diction, including or suppressing phrases as specified in *patfile*. Suggested replacements for bad phrases will also be printed. (The *patfile* can be *\$HOME/lib/ddict*.)

**FILES**

/tmp/\$\$\*            temporary files used by *dictplus*

**SEE ALSO**

*diction*(1), *suggest*(1), *proofr*(1), *wwb*(1), *worduse*(1), *sexist*(1), *dictadd*(1).

**SUPPORT**

*COMPONENT NAME*: Writer's Workbench

*APPROVAL AUTHORITY*: Div 452

*STATUS*: Standard

*SUPPLIER*: Dept 45271

*USER INTERFACE*: Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL*: Class B - unqualified support other than Div 452

**NAME**

diff, diff3 – differential file comparison

**SYNOPSIS**

**diff** [ *option ...* ] *file1 file2*

**diff3** [ **-ex3** ] *file1 file2 file3*

**DESCRIPTION**

When run on regular files *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of differences. If neither file is a directory, then one may be meaning the standard input. If one file is a directory, then a file in that directory with basename the same as that of the other file is used.

If both files are directories, similarly named files in the two directories are compared by the method of *diff* for text files and *cmp(1)* otherwise. Options when comparing directories are:

- r** Apply *diff* recursively to similarly named subdirectories.
- s** Report files that are the same (normally not mentioned).

There are several options for output format; the default output format contains lines of these forms:

```
n1a n3,n4
n1,n2d n3
n1,n2c n3,n4
```

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging for and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where  $n1=n2$  or  $n3 = n4$  are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by then all the lines that are affected in the second file flagged by

- e** Produce a script of and for *ed(1)* to recreate *file2* from *file1*. When comparing directories, produce a *sh(1)* script to convert text files common to the two directories.
- cn** Include *n* extra lines of context with each set of differences. The output format is modified: the output begins with identification of the files involved and their creation dates and changes are separated by lines of \*'s. Lines removed from *file1* are marked with those added to *file2* are marked Lines which are changed from one file to the other are marked in both files with
- h** Do a fast, half-hearted job, useful only when changed stretches are short and well separated, but does work on files of unlimited length.
- b** Ignore trailing blanks (spaces and tabs) and treat other strings of blanks as if they were a single space.
- B** Ignore all blanks.

*Diff3* compares three versions of a file and publishes the various disagreeing ranges of text. One of the following indicators introduces each reported difference.

```
==== All three files differ.
```

```
====f
```

File *f* differs, where *f* is 1, 2, or 3.

Disagreeing fragments from the three files follow the ===== line, each identified by a *diff*-like range indication:

*f:n1a* File *f* lacks text that other files have; their text would be appended after line *n1*.

*f:n1,n2c*

*f:n1c* Lines *n1* through *n2* (or line *n1* only) of file *f* would have to be changed to agree with some other file. The original contents follow, unless a higher-numbered file has the same contents.

Under option **-e**, *diff3* publishes a script for *ed(1)* that will incorporate into *file1* all changes between *file2* and *file3*, i.e. the changes that normally would be flagged ===== and =====**3**. Option **-x** (**-3**) produces a script to incorporate only changes flagged ===== (=====**3**).

**EXAMPLES**

```
(cat diff0-1 diff1-2 diff2-3; echo '1,$p') | ed - file0 >file3
```

An ancestral has been kept along with a chain of version-to-version difference files made thus:  
The shell command reconstructs the latest version.

```
if diff3 mod1 old mod2 | grep -s '^====$'  
then :  
else (diff3 -e mod1 old mod2; echo '1,$p') | ed - mod1 >new  
fi
```

Compare two different modified versions with an old file. If no modifications interfere with each other (*grep* finds no `====` lines), make a new file incorporating both modifications.

**FILES**

for **-h**

**SEE ALSO**

[cmp\(1\)](#), [comm\(1\)](#), [ed\(1\)](#), [idiff\(1\)](#)

**DIAGNOSTICS**

*Diff* yields exit status is 0 for no differences, 1 for some, 2 for trouble.

**BUGS**

Text lines that consist of a single `.` will defeat options **-e**, **-x**, and **-3**.

Superfluous output may result for files that differ only in insignificant blanks when comparing directories under option **-b**.

Option **-c** is unpardonably bizarre.

**NAME**

dired – directory editor

**SYNOPSIS**

**dired** [ *option ...* ] [ *file* ]

**DESCRIPTION**

*Dired* displays a directory listing like (see [ls\(1\)](#)) and allows you to prowl around the listed entries, deleting, editing, and displaying them. It requires a cursor-addressed terminal identified in environment variable **TERM**; see [term\(9\)](#) to simulate such terminals in [mux\(9\)](#)

With no *file* argument, the current directory is listed. With only one *file* argument, the argument is interpreted as a directory and it is listed. With multiple arguments, the arguments are interpreted as filenames. The options are:

**-[sr][nsrw]**

Sort the file list by name (default), size, access time, or modification time. Ordering for **s** is increasing if by name, decreasing otherwise. Ordering is opposite for **r**.

**-wn** If *n* is **f**, use the full screen; if **h**, use half the screen (default); if a number, use *n* lines for the directory listing, reserving the rest of the screen for quick file display.

The fields of a *dired* listing are: mode, link count, owner, size, write date and name. A cursor shows the current entry.

Commands consist of single characters; arguments are prompted for at the bottom of the screen. To get a complete list, use the help command. Fuller descriptions of less obvious commands are given below.

- !** Prompt for a shell command. The command is executed with characters in the command are replaced by the pathname of the current entry, and characters by the basename.
- .** Repeat the previous **!** shell command, using the current entry to replace or
- d** Mark the current entry for deletion. Deletion of a directory is recursive.
- e** If the current entry is a file, edit it with the editor *e*, or an editor named by the environment variable **EDITOR**. If it is a directory, invoke *dired* recursively for that directory.
- h**
- ?** Display a help file.
- m** Display the current file with the pager [p\(1\)](#), or another pager named by the environment variable **PAGER**.
- q** Quit this directory level of *dired*. List the files marked for deletion and request confirmation before deleting them.
- t** Type. Display the current file. In two-window mode pause after each screenfull until you type a carriage return. The display may be interrupted.

**FILES**

help file

**SEE ALSO**

[ls\(1\)](#)

**DIAGNOSTICS**

While *dired* is preparing a listing it reports ‘Reading’, and types a dot for each 10 files.

**BUGS**

Long lines and diagnostics can foul up the display.  
Needs a command to search for a given file.  
The off-line print command is broken.

**NAME**

`dis` – display input as refreshed page on VDU output

**SYNOPSIS**

`dis [-ttimeout] [-crefresh]`

**DESCRIPTION**

*Dis* looks up the terminal capability database for the characteristics of the device represented by the environment variable "TERM". Assuming that the standard input consists of repetitive pages, *Dis* then uses cursor addressing to write changed data only on its standard output. Pages are delimited by a *form-feed* or by a timeout, if requested.

The timeout is specified by the flag `-t` followed by a number representing seconds.

If the `-c` flag is specified, the screen is completely redrawn every *refresh* updates.

**EXAMPLES**

```
(while true; do date; echo '\f\c'; sleep 10; done)|dis
```

**FILES**

`/etc/termcap`

**SEE ALSO**

`termLib(3)`, or `curses(3)`.

**DIAGNOSTICS**

If your terminal doesn't have cursor addressing.

**NAME**

`dist` — distribute files to a remote machine

**SYNOPSIS**

```
dist [-nv] [-d hosts] [-[Ff] hostfile] [-D old=new] [-[Xx] command] file ...
```

```
dist -q [-v] [systems ...]
```

```
dist -r [-v] [-D old=new] [-R rootdir] system [job ...]
```

**DESCRIPTION**

*Dist* distributes files to other systems, where they are installed under the same names. *Dist* operates by packing the files with [mkdist\(1\)](#), and queuing the resulting package in a spool directory to be picked up by the remote systems.

*Dist* has several forms of use. In the first (default) form, *dist* packages the named files and queues them for remote systems. By default, the list of remote systems is taken from `/usr/lib/dist/destinations/default`. The **-d** option allows a list of destination systems to be specified as a single argument (containing system names separated by spaces). Similarly, the **-f** option allows a list of systems to be taken from a file. The **-F** option is like **-f**, except it looks in a standard place for the file. Multiple **-d**, **-f**, and **-F** options may be combined. If any destinations are specified via the command line, the default destinations file is not read. The **-D**, **-X**, and **-x** options are passed to *mkdist*. By default, after packaging and queuing the files for distribution, *dist* notifies each remote system that the package is available, and the remote system then dials back and immediately downloads the package. The **-n** option suppresses this notification.

In the second form, when the **-q** option is given, *dist* displays the queue contents on each remote system named in the command line. If no remote systems are named, *dist* displays the local queue.

The third form, with the **-r** option, makes a network call to the named system and attempts to download and install the named jobs. If no jobs are named, it attempts to download all jobs on the remote system. The **-D** and **-R** options are passed to *insdist(1)*-**r** form is rarely used, since the default behavior is for remote systems to automatically dial back when a package is announced.

In all three forms of the command, the **-v** option enables verbose output.

**FILES**

Supporting programs.

List of default destination systems.

Destination files for the  
**-F** option.

Spool directory.

Spool subdirectories.

Log file.

**SEE ALSO**

[mkdist\(1\)](#)

**BUGS**

**-v** should provide more verbose output.

Logging needs improvement.

**NAME**

docgen – generate a document from a script

**SYNOPSIS**

**docgen** [ *option ...* ] [ *ofile* ]

**DESCRIPTION**

*Docgen* guides interactive preparation of standard documents according to canned scripts and places the output in *ofile*. The output typically takes the form of *troff(1)* input. These options invoke standard scripts:

- mcs** (default) Bell Labs cover sheet; output (by default) may be typeset thus:
- ms** documents using the macro package *ms(6)*; output by default) may be typeset thus:
- mm** similarly for the MM macro package of System V

Other options are:

- f file** take script from *file*
- v** (verbose) print document as it is generated
- d** (debug) print information about the script as it is read

The reference tells how to construct scripts.

**FILES**

scripts

**SEE ALSO**

*troff(1)*, *ms(6)*, *mcs(6)*

**BUGS**

Not all document types are implemented.

**NAME**

docsubmit – send document to library

**SYNOPSIS**

**docsubmit** [ **-c** *cover-file* ] [ **-C** ] [ **-r** *ref-file* ] [ **-t** ] [ **-f** *copyf* ] [ **-d** ] *file* ...

**DESCRIPTION**

*Docsubmit* sends the full text of a TM, including cover sheet, to the Bell Laboratories library network, for inclusion in the libraries' Linus database. The *files* are those that would be mentioned in a *troff* command to print the paper; files included by *.so* macros or by preprocessors such as *pic(1)* should not be mentioned. The options are

**-c** *coverfile*

Cover sheet is in a file by itself.

**-C** Cover sheet is in the document. One of **-c** or **-C** must be specified.

**-r** *ref-file*

Specify a separate reference file for *refer(1)* or *prefer(1)*.

**-t** The single *file* is *tex(1)* source; only the base name, without *.tex*, should be specified.

**-f** *copyf*

Include a copy of *copyf* in the cpio file. This flag should only be necessary for things like awk scripts executed inside the paper with *.sy* commands.

**-d** Don't consider include files that can't be found as fatal errors.

Electronic submission is not a substitute for the official paper submission. For more information contact your local library or call (201)582-4840.

**EXAMPLES**

```
docgen cover-file
eqn cover-file file.1 file.2 | troff | lp
docsubmit -c cover-file file.1 file.2
```

**FILES****SEE ALSO**

*troff(1)*, *docgen(1)*

**BUGS**

Only documents that may be viewed by any AT&T employee can have their full text made available under Linus.

Documents with a complicated construction process, such as a shell script or makefile, cannot be handled directly.



**NAME**

doctype – guess command line for formatting a document

**SYNOPSIS**

**doctype** [ *option ...* ] [ *file* ]

**DESCRIPTION**

*Doctype* guesses and prints on the standard output the command line for printing a document that uses *troff*(1), related preprocessors like *eqn*(1), and the *ms*(6) and *mm* macro packages.

Option **-n** invokes *nroff* instead of *troff*. Other options are passed to *troff*.

**EXAMPLES**

```
eval `doctype chapter.?` | aperiod
Typeset files named chapter.0, chapter.1, ...
```

**SEE ALSO**

*troff*(1), *eqn*(1), *tbl*(1), *refer*(1), *prefer*(1), *pic*(1), *ideal*(1), *grap*(1), *ped*(9) *mcs*(6), *ms*(6), *man*(6)

**BUGS**

It's pretty dumb about guessing the proper macro package.

**NAME**

double – double word finder

**SYNOPSIS**

**double** [ **-flags** ] [ **-ver** ] [file ...]

**DESCRIPTION**

*Double* searches text for consecutive occurrences of words. It skips text contained in tables formatted with *tbl(1)* and ignores consecutive occurrences of any single character except *a*. When *double* finds two words in a row, it prints them with the line number of the first one.

*Double* is one of the programs run under the *proofr(1)* and *wwb(1)* commands.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**SEE ALSO**

*proofr(1)*, *wwb(1)*, *tbl(1)*.

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

*du*, *df* – disk usage

**SYNOPSIS**

**du** [ **-s** ] [ **-a** ] [ *file ...* ]

**df** [ **-i** ] [ *special ...* ] [ *file ...* ]

**DESCRIPTION**

*Du* gives the number of Kbytes allocated to data blocks of named *files* and, recursively, of files in named directories. If *name* is missing, *.* is used. The count for a directory includes the counts of the contained files and directories. The options are

**-s**      Print only the grand total.

**-a**      Print a count for every file in a directory. Normally counts are printed only for contained directories.

A file which has two (hard) links to it is counted only once. Symbolic links are neither counted nor followed.

*Df* prints the amount of free space on the file system contained in *special*, or on the file system in which the specified *file* is contained. If no file system is specified, the free space on all of the currently mounted file systems is printed.

The reported numbers are in Kbytes, independently of the blocksize actually used on the file system. The option is

**-i**      Report also on free and used inodes.

**EXAMPLES**

*df* .

How much space is there where I'm working?

**FILES**

only for the root device

list of currently mounted file systems

**SEE ALSO**

[quot\(8\)](#), [fstab\(5\)](#), [icheck\(8\)](#)

**BUGS**

In the absence of option **-a** non-directories given as arguments to *du* are not listed.

If there are too many distinct linked files, *du* counts the excess files multiply.

Unwritten holes in files count as if real data were present, and indirect blocks are not counted.

**NAME**

echo, printf – print arguments

**SYNOPSIS**

**echo** [ **-n** ] [ **-e** ] [ *arg* ... ]

**printf** [ *format* [ *arg*... ] ]

**DESCRIPTION**

*Echo* writes its arguments separated by blanks and terminated by a newline on the standard output. Option **-n** suppresses the newline.

Option **-e** enables the interpretation of C-style escape codes, *\ddd*, where *d* is an octal digit, plus the special code which terminates the output.

*Echo* is useful for producing diagnostics in shell programs and for writing constant data on pipes.

*Printf* behaves like the library function of the same name; each *arg* is printed on the standard output according to the corresponding %-introduced specification in the *format* string. The standard C escape sequences **\n**, **\r**, **\t**, **\b**, and *\digits* are recognized in *format*. The *arg* will be treated as a string if the corresponding format is **s**; otherwise it is evaluated as a C constant, with the following extensions:

A leading plus or minus is allowed.

If the leading character is a single or double quote, the value is the ASCII code of the next character.

Otherwise, if the leading character is not a digit, the value is its ASCII code.

The format string is reused as often as necessary to satisfy the *arg*'s. Any extra format specifications are evaluated with zero or the null string.

**EXAMPLES**

```
echo "can't open file" $1 1>&2
```

Send a message to the standard error file.

**SEE ALSO**

[printf\(3\)](#)

**BUGS**

*Printf* has no diagnostics for illegal syntax.

**NAME**

ed, e – text editor

**SYNOPSIS**

**ed** [ - ] [ **-o** ] [ *file* ]

**DESCRIPTION**

*Ed* is the standard text editor; *e* is another name for it.

If a *file* argument is given, *ed* simulates an command (see below) on that file: it is read into *ed*'s buffer so that it can be edited. The options are

- Suppress the printing of character counts by and commands and of the confirming by commands.
- o** (for output piping) Place on the standard error file all output except writing by commands. If no *file* is given, make the remembered file; see the command below.

*Ed* operates on a 'buffer', a copy of the file it is editing; changes made in the buffer have no effect on the file until a (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*.

Commands to *ed* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period alone at the beginning of a line.

*Ed* supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus and sometimes
2. A . matches any character.
3. A followed by any character except a digit, or matches that character.
4. A nonempty string *s* bracketed [*s*] (or [^*s*]) matches any character in (or not in) *s*. In *s*, has no special meaning, and may only appear as the first letter. A substring *a-b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by matches a sequence of 0 or more matches of the regular expression.
6. A regular expression, *x*, of form 1-8, bracketed \(*x*\) matches what *x* matches.
7. A followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*th matched.
8. A regular expression of form 1-8, *x*, followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.
9. A regular expression of form 1-8, or a null string, preceded by (and/or followed by is constrained to matches that begin at the left (and/or end at the right) end of a line.
10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.
11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often and to itself).

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current

line is discussed under the description of the command. Addresses are constructed as follows.

1. The character customarily called ‘dot’, addresses the current line.
2. The character addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. ‘*x*’ addresses the line marked with the name *x*, which must be a lower-case letter. Lines are marked with the command described below.
5. A regular expression enclosed in slashes addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.
6. A regular expression enclosed in queries addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.
7. An address followed by a plus sign or a minus sign followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.
8. An address followed by (or followed by a regular expression enclosed in slashes specifies the first matching line following (or preceding) that address. The search wraps around if necessary. The may be omitted, so addresses the *first* line in the buffer with an Enclosing the regular expression in reverses the search direction.
9. If an address begins with or the addition or subtraction is taken with respect to the current line; e.g. is understood to mean
10. If an address ends with or then 1 is added (resp. subtracted). As a consequence of this rule and rule 9, the address refers to the line before the current line. Moreover, trailing and characters have cumulative effect, so refers to the current line less 2.
11. To maintain compatibility with earlier versions of the editor, the character in addresses is equivalent to

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma They may also be separated by a semicolon In this case the current line is set to the previous address before the next address is interpreted. If no address precedes a comma or semicolon, line 1 is assumed; if no address follows, the last line of the buffer is assumed. The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default. ‘Dot’ means the current line.

(.) **a**

<text>

- . Read the given text and append it after the addressed line. Dot is left on the last line input, if there were any, otherwise at the addressed line. Address is legal for this command; text is placed at the beginning of the buffer.

(,..) **b**[+][*pagesize*][**pln**]

Browse. Print a ‘page’, normally 20 lines. The optional (default) or specifies whether the next or previous page is to be printed. The optional *pagesize* is the number of lines in a page. The optional or causes printing in the specified format, initially Pagesize and format are remembered between commands. Dot is left at the last line displayed.

(,..) **c**

<text>

.

Change. Delete the addressed lines, then accept input text to replace these lines. Dot is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(,..) **d** Delete the addressed lines from the buffer. Dot is set to the line following the last line deleted, or to the last line of the buffer if the deleted lines had no successor.

**e** *filename*

Edit. Delete the entire contents of the buffer; then read the named file into the buffer. Dot is set to the last line of the buffer. The number of characters read is typed. The file name is remembered for possible use in later or commands. If *filename* is missing, the remembered name is used.

**E** *filename*

Unconditional see 'DIAGNOSTICS' below.

**f** *filename*

Print the currently remembered file name. If *filename* is given, the currently remembered file name is first changed to *filename*.

(1,\$) **g**/*regular expression/command list*

(1,\$) **g**/*regular expression*/*l*

(1,\$) **g**/*regular expression*

Global. First mark every line which matches the given *regular expression*. Then for every such line, execute the *command list* with dot initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must end with The ':' terminating input mode for an command may be omitted if it would be on the last line of the command list. The commands and are not permitted in the command list. Any character other than space or newline may be used instead of to delimit the regular expression. The second and third forms mean **g**/*regular expression*/**p**.

(.) **i**

<text>

.

Insert the given text before the addressed line. Dot is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

(,..+1) **j**

Join the addressed lines into a single line; intermediate newlines are deleted. Dot is left at the resulting line.

(.) **k***x* Mark the addressed line with name *x*, which must be a lower-case letter. The address form '*x*' then addresses this line.

(,..) **l** List. Print the addressed lines in an unambiguous way: a tab is printed as a backspace as backslashes as and non-printing characters are printed as a backslash followed by three octal digits. Long lines are folded, with the second and subsequent sub-lines indented one tab stop. If the last character in the line is a blank, it is followed by An may be appended, like to any non-I/O command.

(,..) **ma**

Move. Reposition the addressed lines after the line addressed by *a*. Dot is left at the last moved line.

(,..) **n** Number. Perform prefixing each line with its line number and a tab. An may be appended, like to any non-I/O command.

(,..) **p** Print the addressed lines. Dot is left at the last line printed. A appended to any non-I/O command causes the then current line to be printed after the command is executed.

(,..) **P** This command is a synonym for

**q** Quit the editor. No automatic write of a file is done.

**Q** Quit unconditionally; see ‘DIAGNOSTICS’ below.

**( $\$$ ) r *filename***

Read in the given file after the addressed line. If no *filename* is given, the remembered file name is used. The file name is remembered if there were no remembered file name already. If the read is successful, the number of characters read is typed. Dot is left at the last line read in from the file.

**(... ) sn/regular expression/replacement/**

**(... ) sn/regular expression/replacement/g**

**(... ) sn/regular expression/replacement**

Substitute. Search each addressed line for an occurrence of the specified regular expression. On each line in which *n* matches are found (*n* defaults to 1 if missing), the *n*th matched string is replaced by the replacement specified. If the global replacement indicator appears after the command, all subsequent matches on the line are also replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or newline may be used instead of to delimit the regular expression and the replacement. Dot is left at the last line substituted. The third form means *sn/regular expression/replacement/p*. The second may be omitted if the replacement is empty.

An ampersand appearing in the replacement is replaced by the string matching the regular expression. The characters  $\backslash n$ , where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between and. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of starting from the left.

A literal or newline may be included in a replacement by prefixing it with

**(... ) t a**

Transfer. Copy the addressed lines after the line addressed by *a*. Dot is left at the last line of the copy.

**(... ) u** Undo. Restore the preceding contents of the current line, which must be the last line in which a substitution was made.

**(1,  $\$$ ) v/regular expression/command list**

**(1,  $\$$ ) v/regular expression/**

**(1,  $\$$ ) v/regular expression**

This command is the same as the global command except that the command list is executed with dot initially set to every line *except* those matching the regular expression.

**(1,  $\$$ ) w *filename***

Write the addressed lines onto the given file. If the file does not exist, it is created with mode 666 (readable and writable by everyone). If no *filename* is given, the remembered file name, if any, is used. The file name is remembered if there were no remembered file name already. Dot is unchanged. If the write is successful, the number of characters written is printed.

**(1,  $\$$ ) W *filename***

Perform but append to, instead of overwriting, any existing file contents.

**( $\$$ ) =** Print the line number of the addressed line. Dot is unchanged.

**!shell command**

Send the remainder of the line after the to *sh(1)* to be interpreted as a command. Dot is unchanged.

**(.+1) <newline>**

An address without a command is taken as a command. A terminal may be omitted from the address. A blank line alone is equivalent to it is useful for stepping through text.

If an interrupt signal (ASCII DEL) is sent, *ed* prints a and returns to its command level.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.



**FILES**

work is saved here if terminal hangs up

**SEE ALSO**

[sam\(9\)](#) [sed\(1\)](#), [vi\(1\)](#)

**DIAGNOSTICS**

?*name* for inaccessible file; for temporary file overflow; for errors in commands or other overflows.

To protect against throwing away valuable work, a or command is considered to be in error if the buffer has been modified since the last or command.

**NAME**

efl – extended Fortran language preprocessor

**SYNOPSIS**

**efl** [ *option ...* ] [ *filename ...* ]

**DESCRIPTION**

*Efl* compiles a program written in the EFL language into clean Fortran. *Efl* provides the same control flow constructs as does Ratfor (1), which are essentially identical to those in C:

statement grouping with braces;

decision-making with if, if-else, and switch-case; while, for, Fortran do, repeat, and repeat...until loops; multi-level break and next. In addition, EFL has C-like data structures, and more uniform and convenient input/output syntax, generic functions. EFL also provides some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation statement label names (not just numbers),

comments:

# this is a comment

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return (expression)

returns expression to caller from function

define:

define name replacement

include:

include filename

The Efl command option **-w** suppresses warning messages. The option **-C** causes comments to be copied through to the Fortran output (default); **-#** prevents comments from being copied through. If a command argument contains an embedded equal sign, that argument is treated as if it had appeared in an **option** statement at the beginning of the program. *Efl* is best used with [f77\(1\)](#).

**SEE ALSO**

[f77\(1\)](#), [ratfor\(1\)](#).

S. I. Feldman, *The Programming Language EFL*, Bell Labs Computing Science Technical Report #78.

**NAME**

=, ==, =p, ==p – redo previous shell command

**SYNOPSIS**

```
= [ pattern ] [ substitution ... ]
== [ pattern ] [ substitution ... ]
=p [ pattern ] [ substitution ... ]
==p [ pattern ] [ substitution ... ]
```

**DESCRIPTION**

The = command provides a simple history mechanism for the shell, *sh(1)*. The environment variable **HISTORY**, if set, names a file to which the shell appends the text of each command before execution. = searches the history file for the most recent command that matches the *pattern*, performs the *substitutions*, and executes it. The *pattern* must agree with an initial substring of the original command except for variations in spacing. If no *pattern* is specified, the most recent command is selected. If no substitution is specified, the command is executed without modification.

Substitutions have the form

*old=new*

specifying that the string *old* in the command is to be replaced by *new*. Substitutions are made in order and operate on the first match.

The == command is identical to =, but allows the substituted command to be edited before running. The command is printed, and a modification request is read from the terminal. Generally each character in the request specifies how to modify the character immediately above it:

#	Delete the character.
%	Replace the character with a space.
^	Insert the rest of the request line before the character.
\$	Replace the characters in the command from this position on with the rest of the request line.
space or tab	Leave the character(s) unchanged.
=	Must be the first and only edit character. Back up to the next most recent match in the history file and try again.
any other	This character replaces the one above it.

If the request line is longer than the command, the overhang is appended to the command.

=p and ==p behave like = and ==, except that they print the command on their standard output instead of executing it.

**NAME**

eqn, neqn, checkeq – typeset mathematics

**SYNOPSIS**

**eqn** [ *option ...* ] [ *file ...* ]

**neqn** [ *option ...* ] [ *file ...* ]

**checkeq** [ *file ...* ]

**DESCRIPTION**

*Eqn* is a *troff(1)* preprocessor for typesetting mathematics on a phototypesetter, *neqn* on terminals. Usage is almost always

```
eqn file ... | troff
```

```
neqn file ... | nroff
```

If no files are specified, these programs read from the standard input. *Eqn* prepares output for the typesetter named in the **-Tdest** option (Mergenthaler Linotron 202 default, see *troff(1)*). When run with other preprocessor filters, *eqn* usually comes last.

A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as ‘delimiters’; text between delimiters is also *eqn* input. Delimiters may be set to characters *x* and *y* with the option **-dxy** or (more commonly) with **delim xy** between **.EQ** and **.EN**. Left and right delimiters may be identical. (They are customarily taken to be *).* Delimiters are turned off by All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.

*Checkeq* reports missing or unbalanced delimiters and **.EQ/.EN** pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like could appear, a complicated construction enclosed in braces may be used instead. Tilde represents a full space in the output, circumflex half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus makes  $x_i$ , produces  $a_i^2$ , and gives  $e^{x^2+y^2}$ .

Fractions are made with **over**: yields  $\frac{a}{b}$ .

**sqrt** makes square roots: results in  $\frac{1}{\sqrt{ax^2 + bx + c}}$ .

The keywords **from** and **to** introduce lower and upper limits on arbitrary things:  $\lim_{n \rightarrow \infty} \sum_0^n x_i$  is made with

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: produces  $\left[ x^2 + \frac{y^2}{\alpha} \right] = 1$ . The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and **''** for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: produces  $\begin{matrix} a \\ b \\ c \end{matrix}$ . There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: produces  $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$ . In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **prime**, **dot**, **dotdot**, **hat**, **tilde**, **bar**, **under**, **vec**, **dyad**, and **under**: is  $x'_0 = \overline{f(t)} + \underline{g(t)}$ , and is  $\vec{x} = \hat{y}$ .

Sizes and font can be changed with prefix operators **size n**, **size ±n**, **fat**, **roman**, **italic**, **bold**, or **font n**. Size and fonts can be changed globally in a document by **gsiz** *n* and **gfont** *n*, or by the command-line arguments **-sn** and **-fn**.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **-pn**.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: *thing replacement* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The may be any character that does not occur in

Keywords like ( $\Sigma$ ) ( $\int$ ) ( $\infty$ ) and shorthands like ( $\geq$ ) ( $->$ ), and ( $\neq$ ) are recognized. Greek letters are spelled out in the desired case, as in or Mathematical words like are made Roman automatically. *Troff*(1) four-character escapes like ( $\text{\textcircled{R}}$ ) can be used anywhere. Strings enclosed in double quotes " " are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

### SEE ALSO

[troff\(1\)](#), [tbl\(1\)](#), [ms\(6\)](#), [eqnchar\(6\)](#), [doctype\(1\)](#)

B. W. Kernighan and L. L. Cherry, ‘Typesetting Mathematics—User’s Guide’, this manual, Volume 2

J. F. Ossanna and B. W. Kernighan, ‘NROFF/TROFF User’s Manual’, *ibid*.

### BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in

**NAME**

esterel – Esterel compiler

**SYNTAX**

**esterel** [options] [file] ...

**DESCRIPTION**

The *esterel* command invokes the various utilities constituting the Esterel language development tools:

- strlic* The Esterel front-end: receives files containing Esterel source (**.strl** suffix) producing intermediate code (**.ic**);
- iclc* The Esterel binder, performing the expansion of the **copymodule** statements; it receives several **.ic** (or **.lc**) files and builds an unique linked code file (**.lc**);
- lcoc* The Esterel compiler, which produces from an unique **.lc** file, Esterel automata in portable format (**.oc**);
- ocl* A generic name for Esterel code generators, translating portable automata (**.oc**) into a program written in one of the supported target languages (see the **-L** option below).

If no files are given to the *esterel* command, the standard input is used. Any suffix in the list **.strl**, **.ic**, **.lc**, or **.oc** is recognized in the files names: the *esterel* command will arrange for only the appropriate utilities to be called.

**OPTIONS**

The following option is for the *esterel* command itself:

- n** Tell what is to be done, but don't do it.

The option

- version** display the version number of the *esterel* command, as well as the ones of the various utilities including all known code generators.

The following options are passed to all four utilities:

- v** Verbose mode: the *esterel* command and the various utilities tell what they are doing;
- w** Suppress all warning messages;
- W** Display all warning messages (the default is to display only "selected" warnings);
- stat** Display various time statistics;
- memstat**  
Display statistics on dynamically allocated memory.

The three following options enable to stop the compilation process at some intermediate stage:

- ic** Just use *strlic* to convert **.strl** files into **.ic** files (with the same base name), ignoring all other files;
- lc** Stop after running the binder (*iclc*);
- oc** Stop after running the compiler (*lcoc*).

For the **-lc** and **-oc** options, one can specify the output file name(s) with the **-B** and **-D** options.

- B name name** denotes the output files default base name. The appropriate suffix is added automatically (and possibly a working directory name --see the following option). If this option is omitted and if the *esterel* command is invoked with only one file name, *name* defaults to the base name of this unique file with the appropriate suffix; otherwise it defaults to the base name **esterel**, still followed by the appropriate suffix.

- D directory**

Specify a directory where the files produced by the command will be placed. The default is the current directory. The **-B** and **-D** options and the corresponding default rules apply to the files produced by the **-K** (except **-Kic**) and **-L** options below.

The *esterel* command removes all the intermediate files it has created, unless one of the following options is given:

- Kic** Keep all the **.ic** files (their names being the original ones, with the suffix **.strl** replaced by **.ic**);
- Klc** Keep the (unique) **.lc** file;
- Koc** Keep the (unique) **.oc** file;
- K** Keep all the intermediate files.

The binder *iclc* recognizes some specific options:

- Rs** Trace signal captures and renaming;
- Rc** Trace constant captures and renaming;
- R** Trace both signal and constant captures and renaming.

The compiler *lcoc* recognizes also some specific options:

- size** Display the final size (states and bytes) of the generated automata;
- show** Display dynamically the number of states generated so far.

The code generators (*ocl*) recognize an unique option:

- L[*language*][:*specific\_options*]**  
Set the target language: at this time only **c**, **lelisp**, **tex**, **plm**, **auto**, and **debug** are known; it is likely that other languages be added. The code generators have a name of the form **ocl*language***. If the **-L** option or the language are omitted, the default is **c**. The string *specific\_options* allows to transmit language dependent options to a given code generator (see **ocl(1)**).

There can be as many **-L** options as needed.

Finally, there is a particular option to do as much as specified (by the stop options) but producing nothing.

- s** Perform all the compilation process, as specified by the other options, but produce nothing.

## EXAMPLES

The simple command

```
esterel foo.strl
```

performs a full Esterel compilation, leaving the produced automaton, in C language form, in the file **foo.c**.

To produce debug format while keeping the generated automaton in portable format, try

```
esterel -Koc -Ldebug foo.strl
```

A little more complex, the following command

```
esterel -Kic -Koc -Bautom -Llisp f1.strl f2.ic f3.ic
```

will pass **f1.strl** through *strlic* and keep the **f1.ic** file; then it will pass **f1.ic**, **f2.ic**, and **f3.ic** through *iclc* and *lcoc*, producing the file **autom.oc** (the intermediate files are discarded); finally, this last file will be converted into the LeLisp file **autom.ll** by *oclelisp*.

The command

```
esterel -K -Bfoo -D/a/b f1.strl f2.strl f3.ic f4.oc
```

produces the following files: **/a/b/f1.ic**, **/a/b/f2.ic**, **/a/b/foo.ic**, **/a/b/foo.oc**, and **/a/b/foo.c**.

To illustrate the **-s** option, note that

```
esterel -s foo.strl
```

performs a full compilation upto C code generation, but the C file is not produced; similarly,

```
esterel -ic -s foo.strl
```

will only execute the front-end *strlic* without producing any *.ic* file.

Finally,

```
esterel -Lc -Lauto:"-signal EV1,EV2" foo.strl
```

performs a full compilation of the Esterel source file **foo.strl** to auto format (**foo.auto**), passing the arguments **-signal EV1,EV2** untouched to the corresponding code generator (here *ocauto*).

## DIAGNOSTICS

The command returns with exit code 0 if (and only if) no error was detected by the various utilities.

Various error or warning messages indicate incompatible or redundant options, or error conditions related to file handling.

## BUGS

The command checks whether it generates a file which is already present in its argument list, and if so, stops with an error, to avoid clobbering the file.

The corresponding test is based on the name of files as given by the user and is rather rustic. For instance, the following erroneous condition (or any similar one) is not detected

```
esterel -K -Bfoo -D.. foo.strl ../foo.lc
```

and will certainly result in loosing the original content of *../foo.lc* (use the **-n** option to see what will occur).

## FILES

In the following, *\$lib* designates the default library directory for Esterel utilities (usually */usr/local/lib/esterel*). This default path can be modified by the installer of the Esterel system, or by any user setting the environment variable *ESTERELLIB*.

<i>\$lib/strlic</i>	Esterel front-end
<i>\$lib/iclc</i>	Esterel binder
<i>\$lib/lcoc</i>	Esterel compiler (automaton generator)
<i>\$lib/oc*</i>	Esterel code generators
<i>*.strl</i>	Esterel source files
<i>*.ic</i>	Intermediate code files
<i>*.lc</i>	Linked intermediate code file
<i>*.oc</i>	Portable automata file
<i>esterel.*</i>	Default names for keeping intermediate files

## SEE ALSO

*strlic(1)*, *iclc(1)*, *lcoc(1)*, *ocl(1)*  
*Esterel V3 Reference Manual*  
*Esterel V3 System Manuals*

## IDENTIFICATION

Author: Jean-Paul Rigault, Ecole des Mines de Paris, CMA  
\$Revision: 1.1 \$  
\$Date: 88/04/07 13:39:34 \$



**NAME**

*expr* – integer and string-match expression evaluator for shell scripts

**SYNOPSIS**

*expr arg ...*

**DESCRIPTION**

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below in order of increasing precedence, with equal precedence operators grouped.

*expr1* | *expr2*

Value is the value of *expr1* if that is neither empty nor 0, otherwise the value of *expr2*.

*expr1* & *expr2*

Value is the value of *expr1* if neither *expr1* nor *expr2* is empty or 0, otherwise 0.

*expr1* *relop* *expr2*

*Relop* is one of Value is 1 if the indicated comparison is true, 0 if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

*expr1* + *expr2*

*expr1* - *expr2*

Value is the sum or difference of the (integer) values of *expr1* and *expr2*.

*expr1* \* *expr2*

*expr1* / *expr2*

*expr1* % *expr2*

Value is the product, quotient, or remainder of the (integer) values of *expr1* and *expr2*.

*expr* : *regex*

Match the string value of *expr* with the regular expression *regex*; regular expression syntax is the same as in [ed\(1\)](#), but matches are anchored at the left. On success a subexpression `\(...\)`, if present in *regex*, picks out a return value from the matched string. Otherwise, the matching operator yields the number of characters matched (0 on failure).

( *expr* )

Parentheses for grouping.

*arg* Value is the string *arg*.

**EXAMPLES**

```
a=`expr $a + 1`
```

Add 1 to shell variable *a*.

```
expr $a : '.*\(.*\)' |' $a
```

Same as

**SEE ALSO**

[sh\(1\)](#), [test\(1\)](#)

**DIAGNOSTICS**

*Expr* returns exit code 0 if the expression is neither null nor 0, 1 if the expression is null or 0, 2 for invalid expressions.

**NAME**

*f2c* – Convert Fortran 77 to C or C++

**SYNOPSIS**

**f2c** [ *option ...* ] *file ...*

**DESCRIPTION**

*F2c* converts Fortran 77 source code in *files* with names ending in *.f* or *.F* to C (or C++) source files in the current directory, with *.c* substituted for the final *.f* or *.F*. If no Fortran files are named, *f2c* reads Fortran from standard input and writes C on standard output. *File* names that end with *.p* or *.P* are taken to be prototype files, as produced by option *-P*, and are read first.

The following options have the same meaning as in *f77*(1).

- C** Compile code to check that subscripts are within declared array bounds.
- I2** Render INTEGER and LOGICAL as short, INTEGER\*4 as long int. Assume the default *libF77* and *libI77*: allow only INTEGER\*4 (and no LOGICAL) variables in INQUIRES. Option *-I4* confirms the default rendering of INTEGER as long int.
- onetrip** Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
- U** Honor the case of variable and external names. Fortran keywords must be in *lower* case.
- u** Make the default type of a variable ‘undefined’ rather than using the default Fortran rules.
- w** Suppress all warning messages. If the option is *-w66*, only Fortran 66 compatibility warnings are suppressed.

The following options are peculiar to *f2c*.

- A** Produce ANSI C. Default is old-style C.
- a** Make local variables automatic rather than static unless they appear in a DATA, EQUIVALENCE, NAMELIST, or SAVE statement.
- C++** Output C++ code.
- c** Include original Fortran source as comments.
- E** Declare uninitialized COMMON to be **Extern** (overridably defined in *f2c.h* as **extern**).
- ec** Place uninitialized COMMON blocks in separate files: **COMMON /ABC/** appears in file **abc\_com.c**. Option *-e1c* bundles the separate files into the output file, with comments that give an unbundling *sed*(1) script.
- ext** Complain about *f77*(1) extensions.
- g** Include original Fortran line numbers as comments.
- h** Try to align character strings on word (or, if the option is *-hd*, on double-word) boundaries.
- i2** Similar to **-I2**, but assume a modified *libF77* and *libI77* (compiled with **-Df2c\_i2**), so INTEGER and LOGICAL variables may be assigned by INQUIRE and array lengths are stored in short ints.
- kr** Use temporary values to enforce Fortran expression evaluation where K&R (first edition) parenthesization rules allow rearrangement. If the option is *-krd*, use double precision temporaries even for single-precision operands.
- P** Write a *file.P* of ANSI (or C++) prototypes for procedures defined in each input *file.f* or *file.F*. When reading Fortran from standard input, write prototypes at the beginning of standard output. Implies **-A** unless option *-C++* is present. Option **-Ps** implies **-P**, and gives exit status 4 if rerunning *f2c* may change prototypes or declarations.
- p** Supply preprocessor definitions to make common-block members look like local variables.
- R** Do not promote REAL functions and operations to DOUBLE PRECISION. Option *-!R* confirms the default, which imitates *f77*.

- r** Cast values of REAL functions (including intrinsics) to REAL.
- r8** Promote REAL to DOUBLE PRECISION, COMPLEX to DOUBLE COMPLEX.
- Tdir** Put temporary files in directory *dir*.
- w8** Suppress warnings when COMMON or EQUIVALENCE forces odd-word alignment of doubles.
- Wn** Assume *n* characters/word (default 4) when initializing numeric variables with character data.
- z** Do not implicitly recognize DOUBLE COMPLEX.
- lbs** Do not recognize backslash escapes (`\", \', \0, \\, \b, \f, \n, \r, \t, \v`) in character strings.
- lc** Inhibit C output, but produce **-P** output.
- !I** Reject **include** statements.
- lit** Don't infer types of untyped EXTERNAL procedures from use as parameters to previously defined or prototyped procedures.
- !P** Do not attempt to infer ANSI or C++ prototypes from usage.

The resulting C invokes the support routines of *f77*; object code should be loaded by *f77* or with *ld(1)* or *cc(1)* options **-IF77 -II77 -lm**. Calling conventions are those of *f77*: see the reference below.

## FILES

*file.[fF]*  
input file

**\*.c** output file

`/usr/include/f2c.h`  
header file

`/usr/lib/libF77.a`  
intrinsic function library

`/usr/lib/libI77.a`  
Fortran I/O library

`/lib/libc.a`  
C library, see section 3

## SEE ALSO

S. I. Feldman and P. J. Weinberger, 'A Portable Fortran 77 Compiler', *UNIX Time Sharing System Programmer's Manual*, Tenth Edition, Volume 2, AT&T Bell Laboratories, 1990.

## DIAGNOSTICS

The diagnostics produced by *f2c* are intended to be self-explanatory.

## BUGS

Floating-point constant expressions are simplified in the floating-point arithmetic of the machine running *f2c*, so they are typically accurate to at most 16 or 17 decimal places.

Untypable EXTERNAL functions are declared **int**.

**NAME**

*f77* – Fortran 77 compiler

**SYNOPSIS**

*f77* [ *option ...* ] *file ...*

**DESCRIPTION**

*F77* is a Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with *.f* are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with *.o* substituted for *.f*.

Arguments whose names end with *.r* or *.e* are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by *f77*.

In the same way, arguments whose names end with *.c* or *.s* are taken to be C or assembly source programs and are compiled or assembled, producing a *.o* file.

The following options have the same meaning as in *cc(1)*. See *ld(1)* for load-time options.

- c** Suppress loading and produce *.o* files for each source file.
- g** Have the compiler produce additional symbol table information for *sdb(A)* or *pi(9)*
- w** Suppress all warning messages. If the option is *-w66*, only Fortran 66 compatibility warnings are suppressed.
- p** Prepare object files for profiling, see *prof(1)*.
- O** Invoke an object-code optimizer.
- S** Compile the named programs, and leave the assembler-language output on corresponding files suffixed *.s*. (No *.o* is created.)
- o *output***  
Name the final output file *output* instead of a *.out*.

The following options are peculiar to *f77*.

**-onetrip**

Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

- u** Make the default type of a variable ‘undefined’ rather than using the default Fortran rules.
- C** Compile code to check that subscripts are within declared array bounds.
- I2** Render INTEGER and LOGICAL as short, INTEGER\*4 as long. Allow only INTEGER\*4 (and no LOGICAL) variables in INQUIRES.
- U** Honor the case of variable and external names. Fortran keywords must be in lower case.
- F** Apply EFL and Ratfor preprocessors to relevant files, put the results in the files with the suffix changed to *.f*, but do not compile.
- m** Apply the M4 preprocessor to each *.r* or *.e* file before transforming it with the Ratfor or EFL preprocessor.
- Ex** Use the string *x* as an EFL option in processing *.e* files.
- Rx** Use the string *x* as a Ratfor option in processing *.r* files.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name a *.out*.

**FILES**

*file.[fresc]*  
input file

**\*.o** object file  
**a.out** loaded output  
**./fort\*** temporary  
**/usr/lib/f77pass1** compiler  
**/lib/f1** pass 2  
**/lib/c2** optional optimizer  
**/usr/lib/libF77.a** intrinsic function library  
**/usr/lib/libI77.a** Fortran I/O library  
**/lib/libc.a** C library, see section 3

**SEE ALSO**

[prof\(1\)](#), [cc\(1\)](#), [ld\(1\)](#), [efl\(A\)](#), [ratfor\(A\)](#)

S. I. Feldman and P. J. Weinberger, 'A Portable Fortran 77 Compiler', this manual, Volume 2

**DIAGNOSTICS**

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

**NAME**

*factor*, *qfactor*, *primes* – factor a number, generate large primes

**SYNOPSIS**

**factor** [ *number* ]

**qfactor**

**primes** [ *start* [ *finish* ] ]

**DESCRIPTION**

*Factor* prints *number* and its prime factors, each repeated the proper number of times. The number must be positive and less than  $2^{56}$  (about  $7.2 \times 10^{16}$ ).

If no *number* is given, *factor* reads a stream of numbers from the standard input and factors them. It exits on any input not a positive integer. Maximum running time is proportional to  $\sqrt{n}$ .

*Qfactor* reads one number from the standard input and factors it. It will factor numbers up to about 40 digits. For large numbers it is much faster than *factor*.

*Primes* prints the prime numbers ranging from *start* to *finish*, where *start* and *finish* are positive numbers less than  $2^{56}$ . If *finish* is missing, *primes* prints without end; if *start* is missing, it reads the starting number from the standard input.

**NAME**

file – determine file type

**SYNOPSIS**

**file** *file* ...

**file -f** *names*

**DESCRIPTION**

*File* performs a series of tests on a set of files in an attempt to classify their contents by language or purpose. The names may be enumerated or contained in the file specified by the **-f** option.

**BUGS**

It can make mistakes, for example classifying a file of decimal data, .01, .02, etc. as [troff\(1\)](#) input.

**NAME**

find – find files

**SYNOPSIS**

**find** *pathname ... expression*

**DESCRIPTION**

*Find* recursively descends the directory hierarchy for each *pathname*, seeking files that match a boolean *expression*, which consists of one or more arguments. It does not follow symbolic links. In the following descriptions of primary expressions, *n* is a decimal integer; *+n* may be written to specify more than *n* and *-n* to specify less.

**-name** *filename*

True if the *filename* argument matches the current file name. Normal shell filename metacharacters may be used if quoted.

**-perm** *onum*

True if the file permission flags exactly match the octal number *onum* (see [chmod\(1\)](#)). If *onum* is prefixed by a minus sign, more mode bits (017777, see [stat\(2\)](#)) become significant and the modes are compared:  $(mode \& onum) == onum$ .

**-type** *c*

True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **f**, or **L** for block special file, character special file, directory, plain file or symbolic link.

**-links** *n*

True if the file has *n* links.

**-user** *uname*

True if the file belongs to the user *uname* (login name or numeric userid).

**-group** *gname*

True if the file belongs to group *gname* (group name or numeric groupid).

**-size** *n*

True if the file is *n* blocks long (512 bytes per block).

**-inum** *n*

True if the file has inode number *n*.

**-atime** *n*

True if the file has been accessed in *n* days.

**-mtime** *n*

True if the file has been modified in *n* days.

**-ctime** *n*

True if the inode has been changed in *n* days.

**-exec** *command*

True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument { } is replaced by the current pathname.

**-ok** *command*

Like **-exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

**-print**

Always true; causes the current pathname to be printed.

**-newer** *file*

True if the file has been modified more recently than the argument *file*.

**-status** *n*

True if *lstat* (see [stat\(2\)](#)) applied to the file yields error number *n*; see [intro\(2\)](#). Testing **-status** turns off diagnostics that errors normally produce. On ordinary systems a nonzero error number occurs when a file disappears underfoot or a file system is in trouble.

The following operators, listed in order of decreasing precedence, may be used to combine primary expressions.

( *expression* )

Group with parentheses.

! *expression*

Negation. True if and only if *expression* is not true.



*expression expression*

Conjunction. True if both expressions are true.

*expression -o expression*

Disjunction. True if either expression is true.

## EXAMPLES

```
find /\( -name a.out -o -name '*.o' \) -atime +7 -exec rm '{}' \;
```

Remove all files named `a.out` or `*.o` that have not been accessed for a week.

## FILES

`/etc/passwd`

`/etc/group`

## SEE ALSO

[sh\(1\)](#), [test\(1\)](#), [filesystem\(5\)](#)

**NAME**

`fmt` – ultra-simple text formatter

**SYNOPSIS**

`fmt` [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Fmt* copies the given *files* (standard input by default) to its standard output, filling and indenting lines. The options are

**-l** *n*     Output line length is *n*, including indent, (default 70).

**-i** *n*     Indent *n* spaces (default 0).

Empty lines and initial white space in input lines are preserved. Empty lines are inserted between input files.

*Fmt* is idempotent: it leaves already formatted text unchanged.

**BUGS**

Words longer than 256 characters are split.

**NAME**

ftp – internet file transfer program

**SYNOPSIS**

**ftp** [ *option ...* ] [ **host** ]

**pftp** [ *option ...* ] [ **host** ]

**DESCRIPTION**

*Ftp* transfers files to and from a remote network *host* computer via the Internet file transfer protocol. To reach outside Internet sites from v10 machines, prefix the Internet host name with `inet!`; from System V machines, use *pftp*. After attempting to connect to the remote host, if any, *ftp* enters its command interpreter and prompts for instructions. The following commands are recognized:

**!** [ *command* [ *args* ] ]

Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

**\$** *macro-name* [ *args* ]

Execute the macro *macro-name* that was defined with the `macdef` command. Arguments are passed to the macro unglobbed.

**account** [ *passwd* ]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

**append** *local-file* [ *remote-file* ]

Append a local file to a file on the remote machine. If a *remote-file* is not specified, the local file name is used subject to altering *ntrans* or *nmap*. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

**ascii** Set the file transfer *type* to network ASCII. This is the default type.

**bell** Arrange that a bell be sounded after each file transfer command is completed.

**binary**

Set the file transfer *type* to support binary image transfer.

**bye** Terminate the session. An end of file will also terminate the session.

**case** Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

**cd** *remote-directory*

Change the working directory on the remote machine to *remote-directory*.

**cdup** Change the remote machine working directory to the parent of the current remote machine working directory.

**close** Terminate the session. Any defined macros are erased.

**cr** Toggle carriage return stripping during `ascii` type file retrieval. Records are denoted by a carriage return/linefeed sequence during `ascii` type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an `ascii` type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

**delete** *remote-file*

Delete the file *remote-file* on the remote machine.

**debug** [ *debug-level* ]

Toggle debugging or set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string `-->`.

**dir** [ *remote-directory* ] [ *local-file* ]

Place in *local-file* a listing of the contents of *remote-directory*. If *local-file* is - or absent send output to the terminal. If `prompt` is on, *ftp* asks for *local-file* to be confirmed. If no *remote-directory* is specified, the current working directory on the remote machine is used.

**disconnect**

A synonym for **close**.

**form** *format*

Set the file transfer *form* to *format*. The default format is `file`.

**get** *remote-file* [ *local-file* ]

Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to altering by *case*, *ntrans*, and *nmap* settings. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

**glob** Toggle filename expansion for **mdelete**, **mget**, and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in [csh\(1\)](#). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory may be different from expansion of the name of an ordinary file, depending on the foreign operating system and FTP server. It may be previewed by doing '`mls remote-files -`'. Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a [tar\(1\)](#) archive of the subtree (in binary mode).

**hash** Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.

**help** [ *command* ]

Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

**lcd** [ *directory* ]

Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

**ls** [ *remote-directory* ] [ *local-file* ]

List in *local-file* the contents of a directory on the remote machine. If *local-file* is - or absent, the output is sent to the terminal. The form of the list depends on the remote server; most UNIX systems will produce output from the command `ls -l`. (See also **nlist**.) If *remote-directory* is not specified, the current working directory is used.

**macdef** *macro-name*

Define a macro. Subsequent lines are stored under *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets `$` and `\` as special characters. A `$` followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A `$` followed by an `i` signals that macro processor that the executing macro is to be looped. On the first pass `$i` is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A `\` followed by any character is replaced by that character. Use `\` to prevent special treatment of `$`.

**mdelete** [ *remote-files* ]

Delete the *remote-files* on the remote machine.

**mdir** *remote-files local-file*

Like **dir**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

**mget** *remote-files*

Expand *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to *case*, *ntrans*, and *nmap* settings. Files are transferred into the local working directory.

**mkdir** *directory-name*

Make a directory on the remote machine.

**mls** *remote-files local-file*

Like **nlist**, except multiple remote files may be specified, and a *local-file* must be specified. If **prompt** is on, *ftp* asks to confirm the *local-file*.

**mode** [ *mode-name* ]

Set the file transfer mode to *mode-name*. The default mode is **stream**.

**modtime** *file-name*

Show the last modification time of the file on the remote machine.

**mput** *local-files*

Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to *ntrans* and *nmap* settings.

**nlist** [ *remote-directory* ] [ *local-file* ]

Like **ls**, giving only file names.

**nmap** [ *inpattern outpattern* ]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences \$1, \$2, ..., \$9 in *inpattern*. Use \ to prevent special treatment of \$. For example, given *inpattern* \$1.\$2 and the remote file name *mydata.data*, \$1 would have the value *mydata*, and \$2 would have the value *data*. The *outpattern* determines the resulting mapped filename. The sequences \$1, \$2, ..., \$9 are replaced by any value resulting from the *inpattern* template. The sequence \$0 is replaced by the original filename. Additionally, the sequence '*seq1,seq2P*' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command **nmap** \$1.\$2.\$3 [*\$1,\$2*]. [*\$2,file*] would yield the output filename *myfile.data* for input filenames *myfile.data* and *myfile.data.old*, *myfile.file* or the input filename *myfile*, and *myfile.myfile* for the input filename *.myfile*. Spaces may be included in *outpattern*, for example: **nmap** \$1 "|sed 's/ \*\$/' > \$1" .

**ntrans** [ *inchars* [ *outchars* ] ]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

**open** *host* [ *port* ]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see

below).

**prompt**

Toggle file-by-file prompting for **mget**, **mput**, and **mdelete** (on by default).

*proxy ftp-command*

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command `proxy ?` to see other commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection.

*put local-file [ remote-file ]*

Store a local file on the remote machine. If *remote-file* is not specified, the local file name is used after processing according to any *ntrans* or *nmap* settings. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

**pwd** Print the name of the current working directory on the remote machine.

**quit** A synonym for **bye**.

**quote** *arg1 arg2 ...*

The arguments specified are sent, verbatim, to the remote FTP server.

*recv remote-file [ local-file ]*

A synonym for **get**.

*remotehelp [ command-name ]*

Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

*remotestatus [ file-name ]*

With no arguments, show status of remote machine. If *file-name* is specified, show status of *file-name* on the remote machine.

*rename [ from ] [ to ]*

Rename the file *from* on the remote machine, to the file *to*.

**reset** Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

**rmdir** *directory-name*

Delete a directory on the remote machine.

**runique**

Toggle storing of files on the local system with unique filenames. If the target of a **get** or **mget** command already exists locally, a `.1` is appended to the name. If that name, too, matches another existing file, a `.2` is appended and so on until `.99`, when the transfer is aborted. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

*send local-file [ remote-file ]*

A synonym for **put**.

**sendport**

Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which ignore PORT commands but incorrectly indicate they've been accepted.

**size** *file-name*

Return size of *file-name* on the remote machine.

**status** Show the current status of *ftp*.

struct [ *struct-name* ]

Set the file transfer *structure* to *struct-name*. By default `stream` structure is used.

**sunique**

Toggle storing of files on remote machine under unique file names. Default value is off.

**system**

Show the type of operating system running on the remote machine.

**tenex** Set the file transfer type to that needed to talk to TENEX machines.

**trace** Toggle packet tracing.

type [ *type-name* ]

Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [ *password* ] [ *account* ]

Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with `auto-login` disabled, this process is done automatically on initial connection to the FTP server.

**verbose**

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [ *command* ]

A synonym for help.

Command arguments which have embedded spaces may be quoted with quote " **marks**.

**Aborting a file transfer**

The signal processing in the research version of *ftp* has been stripped out. Aborts will generally close the connection.

**File naming conventions**

Files specified as arguments to *ftp* commands are processed according to the following rules.

- 1) If the file name `-` is specified, **stdin** (for reading) or **stdout** (for writing) is used.
- 2) If the first character of the file name is `|`, the remainder of the argument is interpreted as a shell command. *Ftp* reads the standard output of the command, or writes the standard input. If the shell command includes spaces, it must be quoted with double quotes. A useful example of this mechanism is: `|more`.
- 3) Failing the above checks, if `glob` is enabled, local file names are expanded according to the rules used in *csh*(1); c.f. the *glob* command. If the *ftp* command expects a single local file (e.g. **put**), only the first filename generated by the globbing operation is used.
- 4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, subject to altering by **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.
- 5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, subject to altering by **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

### File transfer parameters

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of *ascii*, *image* (binary), *ebcdic*, and *local* (for PDP-10's and PDP-20's mostly). *Ftp* supports the *ascii* and *image* types of file transfer, plus local byte size 8 for **tenex** mode transfers.

*Ftp* supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

### Options

Options may be specified at the command line, or to the command interpreter.

- v** Verbose. Show all responses from the remote server, as well as report on data transfer statistics.
- n** Do not attempt *auto-login* upon initial connection. If *auto-login* is enabled, *ftp* will check the *.netrc* (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.
- i** Do not prompt during multiple file transfers.
- d** Enable debugging.
- g** Disables file name globbing.

### The *.netrc* file

The *.netrc* file contains login and initialization information used by the *auto-login* process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

#### *machine name*

Identify a remote machine name. The *auto-login* process searches *.netrc* for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command argument. Once a match is made, subsequent tokens are processed, until end of file is reached or another **machine** or a **default** token is encountered.

#### **default**

This is the same as **machine name** except that **default** matches any name. There can be only one **default** token, and it must be after all **machine** tokens. This is normally used as:

**default login anonymous password user site**

thereby giving the user automatic anonymous ftp login to machines not specified in *.netrc*.

#### *login name*

Identify a user on the remote machine. If this token is present, the *auto-login* process will initiate a login using the specified name.

#### *password string*

Supply a password. If this token is present, the *auto-login* process will supply the specified string if the remote server requires a password as part of the login process. If this token is present in *.netrc* for any user other than *anonymous*, and *.netrc* is readable by nonowners, *ftp* will abort *auto-login*.

#### *account string*

Supply an additional account password. If this token is present, *auto-login* supplies the *string* when the remote server demands an additional account password; otherwise *auto-login* initiates an *ACCT* command.

#### *macdef name*

Define a macro in the style of **macdef**. If a macro named *init* is defined, it is automatically executed as the last step in *auto-login*.

### SEE ALSO

*ftpd*(8)

### BUGS

Remote servers may not support all features documented here. Interrupts cause *ftp* to exit.



**NAME**

games, demo – some playthings

**SYNOPSIS**

**/usr/jerq/bin/demo** [ *game* ]

**Labyrinth games**

**adventure**  
**zork**  
**rogue**  
**wump**

**Card games**

**fish**  
**canfield**  
**bridge** [ *arg ...* ]  
**mille**

**Board games**

**back**

**Word games**

**hangman** [ **-a** ]  
**word\_clout**  
**ana** [ *n* ]  
**festoon** *length percent*

**System games**

**imp**  
**tso**

**War games**

**mars** [ **-dfhmp** ] [ **-cqsvalue** ] *file ...*  
**ogre** [ *type* ]  
**warp**

**Games of speed**

**atc**  
**snake**  
**worm**

**Educational games**

**quiz** [ **-i file** ] [ **-t** ] [ *question answer* ]  
**arithmetic** [ **+x/** ] [ *range* ]

**Creative games**

**/usr/jerq/bin/twid**  
**banner**

**Sayings**

**fortune** [ *file* ]  
**doctor**  
**say** [ *N* ]

**Coding games**

**bcd** *text*  
**ppt**  
**morse**  
**/usr/bin/number**

**Out-of-layer games**

**/usr/jerq/bin/pen**  
**/usr/jerq/bin/crabs** [ **-i** ] \  
[ **-s gracetime** ] [ **-v speed** ] [ *n* ]

## DESCRIPTION

Game programs exist sporadically on various machines. Manuals for many of them may be obtained by using *man(1)*. For example, to see manual pages for *atc* and *twid*, type **man atc twid**. Unless shown otherwise, games live in which may be put in your shell **PATH** to reach them more conveniently. Some need a cursor-addressed terminal; see the appropriate manual pages and *term(9)*

*Demo* and other games found in need a Teletype 5620 terminal running under *mux(9)* *Demo* comprises many games, which are listed when it is invoked without an argument. Experiment with the mouse to find out they work. Some unobvious interactions are these:

*Swar* is for two players, one using *asdx* on the keyboard, the other 12350 on the keypad.

*Pacman* is controlled by *hjkl* keys as in *vi(1)*.

Here is a list of some *demo* games.

### Watch the time clock

### War games swar

### Games of speed gebam pengo centipede asteroids pacman

### Educational games maxwell

### Out-of-layer games tracks pogo magnet

### Animation EWD road juggle ball fence

### Movies horse dodec explode arno

### Patterns bounce moire fireworks rose disc lunch

**NAME**

gcc – GNU project C Compiler

**SYNOPSIS**

gcc [ option ] ... file ...

**DESCRIPTION**

The *GNU C compiler* uses a command syntax much like the Unix C compiler. The *gcc* program accepts options and file names as operands. Multiple single-letter options may *not* be grouped: ‘**-dr**’ is very different from ‘**-d -r**’. When you invoke *GNU CC* it normally does preprocessing, compilation, assembly and linking. File names which end in ‘.c’ are taken as C source to be preprocessed and compiled; compiler output files plus any input files with names ending in ‘.s’ are assembled; then the resulting object files, plus any other input files, are linked together to produce an executable. Command options allow you to stop this process at an intermediate stage. For example, the ‘**-c**’ option says not to run the linker. Then the output consists of object files output by the assembler. Other command options are passed on to one stage. Some options control the preprocessor and others the compiler itself.

**OPTIONS**

Here are the options to control the overall compilation process, including those that say whether to link, whether to assemble, and so on.

**-o file**

Place linker output in file *file*. This applies regardless to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code. If ‘**-o**’ is not specified, the default is to put an executable file in ‘a.out’, the object file ‘*source.c*’ in ‘*source.o*’, an assembler file in ‘*source.s*’, and preprocessed C on standard output.

**-c** Compile or assemble the source files, but do not link. Produce object files with names made by replacing ‘.c’ or ‘.s’ with ‘.o’ at the end of the input file names. Do nothing at all for object files specified as input.

**-S** Compile into assembler code but do not assemble. The assembler output file name is made by replacing ‘.c’ with ‘.s’ at the end of the input file name. Do nothing at all for assembler source files or object files specified as input.

**-E** Run only the C preprocessor. Preprocess all the C source files specified and output the results to standard output.

**-v** Compiler driver program prints the commands it executes as it runs the preprocessor, compiler proper, assembler and linker. Some of these are directed to print their own version numbers.

**-B prefix**

Compiler driver program tries *prefix* as a prefix for each program it tries to run. These programs are ‘**cpp**’, ‘**cc1**’, ‘**as**’ and ‘**ld**’. For each subprogram to be run, the compiler driver first tries the ‘**-B**’ prefix, if any. If that name is not found, or if ‘**-B**’ was not specified, the driver tries two standard prefixes, which are ‘**/usr/lib/gcc-**’ and ‘**/usr/local/lib/gcc-**’. If neither of those results in a file name that is found, the unmodified program name is searched for using the directories specified in your ‘**PATH**’ environment variable. The run-time support file ‘**gnulib**’ is also searched for using the ‘**-B**’ prefix, if needed. If it is not found there, the two standard prefixes above are tried, and that is all. The file is left out of the link if it is not found by those means. Most of the time, on most machines, you can do without it.

These options control the C preprocessor, which is run on each C source file before actual compilation. If you use the ‘**-E**’ option, nothing is done except C preprocessing. Some of these options make sense only together with ‘**-E**’ because they request preprocessor output that is not suitable for actual compilation.

**-C** Tell the preprocessor not to discard comments. Used with the ‘**-E**’ option.

**-I dir** Search directory *dir* for include files.

**-I-** Any directories specified with ‘**-I**’ options before the ‘**-I-**’ option are searched only for the case of ‘**#include "file"**’; they are not searched for ‘**#include <file>**’. If additional directories are specified with ‘**-I**’ options after the ‘**-I-**’, these directories are searched for all ‘**#include**’ directives. (Ordinarily *all* ‘**-I**’ directories are used this way.) In addition, the ‘**-I-**’ option inhibits the use of the current directory as the first search directory for ‘**#include "file"**’. Therefore, the

current directory is searched only if it is requested explicitly with ‘-I.’. Specifying both ‘-I-’ and ‘-I.’ allows you to control precisely which directories are searched before the current one and which are searched after.

**-nostdinc**

Do not search the standard system directories for header files. Only the directories you have specified with ‘-I’ options (and the current directory, if appropriate) are searched. Between ‘-nostdinc’ and ‘-I-’, you can eliminate all directories from the search path except those you specify.

**-M** Tell the preprocessor to output a rule suitable for **make** describing the dependencies of each source file. For each source file, the preprocessor outputs one **make**-rule whose target is the object file name for that source file and whose dependencies are all the files ‘**#include**’d in it. This rule may be a single line or may be continued ‘\’-newline if it is long. ‘-M’ implies ‘-E’.

**-MM** Like ‘-M’ but the output mentions only the user-header files included with ‘**#include "file"**’. System header files included with ‘**#include <file>**’ are omitted. ‘-MM’ implies ‘-E’.

**-Dmacro**

Define macro *macro* with the empty string as its definition.

**-Dmacro=defn**

Define macro *macro* as *defn*.

**-Umacro**

Undefine macro *macro*.

**-T** Support ANSI C trigraphs. You don’t want to know about this brain-damage. The ‘-ansi’ option also has this effect.

These options control the details of C compilation itself.

**-ansi** Support all ANSI standard C programs. This turns off certain features of GNU C that are incompatible with ANSI C, such as the **asm**, **inline** and **typeof** keywords, and predefined macros such as **unix** and **vax** that identify the type of system you are using. It also enables the undesirable and rarely used ANSI trigraph feature. The ‘-ansi’ option does not cause non-ANSI programs to be rejected gratuitously. For that, ‘-pedantic’ is required in addition to ‘-ansi’. The macro **\_\_STRICT\_ANSI\_\_** is predefined when the ‘-ansi’ option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ANSI standard doesn’t call for; this is to avoid interfering with any programs that might use these names for other things.

**-traditional**

Attempt to support some aspects of traditional C compilers. Specifically:

- \* All **extern** declarations take effect globally even if they are written inside of a function definition. This includes implicit declarations of functions.
- \* The keywords **typeof**, **inline**, **signed**, **const** and **volatile** are not recognized.
- \* Comparisons between pointers and integers are always allowed.
- \* Integer types **unsigned short** and **unsigned char** promote to **unsigned int**.
- \* In the preprocessor, comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.
- \* In the preprocessor, single and double quote characters are ignored when scanning macro definitions, so that macro arguments can be replaced even within a string or character constant. Quote characters are also ignored when skipping text inside a failing conditional directive.

**-pedantic**

Issue all the warnings demanded by strict ANSI standard C; reject all programs that use forbidden extensions. Valid ANSI standard C programs should compile properly with or without this option (though a rare few will require ‘-ansi’. However, without this option, certain GNU extensions and traditional C features are supported as well. With this option, they are rejected. There is no reason to *use* this option; it exists only to satisfy pedants.

- O** Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. Without **'-O'**, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code. Without **'-O'**, only variables declared **register** are allocated in registers. The resulting compiled code is a little worse than produced by PCC without **'-O'**. With **'-O'**, the compiler tries to reduce code size and execution time. Some of the **'-f'** options described below turn specific kinds of optimization on or off.
- g** Produce debugging information in DBX format. Unlike most other C compilers, GNU CC allows you to use **'-g'** with **'-O'**. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops. Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.
- gg** Produce debugging information in GDB(GNU Debugger)'s own format. This requires the GNU assembler and linker in order to work.
- w** Inhibit all warning messages.
- W** Print extra warning messages for these events:

  - \* An automatic variable is used without first being initialized. These warnings are possible only in optimizing compilation, because they require data flow information that is computed only when optimizing. They occur only for variables that are candidates for register allocation. Therefore, they do not occur for a variable that is declared **volatile**, or whose address is taken, or whose size is other than 1,2,4 or 8 bytes. Also, they do not occur for structures, unions or arrays, even when they are in registers. Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by the flow analysis pass before the warnings are printed. These warnings are made optional because GNU CC is not smart enough to see all the reasons why the code might be correct despite appearing to have an error.
  - \* A nonvolatile automatic variable might be changed by a call to **longjmp**. These warnings as well are possible only in optimizing compilation. The compiler sees only the calls to **setjmp**. It cannot know where **longjmp** will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because **longjmp** cannot in fact be called at the place which would cause a problem.
  - \* A function can return either with or without a value. (Falling off the end of the function body is considered returning without a value.) Spurious warning can occur because GNU CC does not realize that certain functions (including **abort** and **longjmp**) will never return.
- Wimplicit**  
Warn whenever a function is implicitly declared.
- Wreturn-type**  
Warn whenever a function is defined with a return-type that defaults to **int**. Also warn about any **return** statement with no return-value in a function whose return-type is not **void**.
- Wcomment**  
Warn whenever a comment-start sequence **'/\*'** appears in a comment.
- p** Generate extra code to write profile information suitable for the analysis program **prof**.
- pg** Generate extra code to write profile information suitable for the analysis program **gprof**.
- library**  
Search a standard list of directories for a library named *library*, which is actually a file named **'liblibrary.a'**. The linker uses this file as if it had been specified precisely by name. The directories searched include several standard system directories plus any that you specify with **'-L'**. Normally the files found this way are library files - archive files whose members are object files.

The linker handles an archive file by through it for members which define symbols that have so far been referenced but not defined. But if the file that is found is an ordinary object file, it is linked in the usual fashion. The only difference between an `-l` option and the full file name of the file that is found is syntactic and the fact that several directories are searched.

`-Ldir` Add directory *dir* to the list of directories to be searched for `-l`.

**`-nostdlib`**

Don't use the standard system libraries and startup files when linking. Only the files you specify (plus `gnulib`) will be passed to the linker.

**`-mmachinespec`**

Machine-dependent option specifying something about the type of target machine. These options are defined by the macro `TARGET_SWITCHES` in the machine description. The default for the options is also defined by that macro, which enables you to change the defaults.

These are the `-m` options defined in the 68000 machine description:

**`-m68020`**

Generate output for a 68020 (rather than a 68000). This is the default if you use the unmodified sources.

**`-m68000`**

Generate output for a 68000 (rather than a 68020).

**`-m68881`**

Generate output containing 68881 instructions for floating point. This is the default if you use the unmodified sources.

**`-msoft-float`**

Generate output containing library calls for floating point.

**`-mshort`**

Consider type `int` to be 16 bits wide, like `short int`.

**`-mnobitfield`**

Do not use the bit-field instructions. `-m68000` implies `-mnobitfield`.

**`-mbitfield`**

Do use the bit-field instructions. `-m68020` implies `-mbitfield`. This is the default if you use the unmodified sources.

**`-mrtld`**

Use a different function-calling convention, in which functions that take a fixed number of arguments return with the `rtld` instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there. This calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler. Also, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); otherwise incorrect code will be generated for calls to those functions. In addition, seriously incorrect code will result if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.) The `rtld` instruction is supported by the 68010 and 68020 processors, but not by the 68000.

These are the `-m` options defined in the VAX machine description:

**`-munix`**

Do not output certain jump instructions ( `ableq` and so on) that the Unix assembler for the VAX cannot handle across long ranges.

**`-mgnu`**

Do output those jump instructions, on the assumption that you will assemble with the GNU assembler.

**`-f flag`**

Specify machine-independent flags. These are the flags:

**-ffloat-store**

Do not store floating-point variables in registers. This prevents undesirable excess precision on machines such as the 68000 where the floating registers (of the 68881) keep more precision than a **double** is supposed to have. For most programs, the excess precision does only good, but a few programs rely on the precise definition of IEEE floating point. Use ‘**-ffloat-store**’ for such programs.

**-frno-asm**

Do not recognize **asm**, **inline** or **typeof** as a keyword. These words may then be used as identifiers.

**-fno-defer-pop**

Always pop the arguments to each function call as soon as that function returns. Normally the compiler (when optimizing) lets arguments accumulate on the stack for several function calls and pops them all at once.

**-fcombine-regs**

Allow the combine pass to combine an instruction that copies one register into another. This might or might not produce better code when used in addition to ‘**-O**’.

**-fforce-mem**

Force memory operands to be copied into registers before doing arithmetic on them. This may produce better code by making all memory references potential common subexpressions. When they are not common subexpressions, instruction combination should eliminate the separate register-load.

**-fforce-addr**

Force memory address constants to be copied into registers before doing arithmetic on them. This may produce better code just as ‘**-fforce-mem**’ may.

**-fomit-frame-pointer**

Don’t keep the frame pointer in a register for functions that don’t need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions. **It also makes debugging impossible.** On some machines, such as the VAX, this flag has no effect, because the standard calling sequence automatically handles the frame pointer and nothing is saved by pretending it doesn’t exist. The machine-description macro **FRAME\_POINTER\_REQUIRED** controls whether a target machine supports this flag.

**-finline-functions**

Integrate all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way. If all calls to a given function are integrated, and the function is declared **static**, then the function is normally not output as assembler code in its own right.

**-fkeep-inline-functions**

Even if all calls to a given function are integrated, and the function is declared **static**, nevertheless output a separate run-time callable version of the function.

**-fwritable-strings**

Store string constants in the writable data segment and don’t uniquize them. This is for compatibility with old programs which assume they can write into string constants. Writing into string constants is a very bad idea; “constants” should be constant.

**-fno-function-cse**

Do not put function addresses in registers; make each instruction that calls a constant function contain the function’s address explicitly. This option results in less efficient code, but some strange hacks that alter the assembler output may be confused by the optimizations performed when this option is not used.

**-fvolatile**

Consider all memory references through pointers to be volatile.

**-funsigned-char**

Let the type **char** be the **unsigned, like unsigned char**. Each kind of machine has a default for what **char** should be. It is either like **unsigned char** by default or like **signed char** by default. (Actually, at present, the default is always signed.) The type **char** is always a distinct type from either **signed char** or **unsigned char**, even though its behavior is always just like one of those two.

**-fsigned-char**

Let the type **char** be the same as **signed char**.

**-ffixed-*reg***

Treat the register named *reg* as a fixed register; generated code should never refer to it (except perhaps as a stack pointer, frame pointer or in some other fixed role). *reg* must be the name of a register. The register names accepted are machine-specific and are defined in the **REGISTER\_NAMES** macro in the machine description macro file.

**-fcall-used-*reg***

Treat the register named *reg* as an allocatable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register *reg*. Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

**-fcall-saved-*reg***

Treat the register named *reg* as an allocatable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register *reg* if they use it. Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results. A different sort of disaster will result from the use of this flag for a register in which function values are may be returned.

**-dletters** Says to make debugging dumps at times specified by *letters*. Here are the possible letters:

- r** Dump after RTL generation.
- j** Dump after first jump optimization.
- J** Dump after last jump optimization.
- s** Dump after CSE (including the jump optimization that sometimes follows CSE).
- L** Dump after loop optimization.
- f** Dump after flow analysis.
- c** Dump after instruction combination.
- l** Dump after local register allocation.
- g** Dump after global register allocation.
- m** Print statistics on memory usage, at the end of the run.

**FILES**

file.c	input file
file.o	object file
a.out	loaded output
/tmp/cc?	temporary
/usr/local/lib/gcc-cpp	preprocessor
/usr/local/lib/gcc-cc1	compiler
/usr/local/lib/gcc-gnulib	library need by GCC on some machines
/lib/crt0.o	runtime startoff
/lib/libc.a	standard library, see <a href="#">intro(3)</a>
/usr/include	standard directory for '#include' files



**SEE ALSO**

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

B. W. Kernighan, *Programming in C*

D. M. Ritchie, *C Reference Manual*

adb(1), ld(1), dbx(1), as(1)

**BUGS**

Bugs should be reported to bug-gcc prep.ai.mit.edu. Bugs tend actually to be fixed if they can be isolated, so it is in your interest to report them in such a way that they can be easily reproduced according to get newer version.

**COPYING**

Copyright (C) 1988 Richard M. Stallman.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the section entitled "GNU CC General Public License" is included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the section entitled "GNU CC General Public License" may be included in a translation approved by the author instead of in the original English.

**AUTHORS**

Richard M. Stallman

**NAME**

getopt – parse command options

**SYNOPSIS**

```
set -- `getopt optstring $*`
```

**DESCRIPTION**

*Getopt* is used to break up options in command lines for easy parsing by shell procedures, and to check for legal options. *Optstring* is a string of recognized option letters (see [getopt\(3C\)](#)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option `--` is used to delimit the end of the options. *Getopt* will place `--` in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments (`$1 $2 . . .`) are reset so that each option is preceded by a `-` and in its own shell argument; each option argument is also in its own shell argument.

**DIAGNOSTICS**

*Getopt* prints an error message on the standard error when it encounters an option letter not included in *optstring*.

**EXAMPLES**

The following code fragment shows how one might process the arguments for a command that can take the options **a** and **b**, and the option **o**, which requires an argument.

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
    echo $USAGE
    exit 2
fi
for i in $*
do
    case $i in
    -a | -b)    FLAG=$i; shift;;
    -o)        OARG=$2;    shift; shift;;
    --)        shift;    break;;
    esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

**SEE ALSO**

[sh\(1\)](#), [getopt\(3C\)](#)

**NAME**

gets – get a string from standard input

**SYNOPSIS**

gets [ default ]

**DESCRIPTION**

**N.B.:** This command was introduced for use in *.login* scripts when the facilities of the *tset(1)* command were not totally adequate in setting the terminal type. This is no longer true, and *gets* should no longer be needed. To boot, a construct “\$<” is available in *cs(1)* now which has the functionality of *gets*:

```
set a=$<
if ($a == ") set a=default
```

replaces

```
set a='gets default'
```

Users of *sh(1)* should use its *read* command rather than *gets*.

*Gets* can be used with *cs(1)* to read a string from the standard input. If a *default* is given it is used if just return is typed, or if an error occurs. The resultant string (either the default or as read from the standard input) is written to the standard output. If no *default* is given and an error occurs, *gets* exits with exit status 1.

**SEE ALSO**

*cs(1)*

**BUGS**

*Gets* is obsolete.

**NAME**

getuid, id – get user identity

**SYNOPSIS**

**getuid** [ - ] [ *arguments* ]

**id**

**DESCRIPTION**

*Getuid* prints on its standard output information about its invoker, based on the effective user id, as presented in the password file. With no arguments, *getuid* prints the login id of its invoker. Arguments select which information to print:

**user** login id  
**group** group id  
**passwd** encrypted password  
**uid** numerical user id  
**gid** numerical group id  
**acct** comp center account number  
**bin** comp center output bin  
**home** home directory  
**shell** default shell

If the optional - or more than one argument is present, the information is displayed in the form

user=name

as suitable for setting environment variables in the shell.

*Id* prints the effective userid and groupid and the login name. The userid and groupid are printed numerically and, if possible, textually.

**FILES**

/etc/passwd

/etc/group

**SEE ALSO**

[who\(1\)](#), [getuid\(2\)](#), [passwd\(5\)](#), [newgrp\(1\)](#)

**BUGS**

*Getuid* reports the default group for the user, not the current effective group id.

The login id reported is the first one in the password file with the correct numerical user id, not necessarily the login for the current session.

**NAME**

*gram* – find split infinitives and incorrect indefinite articles  
*splitrules* – print information about split infinitives

**SYNOPSIS**

**gram** [ **-flags** ] [ **-ver** ] [file ...]

**splitrules** [ **-flags** ] [ **-ver** ]

**DESCRIPTION**

*Gram* uses the *parts*(1) (part of speech assignment) program to look for infinitives that are split by one or more adverbs. It also checks for incorrect indefinite articles.

Grammatical information about split infinitives can be obtained by typing: **splitrules**.

Two options, which apply to both programs, give information about the programs:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

*Gram* is one of the programs run under the *proofr*(1) and *wwb*(1) commands.

**FILES**

/tmp/\$\$                      temporary files

**SEE ALSO**

*proofr*(1), *wwb*(1), *deroff*(1), *parts*(1).

**BUGS**

Because *parts* is not always correct in its assignments, *gram* also makes errors.

**SUPPORT**

**COMPONENT NAME:** Writer's Workbench

**APPROVAL AUTHORITY:** Div 452

**STATUS:** Standard

**SUPPLIER:** Dept 45271

**USER INTERFACE:** Stacey Keenan, Dept 45271, PY x3733

**SUPPORT LEVEL:** Class B - unqualified support other than Div 452

**NAME**

grap – pic preprocessor for drawing graphs

**SYNOPSIS**

**grap** [ *file ...* ]

**DESCRIPTION**

*Grap* is a *pic*(1) preprocessor for drawing graphs on a typesetter. Graphs are surrounded by the *troff* ‘commands’ **.G1** and **.G2**. Data are scaled and plotted, with tick marks supplied automatically. Commands exist to modify the frame, add labels, override the default ticks, change the plotting style, define coordinate ranges and transformations, and include data from files. In addition, *grap* provides the same loops, conditionals and macro processing that *pic* does.

**frame ht e wid e top dotted ...**: Set the frame around the graph to specified **ht** and **wid**; default is 2 by 3 (inches). The line *styles* (**dotted**, **dashed**, **invis**, **solid** (default)) of the *sides* (**top**, **bot**, **left**, **right**) of the frame can be set independently.

**label side "a label" "as a set of strings" adjust**: Place label on specified side; default side is bottom. *adjust* is **up** (or **down left right**) *expr* to shift default position; **width** *expr* sets the width explicitly.

**ticks side in at optname expr, expr, ...**: Put ticks on *side* at *expr*, ..., and label with "*expr*". If any *expr* is followed by "...", label tick with "...", and turn off all automatic labels. If "..." contains %*f*'s, they will be interpreted as **printf** formatting instructions for the tick value. Ticks point **in** or **out** (default out). Tick iterator: instead of **at ...**, use **from expr to expr by op expr** where *op* is optionally **+\*** for additive or multiplicative steps. **by** can be omitted, to give steps of size 1. If no ticks are requested, they are supplied automatically; suppress this with **ticks off**. Automatic ticks normally leave a margin of 7% on each side; set this to anything by **margin = expr**.

**grid side linedesc at optname expr, expr, ...**: Draw grids perpendicular to *side* in style *linedesc* at *expr*, .... Iterators and labels work as with ticks.

**coord optname x min, max y min, max log x log y**: Set range of coords and optional log scaling on either or both. This overrides computation of data range. Default value of *optname* is current coordinate system (each **coord** defines a new coordinate system).

**plot "str" at point; "str" at point**: Put *str* at *point*. Text position can be qualified with **rjust**, **ljust**, **above**, **below** after "...".

**line from point to point linedesc**: Draw line from here to there. **arrow** works in place of **line**.

**next optname at point linedesc**: Continue plot of data in *optname* to *point*; default is current.

**draw optname linedesc ...**: Set mode for **next**: use this style from now on, and plot "..." at each point (if given).

**new optname linedesc ...**: Set mode for **next**, but disconnect from previous.

A list of numbers *x y1 y2 y3 ...* is treated as **plot bullet at x,y1**; **plot bullet at x,y2**; etc., or as **next at x,y1** etc., if **draw** is specified. Abscissae of 1,2,3,... are provided if there is only one input number per line.

A point *optname expr, expr* maps the point to the named coordinate system. A *linedesc* is one of **dot dash invis solid** optionally followed by an expression.

**define name {whatever}**: Define a macro. There are macros already defined for standard plotting symbols like **bullet**, **circle**, **star**, **plus**, etc., in which is included if it exists.

*var = expr*: Evaluate an expression. Operators are **+** **-** **\*** and **/**. Functions are **log** and **exp** (both base 10), **sin**, **cos**, **sqrt**; **rand** returns random number on [0,1); **max(e,e)**, **min(e,e)**, **int(e)**.

**print expr; print "...**": As a debugging aid, print *expr* or *string* on the standard error.

**copy "filename"**: Include this file right here.

**copy thru macro**: Pass rest of input (until **.G2**) through *macro*, treating each field (non-blank, or "...") as an argument. *macro* can be the name of a macro previously defined, or the body of one in place, like **/plot \$1 at \$2,\$3/**.

**copy thru macro until "string"**: Stop copy when input is *string* (left-justified).

**pic remainder of line**: Copy to output with leading blanks removed.

**graph** *Name pic-position*: Start a new frame, place it at specified position, e.g., **graph Thing2** with **.sw** at **Thing1.se + (0.1,0)**. *Name* must be capitalized to keep *pic* happy.

**.anything at beginning of line**: Copied verbatim.

**sh %anything %**: Pass everything between the %'s to the shell; as with macros, % may be any character and *anything* may include newlines.

**# anything**: A comment, which is discarded.

Order is mostly irrelevant; no category is mandatory. Any arguments on the **.G1** line are placed on the generated **.PS** line for *pic*.

## EXAMPLES

```
.G1
frame ht 1 top invis right invis
coord x 0, 10 y 1, 3 log y
ticks left in at 1 "bottommost tick", 2,3 "top tick"
ticks bot in from 0 to 10 by 2
label bot "silly graph"
label left "left side label" "here"
grid left dashed at 2.5
copy thru / circle at $1,$2 /
1 1
2 1.5
3 2
4 1.5
10 3
.G2
frame ht 1 top invis right invis
coord x 0, 10 y 1, 3 log y
ticks left in at 1 "bottommost tick", 2,3 "top tick"
ticks bot in from 0 to 10 by 2
label bot "silly graph"
label left "left side label" "here"
grid left dashed at 2.5
copy thru / circle at $1,$2 /
1 1
2 1.5
3 2
4 1.5
10 3
```

## FILES

/usr/lib/grap.defines  
 definitions of standard plotting characters, e.g., bullet

## SEE ALSO

[graph\(1\)](#), [pic\(1\)](#), [troff\(1\)](#), [plot\(3\)](#)

J. L. Bentley and B. W. Kernighan, 'GRAP—A Language for Typesetting Graphs', this manual, Volume 2

**NAME**

graph – draw a graph

**SYNOPSIS**

**graph** [ *option ...* ]

**DESCRIPTION**

*Graph* with no options takes pairs of numbers from the standard input as abscissas ( $x$ -values) and ordinates ( $y$ -values) of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by *plot(1)* filters.

If the ordinate of a point is followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes " " in which case they may be empty or contain blanks and numbers; labels never contain newlines.

The following options are recognized, each as a separate argument.

- a** Supply abscissas automatically; no  $x$ -values appear in the input. Spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0, or 1 with a log scale in  $x$ , or the lower limit given by **-x**).
- b** Break (disconnect) the graph after each label in the input.
- c** Character string given by next argument is default label for each point.
- g** Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
- l** Next argument is a legend to title the graph. Grid ranges are automatically printed as part of the title unless a **-s** option is present.
- m** Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected. Some devices give distinguishable line styles for other small integers. Mode  $-1$  (default) begins with style 1 and rotates styles for successive curves under option **-o**.
- o** (Overlay.) The ordinates for  $n$  superposed curves appear in the input with each abscissa value. The next argument is  $n$ .
- s** Save screen; no new page for this graph.
- x l** If **l** is present,  $x$ -axis is logarithmic. Next 1 (or 2) arguments are lower (and upper)  $x$  limits. Third argument, if present, is grid spacing on  $x$  axis. Normally these quantities are determined automatically.
- y l** Similarly for  $y$ .
- e** Make automatically determined  $x$  and  $y$  scales equal.
- h** Next argument is fraction of space for height.
- w** Similarly for width.
- r** Next argument is fraction of space to move right before plotting.
- u** Similarly to move up before plotting.
- t** Transpose horizontal and vertical axes. (Option **-a** now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

**SEE ALSO**

*plot(1)*, *grap(1)*, *spline(A)*

**BUGS**

In *graph* segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

Option **-e** actually makes automatic limits, rather than automatic scaling, equal.



**NAME**

*gre*, *grep*, *egrep*, *fgrep* – search a file for a pattern

**SYNOPSIS**

**gre** [ *option ...* ] *pattern* [ *file ...* ]

**grep** [ *option ...* ] *pattern* [ *file ...* ]

**egrep** [ *option ...* ] *pattern* [ *file ...* ]

**fgrep** [ *option ...* ] *strings* [ *file ...* ]

**DESCRIPTION**

*Gre* searches the input *files* (standard input default) for lines (with newlines excluded) that match the *pattern*, a regular expression as defined in [re\(3\)](#). A file name of *-* is interpreted as standard input. Normally, each line matching the pattern is ‘selected’, and each selected line is copied to the standard output. The options are

- 1** Print only the first selected line of each file argument.
- b** Mark each printed line with its byte position in its file. This is sometimes useful in locating patterns in non-text files.
- c** Print only a count of matching lines.
- e *pattern*** Same as a simple *pattern* argument, but useful when *pattern* begins with a *-*.
- E** Simulate *egrep*.
- f *file*** Read the pattern from *file*; there is no *pattern* argument
- F** Simulate *fgrep*.
- G** Simulate *grep*.
- h** Do not print filename tags (headers) with output lines.
- i** Ignore alphabetic case distinctions.
- l** Print the names of files with selected lines; don’t print the lines.
- L** Print the names of files with no selected lines; the converse of **-l**.
- n** Mark each printed line with its line number counted in its file.
- s** Produce no output, but return status.
- v** Reverse: print lines that do not match the pattern.
- x** Exact match: The pattern is  $\wedge(\textit{pattern})\$$ . The implicit parentheses count in back references.

Output lines are tagged by filename when there is more than one input file. (To force this tagging, include */dev/null* as a filename argument.) If the output line exceeds some internal limit, a warning is given and a small block of text surrounding the match is printed.

Care should be taken when using the shell metacharacters  $\$*$ ,  $\wedge$ ,  $()$ , and newline in *pattern*; it is safest to enclose the entire expression in single quotes ‘...’.

*Gre* supplants three classic programs, which are still available:

*Grep* handles only [ed\(1\)](#)-like regular expressions. It uses  $\backslash( \backslash)$  instead of  $( )$ .

*Egrep* handles the same patterns as *gre* except for back-referencing with  $\backslash1, \backslash2, \dots$

*Fgrep* handles no operators except newline (alternation).

**SEE ALSO**

[re\(3\)](#), [awk\(1\)](#), [sed\(1\)](#), [sam\(9\)](#) [strings\(1\)](#)

**DIAGNOSTICS**

Exit status is 0 if any lines are selected, 1 if none, 2 for syntax errors, inaccessible files (even if matches were found). Warnings will be given for input lines that exceed a (generous) internal limit.

**BUGS**

*Grep*, *egrep*, and *fgrep* do not support some options and print (approximate) block numbers rather than byte numbers for option **-b**.

*Egrep* may fail on input containing characters greater than 0176.

**NAME**

grep, egrep, fgrep – search a file for a pattern

**SYNOPSIS**

**grep** [ *option ...* ] *expression* [ *file ...* ]

**egrep** [ *option ...* ] *expression* [ *file ...* ]

**fgrep** [ *option ...* ] *strings* [ *file ...* ]

**DESCRIPTION**

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed(1)*; it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

- f file** Read the pattern from *file*; there is no pattern argument (*egrep* and *fgrep*).
- v** Reverse: print lines that do not match.
- i** Ignore alphabetic case distinctions.
- n** Mark each printed line with its line number counted in its file.
- x** Exact: print only lines matched in their entirety (*fgrep* only).
- c** Print only a count of matching lines.
- l** Print only the names of files with matching lines (once).
- b** Mark each printed line with its block number counted in its file. This is sometimes useful in locating disk block numbers by context. A block is 1024 bytes.
- h** Do not print filename headers with output lines.
- s** Produce no output, but return status.
- e expression**  
Same as a simple *expression* argument; useful when the *expression* begins with a -.

Output lines are tagged by filename when there is more than one input file. Care should be taken when using the shell metacharacters `$*[^\()]` in *expression*; it is safest to enclose the entire *expression* in single quotes `'...'`.

*Fgrep* searches for lines that contain any of the *strings*, which appear as a single argument with embedded newlines.

*Egrep* accepts regular expressions as in *ed(1)*, with the following changes:

1. There is no backreferencing (`\(\)\1\2...`).
2. The characters `^`, `$`, and `*` are always metacharacters unless quoted or contained in brackets `[]`.
3. A regular expression followed by `+` matches one or more occurrences of the expression.
4. A regular expression followed by `?` matches 0 or 1 occurrence.
5. Two regular expressions separated by `|` or newline match occurrences of either.
6. Parentheses `()` specify grouping.

The order of precedence of operators is `[]`, then `*?+`, then concatenation, then `|` and new-line.

**SEE ALSO**

*ed(1)*, *sed(1)*, *sh(1)*.

**DIAGNOSTICS**

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

**BUGS**

Ideally there should be only one *grep*, but we do not know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are truncated at **BUFSIZ** characters; see *setbuf(3)*. Null characters behave as end-of-line in matches.

**NAME**

hang – start a process in stopped state

**SYNOPSIS**

**hang** *command*

**DESCRIPTION**

The given command is executed, but stopped before **main()** is called so that the process can be picked up by a debugger. To ensure that breakpoints may be set, the process does not share its text.

**SEE ALSO**

[pi\(9\)](#) [kill\(1\)](#)

**NAME**

hoc – interactive floating point language

**SYNOPSIS**

**hoc** [ *file ...* ]

**DESCRIPTION**

*Hoc* interprets a simple language for floating point arithmetic, at about the level of Basic, with C-like syntax and functions.

The named *files* are read and interpreted in order. If no *file* is given or if *file* is *- hoc* interprets the standard input.

*Hoc* input consists of *expressions* and *statements*. Expressions are evaluated and their results printed. Statements, typically assignments and function or procedure definitions, produce no output unless they explicitly call *print*.

Variable names have the usual syntax, including *\_*; the name *\_* by itself contains the value of the last expression evaluated. Certain variables are already initialized:

**E** base of natural logs

**PI**

**PHI** golden ratio

**GAMMA**

Euler's constant

**DEG** 180/PI, degrees per radian

**PREC**

maximum number of significant digits in output, initially 15; **PREC=0** gives shortest 'exact' values.

Expressions are formed with these C-like operators, listed by decreasing precedence.

**^** exponentiation

**! - ++ --**

**\* / %**

**+ -**

**> >= < <= == !=**

**&&**

**||**

**= += -= \*= /= %=**

Built in functions include **abs**, **acos**, **atan** (one argument), **cos**, **cosh**, **erf**, **erfc**, **exp**, **gamma**, **int**, **log**, **log10**, **sin**, **sinh**, **sqrt**, **tan**, and **tanh**. The function **read(x)** reads a value into the variable **x**; the statement **print** prints a list of expressions that may include string constants such as **"hello\n"**.

Control flow statements are **if-else**, **while**, and **for**, with braces for grouping. Newline ends a statement. Backslash-newline is equivalent to a space.

Functions and procedures are introduced by the words **func** and **proc**; **return** is used to return with a value from a function. Within a function or procedure, arguments are referred to as **\$1**, **\$2**, etc.; all other variables are global.

**EXAMPLES**

```
func gcd() {
    temp = abs($1) % abs($2)
    if(temp == 0) return abs($2)
    return gcd($2, temp)
}
for(i=1; i<12; i++) print gcd(i,12)
```

**SEE ALSO**

[bc\(1\)](#), [dc\(1\)](#)

B. W. Kernighan and R. Pike, *The Unix Programming Environment*, Prentice-Hall, 1984

**BUGS**

Error recovery is imperfect within function and procedure definitions.

The treatment of newlines is not exactly user-friendly.

**NAME**

hostname, whoami – computer name

**SYNOPSIS**

**hostname**

**DESCRIPTION**

*Hostname* prints the computer name as used by [mail\(1\)](#) or [uucp\(1\)](#).

**FILES**

`/etc/whoami`  
file containing the computer name

**NAME**

*ul*, *hp* – print underlines on screen terminals

**SYNOPSIS**

**ul** [ **-i** ] [ **-t** *terminal* ] [ *file ...* ]

**hp** [ **-e** ] [ **-m** ]

**DESCRIPTION**

*Ul* replaces backspaced, overstruck underscores by control sequences suitable for the terminal given by the environment variable `TERM` or by option **-t**. It reads from the standard input or the named files and writes on the standard output. Option **-i** represents underlining by a separate line of – characters.

*Hp* is a filter that presents most *nroff* output sensibly on HP 2600 series terminals. Option **-s** stops and waits for a newline at the beginning of each page. Option **-e** uses ‘display enhancement’ features to distinguish underlines, superscripts, and subscripts, which are normally all shown in inverse video. Option **-m** squeezes multiple newlines out of the output.

**SEE ALSO**

[column\(1\)](#)

**BUGS**

*Hp* does not reliably handle reverse line feeds as produced by [tbl\(1\)](#); pipe the input through *col* to get rid of them; see [column\(1\)](#).

**NAME**

ican, ibcan, idcan, itcan – interface to Imagen laser-printer spooler

**SYNOPSIS**

**ican** [ *option ...* ] [ *file ...* ]

**ibcan** [ *option ...* ] [ *file ...* ]

**idcan** [ *option ...* ] [ *file ...* ]

**itcan** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

These commands print *files* (standard input by default) on Imagen laser printers. The four commands handle particular kinds of data files:

*ican* ASCII text

*ibcan* bitmap images created by *blitblt*(9)

*idcan* output from *troff*(1)

*itcan* output for a Tektronix 4014 terminal, as produced by *plot*(1)

The destination printer is determined in the following ways, listed in order of decreasing precedence.

option **-d** *dest*  
 environment variable ICANDEST  
 printer named in file */etc/icandest*

Printers at the mother site are:

**1** 1st floor, stair 8 (sid)  
**3** 3rd floor, stair 8 (shannon)  
**5** 5th floor, 2C-5 (hunny)  
**j** 5th floor, 2D-5 (jones)  
*/name* printer attached to machine with Datakit destination *name*

Options:

**-c** *n* Number of copies to be printed.

**-d** *dest*  
 Select the destination printer.

**-f** *font* Set the font (default CW.11) for *can*; see *font*(7)

**-L** (landscape) Rotate *ibcan* pages 90 degrees.

**-l** *n* Set number of lines per page for *can* (default 66).

**-m** *n* Set *ibcan* magnification to a power of 2, where *n* = 0, 1, or 2 (default 1).

**-o** *list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *n-m* means pages *n* through *m*; a range *-b* means from the beginning to page *n*; a range *n-* means from page *n* to the end. **-o** implies **-r**.

**-r** print pages in reverse order (default for *ican* and *idcan*).

**-u** *user*  
 set the name which appears on the banner page; default is login name.

**-x** *n* set the horizontal offset of the print image, measured in dots (default 60). There are 300 dots to the inch.

**-y** *n* set the vertical offset of the print image (default 0), except in *itcan*, where this option specifies *n* extra tekpoints vertically.

**FILES**

*/etc/icandest*  
 default destination



```
/usr/lib/font/devi300
  font directory
/usr/spool/lp
  spool directory
```

**SEE ALSO**

[pr\(1\)](#), [blitblt\(9\)](#) [plot\(1\)](#), [font\(7\)](#)

**BUGS**

The 'landscape' option is supported only by *ibcan*; **-o** is supported only by *ican* and *idcan*.  
There ought to be a way to determine the service class from the input data.

**NAME**

iclc – Esterel binder

**SYNOPSIS**

**iclc** [ option ] ... [ file ]...

**DESCRIPTION**

*iclc* is the Esterel v3 binder. It produces an *lc* format output (or *ic* if some **copymodule** instruction could not be expanded) from one or more *ic* format inputs. If there is no input file, the standard input is used. *ic* format input describes Esterel **modules** to be processed, and *lc* format output describes Esterel **modules** with no **copymodule** instruction. Typical use is:

```
iclc < game1.ic > game.lc
```

or

```
iclc game1.ic game2.ic > game.lc
```

The following options are interpreted by *iclc*:

- v**            Verbose mode. Tells what's going on.
- version**     Prints the version number and exits.
- stat**         Prints times and memory sizes for the main phases.
- memstat**     Gives the memory allocator state at the end of processing.
- Rs**          Signal renaming trace mode.
- Rc**          Constant renaming trace mode.
- cascade**     "Cascade" mode. Creates a file FOO.casc using the -o, -B and -D options to find the name (esterel.casc as a last resort).
- B name**      Basename for the auxiliary output file.
- D name**      The name of the directory where the auxiliary output file will go. For instance, `iclc -D /users/john/wd -B game -cascade game*.ic` will write in the file `"/users/john/wd/game.casc"`.
- d[level]**    Debug mode. Barely for you.
- o name**      Names the final output file *name* (deleting the existing text). Obsolete.
- Specifies the standard input as input stream. Works only once. Obsolete.

**FILES**

The caller of the command must have read/write permission for the directories containing the working files, and execute permission for the *iclc* file itself.

**DIAGNOSTICS**

The diagnostics produced by *iclc* compiler are intended (as usual) to be self-explanatory. They have one of the following forms:

```
"file",line n: iclc error (or warning) : message
```

```
*** iclc: message
```

```
>>>iclc s_trace (or c_trace) : message
```

The first two forms are described in the *Error Messages Manual*. The last one is generated by the **-Rs** or **-Rc** option. The possible messages with **-Rc** are:

root module FOO:

the binder begins to treat the root module FOO.

submodule /FOO/BAR:

the binder begins to treat the module BAR, "called" by module FOO.

CONSTANT added as 33 <<:

CONSTANT is added to the list of final constants with number 33. The "<<" is here to help you find later the name of constant number 33. Just look upward for 33 followed by "<<".

CONSTANT captured by 33 in module /FOO/BAR:

CONSTANT is implicitly captured by final constant number 33 which was defined in module /FOO/BAR.

CONSTANT replaced by 33 in module /FOO/BAR:

CONSTANT is explicitly renamed to final constant number 33 by a copymodule instruction defined in module /FOO/BAR.

Messages generated by -Rs are alike, except that no module name is given (all signals must be defined in the parent module).

## IDENTIFICATION

Author: J-M. Tanzi, CMA, Ecole des Mines de Paris,  
Sophia-Antipolis, 06600 Valbonne, FRANCE

Revision Number: \$Revision: 1.3 \$ ; Release Date: \$Date: 88/07/04 10:32:28 \$ .

## SEE ALSO

Esterel v3 Programming Language Manual  
Esterel v3 System Manuals.  
strlic (1), lcoc(1), ocl (1).

## BUGS

- error messages should point to the Esterel source code and not to an intermediate code input file.
- there is no error message if the same signal or constant appears more than once in a renaming list. Only one renaming is applied, however.
- the "cascade" mode is not fully implemented.

**NAME**

*icont*, *iconc* – Icon language translator and compiler

**SYNOPSIS**

**icont** [ option ... ] file ... [ **-x** arg ... ]

**iconc** [ option ... ] file ...

**DESCRIPTION**

*icont* translates Version 5 of the Icon programming language to an intermediate form, and link edits intermediate files to interpretable files. *iconc* does the same, but finally compiles to machine code. Unless the **-o** option is specified, the name of the linked file is formed by deleting the suffix of the first input file named on the command line. Option **-x** invokes the interpreter and passes the *args* to the Icon program.

Files whose names end in *.icn* are assumed to be Icon source programs; files whose names end in *.u1* or *.u2* are assumed to be intermediate files from a previous translation (only one should be named — the other is assumed). Unnamed *.u1* and *.u2* files are deleted. The argument **-** signifies the use of standard input as a source file.

The following options are recognized by *icont*.

- c** Suppress linking and loading; preserve intermediate files.
- m** Preprocess each *.icn* source file with the *m4(1)* macro processor before translation.
- o output**  
Name the interpretable file *output*.
- s** Suppress informative messages.
- t** Arrange for *trace* to have an initial value of **-1** instead of **0** when the program is executed.
- u** Issue warning messages for undeclared identifiers.

To run either an interpretable or an executable file, simply execute it as a command. The following environment variables – all numeric – affect execution:

**TRACE**

Initialize the value of *trace*, overriding the translation option **-t**.

**NBUFS**

The number of i/o buffers to use for files, normally 3. *input* and *output* are buffered unless they are terminals. *errout* is never buffered.

**STRSIZE**

The initial size of the string space, in bytes, normally 51200.

**HEAPSIZE**

The initial size of the heap, in bytes, normally 51200.

**NSTACKS**

The number of stacks initially available for co-expressions, normally 4.

**STKSIZE**

The size of each co-expression stack, in words, normally, 2000.

**PROFILE**

Turn on execution profiling of the runtime system. The value of this variable specifies the sampling resolution, in words. If the value is zero, profiling is not done. The profiling results are left in a file *mon.out* for interpretation by *prof(1)*.

**FILES**

v5v/int/bin/utran	icon translator
v5v/int/bin/ulink	icon linker
v5v/cmp/bin/libi.a	icon runtime library
v5v/int/bin/iconx	icon interpreter
mon.out	results of profiling
*.u1, *.u2	intermediate files

**SEE ALSO**

*Reference Manual for the Icon Programming Language, Version 5*, Technical Report TR 81-4a, Department of Computer Science, The University of Arizona, Tucson, Arizona, December 1981.

*Co-Expressions in Icon*, Technical Report TR 82-4, Department of Computer Science, The University of Arizona.

iconc(1), m4(1), prof(1), exec(2), monitor(3)

**BUGS**

If the **-m** option is used, line numbers reported in error messages or tracing messages are from the file after, not before, preprocessing.

Integer overflow on multiplication is not detected.

An interpretable file produced on one system will not work on another system unless the Icon interpreter is in the same place on both systems.

Because of the way that co-expressions are implemented, there is a possibility that programs in which they are used may malfunction mysteriously.

**NAME**

*ideal* – troff preprocessor for drawing pictures

**SYNOPSIS**

**ideal** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Ideal* is a constraint-based *troff*(1) preprocessor for typesetting figures in the complex plane. A line beginning **.IS** marks the start of an *ideal* program, **.IE** or **.IF** marks the end. **.IE** leaves the typesetting baseline below the bottom of the picture; **.IF** (flyback) leaves it at the top. The options are

- Tdev** Produce instructions for *troff*(1) device *dev*. **-a** is a synonym for **-Taps**; **-t** for **-T202**.
- tex** Produce output for *tex*(1).
- p** Produce *plot*(1) instructions. Erases come unbidden at every **.IS**.
- 4** Produce instructions for a Tektronix 4014 and wait at each **.IE** for an input character before erasing and starting the next picture.
- n** Produce raw *ideal* output, which passes unharmed through *nroff*.
- v** Print calculated values of variables on standard error.

*Ideal* programs are built of ‘boxes’; boxes look like C functions, in that they are named and delimited by braces. They may include the following kinds of statements, each terminated by a semicolon:

**var** declares one or more complex variables local to the box. Variable names are made up of letters and digits, and start with a letter; do not use any of the following keywords as variable names: *at*, *bdlist*, *boundary*, *box*, *conn*, *construct*, *draw*, *exterior*, *interior*, *left*, *opaque*, *put*, *right*, *spline*, *text*, *to*, *using*, *var*

*equation*

declares relative positions of significant points of the box

**conn** asks for a straight-line path through named points

**pen** asks for a box to be replicated along a line between two points

**left** left-justifies text with respect to a point

**text** centers text with respect to a point

**right** right-justifies text with respect to a point

**spline** draws a spline guided by the named points

**put** asks for an instance of a box to be drawn

**opaque**

asks for a box to erase lines already in the picture that are covered by its bounding polygon

**boundary**

specifies the bounding polygon for an opaque box

**construct**

builds a partial picture on a separate ‘sheet of paper’

**draw** adds the contents of the named constructed box to the current picture

*Ideal* expects all components of a picture to be specified as boxes; instructions to draw the entire picture should comprise a box called *main*. Boxes are remembered across **.IS/.IE** boundaries; if you won’t need a box again, you can reclaim the space it requires by including the command **...forget *boxname*** on a line between any **.IS/.IE** pair after the last use of *boxname*. Box *main* is an exception to this rule: it is always forgotten at **.IE**.

During its first pass, *ideal* solves all the equations to determine the locations of all points it needs to know. These equations must be linear equations in complex variables, although they may include non-linear operators: *ideal* plugs in for as many variables, and does as much function evaluation, as it can before solving the linear equation. It waits until it has absolutely no hope of reducing an equation to a linear equation before complaining. *Ideal* knows about the following functions:

$f[z,w]$  ==  $z+(w-z)f$ , fraction  $f$  of the way from  $z$  to  $w$

**re**( $z$ ) real part of complex number

**im**( $z$ ) imaginary part of complex number

**conj**( $z$ ) complex conjugate of complex number

**abs**( $z$ ) absolute value (modulus) of complex number

**cis**( $x$ ) the unit vector  $\$ \cos^{\sim}x^{\sim} + i^{\sim} \sin^{\sim}x^{\sim} \$$ , (.)if t .ig  $\cos(x) + i \sin(x)$  (.)where  $x = \text{re}(z)$  and  $x$  is measured in degrees (radians if the line **...radians** appeared more recently in the file than the line **...degrees**)

**E**( $x$ ) ==  $\text{cis}(360 x)$  if  $x$  is measured in degrees

**angle**( $z$ ) angle of complex number,  $\arctan(\text{im}(z)/\text{re}(z))$

During the second pass, *ideal* draws the picture.

To draw a circle, include the line **...libfile circle** between the **.IS** and **.IE** lines, and **put** the box named `circle`, giving enough information that the circle can be determined; for instance, give the center and the radius, or give three points through which the circle passes, or give the center and a point on the circle. The circle has center `center`, radius `radius`, and passes through  **$z_1$** ,  **$z_2$** , and  **$z_3$** .

To draw an arc, include the line **...libfile arc** between the **.IS** and **.IE** lines, and **put** the box named `arc`, again giving enough information to determine the arc; for instance, give the center, radius, and starting and ending angles, or give three points on the arc--where to start, where to end, and somewhere in between. The arc has center `center`, radius `radius`, starts at point `start`, passes through point `midway` at angle `midang`, and ends at point `end` at angle `endang`. If no `midway` is specified, the arc is drawn counterclockwise from `start` to `end`.

The picture will be scaled to a default width of four inches and centered in a column of six inches. The default width can be changed by a **...width** command, which includes a number in inches. The default column width can be changed by a **...colwid** command. To defeat *ideal*'s notion of the size of the picture, you can include lines of the form **...minx**, **...miny**, **...maxx**, or **...maxy**; these give the various coordinates of the bounding box of the picture in the coordinate system used by the picture.

*Ideal* supports both C-style comments (between `/*` and `*/` brackets — which nest), and shell-style comments (between `#` and newline).

## EXAMPLES

```
triangle { libfile circle; z1, z2, z3; conn z1 to z2 to z3 to z1; } main { put T: triangle {
triangle { z1 = 0; z2 = 1; z3 = (2,2); } put circle { z1 = T.z1;
z2 = T.z2; z3 = T.z3; } }
```

## SEE ALSO

[troff\(1\)](#), [pic\(1\)](#), [ped\(9\)](#) [doctype\(1\)](#)

C. J. Van Wyk, 'IDEAL User's Manual', this manual, Volume 2

## BUGS

*Ideal* is relatively unforgiving about syntax errors.  
Bounding box computation is naive for arcs and text strings.

```
put circle {
z1 = T.z1; z2 = T.z2; z3 = T.z3;
}
}
```

**NAME**

idiff – interactive file comparison

**SYNOPSIS**

**idiff** [ *option* ] *file1 file2*

**DESCRIPTION**

*Idiff* compares *file1* with *file2* using *diff*, then presents each set of changed lines for selection or processing. *File2* may be a directory; in that case, the basename of *file1* is appended.

For each group, legal responses are

- < to retain the ‘from’ lines
- > to retain the ‘to’ lines
- e to edit both sets of lines
- d to delete both sets
- 1 to retain the rest of the ‘from’ file
- 2 to retain the rest of the ‘to’ file
- ! to invoke a shell command

Lines that compare equal are copied verbatim from *file1*. Lines produced by this process, including the lines written from within the editor, are written to file *idiff.out*. Comparison may be affected by the [diff\(1\)](#) options

- b** Ignore trailing blanks (spaces and tabs) and treat other strings of blanks as if they were a single space.
- B** Ignore all blanks.

**FILES**

idiff.out  
idiff.\*  
/tmp/idiff.\*

**SEE ALSO**

[diff\(1\)](#)

**BUGS**

There is no way to revisit a choice.



**NAME**

imscan – scan greyscale images

**SYNOPSIS**

**imscan** [ **-sN** ] [ **-lN** ] *file*

**DESCRIPTION**

*Imscan* digitizes an image with an Imagitex grey-scale scanner and places the result in the named file in the form of *picfile(5)*. The options are

- sN** Set a scale factor  $1 \leq N \leq 9$ , default 4. With scale factor *N* the image is subsampled: only 1 out of every  $N \times N$  pixels is stored. A larger scale factor, therefore, produces a smaller image.
- lN** Use lens focal length *N*, where *N* is either 5 or 8 (default). The 8-inch lens scans images at 480 dots per inch. The 5-inch lens scans at 754 dots per inch.

**SEE ALSO**

*cscan(1)*, *pico(1)*, *qsnap(1)*, *mugs* in *face(9)* *picfile(5)*

**BUGS**

It is hard to get more than 2000 pixels per scanline reliably. For large originals, higher scale factors work better than smaller ones.

**NAME**

join – relational database operator

**SYNOPSIS**

**join** [ *options* ] *file1 file2*

**DESCRIPTION**

*Join* forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If one of the file names is `-`, the standard input is used.

*File1* and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Input fields are normally separated spaces or tabs; output fields by space. In this case, multiple separators count as one, and leading separators are discarded.

The following options are recognized, with Posix syntax.

- a** *n*     In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
- v** *n*     Like **-a**, omitting output for paired lines.
- e** *s*     Replace empty output fields by string *s*.
- 1** *m*
- 2** *m*     Join on the *m*th field of *file1* or *file2*.
- jn** *m*    Archaic equivalent for **-n m**.
- o** *fields*
- Each output line comprises the designated fields. The comma-separated field designators are either **0**, meaning the join field, or have the form *n.m*, where *n* is a file number and *m* is a field number. Archaic usage allows separate arguments for field designators.
- tc**     Use character *c* as the only separator (tab character) on input and output. Every appearance of *c* in a line is significant.

**EXAMPLES**

```
sort /etc/passwd | join -t: -a 1 -e "" - bdays Add birthdays to password information, leaving
unknown birthdays empty. The layout of /etc/passwd is given in passwd\(5\); bdays contains
sorted lines like ken:Feb 4.
```

```
tr : ' ' </etc/passwd | sort -k 3 3 >temp
join -1 3 -2 3 -o 1.1,2.1 temp temp | awk '$1 < $2' Print all pairs of users with identical userids.
```

**SEE ALSO**

[sort\(1\)](#), [comm\(1\)](#), [awk\(1\)](#)

**BUGS**

With default field separation, the collating sequence is that of **sort -b -ky,y**, with **-t**, the sequence is that of **sort -tx -ky,y**.

One of the files must be randomly accessible.

**NAME**

kill – terminate a process with extreme prejudice

**SYNOPSIS**

kill [ *-sig* ] *processid* ...

kill -l

**DESCRIPTION**

*Kill* sends the SIGTERM signal to the specified processes. If a signal name or number preceded by - is given as first argument, that signal is sent instead; see [signal\(2\)](#). The signal names are listed by kill -l, and are as given in `<signal.h>`.

The terminate signal will kill processes that do not catch the signal. The **SIGKILL** signal is a sure kill, since it cannot be caught. By convention, if process number 0 is specified, all members in the process group (usually processes of the current login or current [mux\(9\)](#) layer) are signaled. Killed processes must belong to the current user unless that is super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a terminate signal by kill 1; see [init\(8\)](#). To force *init* to close and open terminals according to what is currently in `/etc/ttyts` use **kill -SIGHUP 1**.

The process number of an asynchronous process started with & is reported by the shell and by [ps\(1\)](#).

**EXAMPLES**

kill 7151 Kill process 7151 gently; the process can catch the signal.

kill -SIGKILL 7151 Kill peremptorily; this signal cannot be caught.

kill 0 Kill all the background processes in this process group.

**SEE ALSO**

[ps\(1\)](#), [signal\(2\)](#), [signal\(2\)](#), [init\(8\)](#)

**NAME**

ksh – Korn shell, the not standard command programming language

**SYNOPSIS**

**ksh** [ **-acefhikmnrstuvx** ] [ **-o** option ] ... [ arg ... ]

**DESCRIPTION**

*Ksh* is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

**Definitions.**

A *metacharacter* is one of the following characters:

**; & ( ) | < > new-line space tab**

A *blank* is a **tab** or a **space**. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *aliases*, *functions*, and *named parameters*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

**Commands.**

A *simple-command* is a sequence of *blank* separated words which may be preceded by a parameter assignment list. (See *Environment* below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec(2)*). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal(2)* for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe(2)* to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **| |**, and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than that of **&&** and **| |**. The symbols **&&** and **| |** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent Shell using the **-p** option of the special commands **read** and **print** described later. Only one such command can be active at any given time. The symbol **&&** (**| |**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

**for** *identifier* [ **in** *word* ... ] **do** *list* **done**

Each time a **for** command is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

**select** *identifier* [ **in** *word* ... ] **do** *list* **done**

A **select** command prints on standard error (file descriptor 2), the set of *words*, each preceded by a number. If **in** *word* ... is omitted, then the positional parameters are used instead (see *Parameter Substitution* below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed **words**, then the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the parameter *identifier* is set to **null**. The contents of the line read from standard input is saved in the parameter **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered.

**case** *word* **in** [ *pattern* [ | *pattern* ] ... ] *list* ;; ] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* below).

**if** *list* **then** *list* [ **elif** *list* **then** *list* ] ... [ **else** *list* ] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

**while** *list* **do** *list* **done**

**until** *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the *list* is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{ *list* ; }

*list* is simply executed. Note that { is a *keyword* and requires a blank in order to be recognized.

**function** *identifier* { *list* ; }

*identifier* () { *list* ; }

Define a function which is referenced by *identifier*. The body of the function is the *list* of commands between { and }. (See *Functions* below).

**time** *pipeline*

The *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error.

The following keywords are only recognized as the first word of a command and when not quoted:

**if then else elif fi case esac for while until do done { } function select time**

### Comments.

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

### Aliasing.

The first word of each command is replaced by the text of an **alias** if an **alias** for this word has been defined. The first character of an **alias** name can be any printable character, but the rest of the characters must be the same as for a valid *identifier*. The replacement string can contain any valid Shell script including the metacharacters listed above. The first word of each command of the replaced text will not be tested for additional aliases. If the last character of the alias value is a *blank* then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the keywords listed above. Aliases can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported aliases remain in effect for sub-shells but must be reinitialized for separate invocations of the Shell (See *Invocation* below).

*Aliasing* is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the **alias** command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the identifier is read and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The **-h** option of the **set** command makes each command name which is an *identifier* into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```

echo='print - '
false='let 0'
functions='typeset -f'
history='fc -l'
integer='typeset -i'
nohup='nohup '
pwd='print - $PWD'
r='fc -e - '
true=':'
type='whence -v'
hash='alias -t'

```

### Tilde Substitution.

After alias substitution is performed, each word is checked to see if it begins with an unquoted `~`. If it does, then the word up to a `/` is checked to see if it matches a user name in the `/etc/passwd` file. If a match is found, the `~` and the matched login name is replaced by the login directory of the matched user. This is called a *tilde* substitution. If no match is found, the original text is left unchanged. A `~` by itself, or in front of a `/`, is replaced by the value of the `HOME` parameter. A `~` followed by a `+` or `-` is replaced by the value of the parameter `PWD` and `OLDPWD` respectively.

In addition, the value of each *keyword parameter* is checked to see if it begins with a `~` or if a `~` appears after a `:`. In either of these cases a *tilde* substitution is attempted.

### Command Substitution.

The standard output from a command enclosed in a pair of grave accents (```) may be used as part or all of a word; trailing new-lines are removed. The command substitution ``cat file`` can be replaced by the equivalent but faster ``file``. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

### Parameter Substitution.

A *parameter* is an *identifier*, a digit, or any of the characters `*`, `,`, `#`, `?`, `-`, `$`, and `!`. A *named parameter* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Named parameters* can be assigned *values* and *attributes* by using the `typeset` special command. The attributes supported by the Shell are described later with the `typeset` special command. Exported parameters pass values and attributes to sub-shells but only values to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a *subscript*. A *subscript* is denoted by a `[`, followed by an *arithmetic expression* (see Arithmetic evaluation below) followed by a `]`. The value of all subscripts must be in the range of 0 through 511. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the first element.

The *value* of a *named parameter* may also be assigned by writing:

```
name=value [ name=value ] ...
```

If the integer attribute, `-i`, is set for *name* the *value* is subject to arithmetic evaluation as described below. Positional parameters, parameters denoted by a number, may be assigned values with the `set` special command. Parameter `$0` is set from argument zero when the shell is invoked.

The character `$` is used to introduce substitutable *parameters*.

`${parameter}`

The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is a digit then it is a positional parameter. If *parameter* is `*` or `,`, then all the positional parameters, starting with `$1`, are substituted (separated by spaces). If an array *identifier* with subscript `*` or `,` is used, then the value for each of the elements is substituted (separated by spaces).

`${#parameter}`

If *parameter* is not `*`, the length of the value of the *parameter* is substituted. Otherwise, the number of positional parameters is substituted.

**\${#identifier[\*]}**

The number of elements in the array *identifier* is substituted.

**\${parameter:-word}**

If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

**\${parameter:=word}**

If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

**\${parameter:?word}**

If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

**\${parameter:+word}**

If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

**\${parameter#pattern}**

**\${parameter##pattern}**

If the Shell *pattern* matches the beginning of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise the value of this *parameter* is substituted. In the first form the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

**\${parameter%pattern}**

**\${parameter%%pattern}**

If the Shell *pattern* matches the end of the value of *parameter*, then the value of *parameter* with the matched part deleted; otherwise substitute the value of *parameter*. In the first form the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd` }
```

If the colon ( : ) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- #** The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
- ?** The decimal value returned by the last executed command.
- \$** The process number of this shell.
- \_** The last argument of the previous command. This parameter is not set for commands which are asynchronous.
- !** The process number of the last background command invoked.
- PPID** The process number of the parent of the shell.
- PWD** The present working directory set by the **cd** command.
- OLDPWD**  
The previous working directory set by the **cd** command.
- RANDOM**  
Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.
- REPLY**  
This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.

The following parameters are used by the shell:

**CDPATH**

The search path for the **cd** command.

**COLUMNS**

If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.

**EDITOR**

If the value of this variable ends in *emacs*, *gmacs*, or *vi* and the **VISUAL** variable is not set, then the corresponding option (see Special Command **set** below) will be turned on.

**ENV** If this parameter is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the *shell* is invoked. (See *Invocation* below.) This file is typically used for *alias* and *function* definitions.

**FCEDIT**

The default editor name for the **fc** command.

**IFS** Internal field separators, normally **space**, **tab**, and **new-line** that is used to separate command words which result from command or parameter substitution and for separating words with the special command **read**.

**HISTFILE**

If this parameter is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See *Command re-entry* below.)

**HISTSIZE**

If this parameter is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

**HOME**

The default argument (home directory) for the **cd** command.

**MAIL** If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, then the shell informs the user of arrival of mail in the specified file.

**MAILCHECK**

This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds. If set to 0, the shell will check before each prompt.

**MAILPATH**

A colon ( : ) separated list of file names. If this parameter is set then the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a ? and a message that will be printed. The message will undergo parameter and command substitution with the parameter, **\$\_** defined as the name of the file that has changed. The default message is *you have mail in \$\_*.

**PATH** The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh* (except in *.profile* ).

**PS1** The value of this parameter is expanded for parameter substitution to define the primary prompt string which by default is "\$ ". The character ! in the primary prompt string is replaced by the *command* number (see *Command Re-entry* below).

**PS2** Secondary prompt string, by default "> ".

**PS3** Selection prompt string used within a **select** loop, by default "#? ".

**SHELL**

The pathname of the *shell* is kept in the environment. At invocation, if the value of this variable contains an **r** in the basename, then the shell becomes restricted.

**TMOUT**

If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)

**VISUAL**

If the value of this variable ends in *emacs*, *gmacs*, or *vi* then the corresponding option (see Special Command **set** below) will be turned on.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, **TMOUT** and **IFS**, while **HOME**, **SHELL**, **ENV** and **MAIL** are not set at all by the shell (although **HOME** is set by *login(1)*). On some systems **MAIL** and **SHELL** are also set by *login(1)*.

**Blank Interpretation.**

After parameter and command substitution, the results of substitutions are scanned for the field separator characters ( those found in **IFS** ) and split into distinct arguments where such characters are found. Explicit null arguments ( "" or **(fm)(fm)** are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.



**File Name Generation.**

Following substitution, each command *word* is scanned for the characters \*, ?, and [ unless the **-f** option has been **set**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly. In other instances of pattern matching the / and . are not treated specially.

- \* Matches any string, including the null string.
- ? Matches any single character.
- [... ] Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening "[ " is a "!" then any character not enclosed is matched. A – can be included in the character set by putting it as the first or last character.

**Quoting.**

Each of the *metacharacters* listed above (See *Definitions* above). has a special meaning to the shell and cause termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ', ", and \$. "\$\*" is equivalent to "\$1 \$2 ...", whereas "\$ " is equivalent to "\$1" "\$2" ....

The special meaning of keywords can be removed by quoting any character of the keyword. The recognition of special command names listed below cannot be altered by quoting them.

**Arithmetic Evaluation.**

An ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using *long* arithmetic. Constants are of the form [*base#*] *n* where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted then base 10 is used.

An internal integer representation of a *named parameter* can be specified with the **-i** option of the **typeset** special command. When this attribute is selected the first assignment to the parameter determines the arithmetic base to be used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command which begins with a ((, all the characters until a matching )) are treated as a quoted expression. More precisely, ((... )) is equivalent to **let** " ...".

**Prompting.**

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

**Input/Output.**

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

- <*word* Use file *word* as standard input (file descriptor 0).
- >*word* Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
- >>*word* Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
- <<[–]*word* The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of

the document; otherwise, parameter and command substitution occurs, **\new-line** is ignored, and **\** must be used to quote the characters **\**, **\$**, **^**, and the first character of *word*. If **-** is appended to **<<**, then all leading tabs are stripped from *word* and from the document.

**<&digit** The standard input is duplicated from file descriptor *digit* (see [dup\(2\)](#)). Similarly for the standard output using **>& digit**.

**<&-** The standard input is closed. Similarly for the standard output using **>&-**.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

```
... 1>fname 2>&1
```

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by **&** and job control is not active, then the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

### Environment.

The *environment* (see [environ\(7\)](#)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value and marking it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the **export** or **typeset -x** commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in **export** or **typeset -x** commands.

The environment for any *simple-command* or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word of the form *identifier=value*. Thus:

```
TERM=450 cmd args                                and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* parameter assignment arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

### Functions.

The **function** keyword, described in the *Commands* section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See *Execution* below).

Functions execute in the same process as the caller and share all files, traps (other than **EXIT** and **ERR**) and present working directory with the caller. A trap set on **EXIT** inside a function is executed after the function completes. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **-f** option of the **typeset** special command. The text of functions will also be listed. Function can be undefined with the **-f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **-xf** option of the **typeset** command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be placed in the **ENV** file.

### Jobs.

If the **monitor** option of the **set** command is turned on, an interactive shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

This paragraph and the next require features that are not in all versions of UNIX and may not apply. If you are running a job and wish to do something else you may hit the key **^Z** (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been ‘Stopped’, and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A **^Z** takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command “stty tostop”. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character **%** introduces a job name. If you wish to refer to job number 1, you can name it as **%1**. Jobs can also be named by prefixes of the string typed in to **kill** or restart them. Thus, on systems that support job control, **fg %ed** would normally restart a suspended *ed(1)* job, if there were a suspended job whose name began with the string ‘ed’.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a -. The abbreviation **%+** refers to the current job and **%-** refers to the previous job. **%%** is also a synonym for the current job.

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

When you try to leave the shell while jobs are running or stopped, you will be warned that ‘You have stopped(running) jobs.’ You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

### Signals.

The INT and QUIT signals for an invoked command are ignored if the command is followed by **&** and job **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

### Execution.

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a **return**, the positional parameter list is restored and any trap set on **EXIT** within the function is executed. The value of a *function* is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a *special command* or a user defined *function*, a process is created and an attempt is made to execute the command via *exec(2)*.

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between colon delimiters, or at the end of the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and named parameters are removed in this case. A parenthesized command is also executed in a sub-shell.

### Command Re-entry.

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **\$HOME/.history** is used if the **HISTFILE** variable is not set or is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to **fc** then the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined then **/bin/ed** is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name **-** is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if **r** is aliased to **'fc -e -'** then typing **'r bad=good c'** will re-execute the most recent command which starts with the letter **c**, replacing the string **bad** with the string **good**.

### In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a new-line ('RETURN' or 'LINE FEED'). If either the *emacs*, *gmacs*, or *vi* option is active, the user can edit the command line. To be in either of these edit modes **set** the corresponding option. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept 'RETURN' as carriage return without line feed and that a space (' ') must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to 'space'. Hewlett-Packard series 2621 terminal users should set the straps to 'bcGHxZ etX'.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it is defined, otherwise 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, \*) if the line extends on the right (left, both) side(s) of the window.

### Emacs Editing Mode

This mode is entered by enabling either the *emacs* or *gmacs* option. The only difference between these two modes is the way they handle **^T**. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, **^F** is the notation for control **F**. This is entered by depressing 'f' while holding down the 'CTRL' (control) key. The 'SHIFT' key is *not* depressed. (The notation **^?** indicates the DEL (delete) key.)

The notation for escape sequences is **M-** followed by a character. For example, **M-f** (pronounced Meta f) is entered by depressing ESC (ascii **033**) followed by 'f'. (**M-F** would be the notation for ESC followed by 'SHIFT' (capital) 'F'.)

All edit commands operate from any place on the line (not just at the beginning). Neither the "RETURN" nor the "LINE FEED" key is entered after edit commands except when noted.

**^F** Move cursor forward (right) one character.  
**M-f** Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)

<b>^B</b>	Move cursor backward (left) one character.
<b>M-b</b>	Move cursor backward one word.
<b>^A</b>	Move cursor to start of line.
<b>^E</b>	Move cursor to end of line.
<b>^]char</b>	Move cursor to character <i>char</i> on current line.
<b>^X^X</b>	Interchange the cursor and mark.
<i>erase</i>	(User defined erase character as defined by the stty command, usually <b>^H</b> or #.) Delete previous character.
<b>^D</b>	Delete current character.
<b>M-d</b>	Delete current word.
<b>M-^H</b>	(Meta-backspace) Delete previous word.
<b>M-h</b>	Delete previous word.
<b>M-^?</b>	(Meta-DEL) Delete previous word (if your interrupt character is <b>^?</b> (DEL, the default) then this command will not work).
<b>^T</b>	Transpose current character with next character in <i>emacs</i> mode. Transpose two previous characters in <i>gmacs</i> mode.
<b>^C</b>	Capitalize current character.
<b>M-C</b>	Capitalize current word.
<b>^K</b>	Kill from the cursor to the end of the line. If given a parameter of zero then kill from the start of line to the cursor.
<b>^W</b>	Kill from the cursor to the mark.
<b>M-p</b>	Push the region from the cursor to the mark on the stack.
<i>kill</i>	(User defined kill character as defined by the stty command, usually <b>^G</b> or .) Kill the entire current line. If two <i>kill</i> characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
<b>^Y</b>	Restore last item removed from line. (Yank item back to the line.)
<b>^L</b>	Line feed and print current line.
<b>^</b>	(Null character) Set mark.
<b>M-</b>	(Meta space) Set mark.
<b>^J</b>	(New line) Execute the current line.
<b>^M</b>	(Return) Execute the current line.
<i>eof</i>	End-of-file character, normally <b>^D</b> , will terminate the shell if the current line is null.
<b>^P</b>	Fetch previous command. Each time <b>^P</b> is entered the previous command back in time is accessed.
<b>M-&lt;</b>	Fetch the least recent (oldest) history line.
<b>M-&gt;</b>	Fetch the most recent (youngest) history line.
<b>^N</b>	Fetch next command. Each time <b>^N</b> is entered the next command forward in time is accessed.
<b>^Rstring</b>	Reverse search history for a previous command line containing <i>string</i> . If a parameter of zero is given the search is forward. <i>String</i> is terminated by a "RETURN" or "NEW LINE".
<b>^O</b>	Operate – Execute the current line and fetch the next line relative to current line from the history file.
<b>M-digits</b>	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are <b>^F</b> , <b>^B</b> , <i>erase</i> , <b>^D</b> , <b>^K</b> , <b>^R</b> , <b>^P</b> and <b>^N</b> .
<b>M-letter</b>	Soft-key – Your alias list is searched for an alias by the name <i>_letter</i> and if an alias of this name is defined, its value will be inserted on the line. The <i>letter</i> must not be one of the above meta-functions.
<b>M-_</b>	The last parameter of the previous command is inserted on the line.
<b>M-.</b>	The last parameter of the previous command is inserted on the line.
<b>M-*</b>	Attempt file name generation on the current word.
<b>^U</b>	Multiply parameter of next command by 4.
<b>\</b>	Escape next character. Editing characters, the user's erase, kill and interrupt (normally <b>^?</b> ) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
<b>^V</b>	Display version of the shell.

### Vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC ( **033** ) and moves the cursor to the point needing correction and

then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command.

When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option **viraw** is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

### Input Edit Commands

By default the editor is in input mode.

<i>erase</i>	(User defined erase character as defined by the stty command, usually <b>^H</b> or <b>#</b> .) Delete previous character.
<b>^W</b>	Delete the previous blank separated word.
<b>^D</b>	Terminate the shell.
<b>^V</b>	Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a <b>^V</b> . The <b>^V</b> removes the next character's editing features (if any).
<b>\</b>	Escape the next <i>erase</i> or <i>kill</i> character.

### Motion Edit Commands

These commands will move the cursor.

<i>[count]</i> <b>l</b>	Cursor forward (right) one character.
<i>[count]</i> <b>w</b>	Cursor forward one alpha-numeric word.
<i>[count]</i> <b>W</b>	Cursor to the beginning of the next word that follows a blank.
<i>[count]</i> <b>e</b>	Cursor to end of word.
<i>[count]</i> <b>E</b>	Cursor to end of the current blank delimited word.
<i>[count]</i> <b>h</b>	Cursor backward (left) one character.
<i>[count]</i> <b>b</b>	Cursor backward one word.
<i>[count]</i> <b>B</b>	Cursor to preceding blank separated word.
<i>[count]</i> <b>fc</b>	Find the next character <i>c</i> in the current line.
<i>[count]</i> <b>Fc</b>	Find the previous character <i>c</i> in the current line.
<i>[count]</i> <b>tc</b>	Equivalent to <b>f</b> followed by <b>h</b> .
<i>[count]</i> <b>Tc</b>	Equivalent to <b>F</b> followed by <b>l</b> .
<b>;</b>	Repeats the last single character find command, <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> .
<b>,</b>	Reverses the last single character find command.
<b>0</b>	Cursor to start of line.
<b>^</b>	Cursor to first non-blank character in line.
<b>\$</b>	Cursor to end of line.

### Search Edit Commands

These commands access your command history.

<i>[count]</i> <b>k</b>	Fetch previous command. Each time <b>k</b> is entered the previous command back in time is accessed.
<i>[count]</i> <b>-</b>	Equivalent to <b>k</b> .
<i>[count]</i> <b>j</b>	Fetch next command. Each time <b>j</b> is entered the next command forward in time is accessed.
<i>[count]</i> <b>+</b>	Equivalent to <b>j</b> .
<i>[count]</i> <b>G</b>	The command number <i>count</i> is fetched. The default is the least recent history command.
<i>/string</i>	Search backward through history for a previous command containing <i>string</i> . <i>String</i> is terminated by a "RETURN" or "NEW LINE". If <i>string</i> is null the previous string will be used.
<i>?string</i>	Same as <i>/</i> except that search will be in the forward direction.
<b>n</b>	Search for next match of the last pattern to <i>/</i> or <i>?</i> commands.

**N** Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the *string* entered by the previous / command.

### Text Modification Edit Commands

These commands will modify the line.

**a** Enter input mode and enter text after the current character.

**A** Append text to the end of the line. Equivalent to **\$a**.

**[count]c***motion*

**c***[count]motion*

Delete current character through the character *motion* moves the cursor to and enter input mode. If *motion* is **c**, the entire line will be deleted and input mode entered.

**C** Delete the current character through the end of line and enter input mode. Equivalent to **c\$**.

**S** Equivalent to **cc**.

**D** Delete the current character through the end of line.

**[count]d***motion*

**d***[count]motion*

Delete current character through the character *motion* moves the cursor to. Equivalent to **d\$**. If *motion* is **d**, the entire line will be deleted.

**i** Enter input mode and insert text before the current character.

**I** Insert text before the beginning of the line. Equivalent to the two character sequence **^i**.

**[count]P** Place the previous text modification before the cursor.

**[count]p** Place the previous text modification after the cursor.

**R** Enter input mode and replace characters on the screen with characters you type overlay fashion.

**rc** Replace the current character with *c*.

**[count]x** Delete current character.

**[count]X** Delete preceding character.

**[count].** Repeat the previous text modification command.

**~** Invert the case of the current character and advance the cursor.

**[count]\_** Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.

**\*** Causes an **\*** to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

### Other Edit Commands

Miscellaneous commands.

**u** Undo the last text modifying command.

**U** Undo all the text modifying commands performed on the line.

**[count]v** Returns the command **fc -e \${VISUAL:-\${EDITOR:-vi}}** *count* in the input buffer. If *count* is omitted, then the current line is used.

**^L** Line feed and print current line. Has effect only in control mode.

**^J** (New line) Execute the current line, regardless of mode.

**^M** (Return) Execute the current line, regardless of mode.

### Equivalent to

**I#<cr>**. Useful for causing the current line to be inserted in the history without being executed.

### Special Commands.

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. File descriptor 1 is the default output location. Parameter assignment lists preceding the command do not remain in effect when the command completes unless noted.

**:[ arg ... ]**

Parameter assignments remain in effect after the command completes. The command only expands parameters. A zero exit code is returned.

**.file** [ *arg* ... ]

Parameter assignments remain in effect after the command completes. Read and execute commands from *file* and return. The commands are executed in the current Shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged.

**alias** [ **-tx** ] [ *name*[ =*value* ] ... ]

*Alias* with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **-t** flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **-t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **-x** flag is used to set or print exported aliases. An exported alias is defined across sub-shell environments. *Alias* returns true unless a *name* is given for which no alias has been defined.

**bg** [ %*job* ]

This command is only built-in on systems that support job control. Puts the specified *job* into the background. The current job is put in the background if *job* is not specified.

**break** [ *n* ]

Exit from the enclosing **for while until** or **select** loop, if any. If *n* is specified then break *n* levels.

**continue** [ *n* ]

Resume the next iteration of the enclosing **for while until** or **select** loop. If *n* is specified then resume at the *n*-th enclosing loop.

**cd** [ *arg* ]**cd** *old new*

This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is **-** the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / then the search path is not used. Otherwise, each directory in the path is searched for *arg*.

The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

The **cd** command may not be executed by *rsh*.

**eval** [ *arg* ... ]

The arguments are read as input to the shell and the resulting command(s) executed.

**exec** [ *arg* ... ]

Parameter assignments remain in effect after the command completes. If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

**exit** [ *n* ]

Causes the shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed. An end-of-file will also cause the shell to exit except for a shell which has the *ignoreeof* option (See **set** below) turned on.



**export** [ *name* ... ]

The given *names* are marked for automatic export to the *environment* of subsequently-executed commands.

**fc** [ **-e** *ename* ] [ **-nlr** ] [ *first* ] [ *last* ]

**fc -e -** [ *old=new* ] [ *command* ]

In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag **-l**, is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the parameter **FCEDIT** (default `/bin/ed`) is used as the editor. When editing is complete, the edited command(s) is executed. *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and `-16` for listing. The flag **-r** reverses the order of the commands and the flag **-n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old=new* is performed.

**fg** [ *%job* ]

This command is only built-in on systems that support job control. If *job* is specified it brings it to the foreground. Otherwise, the current job is brought into the foreground.

**jobs** [ **-l** ]

Lists the active jobs; given the **-l** options lists process id's in addition to the normal information.

**kill** [ **-sig** ] *process* ...

Sends either the **TERM** (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in `/usr/include/signal.h`, stripped of the prefix "SIG"). The signal names are listed by **kill -l**. There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is **TERM** (terminate) or **HUP** (hangup), then the job or process will be sent a **CONT** (continue) signal if it is stopped. The argument *process* can be either a process id or a job.

**let** *arg* ...

Each *arg* is an *arithmetic expression* to be evaluated. All calculations are done as long integers and no check for overflow is performed. Expressions consist of constants, named parameters, and operators. The following set of operators, listed in order of decreasing precedence, have been implemented:

-	unary minus
!	logical negation
* / %	multiplication, division, remainder
+ -	addition, subtraction
<= >= < >	comparison
== !=	equality inequality
=	arithmetic replacement

Sub-expressions in parentheses ( ) are evaluated first and can be used to override the above precedence rules. The evaluation within a precedence group is from right to left for the = operator and from left to right for the others.

A parameter name must be a valid *identifier*. When a parameter is encountered, the value associated with the parameter name is substituted and expression evaluation resumes. Up to nine levels of recursion are permitted.

The return code is 0 if the value of the last expression is non-zero, and 1 otherwise.

**newgrp** [ *arg* ... ]

Equivalent to **exec newgrp** *arg* ....

**print** [ **-Rnrpsu**[ *n* ] ] [ *arg* ... ]

The shell output mechanism. With no flags or with flag **-**, the arguments are printed on standard output as described by *echo(1)*. In raw mode, **-R** or **-r**, the escape conventions of *echo* are ignored. The **-R** option will print all subsequent arguments and options other than **-n**. The **-p** option causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output. The **-s** option causes the arguments to be written onto the history file instead of standard output. The **-u** flag can be used to specify a one digit file descriptor unit number **n** on which the output will be placed. The default is 1. If the flag **-n** is used, no **new-line** is added to the output.

**read** [ **-prsu**[ *n* ] ] [ *name?prompt* ] [ *name* ... ]

The shell input mechanism. One line is read and is broken up into words using the characters in **IFS** as separators. In raw mode, **-r**, a **\** at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with left-over words assigned to the last *name*. The **-p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **-s** flag is present, the input will be saved as a command in the history file. The flag **-u** can be used to specify a one digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted then **REPLY** is used as the default *name*. The return code is 0 unless an end-of-file is encountered. An end-of-file with the **-p** option causes cleanup for this process so that another can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device then the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is 0 unless an end-of-file is encountered.

**readonly** [ *name* ... ]

The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

**return** [ *n* ]

Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted then the return status is that of the last command executed. If **return** is invoked while not in a *function* then it is the same as an **exit**.

**set** [ **-aefhkmnostuvx** ] [ **-o** *option* ... ] [ *arg* ... ]

The flags for this command have meaning as follows:

- a** All subsequent parameters that are defined are automatically exported.
- e** If the shell is non-interactive and if a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.
- f** Disables file name generation.
- h** Each command whose name is an *identifier* becomes a tracked alias when first encountered.
- k** All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.
- m** Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.
- n** Read commands but do not execute them.
- o** The following argument can be one of the following option names:
  - allexport** Same as **-a**.
  - errexit** Same as **-e**.
  - emacs** Puts you in an *emacs* style in-line editor for command entry.
  - gmacs** Puts you in a *gmacs* style in-line editor for command entry.
  - ignoreeof** The shell will not exit on end-of-file. The command **exit** must be used.
  - keyword** Same as **-k**.

**markdirs**

All directory names resulting from file name generation have a trailing / appended.

**monitor**

Same as **-m**.

**noexec** Same as **-n**.

**noglob** Same as **-f**.

**nounset**

Same as **-u**.

**verbose**

Same as **-v**.

**trackall**

Same as **-h**.

**vi** Puts you in insert mode of a *vi* style in-line editor until you hit escape character **033**. This puts you in move mode. A return sends the line.

**viraw** Each character is processed as it is typed in *vi* mode.

**xtrace** Same as **-x**.

If no option name is supplied then the current option settings are printed.

**-s** Sort the positional parameters.

**-t** Exit after reading and executing one command.

**-u** Treat unset parameters as an error when substituting.

**-v** Print shell input lines as they are read.

**-x** Print commands and their arguments as they are executed.

**-** Turns off **-x** and **-v** flags and stops examining arguments for flags.

**--** Do not change any of the flags; useful in setting **\$1** to a value beginning with **-**. If no arguments follow this flag then the positional parameters are unset.

Using **+** rather than **-** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **\$-**. The remaining arguments are positional parameters and are assigned, in order, to **\$1**, **\$2**, ... If no arguments are given then the values of all names are printed on the standard output.

**shift** [ *n* ]

The positional parameters from **\$n+1** ... are renamed **\$1** ... , default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **\$#**.

**test** [ *expr* ]

Evaluate conditional expression *expr*. See [test\(1\)](#) for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional primitive expressions are allowed:

**-L file**

True if *file* is a symbolic link.

*file1* **-nt** *file2*

True if *file1* is newer than *file2*.

*file1* **-ot** *file2*

True if *file1* is older than *file2*.

*file1* **-ef** *file2*

True if *file1* has the same device and i-node number as *file2*.

**times**

Print the accumulated user and system times for the shell and for processes run from the shell.

**trap** [ *arg* ] [ *sig* ] ...

*arg* is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is omitted or is **-**, then all trap(s) *sig* are reset to their original values. If *arg* is the null string then this signal is ignored

by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit code. This trap is not inherited by functions. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

**typeset** [ **-FLRZefilprtux**[ *n* ] [ *name*[ *=value* ] ] ... ]

Parameter assignments remain in effect after the command completes. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

- F** This flag provides UNIX to host-name file mapping on non-UNIX machines.
- L** Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. When the parameter is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the **-Z** flag is also set. The **-R** flag is turned off.
- R** Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the parameter is reassigned. The **L** flag is turned off.
- Z** Right justify and fill with leading zeros if the first non-blank character is a digit and the **-L** flag has not been set. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment.
- e** Tag the parameter as having an error. This tag is currently unused by the shell and can be set or cleared by the user.
- f** The names refer to function names rather than parameter names. No assignments can be made and the only other valid flag is **-x**.
- i** Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base, otherwise the first assignment determines the output base.
- l** All upper-case characters converted to lower-case. The upper-case flag, **-u** is turned off.
- p** The output of this command, if any, is written onto the two-way pipe
- r** The given *names* are marked readonly and these names cannot be changed by subsequent assignment.
- t** Tags the named parameters. Tags are user definable and have no special meaning to the shell.
- u** All lower-case characters are converted to upper-case characters. The lower-case flag, **-l** is turned off.
- x** The given *names* are marked for automatic export to the *environment* of subsequently-executed commands.

Using **+** rather than **-** causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values* ) of the *parameters* which have these flags set is printed. (Using **+** rather than **-** keeps the values to be printed.) If no *names* and flags are given, the *names* and *attributes* of all *parameters* are printed.

**ulimit** [ **-cdfmpt** ] [ *n* ]

- c** imposes a size limit of *n* blocks on the size of core dumps (BSD only).
- d** imposes a size limit of *n* blocks on the size of the data area (BSD only).
- f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read).
- m** imposes a soft limit of *n* blocks on the size of physical memory (BSD only).
- p** changes the pipe size to *n* (UNIX/RT only).
- t** imposes a time limit of *n* seconds to be used by each process (BSD only).

If no option is given, **-f** is assumed. If *n* is not given the current limit is printed.

**umask** [ *nnn* ]

The user file-creation mask is set to *nnn* (see [umask\(2\)](#)). If *nnn* is omitted, the current value of the mask is printed.

**unalias** *name* ...

The parameters given by the list of *names* are removed from the *alias* list.

**unset** [ **-f** ] *name* ...

The parameters given by the list of *names* are unassigned, i. e., their values and attributes are erased. Readonly variables cannot be unset. If the flag, **-f**, is set, then the names refer to *function* names.

**wait** [ *n* ]

Wait for the specified process and report its termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

**whence** [ **-v** ] *name* ...

For each *name*, indicate how it would be interpreted if used as a command name.

The flag, **-v**, produces a more verbose report.

**Invocation.**

If the shell is invoked by [exec\(2\)](#), and the first character of argument zero (**\$0**) is **-**, then the shell is assumed to be a *login* shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **\$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV** if the file exists. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

- c** *string* If the **-c** flag is present then commands are read from *string*.
- s** If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Shell output, except for the output of some of the *Special commands* listed above, is written to file descriptor 2.
- i** If the **-i** flag is present or if the shell input and output are attached to a terminal (as told by [gty\(2\)](#)) then this shell is *interactive*. In this case **TERMINATE** is ignored (so that **kill 0** does not kill an interactive shell) and **INTERRUPT** is caught and ignored (so that **wait** is interruptible). In all cases, **QUIT** is ignored by the shell.
- r** If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

**Rsh Only.**

*Rsh* is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

- changing directory (see [cd\(1\)](#)),
- setting the value of **SHELL** or **PATH**,
- specifying path or command names containing */*,
- redirecting output (**>** and **>>**).

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

## EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

## FILES

/etc/passwd  
/etc/profile  
\$HOME/.profile  
/tmp/sh\*  
/dev/null

## SEE ALSO

cat(1), cd(1), echo(1), emacs(1), env(1), gmacs(1), newgrp(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), gtty(2), pipe(2), signal(2), umask(2), ulimit(2), wait(2), rand(3), a.out(5), profile(5), environ(7).

## CAVEATS

If a command which is a *tracked alias* is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **-t** option of the **alias** command to correct this situation

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Some very old shell scripts contain a **^** as a synonym for the pipe character **|**.

**NAME**

lab – label maker

**SYNOPSIS**

**lab** [ **-m** ] [ *file ...* ]

**DESCRIPTION**

*Lab* causes the *files* to be queued for printing as mailing labels. If no files are named, the standard input is read. The option **-m** causes notification via [mail\(1\)](#) to be sent when the job completes.

The last line of each label is identified by ending it with one or more spaces and a hyphen. The hyphen must be the last character in the line; no white space may follow it.

**FILES**

/usr/spool/lab/\*  
spool area

/usr/lib/lab  
printer daemon

**SEE ALSO**

[pr\(1\)](#)

**BUGS**

Queued jobs print in directory (seemingly random) order.

**NAME**

altran, cospan, esterel, icon, lisp, macsyma, maple, ops5, pascal, ratfor, S, smp, sno, spitbol, struct, twig – languages

**SYNOPSIS**

**altran** [ *option ...* ] *file ...*  
**cospan** [ *option ...* ] *file*  
**esterel**  
**iconc**  
**icont**  
**lisp**  
**/usr/lbin/macsyma**  
**maple**  
**/usr/lbin/ops5**  
**ratfor** [ *option ...* ] *file ...*  
**S**  
**smp**  
**sno** [ *file ...* ]  
**spitbol** [ *option ...* ] *file ...*  
**struct** [ *option ...* ] *file ...*  
**twig** [ *-wxx* ] *file*

**DESCRIPTION**

*Altran*, a language for rational algebra, is described in W. S. Brown, *ALTRAN User's Manual*. For more information execute `man altran`.

*Cospan*, a system which analyzes concurrent programs written in the data-flow language S/R for properties defined by automata, is described in Z. Har'El and R. P. Kurshan, *COSPAN User's Guide*, 1121-871009-21TM, AT&T Bell Laboratories, 1987. For more information, execute `man cospan`.

*Esterel* compiles single-process implementations of programs expressed in terms of asynchronously cooperating automata. For more information, execute `man esterel`.

*Icon*, a general-purpose language with stream-based coroutines is described in R. E. Griswold, *The Icon Programming Language*, Prentice-Hall, 1983. For more information, execute `man icont`.

*Lisp*, the symbol manipulation language, is described in J. K. Foderara, 'The Franz Lisp Manual', in *Unix Programmer's Manual*, Seventh Edition, Virtual VAX-11 Version, 1980, Volume 2C (Berkeley)

*Macsyma*, another symbolic algebra language, is described in *Macsyma Reference Manual*, Laboratory for Computer Science, MIT, 1977. It breaks if the environment contains shell functions.

*Maple*, a third symbolic algebra language, is described in K. O. Geddes, G. H. Gonnet, and B. W. Char, *MAPLE User's Manual, Third Edition*, Research Report CS-83-41 Dept. of Computer Science, University of Waterloo, 1983. For more information execute `man maple`.

*Ops5* is a production-system interpreter described in C. L. Forgy *OPSS User's Manual*, Department of Computer Science, Carnegie-Mellon University, July, 1981. For more information execute `man ops5`.

*Pascal* is an interpreter and *pc* is a compiler for the well known language. For more information, type `man pascal pc pxp`.

*Ratfor* accepts Fortran extended with C-like control constructs and compiles into Fortran. For more information execute `man ratfor`. For a full description see B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

*S*, a system for data analysis and graphics, is described in R. A. Becker, J. M. Chambers, and A. R. Wilks, *The New S Language*, Wadsworth, 1988.

*Smp*, a fourth symbolic algebra language, is described in C. Cole and S. Wolfram, *SMP Handbook*, California Institute of Technology, 1981.

*Sno* is a dialect of Snobol 3. For more information execute `man sno`.

*Spitbol* is a dialect of Snobol 4. For more information execute `man spitbol`. The full story is in R. B. K. Dewar, A. P. McCann, R. E. Goldberg, and S. G. Duff, *Macro SPITBOL Program Reference Manual*,



and R. E. Griswold, J. F. Poage, and I. P. Polonsky, *The SNOBOL Programming Language* Academic Press, 1968.

*Struct*, an inverse of *ratfor*, deduces rational control structure from pure Fortran. For more information execute `man struct`.

*Twig* is a language for tree manipulation, useful for code generation and other applications. For more information execute `man twig`. See also S. W. K. Tjiang, *Twig Reference Manual*, AT&T Bell Laboratories Computing Science Technical Report 120.

## **BUGS**

These language processors are available on a haphazard collection of machines. Many are maintained in the same spirit.

**NAME**

lcoc – Esterel compiler

**SYNOPSIS**

**lcoc** [ option ] ... [ file ]...

**DESCRIPTION**

*lcoc* is the Esterel v3 compiler. It produce an *oc* format output from one or more *lc* format inputs (or *ic* if no Esterel **copymodule** instruction is used). If there is no input files, the standard input is used. *ic* format inputs describes Esterel **modules** to be processed, and *oc* format output describes the computed automata. Typical use is:

```
lcoc < game1.ic > game1.oc
```

or

```
lcoc game1.ic > game1.oc
```

The following options are interpreted by *lcoc*.

- version** Gives the version name and terminates ignoring all others arguments.
- v** Verbose option: gives names of the modules compiled.
- stat** Prints statistic informations into the standard error stream: parsing and compiling times and size of the process.
- size** Prints size informations into the standard error stream: how many states, actions and action calls are produced.
- memstat** Memory state after compiling.
- W** Give warnings about .ic unused actions and .ic dead code.
- show** During the compiling process, gives two informations into the standard error stream: how many states are already created and how many states are already analysed.

**FILES**

The caller of the command must have read/write permission for the directories containing the working files, and execute permission for the *lcoc* file itself.

**DIAGNOSTICS**

The diagnostics produced by *lcoc* compiler are intended to be self-explanatory.

**IDENTIFICATION**

Author: F. Boussinot, CMA, Ecole des Mines de Paris,

Sophia-Antipolis, 06600 Valbonne, FRANCE

Revision Number: \$Revision: 1.3 \$ ; Release Date: \$Date: 88/06/30 12:08:10 \$ .

**SEE ALSO**

Esterel v3 Programming Language Manual

Esterel v3 System Manuals.

strlic (1), iclc(1), ocl (1).

**BUGS**

**NAME**

`lcomp`, `lprint` – line-by-line profiler

**SYNOPSIS**

`lcomp` [ *option ...* ] *file ...*

`lprint` [ *option* ] [ *file ...* ]

**DESCRIPTION**

`Lcomp` is used in place of `cc(1)` or `f77(1)` to insert instruction-counting code into programs. It shares with those commands options whose initial letters are taken from the string `cwpDUIRd1Nnz`, and accepts files whose names end in `.c`, `.f`, `.s`, or `.o`. From each source file it derives a `.o` file and a `.sL` file which `lprint` uses to correlate source lines with basic blocks.

Option `-C` declares that `.c` files (and `.o` files, if no source files are named) are C++ files. If the `-c` option is not present `lcomp` creates Each time a `.out` is run statistics are added to a profiling file

`Lprint` produces on the standard output a listing (in the style of `pr(1)`) of the programs compiled by `lcomp`. Without arguments or files, each line of the listing is preceded by the number of times it was executed, as determined from the data in `Lprint` interprets the following options.

- a** Detailed listing of every machine instruction and how often it was executed.
- b** How often each basic block was executed.
- c** Compress the `prof.out` file, which otherwise grows with every execution of a `.out`.
- f** Print summary information by function: instruction executions, number of invocations, source instructions, and number of instructions never executed.
- i** Before each line of source print the number of machine instructions executed.
- p** Before each line of source print the number of times the first basic block in that line was executed.
- s** Summarize the counts by source file: instruction executions, source instructions, instructions never executed, basic block executions, total number of source basic blocks, and how many were never executed.

If any file names are given, the options `abip` apply only to them. If no options are given, **-p** is assumed. Any combination of options is allowed.

**FILES**

`prof.out`  
counts

\*`.sL` for correlating with source

`/usr/lib/bb`  
for finding basic blocks and inserting counting code

`/usr/lib/nexit.o`  
for printing counts when a `.out` exits

**SEE ALSO**

`cc(1)`, `f77(1)`, `prof(1)`

**BUGS**

A line in the source file may be in zero, one, or more basic blocks; the count given in the listing corresponds to some particular choice of the basic block to associate with the line.

Processing the output of `yacc(1)` without removing `#line` directives will produce unsatisfactory results.

Option `-C` masks an option of `cc(1)`.

**NAME**

ld – link editor or loader

**SYNOPSIS**

ld [ *option ...* ] *file ...*

**DESCRIPTION**

*Ld* combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of *ld* is left on This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine (unless the **-e** option is specified).

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib* (see [ar\(1\)](#)), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important; see [lorder\(1\)](#). The first member of a library should be a file named **\_\_SYMDEF**, which is understood to be a dictionary for the library as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols `_etext`, `_edata`, and `_end` `edata`, and `end` in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data, respectively. It is erroneous to define these symbols.

*Ld* understands several options. Except for **-I** and **-o**, they should appear before the file names.

- A** Load incrementally, so that the resulting object code may be read into an already executing program. The next argument names an object file whose symbol table will be added to. Only newly linked material will be entered into the text and data portions of but the new symbol table will reflect every symbol defined before and after the incremental load. **-A** must not follow any object file names.
- D** Take the next argument as a hexadecimal number and pad the data segment with zeros to the indicated length.
- d** Force definition of common storage even if the **-r** flag is present.
- e** The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.
- l*x*** This option is an abbreviation for the library name **/lib/lib*x*.a**, where *x* is a string. If that does not exist, *ld* tries **/usr/lib/lib*x*.a** A library is searched when its name is encountered, so the placement of the option is significant.
- M** produce a primitive load map, listing the names of the files which will be loaded.
- N** Do not make the text portion read-only or sharable. (Use ‘magic number’ 0407.)
- n** Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. (Use magic number 0410 and move the data segment to a 1024 byte boundary.)
- o** The *name* argument after **-o** is used as the name of the *ld* output file, instead of
- r** Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final fixing of ‘common’ symbols (uninitialized C variables or Fortran common variables), and suppresses ‘undefined symbol’ diagnostics.
- s** Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by [strip\(1\)](#).
- S** Partially strip; remove all symbols that were not in the source.

- T** The next argument is a hexadecimal number which sets the text segment origin. With option **-A** this origin must be a multiple of 1024. The default is 0, or **\_end** with **-A**.
- t** (trace) Print the name of each file as it is processed.
- u** Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.
- X** Save local symbols except for those whose names begin with **L**. This option is used by [cc\(1\)](#) to discard internally-generated labels while retaining symbols local to routines.
- x** Do not preserve local symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.
- ysym** Indicate each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. (It is usually necessary to begin *sym* with an underscore **\_**, as external C, Fortran, and Pascal variables begin with underscores.)
- z** Arrange for the process to be loaded on demand from the resulting executable file (magic number 413) rather than preloaded. This (default) output format has a 1024-byte header followed by a text and data segment each of which have size a multiple of 1024 bytes (being padded out with zeros if necessary). The first few BSS segment symbols may appear in the data segment to avoid wasting space at the end of that segment.

## FILES

`/lib/lib*.a`  
libraries

`/usr/lib/lib*.a`  
more libraries

`a.out`  
output file

## SEE ALSO

[as\(1\)](#), [ar\(1\)](#), [cc\(1\)](#), [f77\(1\)](#), [size\(1\)](#), [nm\(1\)](#), [lorder\(1\)](#), [a.out\(5\)](#)

## BUGS

There is no way to force data to be page aligned.

**NAME**

"ld80" link editor for the 8080/Z80 load moduals.

**SYNOPSIS**

**ld80** [ **ulddb** ] name ...

**DESCRIPTION**

*ld80* combines several object programs into one; resolves external references; and searches libraries. In the simplest case the names of several object programs are given, and *ld80* combines them, producing an object module which can be either executed or become the input for a further *ld80* run. The output of *ld80* is left on **80.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

*ld80* understands several flag arguments which are written preceded by a '. Except for **l**, they should appear before the file names.

**b** This option is used to provide an absolute origin for the bss segment of the resultant "80.out". The supplied origin must be the next argument on the command line and must be a positive or negative octal (leading 0) or decimal number. The default is for the bss segment to immediately follow the data segment. Use of this option will cause the relocation information to be suppressed from the output.

**d** This option is used to provide an absolute origin for the data segment of the resultant "80.out". The supplied origin must be the next argument on the command line and must be a positive or negative octal (leading 0) or decimal number. The default is to have the data placed directly after the text. Use of this option will cause the relocation information to be suppressed from the output.

**u** take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

**t** This option is used to provide an absolute origin for the text segment of the resultant "80.out". The supplied origin must be the next argument on the command line and must be a positive or negative octal (leading 0) or decimal number. The default origin is 0. Use of this option will cause the relocation information to be suppressed from the output.

**l** This option is an abbreviation for a library name. **l** alone stands for '/usr/z8080/lib/z80', which is the standard library for assembly language programs. **lx** stands for '/usr/z8080/lib/z80x.a' where *x* is any character. A library is searched when its name is encountered, so the placement of a **l** is significant.

**FILES**

/usr/z8080/lib/z80	libraries
"80.out"	output file

**SEE ALSO**

"as80" (I), ar (I)

**BUGS**

Most diagnostics are self explanatory. The strangest is 'origin - conflict' and occurs whenever an origin supplied by the user via the -t -d or -b options causes segments to overlap. The numbers printed out correspond origins and sizes(both in octal) of each resultant segment. **80.out** is produced.

**NAME**

learn – computer aided instruction about UNIX

**SYNOPSIS**

**learn** [ *-directory* ] [ *subject* [ *lesson* [ *speed* ] ] ]

**DESCRIPTION**

*Learn* gives CAI courses and practice in the use of UNIX. To get started simply type ‘learn’. The program will ask questions to find out what you want to do. The questions may be bypassed by naming a *subject*, and the last *lesson* number that *learn* told you in the previous session. You may also include a *speed* number that was given with the lesson number (but without the parentheses that *learn* places around the speed number). If *lesson* is –, *learn* prompts for each lesson; this is useful for debugging.

The *subjects* presently handled are

```
editor
eqn
files
macros
morefiles
C
```

The special command *bye* terminates a *learn* session.

The *-directory* option allows one to exercise a script in a nonstandard place.

**FILES**

/usr/learn/\*

**BUGS**

The main strength of *learn*, that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped, but it takes some sophistication to recognize the situation.

**NAME**

lex – generator of lexical analysis programs

**SYNOPSIS**

lex [ **-tvfn** ] [ *file ...* ]

**DESCRIPTION**

*Lex* generates programs to be used in simple lexical analysis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, `lex.yy.c` is generated, to be compiled thus:

```
cc lex.yy.c -ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The options have the following meanings.

- t** Place the result on the standard output instead of in file
- v** Print a one-line summary of statistics of the generated analyzer.
- n** Opposite of **-v**; **-n** is default.
- f** ‘Faster’ compilation: don’t bother to pack the resulting tables; limited to small programs.

**EXAMPLES**

This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%  
[A-Z] putchar(yytext[0]+'a'-'A');  
[ ]+$  
[ ]+ putchar(' ');
```

**FILES**

`lex.yy.c`

**SEE ALSO**

[yacc\(1\)](#), [sed\(1\)](#)

M. E. Lesk and E. Schmidt, ‘LEX—Lexical Analyzer Generator’, this manual, Volume 2



**NAME**

*library* – send information requests to appropriate organization

**SYNOPSIS**

**library [-1234567] [request string]**

**DESCRIPTION**

*library* sends document/information requests to the appropriate organization within the Library Network. It also handles a variety of requests for other AT&T organizations (e.g., Engineering Information Services). It requires the user to discriminate between seven classes of requests as indicated by the menu it displays:

- 1) Order by number - this includes nearly everything announced by the AT&T Library Network (e.g. TMs, bulletin items) or handled by the Engineering Information Services (e.g. J docs, CPSs).
  - 2) Order item not announced by the AT&T Library Network. Available: Buy a copy of a book; Address labels; Technical Reports; Internal document by date (id unknown); Photocopies; Subscribe to a journal; foreign language services.
  - 3) Subscribe or unsubscribe to a bulletin (e.g. Mercury, CTP)
  - 4) Submit a database search. Examples of available databases: book catalog, internal documents, AT&T personnel, released papers
  - 5) Request human assistance / interaction. Type(s) available: General AT&T Library Network assistance; Reference question.
  - 6) Request AT&T Library Network information/services. Examples: LINUS info, products/services descriptions, loan/reserve status
  - 7) Read AT&T Library Network email transmissions
- Note that the first class includes anything announced by the Library Network operated by AT&T Bell Laboratories.

The main menu level of *library* can be skipped by giving the number of the desired option (1 - 7) as the first parameter to the *library*.

The secondary menu levels in *library* (presently in options 2, 4, 5 and 6) can be bypassed by selecting the desired option as the second parameter. For example, to do order a photocopy, use the command

```
library -2 -p
```

Similarly, if you want to do a search of the personnel database, you can do the command line

```
library -4 -p waldstein, r k
```

Requests for objects that can be meaningfully described with one line can be entered on the command line. This includes the things orderable by options 1, 3, 4, and 6. Note that *library* tries in this case to function with a minimum of interaction. For example, if you request a TM this way, you will not be given a chance to enter remarks connected with the request.

When permitted, the command line requests can include more than one item (presently only options 1 and 3 support this). An example command line is

```
library -1 123456-851234-56tm 5d123 ad-123456
```

In options 2, 5, and 6, *library* will prompt for a variety of information of varying complexity. A period (.) at any point in this session will delete the request being entered. Blank lines (just hit return) will cause optional information to be left out of the request. A line consisting of tilde e ( e ) will, when a long response is permitted, put you into an editor. This editor defaults to ed(1). However, if the environment variable EDITOR is set, the specified editor is used. A line consisting of tilde r ( r ) will, when a long response is permitted, read in the indicated file.

Option 7 is a misfit, in that it is primarily a reader, not a request transmitter; although it does allow requesting items. This option is intended for reading electronic transmissions from the library network: primarily responses to option 4 search requests and ASAP (specialized searches like electronic Mercury). A convenient way to use option 7 is to invoke it via a pipe from mail(1), mailx(1) or post(1).

```
| 3 "library -7"
```

This assumes that mail message 3 consists of a search result. The quote marks are required due to post(1)

and mailx(1) syntax.

Break causes *library* to exit without sending any requests.

In general for more information about what the library command can do, go into each option and enter a question mark. This will cause a description of how the option works and what it can request.

Several other pieces of information can be passed to *library* to ease and improve its usage. This information is looked up in a file called *.lib* (or the file indicated by the LIBFILE shell variable). It expects this file to contain lines of the form:

```
ID: individual's PAN or Social Security Number
libname: individual's last name
liblog: name of log file
libcntl: control information
reader: reader control information
liblocal: control information
```

If this file is not found or lines of this form are not found, then *library* prompts for name and ID (PAN or SS#).

This information can also be passed to *library* as the shell parameters: LIBID, LIBNAME, LIBLOG, LIBCNTL, and LIBLOCAL.

*library* keeps a log of requests sent via *library* if a line in the *.lib* file exists giving a log file name, i.e., if you have in your *.lib* file a line of the form

```
liblog: name of log file
```

*library* keeps a log of requests in that file. This file is created in a form that can be read and manipulated by the *mail* command. To read or modify the log file, type

```
mail -f name of log file
```

*library* creates the log file in your HOME directory unless the file name given starts with a slash (/). *library* will automatically check option 1 requests for duplicates in the log file.

The libcntl information is sent with the request to the program that receives the requests for the library networks. Control information containing the letter "a" will cause an acknowledgement to be mailed back to you that your request has been received. Control information of the form "mnumber" will determine the maximum number of items retrieved by a search request. For example, a control line of the form

```
libcntl: am100
```

will cause requests to be acknowledged and a maximum of 100 retrieved search items to be mailed back to you.

The reader control information is intended to let you personalize the way *library* option 7 works for you. Each letter after the colon indicates a different option turned on or off. Presently available are the following:

- b causes *library* -7 to leave a blank line between records when more than one is displayed on the screen.
- c causes *library* to confirm that you want the entered requests transmitted. It does this at the end of the session, before finishing.
- n is an interesting features causing no introductory menu of available announcements to be displayed. The reader then goes straight into the first announcement to be read, and moves directly from one announcement to the next, without displaying the menu of those available at each stage.

The liblocal information is used to control the execution of *library*. Presently the only meaningful control is *x*. This causes *library*(1) to assume you are an expert and the prompts are generally much shorter.

*library* also uses your *.lib* file to save various repetitious responses for its own use. These will prevent you from having to duplicate responses.

**BUGS**

*library* checks upon input whether the request is reasonable. New styles of request numbers require program modification before they are valid.

**FILES**

\$HOME/.lib        This optional file contains a PAN and name for *library* to use.  
/usr/lib/library/library.help  
                  The help message displayed by *library*.

**SEE ALSO**

mail(1), post(1), mailx(1)

**NAME**

lim – change shares for users

**SYNOPSIS**

**lim** limit{+|-|=}string[,string...][;...] [-name|uid-uid|uid.. ...

**DESCRIPTION**

*Lim* changes shares file records for several users, a range of uids or a list from standard input. The changeable limits are designated by their names as defined in the files *<sys/lnode.h>*, and *<shares.h>* as follows:

<b>charge</b>	Long term account charge.
<b>flags</b>	The only specifiable flag is: <i>notshared</i> . Only the first few letters needed to ensure a unique match are required.
<b>lastused</b>	Date account last used.
<b>sgroup</b>	Scheduling group for the account.
<b>shares</b>	Allocated shares.
<b>usage</b>	Usage for scheduling.

The next character designates that the limit is to be incremented (+), decremented (–), or set (=).

The third group of characters is interpreted as a number, a date (if the string contains a ‘/’), or as a string (or as a comma-separated list) depending on the type of limit being changed. However, if the first character is ‘?’, then an explanation of the options available with the given limit will be listed.

Additional limits are specified by a semi-colon separated list.

If any following argument is just a ‘–’, then the standard input is read for a list of user names, one per line. If any following argument *contains* a ‘–’ then it is interpreted as a range of uids, otherwise if it ends in trailing dots (eg: 100..) it is interpreted as a range running from the first uid up to the maximum number of registered users. Otherwise the argument is interpreted as a name.

A list of valid limits is printed out if *lim* is invoked with invalid arguments (or no arguments).

**FILES**

<i>/etc/shares</i>	for share details.
<i>/etc/passwd</i>	for user names and IDs.

**SEE ALSO**

pl(1), lnode(5), shares(5).

**DIAGNOSTICS**

... could not change kernel lnode ...

The limits system call failed for a logged in user, usually because you are attempting to change a scheduling group to one that isn’t currently active.

**BUGS**

*Lim* does not use *getshput(3)*, so be careful something else is not updating the same entry simultaneously.

**NAME**

lint, cyntax, cem – C program verifiers

**SYNOPSIS**

**lint** [ **-abchnpuvx** ] [ *option ...* ] *file ...*

**cyntax** [ *option ...* ] *file ...*

**/usr/lib/cyntax/cem** [ *option ...* ] *file ...*

**DESCRIPTION**

*Lint* checks, more thoroughly than *cc(1)*, the syntactic validity and semantic consistency of one or more C program *files*. It is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as *-lm*, in the style of *ld(1)*.

Any number of the option letters in the following list may be used. The **-D**, **-U**, and **-I** options of *cc(1)* are also recognized as separate arguments.

- p** Attempt to check portability to some other dialects of C.
- h** Apply heuristics to intuit bugs, improve style, and reduce waste.
- b** Report *break* statements that cannot be reached.
- v** Suppress complaints about unused arguments in functions.
- x** Report variables referred to by extern declarations, but never used.
- a** Report assignments of long values to int variables.
- c** Complain about casts which have questionable portability.
- u** Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).
- n** Do not check compatibility against the standard library.

Certain conventional comments in the C source will change the behavior of *lint*:

**/\*NOTREACHED\*/**

at appropriate points. Stop comments about unreachable code.

**/\*VARARGS*n*\*/**

Suppress the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

**/\*PRINTFLIKE*n*\*/**

The data types of the first *n* arguments are checked as usual. The remaining arguments are checked against the *n*th argument, which is interpreted as a *printf(3)* format string.

**/\*SCANFLIKE*n*\*/**

Similarly for *scanf(3)*.

**/\*NOSTRICT\*/**

Shut off strict type checking in the next expression.

**/\*ARGSUSED\*/**

Turn on the **-v** option for the next function.

**/\*LINTLIBRARY\*/**

at the beginning of a file. Shut off complaints about unused functions in this file.

*Cyntax* makes checks similar to those of *lint*, more stringent about syntax, less observant of portability issues. It keeps type information gleaned from source files, whose names end with *.c*, in ‘object files’ with corresponding names ending in *.O*. If all goes well it will cross check among all the *.c* and *.O* input *files*.

Options **-D**, **-I**, **-U**, **-o**, **-l** are as in *cc(1)*. Options **-O**, **-P**, **-g**, **-p**, and **-s** are ignored. Other options are:

- c** Suppress cross checking.
- d** Passed to *cem*.
- h** Base object files on the basename of the source file rather than the full pathname.
- n** Do not check compatibility against the standard library.
- G** Change default directory of include files to pass **-lg** to *cem* instead of **-lc**.
- j** Change default directory of include files to pass **-lj** to *cem* instead of **-lc**.

- k** Change default directory of include files to pass **-lk** to *cem* instead of **-lc**.
- w** Enable pedantic warning diagnostics.
- m** equivalent to **-j -DMUX**.
- v** Report what *cyntax* is actually doing.
- V func:n**  
Declare function *func* to have a variable number of arguments, the first *n* of which are to be type checked.

*Cem* (cemantics), the cross-checker, is normally invoked only by *cyntax*. It also has options, some of which *cyntax* can't be coerced into providing. Besides **-o** and **-l**, they are:

- m** Don't believe file modification times. These are normally used to avert redundant type checking.
- d** Debug: print ascii version of *.O* files on standard output.
- p** Be pedantic about type checking.
- t** Unconditionally include file modification times in diagnostics.
- v** Use a verbose format for type names.

## FILES

```

/usr/lib/lint/lint[12]
    programs

/usr/lib/lint/l1ib-lc
    declarations for standard functions

/usr/lib/lint/l1ib-lport
    declarations for portable functions

/usr/include/*

/usr/lib/cyntax/ccom
    cyntax proper

/usr/lib/cyntax/libc
    type library

/usr/lib/cyntax/libj

```

## SEE ALSO

[cc\(1\)](#), [cin\(1\)](#)

## BUGS

*Lint's* understanding of the type system of C is outmoded: its handling of `void` and `@is` is simply wrong. The unnatural default setting of *lint* option **-b** is intended to hide the ugliness of C code produced by [yacc\(1\)](#) and [lex\(1\)](#).

**NAME**

*lisp*, *liszt*, *lxref* – lisp interpreter and compiler

**SYNOPSIS**

***lisp***

***liszt*** [ *option ...* ] [ *source* ]

***lxref*** [ *-n* ] *file ...*

**DESCRIPTION**

*Lisp* interprets Franz Lisp, which closely resembles MIT's Maclisp. Interpreted functions may be mixed with code compiled by *liszt*, and both may be debugged using the 'Joseph Lister' trace package.

There are too many functions to list here; one should refer to the manuals listed below.

*Liszt* compiles the lisp *source* file, whose name ends in **.l**, into an object file, whose name ends in **.o**. The following options are available.

- w** suppress warning diagnostics
- q** suppress compilation statistics
- o *object***  
put object code in specified file
- m** source is Maclisp
- u** source is UCI Lisp
- S** leave assembler input in file suffixed **.s**; do not finish compilation
- x** place cross-reference list in file suffixed **.x** to be used by *lxref*.

*Liszt* with no arguments is the same as *lisp*. The compiler may be invoked from the interpreter:

```
( liszt [options] foo )
```

compiles file '*foo.l*'.

*Lxref* writes to the standard output a readable form of the named cross-reference files. Not more than *n* (default 50) references to any function will be printed.

**FILES**

*/usr/lib/lisp/auxfns0.l* common functions  
*/usr/lib/lisp/auxfns1.l* less common functions  
*/usr/lib/lisp/trace.l* Joseph Lister trace package  
*/usr/lib/lisp/toplevel.l* top level read-eval-print loop  
*/usr/lib/lisp/machacks.l* Maclisp compatibility package  
*/usr/lib/lisp/ucifnc.l* UCI Lisp compatibility package

**SEE ALSO**

'FRANZ LISP Manual, Version 1' by John K. Foderaro  
MACLISP Manual

**BUGS**

The error system is in a state of flux and not all error messages are as informative as they could be.

**NAME**

load – load statistics

**SYNOPSIS**

**load** [ *interval* [ *count* [ *sysfile* [ *corefile* ] ] ] ]

**DESCRIPTION**

*Load* reports the number of processes ready to run averaged over the preceding 1, 5, and 15 minutes.

The optional *interval* argument causes a report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

The optional arguments *sysfile* and *corefile* cause the named files to be consulted instead of the defaults, */unix* and

**FILES**

*/dev/kmem*

*/unix*

**SEE ALSO**

*vismon(9)* *ps(1)*, *vmstat(8)*



**NAME**

look – find lines in a sorted list

**SYNOPSIS**

**look** [ **-dfnixtc** ] [ *string* ] [ *file* ]

**DESCRIPTION**

*Look* consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The following options are recognized. Options **dfntc** affect comparisons as in [sort\(1\)](#).

- i** Interactive. There is no *string* argument; instead *look* takes lines from the standard input as strings to be looked up.
- x** Exact. Print only lines of the file whose key matches *string* exactly.
- d** ‘Directory’ order: only letters, digits, tabs and blanks participate in comparisons.
- f** Fold. Upper case letters compare equal to lower case.
- n** Numeric comparison with initial string of digits, optional minus sign, and optional decimal point.
- tc** ‘Tab character’ *c* terminates the sort key in the *file*.

If no *file* is specified, `/usr/dict/words` is assumed, with collating sequence **df**.

**FILES**

`/usr/dict/words`

**SEE ALSO**

[sort\(1\)](#), [gre\(1\)](#), [dict\(7\)](#)

**DIAGNOSTICS**

*Look* returns exit status 0 if *string* is found, 1 if not found, 2 for error.

**NAME**

lorder – find ordering relation for an object library

**SYNOPSIS**

**lorder** *file* ...

**DESCRIPTION**

The input is one or more object or library archive (see [ar\(1\)](#)) *files*. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by [tsort\(1\)](#) to find an ordering of a library suitable for one-pass sequential access by [ld\(1\)](#).

**EXAMPLES**

```
ar cr libnew.a `lorder *.o | tsort` Build a new library from existing .o files.
```

**FILES**

/tmp/\*symref

/tmp/\*symdef

**SEE ALSO**

[ar\(1\)](#), [tsort\(1\)](#), [ld\(1\)](#)

**BUGS**

The names of object files, in and out of libraries, must end with **.o**; nonsense results otherwise.

**NAME**

lp – printer output

**SYNOPSIS**

lp [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Lp* is a generalized output printing service. It can be used to queue files for printing, check a queue, or kill jobs in a queue. The options are:

**-d** *dest*

Select the destination printer. If *dest* is `?`, list the currently available printers. In the absence of `-d`, the destination is taken from the environment variable **LPDEST**, or finally from the file `Destination stdout` is the standard output.

**-p** *proc*

The given preprocessor is invoked. The default preprocessor is `generic`, which tries to do the right thing for regular text, *troff*(1) output, or *bitfile*(9) bitmaps. If no preprocessing is desired (e.g. an `ImPress` file from `dviimp` is to be printed) `noproc` may be specified.

**-q** Print the queue for the given destination. For some devices, include printer status.

**-k** Kill the job(s) given as subsequent arguments instead of file names for the given destination.

The remaining options may be used to affect the output at a given device. These options may not be applicable to all devices.

**-c** *n* Print *n* copies.

**-f** *font* Set the font (default `CW.11`).

**-H** Suppress printing of header page.

**-i** *n* Select paper input tray options *n*. The value *n* may be a comma separated list.

**-l** *n* Set the number of lines per page to *n*.

**-L** Print pages in landscape mode (i.e. turned 90 degrees).

**-m** *n* Set magnification to *n*.

**-n** *n* Print *n* logical pages per physical page.

**-o** *list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *n-m* means pages *n* through *m*; a range *-n* means from the beginning to page *n*; a range *n-* means from page *n* to the end.

**-r** Reverse the order of page printing (currently not functional).

**-x** *n* Set the horizontal offset of the print image, measured in inches.

**-y** *n* Set the vertical offset of the print image, measured in inches.

**EXAMPLES**

`eqn paper | troff -ms | lp` Typeset and print a paper that contains equations.

`pr -l100 file | lp -l100 -fCW.8` Print a file in a small font at 100 lines per page.

`lp -dstdout -H <bitfile >postfile` Convert a bitmap to postscript form. Use *mpictures*(6) macros to insert the output into a *troff* document.

`lp -du -H -i2,simplex viewgraphs.dvi` will take input from the second paper tray and print single sided, even if the printer defaults to double sided (duplex) output. Do not print a header page.

**FILES**

`/usr/spool/lp/defdevice`  
default printer name

`/usr/spool/lp/devices`  
printer list with interface specification

`/usr/spool/lp/process`  
directory of preprocessors

`/usr/spool/lp/prob/*`  
where printer jobs go when things go awry

**SEE ALSO**

*pr(1)*, *blitblt(9)* *plot(1)*, *font(6)*, *postio(8)*, *postscript(8)*

**BUGS**

Not all options work with all output devices.  
Any user can kill any job.

**NAME**

`lpr` – line printer spooler

**SYNOPSIS**

`lpr` [ **-m** ] [ *name* ... ]

`lp0` [ *name* ... ]

`lp1` [ *name* ... ]

**DESCRIPTION**

*Lpr* causes the named files to be queued and then printed off line. If no files are named, the standard input is read. The option **-m** causes notification via [mail\(1\)](#) to be sent when the job completes.

*Lp0* and *lp1* direct output to particular printers.

**FILES**

`/usr/spool/lpd/*`  
spool area

`/usr/lib/lpd`  
printer daemon

`/usr/lib/lpfx`  
filter to handle banners and underlining

**SEE ALSO**

[pr\(1\)](#), [thinkblt\(9\)](#)

**BUGS**

Queued jobs print in directory (seemingly random) order.

**NAME**

*ls*, *lc* – list contents of directory

**SYNOPSIS**

**ls** [ **-acdfilrstuFLR** ] *name* ...

**lc** [ *options* ] *name* ...

**DESCRIPTION**

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted, but file arguments appear before directories and their contents.

*Lc* is the same as *ls*, but prints the list in multiple columns.

There are an unbelievable number of options:

- l** List in long format, giving mode (see below), number of links, owner, group, size in bytes, and time of last modification for each file. Symbolic links are identified by a link count marked **L**; the link count is that of the ultimate file. If the file is a special file the size field will instead contain the major and minor device numbers.
- d** If argument is a directory, list its name, not its contents.
- t** Sort by time modified (latest first) instead of by name, as is normal.
- L** Under **-l** for each symbolic link give the immediate, not the ultimate, link count and append the name pointed to.
- a** List all entries; usually **.** and **..** are suppressed.
- c** Under **-t** sort by time of inode change; under **-l** print time of inode change.
- f** Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.
- F** Mark directories with a trailing **/** and executable files with a trailing **\***
- i** Print i-number in first column of the report for each file listed.
- r** Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
- R** recursively list subdirectories encountered.
- s** Give size in Kbytes for each entry.
- u** Under **-t** sort by time of last access; under **-l** print time of last access.

The mode printed under the **-l** option contains 11 characters which are interpreted as follows: the first character is

- d** if the entry is a directory;
- b** if the entry is a block-type special file;
- c** if the entry is a character-type special file;
- l** if the entry is a symbolic link and option **-L** is in effect;
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, ‘execute’ permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

- r** if the file is readable;
- w** if the file is writable;

- x** if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as **s** if the file has set-user-ID mode.

The last character of the mode (normally a blank) indicates the type of concurrency control:

- e** if the file is set for exclusive access (1 writer or *n* readers);
- y** if the file is set for synchronized access (1 writer and *n* readers);

## **FILES**

## **SEE ALSO**

[stat\(2\)](#)

## **BUGS**

Option **-s** counts unwritten holes as if they were real data.

**NAME**

m4 – macro processor

**SYNOPSIS**

**m4** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*M4* is a macro processor intended as a front end for C and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is `-`, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

- e** Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.
- s** Enable line sync output for the C preprocessor, (**#line** ...)
- Bint** Change the size of the push-back and argument collection buffers from the default of 4,096.
- Hint** Change the size of the symbol table hash array from the default of 199. The size should be prime.
- Sint** Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
- Tint** Change the size of the token buffer from the default of 512 bytes.

The preceding options must appear before any file names or **-D** or **-U** options.

**-Dname**[=*val*]

Defines *name* to *val* or to null if *val* is missing.

**-Uname**

undefines *name*.

Macro calls have the form:

```
name ( arg1 , arg2 ,
```

The `(` must immediately follow the name of the macro. If a defined macro name is not followed by a `(`, it is deemed to have no arguments. Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore `_`, where the first character is not a digit.

Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

The value of a macro is obtained by replacing each occurrence of `$n` in the replacement text, where *n* is a digit, with the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; `$#` is replaced by the number of arguments; `$*` is replaced by a list of all the arguments separated by commas; `$` is like `$*`, but each argument is quoted (with the current quotes).

*M4* makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

- define** the second argument is installed as the replacement text of the macro whose name is the first argument.
- undefine** Remove the definition of the macro named in the argument.
- defn** Return the quoted definition of the argument(s); useful for renaming macros, especially built-ins.



<b>pushdef</b>	Like <i>define</i> , but save any previous definition.
<b>popdef</b>	Remove current definition of the argument(s), exposing the previous one if any.
<b>ifdef</b>	If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word <code>unix</code> is predefined on UNIX versions of <i>m4</i> .
<b>shift</b>	Return all but the first argument. The other arguments pushed back with commas in between and quoted to nullify the effect of the extra scan.
<b>changequote</b>	Change quote symbols to the first and second arguments. The symbols may be up to five characters long. <b>Changequote</b> without arguments restores the original values (i.e., ' ').
<b>changeocom</b>	Change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
<b>divert</b>	<i>m4</i> Switch output to one of 10 streams, numbered 0-9 designated by the argument. The final output is the concatenation of the streams in numerical order; stream 0 is the current initially. Output to a stream other than 0 through 9 is discarded.
<b>undivert</b>	Cause immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Once undiverted, the diverted text is no longer contained in that diversion.
<b>divnum</b>	Return the name of the current output stream.
<b>dnl</b>	reads and discards characters up to and including the next new-line.
<b>ifelse</b>	If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if that is not present, null.
<b>incr</b>	Return the value of the argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
<b>decr</b>	Return the value of the argument decremented by 1.
<b>eval</b>	Evaluate the argument as an arithmetic expression, using 32-bit arithmetic. C-like operators include <code>+*/%</code> , bitwise <code>&amp; ^~</code> ; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
<b>len</b>	Returns the number of characters in the argument.
<b>index</b>	Return the position in the first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.
<b>substr</b>	Return a substring of the first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
<b>translit</b>	Transliterate the characters in the first argument from the set given by the second argument to the set given by the third, deleting characters that lack a correspondent in the third set. There is no character-range notation.
<b>include</b>	Return the contents of the file named in the argument.
<b>sinclude</b>	Same, but give no diagnostic if the file is inaccessible.
<b>syscmd</b>	Execute the UNIX command given in the first argument. No value is returned.
<b>sysval</b>	The return code from the last call to <i>syscmd</i> .
<b>maketemp</b>	Fill in a string of X characters in the argument with the current process id.

<b>m4exit</b>	Exit immediately from <i>m4</i> . Argument 1, if given, is the exit code; the default is 0.
<b>m4wrap</b>	Push the argument back at the end of the input. Example: <code>m4wrap( 'cleanup()' )</code>
<b>errprint</b>	Prints the argument on the standard error file.
<b>dumpdef</b>	Print current names and definitions, for the named items, or for all if no arguments are given.
<b>traceon</b>	If there are no arguments, turn on tracing for all macros (including built-ins). Otherwise, turn on tracing for named macros.
<b>traceoff</b>	Turn off trace globally and for any macros specified. Macros specifically traced by <b>traceon</b> can be untraced only by specific calls to <b>traceoff</b> .

### EXAMPLES

```
define(fib, 'ifelse(define('n', eval($1))n, 0, 1, n, 1, 1, 1, dnl()
'eval(fib(n-1)+fib($1-2))')')dnl()
fib(2*3)
```

Recursively evaluate a Fibonacci number. The inner **define** avoids some reevaluations.

**NAME**

Mail – send and receive mail

**SYNOPSIS**

**Mail** [ *option* ] (.,.)[ *person* ] (.,.)SH DESCRIPTION *Mail* with *persons* named reads a letter from the standard input and sends it to them.

*Mail* otherwise presents your mail. It responds to commands, each typed on a line by itself, possibly with arguments. A command need not be typed in its entirety – the first command that matches the typed prefix is used. A missing message list is interpreted as a singleton: whichever of the current message, the nearest message ahead, or the nearest message back satisfies the command's requirements.

The following table describes the commands:

!	"Single	commandescape
-	"Back	upto
Reply	"Compose	areply
alias	"Define	analias
alternates	"List	othernames
chdir	"Change	workingdirectory,
copy	"Copy	amessage
delete	"Delete	alist
dt	"Delete	message,type
endif	"End	ofif
edit	"Edit	alist
else	"Part	ofif
exit	"Leave	mailwithout
file	"Interrogate/change	currentmail
folder	"Same	asfile"
folders	"List	yourfolder
from	"List	headersof
headers	"List	currentwindow
help	"Print	briefsummary
hold	"Same	aspreserve"
if	"Conditionalexecutionof	
ignore	"Do	notprint
mail	"Send	mailto
mbox	"Arrange	tosave
next	"Go	tonext
preserve	"Arrange	toleave
quit	"Leave	Mail;update
reply	"Reply	toauthor
save	"Append	messages,with
set	"Set	binaryor
shell	"Invoke	aninteractive
top	"Print	firstso
type	"Print	messages"
undelete	"Undelete	listof
unset	"Undo	theoperation
visual	"Invoke	visualeditor
write	"Append	messages,without
z	"Scroll	tonext/previous

The following table describes the options for **set**. Each option is shown as being either a binary or valued option.

EDITOR	valued	"Pathnameofeditor
SHELL	valued	"Pathnameofshell
VISUAL	valued	"Pathnameofscreen
append	binary	"Alwaysappendmessages
ask	binary	"Promptuserfor

askcc	binary	"Promptuserfor
autoprint	binary	"Printnextmessage
crt	valued	"Minimumnumberof
dot	binary	"Accept.alone
escape	valued	"Escapecharacter to
folder	valued	"Directory to store
hold	binary	"Hold messages in
ignore	binary	"Ignore RUBOUT while
ignoreeof	binary	"Don't terminate letters/command
keep	binary	"Don't unlink <i>mbox</i>
keepsave	binary	"Don't delete saved
metoo	binary	"Includesendinguser
nosave	binary	"Don't save partial
quiet	binary	"Suppress printing of
record	valued	"File to save
screen	valued	"Size of window
sendmail	valued	"Choose alternate mail
toplines	valued	"Number of lines

The following table summarizes tilde escapes available while entering a letter.

Escape	Arguments	Description
~!	command	"Executes <i>shell</i> command"
~c	name	... " <i>Addnames</i>
~d		"Read dead.letter into
~e		"Invoke <i>text</i> editor
~f	messages	"Read <i>named</i> messages"
~h		"Edit <i>the</i> header
~m	messages	"Read <i>named</i> messages,
~p		"Print <i>message</i> entered
~q		"Abort entry of
~r	filename	"Read <i>file</i> into
~s	string	"Set <i>Subject:</i> field
~t	name	... " <i>Addnames</i>
~v		"Invoke <i>screen</i> editor
~w	filename	"Write <i>message</i> on
~	command	"Pipe <i>message</i> through
~~	string	"Quote a ~

The following table shows the command line flags.

-N	"Suppress <i>the</i> <b>initial</b>
-T	file "Article-id" <i>sof</i> read/deleted
-d	"Turn on debugging"
-f	file "Show <i>messages</i> in
-h	number "Pass on hop
-i	"Ignore <i>ty</i> interrupt
-n	"Inhibit <i>reading</i> of
-r	name "Pass on <i>name</i>
-s	string "Use <i>string</i> as
-u	name "Read <i>name</i> 's mail

Notes: **-T**, **-d**, **-h**, and **-r** are not intended for human consumption.

## FILES

/usr/spool/mail/\*  
 post office

\$HOME/mbox  
 your old mail

`$HOME/.mailrc`  
file giving initial mail commands

`/tmp/R#`  
temporary for editor escape

`/usr/lib/Mail.help*`  
help files

`/usr/lib/Mail.rc`  
system initialization file

`/bin/mail`  
to do actual mailing

**SEE ALSO**

[mail\(1\)](#)

'The Mail Reference Manual,' *Berkeley BSD 4.1 UNIX User's Manual*

**NAME**

mail – send or receive mail

**SYNOPSIS**

**mail** [ **-mpren** ] [ **-f** *file* ]

**mail** [ **-#** ] *person* ...

**mail**

**/usr/lib/upas/gone.fishing** [ *msg* ]

**DESCRIPTION****Printing Mail**

When *persons* are not named, *mail* displays your incoming computer mail. The options are:

- r** Print mail in first-in, first-out order.
- p** Print all the mail messages without prompting for commands.
- m** Use a manual style of interface, i.e., print no messages unless directed to.
- f *file*** Use *file*, e.g. *mbox*, as if it were the mailbox.
- e** Check silently if there is anything in the mailbox; return zero (true) if so, non-zero otherwise.
- n** Announce mail to the control terminal when it arrives. Do not print mail now.

*Mail* prints a user's mail, message by message, prompting between messages. After printing a prompt *mail* reads a line from the standard input to direct disposition of the message. Commands, as in *ed*(1), are of the form '*[range] command [arguments]*'. The command is applied to each message in the (optional) range addressed by message number and/or regular expressions in the style of *regexp*(3). A regular expression in slashes searches among header (postmark) lines; an expression in backslashes searches on message content.

*address* to indicate a single message header  
*address,address* to indicate a range of contiguous message headers  
*/expression/* to indicate all message headers matching the regular *expression*.

The commands are:

- b** Print the headers for the next ten messages.
- d** Mark message for deletion on exiting mail.
- h** Print the disposition, size in characters, and header line of the message.
- m *person* ...** Mail the message to the named *persons*.
- M *person* ...** Same as **m** except that lines typed on the terminal (terminated by **EOT** or **.**) are prepended to the message.
- p** Print message. An interrupt stops the printing.
- r** Reply to the sender of the message.
- R** Like **r** but with the message appended to the reply.
- s *file*** (Save) Append the message to the named *file* default, in **HOME** directory if known, see *environ*(5).
- q** Put undeleted mail back in the mailbox and stop.
- EOT (control-D)** Same as **q**.
- w *file*** Same as **s** with the mail header line(s) stripped.
- u** Remove mark for deletion.
- x** Exit, without changing the mailbox file.
- ?** Print a command summary.
- |*command*** Run the *command* with the message as standard input.
- !*command*** Escape to the shell to do *command*.
- =** Print the number of the current message.

**Sending Mail**

When *persons* are named, *mail* takes the standard input up to an end-of-file, or (if input is from a terminal) to a line consisting of a single **.** and adds it to each *person*'s mailbox. The message is automatically postmarked with the sender's name and date. Lines that look like postmarks are prefixed with **>**.

*Person* is a login name on the local system or a network name for a remote system; see *mail*(6).

Option **-#** does not send mail, but reports instead how the mail would be sent: the sender, the next machine

to handle the mail, and the recipient's address relative to that machine. The report reflects address translation; see [mail\(6\)](#) and [upas\(8\)](#).

[Sh\(1\)](#) and [vismon\(9\)](#) have mechanisms for timely notification of incoming mail.

### Mailboxes

Each user owns a mailbox for incoming mail, normally `/usr/spool/mail/person`. *Mail* creates mailboxes as necessary, and never removes them. Mailboxes are created readable but not writable by others. For more privacy, a mailbox's owner may make it unreadable; see [chmod\(2\)](#).

If a mailbox contains the sole line

**Forward to** *name*,

mail for that mailbox is sent instead to *name*. *Name* may be a list of names. If the mailbox contains

**Pipe to** *command*

the mail is sent to the standard input of *command* instead of being appended to the mailbox. The command is run with the userid and groupid of the mailbox's owner. The command is sent (see [signal\(2\)](#)) after two minutes. (On System V machines, the set userid bit must be set.)

*Mail* checks centralized forwarding lists before looking in mailboxes. If you have accounts on many machines, but wish to receive mail on only one, it is usually easier to register in forwarding lists than to install `Forward` in many mailboxes; see [upas\(8\)](#).

To use *mail* as an answering machine while you are away, replace the contents of your mailbox with a single line like

**Pipe to** `/usr/lib/upas/gone.fishing /usr/you/mesg`

The *mesg* file will be sent (just once) to everyone who sends you mail; arriving messages will be collected in **gone.mail** in your home directory. If you do not name a *mesg* file, will be used by default.

### FILES

```
/usr/spool/mail/mail.log
    mail log file

/usr/spool/mail/*
    mailboxes

/etc/passwd
    to identify sender and locate persons

$HOME/mbox
    saved mail

$HOME/dead.letter
    unmailable text

/usr/lib/upas/edmail
    the program for editing mail

/usr/lib/upas/send
    the program for sending mail

/bin/rmail
    a link to used to receive remote mail

/usr/lib/upas/gone.msg
$HOME/gone.mail
$HOME/gone.addr
    list of senders answered by gone.fishing
```

### SEE ALSO

[write\(1\)](#), [vismon\(9\)](#) [uucp\(1\)](#), [verify\(1\)](#), [mail\(6\)](#), [upas\(8\)](#), [smtp\(8\)](#)

**BUGS**

Long headers are truncated for header search.

Backslash quoting is impossible in content regular expressions.



**NAME**

*mailx* – interactive message processing system

**SYNOPSIS**

**mailx** [options] [address ...]

**DESCRIPTION**

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they may be marked to be moved to a secondary file for storage (see “hold”), unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user’s HOME directory (see “MBOX”). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **-f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *mailbox*. This gives rise within these pages to the notion of a current *mailbox*.

On the command line, *options* start with a dash (–) and any other arguments are taken to be addresses. If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

- e** Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.
- f [filename]** Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.
- F** Record the message in a file named after the first recipient. Overrides the “record” variable, if set.
- H** Print header summary only.
- i** Ignore interrupts. See “ignore”.
- n** Do not initialize from the system default *mailx.rc* file.
- N** Do not print initial header summary.
- s subject** Set the Subject header field to *subject*.
- u user** Read *user*’s *mailbox*. This is only effective if *user*’s *mailbox* is not read protected.

When reading mail, *mailx* is in command mode. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in input mode. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands. Outgoing mail may be saved in a local file (see “record”).

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the “set” and “unset” commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

*mailx* accepts *post* addressing. Login names may be any network address, including mixed network addressing. If mail is found to be undeliverable, an attempt is made to return it to the sender’s *mailbox*. If the recipient name begins with a plus (+), the rest of the name is a file to append the mail to. If the file is a relative path name, it is prepended with the value of the “folder” variable. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp(1)* for recording outgoing mail on paper. Alias groups are set by the “alias” command and are lists of recipients of any type.

Regular commands are of the form

[ **command** ] [ *msglist* ] [ *arguments* ]

If no command is specified in command mode, **print** is assumed. In input mode, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

<b>n</b>	Message number <b>n</b> .
<b>.</b>	The current message.
<b>^</b>	The first message.
<b>\$</b>	The last message.
<b>*</b>	All messages.
<b>n-m</b>	An inclusive range of message numbers.
<b>user</b>	All messages from <b>user</b> , where <b>user</b> is the network address of the sender.
<b>/string</b>	All messages with <b>string</b> in the subject line (case ignored).
<b>:c</b>	All messages of type <i>c</i> , where <i>c</i> is one of:
	<b>d</b> deleted messages
	<b>n</b> new messages
	<b>o</b> old messages
	<b>r</b> read messages
	<b>u</b> unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh(1)*). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* tries to execute commands from the optional system-wide file (*/usr/lib/mailx.rc*) to initialize certain parameters, then from a private start-up file (*\$HOME/.mailrc*) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: “!”, “Copy”, “edit”, “followup”, “Followup”, “hold”, “mail”, “reply”, “Reply”, “shell”, and “visual”. An error in the start-up file causes the remaining lines in the file to be ignored. The *.mailrc* file is optional, and must be constructed locally.

## COMMANDS

The following is a complete list of *mailx* commands:

<b>!shell-command</b>	Escape to the shell. See “SHELL”.
<b># comment</b>	Null command (comment). This may be useful in <i>.mailrc</i> files.
<b>=</b>	Print the current message number.
<b>?</b>	Prints a summary of commands.
<b>alias alias address ...</b>	Declare an alias for the given addresses. The addresses will be substituted when <i>alias</i> is used as a recipient. Useful in the <i>.mailrc</i> file.
<b>alternates address ...</b>	Declares a list of alternate addresses for your login. When responding to a message, these addresses are removed from the list of recipients for the response. With no arguments, <b>alternates</b> prints the current list of alternate addresses. See also “allnet”.
<b>cd [directory]</b>	Change directory. If <i>directory</i> is not specified, \$HOME is used.
<b>copy [filename]</b>	
<b>copy [msglist] filename</b>	Copy messages to the file without marking the messages as saved. Otherwise equivalent to the

“save” command.

**Copy** [*msglist*]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the “Save” command.

**delete** [*msglist*]

Delete messages from the *mailbox*. If “autoprint” is set, the next message after the last one deleted is printed.

**discard** [*header-field ...*]

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are “status” and “cc”. The fields are included when the message is saved. The “Print” command overrides this command.

**dp** [*msglist*]

Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a “delete” command followed by a “print” command.

**echo** *string ...*

Echo the given strings (like *echo(1)*).

**edit** [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the “EDITOR” variable is used to get the name of the editor. Default editor is *ed(1)*.

**exit**

**xit**

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

**file** [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	current <i>mailbox</i> .
% <b>user</b>	<i>mailbox</i> for <b>user</b> .
#	previous file.
&	current <i>mbox</i> .

Default file is the current *mailbox*.

**folders**

Print the names of the files in the directory set by the “folder” variable.

**followup** [*msglist*]

Reply to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the “Followup”, “Save”, and “Copy” commands, and “outfolder” and “flipf”.

**Followup** [*message*]

Reply to the specified message, recording the response in a file whose name is derived from the author of the message. Overrides the “record” variable, if set. See also the “followup”, “Save”, and “Copy” commands, and “outfolder” and “flipf”.

**from** [*msglist*]

Prints the header summary for the specified messages.

**headers** [*message*]

Prints the page of headers which includes the message specified. The “screen” variable sets the number of headers per page. See also the “z” command.

**help**

Prints a summary of commands.

**hold** [*msglist*]

Holds the specified messages in the *mailbox*.

**if s | r**

*mail-commands*

**else**

*mail-commands*

- endif**  
Conditional execution, where **s** will execute following *mail-commands*, up to an **else** or **endif**, if the program is in send mode, and **r** causes the *mail-commands* to be executed only in receive mode. Useful in the *.mailrc* file.
- list**  
Prints all commands available. No explanation is given.
- mail address ...**  
Mail a message to the specified *address(es)*. If “record” is set to a file name, the reply is saved at the end of that file.
- Mail address**  
Mail a message to the specified *address* and record a copy of it in a file named after that *address*.
- mbox [msglist]**  
Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See “MBOX” for a description of this file. See also the “exit” and “quit” commands.
- next [message]**  
Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.
- pipe [msglist] [shell-command]**  
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the “cmd” variable. If the “page” variable is set, a form feed character is inserted after each message.
- print [msglist]**  
Print the specified messages. If “crt” is set, the messages longer than the number of lines specified by the “crt” variable are paged through the command specified by the “PAGER” variable. The default command is *pg(1)*.
- Print [msglist]**  
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the “ignore” command.
- quit**  
Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.
- reply [msglist]**  
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If “record” is set to a file name, the response is saved at the end of that file. See “flpr”.
- Reply [message]**  
Reply to the specified message, including all other recipients of the message. If “record” is set to a file name, the response is saved at the end of that file. See “flpr”.
- save [filename]**  
**save [msglist] filename**  
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless “keepsave” is set (see the “exit” and “quit” commands).
- Save [msglist]**  
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author’s name with all network addressing stripped off. See also the “Copy”, “followup”, and “Followup” commands and “outfolder”.
- set**  
**set variable**  
**set variable=string**  
**set variable=number**  
Define a variable called *variable*. The variable may be given a null, string, or numeric value. “Set” by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for

detailed descriptions of the *mailx* variables.

**shell**

Invoke an interactive shell (see “SHELL”).

**size** [*msglist*]

Print the size in characters of the specified messages.

**source** *filename*

Read commands from the given file and return to command mode.

**top** [*msglist*]

Print the top few lines of the specified messages. If the “toplines” variable is set, it is taken as the number of lines to print. The default is 5.

**touch** [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox*, or the file specified in the MBOX environment variable, upon normal termination. See “exit” and “quit”.

**undelete** [*msglist*]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If “autoprint” is set, the last message of those restored is printed.

**undiscard** [*header-field ...*]

Restore printing of the specified header fields when displaying messages on the screen.

**unset** *variable ...*

Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

**version**

Prints the current version and release date.

**visual** [*msglist*]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the “VISUAL” variable is used to get the name of the editor.

**write** [*msglist*] *filename*

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the “save” command.

**z**[+|-]

Scroll the header display forward or backward one screen–full. The number of headers displayed is set by the “screen” variable.

**TILDE ESCAPES**

The following commands may be entered only from input mode, by beginning a line with the tilde escape character (~). See “escape” for changing this special character.

~! *shell-command*

Escape to the shell.

~: *mail-command*

Perform the command-level request.

~?

Print a summary of tilde escapes.

~A

Insert the autograph string “Sign” into the message.

~a

Insert the autograph string “sign” into the message.

~b *address ...*

Add the *addresses* to the blind carbon copy (Bcc) list. See “askcc”.

~c *address ...*

Add the *addresses* to the carbon copy (Cc) list. See “askbcc”.

~d

Read in the **dead.letter** file. See “DEAD”.

~e

Invoke the editor on the partial message. See “EDITOR”.

- ~f** [*msglist*]  
Forward the specified messages. The messages are inserted into the message without alteration.
- ~h**  
Prompt for Subject, To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.
- ~i** *string*  
Insert the value of the named variable into the text of the message. For example, “~A” is equivalent to “~i Sign”. Environment variables set and exported in the shell are also accessible by “~i”.
- ~m** [*msglist*]  
Insert the specified messages into the letter, shifting the new text to the right one tab stop (see “mprefix”). Valid only when sending a message while reading mail.
- ~p**  
Print the message being entered.
- ~q**  
Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in **dead.letter**. See “DEAD” for a description of this file.
- ~r** *filename*  
**~r !shell-command**  
Read in the specified file. If the argument begins with an exclamation point, the rest of the string is taken as a shell command and is executed, with the standard output inserted into the message.
- ~R**  
Request a return-receipt when the recipient reads the mailbox. The return-receipt is generated if the recipient reads the mail using *post(1)*, AT&T Mail, or OTS.
- ~s** *string* ...  
Set the subject line to *string*.
- ~t** *address* ...  
Add the given *addresses* to the To list.
- ~v**  
Invoke a preferred screen editor on the partial message. See “VISUAL”.
- ~w** *filename*  
**~w !shell-command**  
Write the message without the headers onto the given file. If the argument begins with an exclamation point, the rest of the string is taken as a shell command and is executed, with the standard input being the message without the headers.
- ~x**  
Exit as with **~q** except the message is not saved in **dead.letter**.
- ~|** *shell-command*  
Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

## ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

**HOME**=*directory*

The user’s base of operations.

**MAILRC**=*filename*

The name of the start-up file. Default is **\$HOME/.mailrc**.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the “set” command at any time. The “unset” command may be used to erase variables.

**allnet**

**allnet=uucp**

**allnet=any**

**allnet=header**

All network names whose last component (login name) match are treated as identical. If **uucp** is the argument, all network names whose two last components (system and login name) that match are treated as identical. This causes the *msglist* message specifications to behave similarly. If

**any** is the argument, *user* is treated as a pattern to be matched anywhere in the first line of a message (similar to /bin/mail). **header** is the argument, *user* is treated as a pattern to be used in a case-independent match of either the network address in the first line of a message or the full name, as printed by the “headers” command. Default is “allnet=any”. See also the “alternates” command and the “metoo” variable.

**append**

Upon termination, append messages to the end of the *mbox* file instead of prepending them. Enabled by default.

**askbcc**

Prompt for the Bcc list after message is entered. Default is “noaskbcc”.

**askcc**

Prompt for the Cc list after message is entered. Default is “noaskcc”.

**asksub**

Prompt for subject if it is not specified on the command line with the `-s` option. Enabled by default.

**autoprint**

Enable automatic printing of messages after “delete” and “undelete” commands. Default is “noautoprint”.

**bang**

Enable the special-casing of exclamation points (!) in shell escape command lines as in `vi(1)`. Default is “nobang”.

**cmd=shell-command**

Set the default command for the “pipe” command. No default value.

**crt****crt=number**

Pipe messages having more than *number* lines through the command specified by the value of the “PAGER” variable. Enabled by default.

**DEAD=filename**

The name of the file in which to save partial letters in case of untimely interrupt. Default is **\$HOME/dead.letter**.

**debug**

Enable verbose diagnostics for debugging. Messages are not delivered. Default is “nodebug”.

**dot**

Take a period on a line by itself during input from a terminal as end-of-file. Enabled by default.

**EDITOR=shell-command**

The command to run when the “edit” or “~e” command is used. Default is “ed”.

**escape=c**

Substitute *c* for the ~ escape character.

**flipf**

Reverse the meanings of the “followup” and “Followup” commands in composition mode. Enabled by default.

**flipr**

Reverse the meanings of the “reply” and “Reply” commands in composition mode. Enabled by default.

**folder=directory**

The directory for saving standard mail files. User-specified relative file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), \$HOME is prepended to it. Default is **\$HOME**. See also “outfolder” below.

**header**

Enable printing of the header summary when entering *mailx*. Enabled by default.

**hold**

Hold all messages that are read in the mailbox instead of putting them in the standard *mbox* file. Enabled by default.

**ignore**

Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

**ignoreeof**

Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the “~.” command (see “dot”). Default is “noignoreeof”.

**iprompt**=*string*

Set the composition mode prompt to *string*. Default is no prompt.

**keepsave**

Keep messages that have been saved in other files in the mailbox instead of deleting them. Default is “nokeepsave”.

**LISTER**=*shell-command*

The command (and options) to use when listing the contents of the “folder” directory. The default is ls.

**MBOX**=*filename*

The name of the file to save messages which have been read. The “xit” command overrides this function, as does saving the message explicitly in another file. Default is **\$HOME/mbox**.

**metoo**

If your login appears as a recipient, do not delete it from the list. Default is “nometoo”.

**mprefix**=*string*

Set the leading string to be used with the “~m” command. Default is a tab character.

**onehop**

When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author’s machine for the response. This flag disables alteration of the recipients’ addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away). Disabled by default.

**outfolder**

Causes the files used to record outgoing messages to be located in the directory specified by the “folder” variable unless the path name is absolute. Default is “nooutfolder”. See “folder” and the “Save”, “Copy”, “followup”, and “Followup” commands.

**page**

Used with the “pipe” command to insert a form feed after each message sent through the pipe. Default is “nopage”.

**PAGER**=*shell-command*

The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is “pg -e”.

**prompt**=*string*

Set the command mode prompt to *string*. Default is “? ”.

**quiet**

Don’t print the opening message and version when entering *mailx*. Default is “quiet”.

**record**=*filename*

Record all outgoing mail in *filename*. Disabled by default. See also “outfolder” above.

**save**

Enable saving of messages in **dead.letter** on interrupt or delivery error. See “DEAD” for a description of this file. Enabled by default.

**screen**=*number*

Sets the number of lines in a screen-full of headers for the “headers” command. Default depends on baud rate.

**sendmail**=*shell-command*

Alternate command for delivering messages. Default is **/bin/mail**.

**sendwait**

Wait for background mailer to finish before returning. Enabled by default.

**SHELL**=*shell-command*

The name of a preferred command interpreter. Default is **/bin/sh**.

**showto**

When displaying the header summary and the message is from you, print the recipient’s address instead of the author’s address. Enabled by default.



**sign**=*string*

The variable inserted into the text of a message when the “~a” command is given. No default (see also ~i).

**Sign**=*string*

The variable inserted into the text of a message when the “~A” command is given. No default (see also ~i).

**toplines**=*number*

The number of lines of header to print with the “top” command. Default is 5.

**translate**=*command*

Run the given address(es) through *command* for resolution. Disabled by default.

**VISUAL**=*shell-command*

The name of a preferred screen editor. Default is “vi”.

**FILES**

\$HOME/.mailrc	personal start-up file
\$HOME/mbox	secondary storage file
/usr/mail/*	mailboxes
/usr/lib/mailx.help*	help message files
/usr/lib/mailx.rc	optional global start-up file
/tmp/R[emqsx]*	temporary files

**SEE ALSO**

ls(1), mail(1), post(1), pg(1).

**NAME**

make – maintain collections of programs

**SYNOPSIS**

**make** [ **-f** *makefile* ] [ *option ...* ] [ *name ...* ]

**DESCRIPTION**

*Make* executes recipes in *makefile* to update the target *names* (usually programs). If no target is specified, the targets of the first rule in *makefile* are updated. If no **-f** option is present, *makefile* and *Makefile* are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-f** option may appear.

*Make* updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist. The prerequisites are updated before the target.

The makefile comprises a sequence of rules and macro definitions. The first line of a rule is a blank-separated list of targets, then a single or double colon, then a list of prerequisite files terminated by semicolon or newline. Text following a semicolon, and all following lines that begin with a tab, are shell commands: the recipe for updating the target.

If a name appears as target in more than one single-colon rule, it depends on all of the prerequisites of those rules, but only one recipe may be specified among the rules. A target in a double-colon rule is updated by the following recipe only if it is out of date with respect to the prerequisites of that rule.

Two special forms of name are recognized. A name like *a(b)* means the file named *b* stored in the archive named *a*. A name like *a((b))* means the file stored in archive *a* and containing the entry point *b*.

Sharp and newline surround comments.

In this makefile *pgm* depends on two files *a.o* and *b.o*, and they in turn depend on *.c* files and a common file *ab.h*:

```
pgm: a.o b.o
    cc a.o b.o -lplot -o pgm

a.o: ab.h a.c
    cc -c a.c

b.o: ab.h b.c
    cc -c b.c
```

Makefile lines of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of  $\$(string1)$  are replaced by *string2*. If *string1* is a single character, the parentheses are optional;  $\$\$$  is replaced by  $\$$ . Each entry in the environment (see [sh\(1\)](#)) of the *make* command is taken as a macro definition, as are command arguments with embedded equal signs.

Lines of the form *string1 := string2* occurring in a recipe are assignments: macro definitions that are made in the course of executing the recipe.

A target containing a single **%** introduces a pattern rule, which controls the making of names that do not occur explicitly as targets. The **%** matches an arbitrary string called the stem: *A%B* matches any string that begins with *A* and ends with *B*. A **%** in a prerequisite name stands for the stem; and the special macro  $\$\%$  stands for the stem in the recipe. A name that has no explicit recipe is matched against the target of each pattern rule. The first pattern rule for which the prerequisites exist specifies further dependencies.

The following pattern rule maintains an object library where all the C source files share a common include file *defs.h*.

```
arch.a(%o) : %.c defs.h
    cc -c  $\$\%$ .c
    ar r arch.a  $\$\%$ .o
    rm  $\$\%$ .o
```

A set of default pattern rules is built in, and effectively follows the user's list of rules. Assuming these rules, which tell, among other things, how to make *.o* files from *.c* files, the first example becomes:

```

pgm: a.o b.o
      cc a.o b.o -lplot -o pgm
a.o b.o: ab.h

```

Here, greatly simplified, is a sample of the built-in rules:

```

CC = cc
%.o: %.c
      $(CC) $(CFLAGS) -c $%.c
%.o: %.f
      f77 $(FFLAGS) -c $%.f
% : %.c
      $(CC) $(CFLAGS) -o $% $%.c

```

The first rule says that a name ending in `.o` could be made if a matching name ending in `.c` were present. The second states a similar rule for files ending in `.f`. The third says that an arbitrary name can be made by compiling a file with that name suffixed by `.c`.

Macros make the builtin pattern rules flexible: **CC** names the particular C compiler, **CFLAGS** gives `cc(1)` options, **FFLAGS** for `f77(1)`, **LFLAGS** for `lex(1)`, **YFLAGS** for `yacc(1)`, and **PFLAGS** for `pascal(A)`

An older, now disparaged, means of specifying default rules is based only on suffixes. Prerequisites are inferred according to selected suffixes listed as the ‘prerequisites’ for the special name **.SUFFIXES**; multiple lists accumulate; an empty list clears what came before.

The rule to create a file with suffix `s2` that depends on a similarly named file with suffix `s1` is specified as an entry for the ‘target’ `s1s2`. Order is significant; the first possible name for which both a file and a rule exist is inferred. An old style rule for making optimized `.o` files from `.c` files is

```

.SUFFIXES: .c .o
.c.o: ; cc -c -O -o $ $*.c

```

The following two macros are defined for use in any rule:

```

$( $ )    full name of target
$( $ / )  target name beginning at the last slash, if any

```

A number of other special macros are defined automatically in rules invoked by one of the implicit mechanisms:

```

$*        target name with suffix deleted
$         full target name
$<        list of prerequisites in an implicit rule
$?        list of prerequisites that are out of date
$^        list of all prerequisites

```

The following are included for consistency with System V:

```

$( D )    directory part of $ (up to last slash)
$( F )    file name part of $ (after last slash)
$( * D )  directory part of $* (up to last slash)
$( * F )  file name part of $* (after last slash)
$( < D )  directory part of $< (up to last slash)
$( < F )  file name part of $< (after last slash)

```

Recipe lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target **.SILENT** is in the makefile, or the first character of the command is `.`

Commands that return nonzero status cause *make* to terminate unless the special target **.IGNORE** is in the makefile or the command begins with `<tab><hyphen>`.

Interrupt and quit cause the target to be deleted unless the target depends on the special name **.PRECIOUS**.

*Make* includes a rudimentary parallel processing ability. If the separation string is `:&` or `::&`, *make* can run the command sequences to create the prerequisites simultaneously. If two names are separated by an ampersand on the right side of a colon, those two may be created in parallel.

Other options:

- i** Equivalent to the special entry `.IGNORE:`
- k** When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.
- n** Trace and print, but do not execute the commands needed to update the targets.
- t** Touch, i.e. update the modified date of targets, without executing any commands.
- r** Turn off built-in rules.
- s** Equivalent to the special entry `.SILENT:`.
- e** Environment definitions override conflicting definitions in arguments or in makefiles. Ordinary precedence is argument over makefile over environment.
- o** Assume old style default suffix list: `.SUFFIXES: .out .o .c .e .r .f .y .l .s .p`
- Pn** Permit *n* command sequences to be done in parallel with `&`.
- z** Run commands by passing them to the shell; normally simple commands are run directly by [exec\(2\)](#).

## FILES

makefile  
Makefile

## SEE ALSO

[sh\(1\)](#), [touch](#) in [chdate\(1\)](#), [ar\(1\)](#), [mk\(1\)](#)

## BUGS

Comments can't appear on recipe lines.  
Archive entries are not handled reliably.

**NAME**

`man` – print pages of this manual

**SYNOPSIS**

`man` [ *option ...* ] [ *chapter* ] *title ...*

`man -k` *pattern*

**DESCRIPTION**

*Man* locates and prints pages of this manual named *title* in the specified *chapter*. *Title* is given in lower case. The *chapter* number is a single digit, 1-9; pages marked (3S) or (9.1), for example, belong to chapters 3 and 9 respectively. If no *chapter* is specified, pages in all chapters, including the unprinted appendix, are printed.

On some machines a cache of preformatted pages is available. If so, and if neither option `-t` or `-n` is present, option `-q`, for quick printing, is assumed.

The options are:

- `-q` Copy cached pages to the standard output or, if they are out of date or unavailable, act as `-n`. Any name from the NAME list at the top of the page will serve as a *title* in the cache. If the standard output is directed to a terminal, filter it through [ul\(1\)](#).
- `-t` Use [troff\(1\)](#) to place on the standard output intermediate code to drive the typesetting devices of [lp\(1\)](#), [apsend\(1\)](#), or [proof\(9\)](#)
- `-n` Print the pages on the standard output using *nroff*.
- `-f` The *titles* are actual names of manual source files. Assume option `-t` unless `-n` is present.
- `-k` Report the NAME lines of all manual pages that matches the *pattern*. The *pattern* is as in [egrep](#); see [gre\(1\)](#).

Under `-t` and `-n` further options, e.g. to specify the kind of terminal you have, are passed on to *troff* or *nroff*.

From time to time a daemon updates cached pages that are out of date and supplies links for subsidiary entries.

**EXAMPLES**

`man man`

Reproduce this page and [man\(6\)](#).

`man -t eqn eqnchar | lp`

Format the *eqn* and *eqnchar* pages and send them to the laser printer.

`man -k 'copy.*file|file.*copy'`

Look for file-copying utilities.

**FILES**

`/usr/man/man?/*.*?`

*troff* source for manual; this page is `/usr/man/man1/man.1`

`/usr/spool/man/man?/*.*?`

manual cache

`/usr/man/man0/cache`

cache daemon; invoked by [cron\(8\)](#)

**SEE ALSO**

[troff\(1\)](#), [man\(6\)](#), [apsend\(1\)](#), [lp\(1\)](#), [proof\(9\)](#)

**BUGS**

The manual was intended to be typeset; some detail is sacrificed on terminals.

You can't ask for manual pages named 1 through 9.

Pages not available in the cache can be located by their proper titles only.

Cache entries by subsidiary names are always deemed up to date.



**SEE ALSO**

*Maple: A Sample Interactive Session* issued by the Symbolic Computation Group as Research Report CS-85-01 available from the Department of Computer Science, University of Waterloo,  
*Maple User's Guide* by B.W. Char et al, Watcom Publications Limited, Waterloo, Ontario (1985).

[mint\(1\)](#)

**FILES**

.mapleinit

/usr/maple/lib – Maple library (Pathname subject to change at each installation.)

**AUTHOR**

Symbolic Computation Group, University of Waterloo

**FOR HELP**

At Waterloo, there is the newsgroup uw.maple which contains broadcasts and discussions which would be of interest to general Maple users. You should subscribe to this newsgroup if you intend to use Maple in more than just a casual manner. Users are encouraged to post their questions regarding Maple to this newsgroup if they feel that their enquiries are of a general nature. Replies will be posted to the newsgroup for all to see. If you have a question that you think is of a very specific nature and not of interest to others, you may send a mail message to maple\_help watmum.

**NAME**

match – compare style tables from two or more texts

**SYNOPSIS**

**match** [ **-flags** ] [ **-ver** ] [style-file1 [style-file2 ...]

**DESCRIPTION**

*Match* collates selected variables from tables produced by the *style(1)* command and prints values from the different files one below the other for easy comparison. The *style-files* must contain tables produced by *style*.

*Match* can also run on one file to produce an abbreviated version of the *style* table.

When comparing texts, it is advisable to use texts of similar length.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**USES**

This program is useful for visually inspecting similarities and differences among stylistic features of different documents or drafts and their revisions.

**SEE ALSO**

*style(1)*, *prose(1)*.

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452



**NAME**

matlab – interactive matrix desk calculator

**SYNOPSIS**

**/usr/lbin/matlab**

**DESCRIPTION**

*Matlab* manipulates complex matrices interactively. Special cases include real matrices and scalars. Operations include pseudoinversion (which is inversion for square nonsingular matrices), eigendecomposition, various other factorizations, solution of linear equations, matrix products (including inner and outer products), Kronecker products, log, exponential and trigonometric functions of matrices, and rank and condition estimation.

Variables are alphanumeric strings of at most 4 characters. Case is ignored. Expressions and assignment statements are written as in Fortran. Multiple statements can be put on one line, separated by either comma or semicolon; the result of a statement is written on the standard output unless a semicolon follows the statement. Extensions to Fortran notation include:

*Matrix construction* from elements. Elements in a row are separated by commas; columns are separated by semicolons; matrices are surrounded by < brackets.

*Transpose* is indicated by postfix prime ' .

*Consecutive integers* are denoted by colons in the style 1:4 or 1:2:8 (meaning 1,3,5,7).  $A(2:5)$  is a subarray;  $A(:,j)$  is a column.

*Identity matrix* is denoted `eye`; its dimensions are dictated by context.

*Reverse division* is denoted by `\`. For example,  $x$  is roughly the same as  $x = \text{inv}(A) * b$ , except that Gaussian elimination, if applicable, is used to compute  $x$ .

Some *matlab* commands:

**help**

**help** *word*

List commands and functions, or specific information about a *word*:

ABS ANS ATAN BASE CHAR CHOL CHOP CLEA COND CONJ COS  
DET DIAG DIAR DISP EDIT EIG ELSE END EPS EXEC EXIT  
EXP EYE FILE FLOP FLPS FOR FUN HESS HILB IF IMAG  
INV KRON LINE LOAD LOG LONG LU MACR MAGI NORM ONES  
ORTH PINV PLOT POLY PRIN PROD QR RAND RANK RCON RAT  
REAL RETU RREF ROOT ROUN SAVE SCHU SHOR SEMI SIN SIZE  
SQRT STOP SUM SVD TRIL TRIU USER WHAT WHIL WHO WHY

**save**(' *file*')

**save**(' *file*[,*var*]...')

Save all current variables, or just the designated variables in *file*.

**load**(' *file*')

Restore saved variables.

**exec**(' *file*')

Execute the commands in *file* before reading more commands from the standard input.

*Matlab* can be called as a subroutine. For details, see the reference or type `help user`.

**FILES**

/usr/lib/mathhelp.dac

/usr/lib/mathhelp.idx

**SEE ALSO**

Cleve Moler, *MATLAB User's Guide*, Technical Report CS81-1 (Revised), Dept. of Computer Science, University of New Mexico, 1982. (Available in troff form with the Matlab source.)

**NAME**

memo – produce macros for MM interactively

**SYNOPSIS**

**memo** *filename* [ *argument* ]

**DESCRIPTION**

*Memo* is designed as a computer aid to help new or infrequent users of the PWB/MM Memorandum Macros. It asks questions interactively to produce a file which contains the necessary introductory and concluding macros for PWB/MM. In order to enter the text of the memo, you must enter the editor. For help to any question enter a *A detailed explanation of the proper* answer will be printed.

The following are optional arguments to be used when filename already exists to avoid lengthy printout:

- a**     Enter editor to add/modify text.
- b**     Be prompted for concluding macros.
- !**     Overwrite filename and begin introductory macros.

**SEE ALSO**

*nroff*(1)

**NAME**

mint – produce usage report from a maple program

**SYNOPSIS**

mint [ **-i** info\_level ] [ **-l** ] [ **-d** library\_database ] [ **-a** database\_file ] [ **-q** ] [ file ]

**DESCRIPTION**

*Mint* produces a report about possible errors in a Maple source file and also reports about how variables are used in the file. If *file* is not given, then the standard input file is used to read Maple source statements. Unlike *maple*, *mint* is not terminated when it reads a quit statement. It is terminated when it reaches the end of file. When started, *mint* normally produces a mint leaf logo. This can be suppressed by the use of the **-q** (quiet) option. The amount of information to be produced by *mint* is specified by the *info\_level* argument. The values allowed for this argument are:

- 0 – Display no information.
- 1 – Display only severe errors
- 2 – Display severe and serious errors
- 3 – Display warnings as well as severe and serious errors
- 4 – Give a full report on variable usage as well as displaying errors and warnings

A report for each procedure in the file is displayed separately followed by a global report for statements not contained within any procedure. If the severity of errors found within a procedure is less than what *info\_level* specifies, then no report is produced for that procedure. In all cases, the most severe error found in the file will be used to set the exit status for *mint*. Thus, by using an *info\_level* of 0, *mint* can be used to determine the severity of errors in a file without actually producing any output at all. If no value is given for *info\_level* on the command line, a default value of 2 (severe and serious errors) is used. The types of errors and warnings found are classified as severe, serious, and warning. A severe error is an undisputable error. A serious error is almost certainly an error. However, persons defining procedures for addition to the Maple library may choose to ignore these “errors”. Warnings are possible errors. They point to constructs that may be correct in some contexts, but probable errors in other contexts. The types of errors and warnings produced are:

**SEVERE**

Syntax errors

A caret symbol will point to the token that is being read when the error occurred.

Duplicated parameter

A name appears more than once in a parameter list for a procedure.

Duplicated local

A name is declared more than once in the list of local variables for a procedure.

Local variable and parameter conflict

A name is used both as a parameter and a local variable within a procedure. In further analysis, the name is treated as a parameter.

Local variable and system-defined name conflict

The name of a local variable is also used by Maple as a system-defined name.

Parameter and system-defined name conflict

The name of a parameter is also used by Maple as a system-defined name.

Duplicated loop name

A loop nested within another loop uses as its loop control variable the same name that the outer loop uses.

Break or next statement outside of a loop

A break or a next statement occurs outside of any loop. (Break or next may still be used as names within an expression outside of a loop.)

RETURN or ERROR function call outside of a procedure

A function call to RETURN or ERROR occurs outside of a procedure body. (RETURN or ERROR may still be used as names if they are not invoked as functions.)

Unreachable code

There are statements which follow directly after a goto type of statement. These statements are unreachable and will never be executed. A goto statement is a next statement, a break statement, a quit, stop, or done statement, a RETURN() call, an ERROR() call. An if statement all branches of which end in a goto statement is also considered a goto statement.

**SERIOUS**

## Overly long name

A name whose length is too long is used. The length of the name is truncated to the maximum allowed.

## Unused local variable

A local variable is declared for a procedure but never used within the procedure body.

## Local variable assigned a value but not used otherwise

A local variable is assigned a value within a procedure but is not otherwise used.

## Local variable never assigned a value but used as a value

A local variable was never assigned a value in a procedure but within the procedure its value is used in an expression. Such an expression would contain a pointer to a non-existent local variable if the expression were returned or assigned to a global variable.

## System-defined name is overwritten

A name which is treated as a system-defined name by Maple is assigned a value. The class of system-defined names includes names which are special names for the Maple kernel, e.g., true and Digits, names of built-in functions, e.g., anames and lprint, names of functions which are automatically readlib-defined, e.g., cat or help. Also included are names that are special to routines for evalf, diff, expand, etc. Examples of these are Pi and sinh. These special names generally should not be assigned a value in order for some library routines to work properly. Included in the report is an indication of which parts of Maple use the system-defined names.

## Dubious global name in a procedure

A global name is used within a procedure. A global name is a name which is not a parameter, a local name, a system-defined name, or a catenated name. A quoted name used as an argument to the routines lprint, print, and ERROR is probably used just for output and is not considered a name. Global names used as procedure names in a function call are not considered errors. Also excluded are names of files in the Maple library, e.g., 'convert/ratpoly'. All remaining names are considered as global names. By convention, global names used in a package of routines should begin with the '\_' (underscore) character. Those that do not are considered dubious and are reported here.

## Library file name overwritten

The name of a library file, e.g., 'convert/ratpoly', is assigned a value. It is usual for the name of a library file to also be the name of a library function. Hence, the library function 'convert/ratpoly' is no longer accessible. (The -I (library file) option will downgrade these messages from a serious error to a report.)

## Unused parameter in a procedure

A name specified in the parameter list of a procedure is never used in the procedure. This is considered a serious error if 'args' is never used in the procedure either. If args is used in the procedure, then it's possible that the parameter may be accessed through a construct using 'arg' and this error is downgraded to a warning.

## Wrong argument count in a procedure call

The number of arguments passed in a procedure call doesn't match the number of formal arguments in the definition of a procedure of the same name recorded in the library database file. A library database file (cf. **DATABASE FILES**) contains information about the minimum number of arguments expected for a procedure, the maximum number of arguments, whether 'nargs' is used in the procedure body, and the name of the file in which the procedure is defined. If the number of actual arguments passed is either less than the minimum arguments expected or more than the maximum number expected *and* 'nargs' is not used in the procedure body, then a warning is generated. This warning is suppressed if one of the arguments passed is 'args'. It is a common practice for a procedure to take its argument list, contained in the expression sequence 'args', and pass that on to other procedures. What appears to *mint* as one argument is in reality a sequence of arguments.

**WARNING**

## Equation used as a statement

This may be intentional. On the other hand, it's common for many Fortran and C programmers to mistype '=' for the assignment operator which is ':=' in Maple.

## Unused parameter in a procedure

See similar entry under serious errors.

Global name used

A global name which may or may not start with ' \_ ' is used within this procedure.

Catenated name used

A name is formed through the catenation operator.

## OTHER REPORTS

If *info\_level* is 4, then a usage report is given for each procedure as well as global statements within the file. Each usage report shows how parameters, local variables, global variables, system-defined names and catenated names are used. As well can easily be done, the following information about how a variable is used may be provided:

1. Used as a value
2. Used as a table or list element
3. Used as a call-by-value parameter
4. Used as a call-by-name parameter (a quoted parameter)
5. Called as a function
6. Assigned a procedure
7. Assigned a list
8. Assigned a set
9. Assigned a range
10. Assigned a value as a table or list element
11. Assigned a function value

(assigned a value to remember as a function value)

In addition, a list of all the error messages generated is given.

## COMMAND OPTIONS

The **-i** (info level) and **-q** (quiet) options are explained above. The **-I** (library file) option will suppress the catenated name warning and the global name warning if only one of each is used outside of any procedure. Typically, a Maple library source file will contain one of each for use in loading the library file. This option will also suppress error messages about library file names being overwritten since one of the purposes of a library file is to assign a procedure to a library file name. Moreover, warnings about the assignment of values to the system-defined names *Digits* and *printlevel* are suppressed since this often happens in a library file.

## INITIALIZATION FILE

If there is a file named *.mintrc* in your home directory, *mint* will read this file for command line options. This file may contain several lines containing command line options or arguments as you would type them on a command line. Since *mint* reads this file and then scans the actual command line, arguments on the actual command line can override arguments in the initialization file. A good use of the initialization file may be to enter the name of the Maple library procedure database file when using the **-d** option, obviating the need to type this each time *mint* is used.

## PROCEDURE DATABASE FILES

A procedure database file contains information about the definition of procedures which is useful in ensuring that these procedures are used correctly. Each line in a database file contains the following:

<name> <min args> <max args> <nargs used> <file name>

where <name> is a legal Maple name without any embedded blanks, <min args> is the minimum number of arguments expected for <name>, <max args> is the maximum number of arguments, <nargs used> is 1 if 'nargs' is used in the procedure body for <name> and 0 otherwise, <file name> is the name of the file in which <name> is defined. The entries on each line are in free format but must be separated from one another by at least one space character. The values for <min args> and <max args> should be numbers in the range 0 to 999. If <max args> is 999 for an entry, that denotes that the procedure has no upper limit on the number of arguments. There may be multiple entries for a particular procedure. Later entries supersede earlier ones. A procedure database file for the entire Maple library is generated or updated periodically. This file is */usr/maple/data/mint.db* and contains close to 1200 entries and it takes *mint* about 7 seconds to read this file. A private database file can be generated through the use of the **-a** command line option for *mint*. A file name must follow **-a** on the command line and is taken to be a procedure database file. As *mint* scans procedure definitions in the input file, it will append procedure database entries into the database file. For information gathered automatically by *mint* about a procedure, <min args> and

<max args> will both be the number of formal arguments used in the procedure definition. You can edit the database file to adjust these values. Remember that use of 'nargs' in a procedure body sets the <nargs seen> field to 1 in the database entry and that this will turn off argument count checking for that procedure.

### EXAMPLES

```
mint -d /usr/maple/data/mint.db -a my.db -i 4 rat_poisson
mint -d /usr/maple/data/mint.db -d my.db      rat_trap
mint -i 1 -q                                  warfarin
```

The first example gives a full report (info\_level = 4) for the Maple source file `rat_poisson`. It reads the Maple library database file and uses this to check that procedures defined in the Maple library are called with the correct number of arguments. Information about procedures defined in `rat_poisson` is *appended* to `my.db`. In the second example, both the Maple library database file and the private database file `my.db` are used to check number of arguments used in procedure calls in the file `rat_trap`. Entries in `my.db` supersede entries in the library database file if the name of a library procedure has been redefined in `my.db`. In the third example, no argument count checking is done. Since the info\_level is set to 1, only severe errors are reported. Since the `-q` (quiet) option is used, the printing of the *mint* leaf logo is suppressed in the output.

### FILES USED

`.mintrc` – Mint initialization file `/usr/maple/data/mint.db` – Maple library procedure database  
(The location of the database may be different for each site)

### SEE ALSO

`maple`

### STATUS

*Mint* will return an exit status of 1, 2, or 3 if the worst error it detects is a warning, serious error, or severe error, respectively. An exit status of 0 is returned if no errors or warnings are found.

**NAME**

mk, mkconv, membername – maintain (make) related files

**SYNOPSIS**

**mk** [ **-f** *mkfile* ] ... [ *option* ... ] [ *name* ... ]

**mkconv** *makefile*

**membername** *aggregate* ...

**DESCRIPTION**

*Mk* is most often used to keep object files current with the source they depend on.

*Mk* reads *mkfile* and builds and executes dependency dags (directed acyclic graphs) for the target *names*. If no target is specified, the targets of the first non-metarule in the first *mkfile* are used. If no **-f** option is present, *mkfile* is tried. Other options are:

- a** Assume all targets to be out of date. Thus, everything gets made.
- d[egp]** Produce debugging output (**p** is for parsing, **g** for graph building, **e** for execution).
- e** Explain why each target is made.
- i** Force any missing intermediate targets to be made.
- k** Do as much work as possible in the face of errors.
- m** Generate an equivalent makefile on standard output. Recipes are not handled well.
- n** Print, but do not execute, the commands needed to update the targets.
- t** Touch (update the modified date of) non-virtual targets, without executing any recipes.
- u** Produce a table of clock seconds spent with *n* recipes running.
- wname1,name2,...**  
Set the initial date stamp for each name to the current time. The names may also be separated by blanks or newlines. (Use with **-n** to find what else would need to change if the named files were modified.)

*Mkconv* attempts to convert a [make\(1\)](#) *makefile* to a *mkfile* on standard output. The conversion is not likely to be faithful.

The shell script *membername* extracts member names (see ‘Aggregates’ below) from its arguments.

**Definitions**

A *mkfile* consists of *assignments* (described under ‘Environment’) and *rules*. A rule contains *targets* and a *tail*. A target is a literal string, or *label*, and is normally a file name. The tail contains zero or more *prerequisites* and an optional *recipe*, which is a shell script.

A *metarule* has a target of the form *A%B* where *A* and *B* are (possibly empty) strings. A metarule applies to any label that matches the target with **%** replaced by an arbitrary string, called the *stem*. In interpreting a metarule, the stem is substituted for all occurrences of **%** in the prerequisite names. A metarule may be marked as using regular expressions (described under ‘Syntax’). In this case, **%** has no special meaning; the target is interpreted according to [regexp\(3\)](#). The dependencies may refer to subexpressions in the normal way, using **\n**. The *dependency dag* for a target consists of *nodes* connected by directed *arcs*. A node consists of a label and a set of arcs leading to prerequisite nodes. The root node is labeled with an original target *name*.

**Building the Dependency Dag**

Read the *mkfiles* in command line order and distribute rule tails over targets to get single-target rules.

For a node *n*, for every rule *r* that matches *n*’s label generate an arc to a prerequisite node. The node *n* is then marked as done. The process is then repeated for each of the prerequisite nodes. The process stops if *n* is already done, or if *n* has no prerequisites, or if any rule would be used more than **\$NREP** times on the current path in the dag. A probable node is one where the label exists as a file or is a target of a non-metarule.

After the graph is built, it is checked for cycles, and subdags containing no probable nodes are deleted. Also, for any node with arcs generated by a non-metarule with a recipe, arcs generated by a metarule with a recipe are deleted. Disconnected subdags are deleted.

## Execution

Labels have an associated date stamp. A label is *ready* if it has no prerequisites, or all its prerequisites are made. A ready label is *trivially uptodate* if it is not a target and has a nonzero date stamp, or it has a nonzero date stamp, and all its prerequisites are made and predate the ready label. A ready label is marked *made* (and given a date stamp) if it is trivially uptodate or by executing the recipe associated with the arcs leading from the node associated with the ready label. The **P** attribute can be used to generalize *mk*'s notion of determining if prerequisites predate a label. Rather than comparing date stamps, it executes a specified program and uses the exit status.

Date stamps are calculated differently for virtual labels, for labels that correspond to extant files, and for other labels. If a label is *virtual* (target of a rule with the **V** attribute), its date stamp is initially zero and upon being made is set to the most recent date stamp of its prerequisites. Otherwise, if a label is nonexistent (does not exist as a file), its date stamp is set to the most recent date stamp of its prerequisites, or zero if it has no prerequisites. Otherwise, the label is the name of a file and the label's date stamp is always that file's modification date.

Nonexistent labels which have prerequisites and are prerequisite to other label(s) are treated specially unless the **-i** flag is used. Such a label *l* is given the date stamp of its most recent prerequisite and if this causes all the labels which have *l* as a prerequisite to be trivially uptodate, *l* is considered to be trivially uptodate. Otherwise, *l* is made in the normal fashion.

Two recipes are called identical if they arose by distribution from a single rule as described above. Identical recipes may be executed only when all their prerequisite nodes are ready, and then just one instance of the identical recipes is executed to make all their target nodes.

Files may be made in any order that respects the preceding restrictions.

A recipe is executed by supplying the recipe as standard input to the command

```
/bin/sh -e
```

The environment is augmented by the following variables:

<b>\$alltarget</b>	all the targets of this rule.
<b>\$newprereq</b>	the prerequisites that caused this rule to execute.
<b>\$nproc</b>	the process slot for this recipe. It satisfies $0 \leq \$nproc < \$NPROC$ , where <b>\$NPROC</b> is the maximum number of recipes that may be executing simultaneously.
<b>\$pid</b>	the process id for the <i>mk</i> forking the recipe.
<b>\$prereq</b>	all the prerequisites for this rule.
<b>\$stem</b>	if this is a metarule, <b>\$stem</b> is the string that matched <b>%</b> . Otherwise, it is empty. For regular expression metarules, the variables <i>stem0</i> , ..., <i>stem9</i> are set to the corresponding subexpressions.
<b>\$target</b>	the targets for this rule that need to be remade.

Unless the rule has the **Q** attribute, the recipe is printed prior to execution with recognizable shell variables expanded. To see the commands print as they execute, include a *set* in your rule. Commands returning nonzero status (see *intro(1)*) cause *mk* to terminate.

## Aggregates

Names of the form *a(b)* refer to member *b* of the aggregate *a*. Currently, the only aggregates supported are *ar(1)* archives.

## Environment

Rules may make use of shell (or environment) variables. A legal shell variable reference of the form **\$OBJ** or **\${name}** is expanded as in *sh(1)*. A reference of the form **\${name:A%B=C%D}**, where *A*, *B*, *C*, *D* are (possibly empty) strings, has the value formed by expanding **\$name** and substituting *C* for *A* and *D* for *B* in each word in **\$name** that matches pattern *A%B*.

Variables can be set by assignments of the form

```
var=[attr=]tokens
```

where *tokens* and the optional attributes are defined under 'Syntax' below. The environment is exported to recipe executions. Variable values are taken from (in increasing order of precedence) the default values below, the environment, the mkfiles, and any command line assignment. A variable assignment argument



overrides the first (but not any subsequent) assignment to that variable.

```

AS=as                FFLAGS=                NPROC=1
CC=cc                LEX=lex                NREP=1
CFLAGS=              LFLAGS=                YACC=yacc
FC=f77               LDFLAGS=               YFLAGS=
BUILTINS='
%.o: %.c
    $CC $CFLAGS -c $stem.c
%.o: %.s
    $AS -o $stem.o $stem.s
%.o: %.f
    $FC $FFLAGS -c $stem.f
%.o: %.y
    $YACC $YFLAGS $stem.y &&
    $CC $CFLAGS -c y.tab.c && mv y.tab.o $stem.o; rm y.tab.c
%.o: %.l
    $LEX $LFLAGS -t $stem.l > $stem.c &&
    $CC $CFLAGS -c $stem.c && rm $stem.c'
ENVIRON=

```

The builtin rules are obtained from the variable **BUILTINS** after all input has been processed. The **ENVIRON** variable is split into parts at control-A characters, the control-A characters are deleted, and the parts are placed in the environment. The variable **MKFLAGS** contains all the option arguments (arguments starting with - or containing =) and **MKARGS** contains all the targets in the call to *mk*.

## Syntax

Leading white space (blank or tab) is ignored. Input after an unquoted # (a comment) is ignored as are blank lines. Lines can be spread over several physical lines by placing a \ before newlines to be elided. Non-recipe lines are processed by substituting for *cmd* and then substituting for variable references. Finally, the filename metacharacters []\*? are expanded. Quoting by ", "'", and \ is supported. The semantics for substitution and quoting are given in *sh(1)*.

The contents of files may be included by lines beginning with < followed by a filename.

Assignments and rule header lines are distinguished by the first unquoted occurrence of : (rule header) or = (assignment).

A rule definition consists of a header line followed by a recipe. The recipe consists of all lines following the header line that start with white space. The recipe may be empty. The first character on every line of the recipe is elided. The header line consists of at least one target followed by the rule separator and a possibly empty list of prerequisites. The rule separator is either a single : or is a : immediately followed by attributes and another :. If any prerequisite is more recent than any of the targets, the recipe is executed. This meaning is modified by the following attributes

- < The standard output of the recipe is read by *mk* as an additional mkfile. Assignments take effect immediately. Rule definitions are used when a new dependency dag is constructed.
- D** If the recipe exits with an error status, the target is deleted.
- N** If there is no recipe, the target has its time updated.
- P** The characters after the **P** until the terminating : are taken as a program name. It will be invoked as **sh -c prog 'arg1' 'arg2'** and should return 0 exit status if and only if arg1 is not out of date with respect to arg2. Date stamps are still propagated in the normal way.
- Q** The recipe is not printed prior to execution.
- R** The rule is a metarule using regular expressions.
- U** The targets are considered to have been updated even if the recipe did not do so.
- V** The targets of this rule are marked as virtual. They are distinct from files of the same name.

Similarly, assignments may have attributes terminated by =. The only assignment attribute is

**U**

Do not export this variable to recipe executions.

**EXAMPLES**

A simple mkfile to compile a program.

```
prog: a.o b.o c.o
      $CC $CFLAGS -o $target $prereq
```

Override flag settings in the mkfile.

```
$ mk target CFLAGS='-O -s'
```

To get the prerequisites for an aggregate.

```
$ membername 'libc.a(read.o)' 'libc.a(write.o)'
read.o write.o
```

Maintain a library.

```
libc.a(%o):N:%o
libc.a:libc.a(abs.o) libc.a(access.o) libc.a(alarm.o) ...
      names='membername $newprereq'
      ar r libc.a $names && rm $names
```

Backquotes used to derive a list from a master list.

```
NAMES=alloc arc bquote builtins expand main match mk var word
OBJ='echo $NAMES|sed -e 's/[^ ][^ ]*/&.o/g''
```

Regular expression metarules. The single quotes are needed to protect the \s.

```
'([^\s]*)/(\.*)\.o':R: '\1/\2.c'
      cd $stem1; $CC $CFLAGS -c $stem2.c
```

A correct way to deal with [yacc\(1\)](#) grammars. The file **lex.c** includes the file **x.tab.h** rather than **y.tab.h** in order to reflect changes in content, not just modification time.

```
YFLAGS=-d
lex.o: x.tab.h
x.tab.h:y.tab.h
      cmp -s x.tab.h y.tab.h || cp y.tab.h x.tab.h
y.tab.c y.tab.h:gram.y
      $YACC $YFLAGS gram.y
```

The above example could also use the **P** attribute for the **x.tab.h** rule:

```
x.tab.h:Pcmp -s:y.tab.h
      cp y.tab.h x.tab.h
```

**SEE ALSO**

[make\(1\)](#), [chdate\(1\)](#), [sh\(1\)](#), [regex\(3\)](#)

A. Hume, 'Mk: a Successor to Make', this manual, Volume 2

**BUGS**

Identical recipes for regular expression metarules only have one target.

Seemingly appropriate input like **CFLAGS=-DHZ=60** is parsed as an erroneous attribute; correct it by inserting a space after the first =.

**NAME**

mkdir – make a directory

**SYNOPSIS**

**mkdir** *dirname* ...

**DESCRIPTION**

*Mkdir* creates specified directories in mode *777*, subject to masking by *umask(2)*. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

*Mkdir* requires write permission in the parent directory.

**SEE ALSO**

*rm(1)* *cd* in *sh(1)*

**DIAGNOSTICS**

*Mkdir* returns exit code 0 if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

**NAME**

mkdist, insdist — make and install distribution packages

**SYNOPSIS**

```
/usr/lib/dist/mkdist [-v] [-D old=new] [-[Xx] command] files ...
```

```
/usr/lib/dist/insdist [-v] [-D old=new] [-R rootdir]
```

**DESCRIPTION**

*Mkdist* packages the named files into a distribution package on the standard output. A distribution package is an ordinary *tar(1)* file, containing the files of the package as well as installation information for use by *insdist*.

The **-D** option to both *mkdist* and *insdist* allows prefix substitution to be performed on pathnames going into the distribution. (*Mkdist* arranges for all file names in the distribution to be absolute path names, by prepending the current directory to any relative pathname arguments.) At most one **-D** option will be applied to any name in the distribution, so there are no substitution loops. If multiple **-D** options might match a given file name, the leftmost one from the command line is chosen. The **-R** option (*insdist* only) additionally specifies that all files are to be unpacked relative to the given root directory.

The **-X** and **-x** options to *mkdist* allow a command to be given that will be executed when the distribution is unpacked by *insdist*. These options are identical, except that pathname prefix substitution from **-D** options will be applied to a command specified in a **-X** option.

The **-v** option turns on verbose output describing what's going on.

**SEE ALSO**

*tar(1)*

**BUGS**

The **-v** option should show more.

**NAME**

mkstand – compile style standards for prose program

**SYNOPSIS**

**mkstand** [ **-flags** ] [ **-ver** ] [ **-mm** | **-ms** ] [ **-li** | **+li** ] [ **-o** *outfile* ] file1 file2 ...

**DESCRIPTION**

*Mkstand* enables users to compile their own set of *style*(1) standards for use by *prose*(1). *Prose* describes stylistic features of a text and compares them to specified standards. If a user or group, for example a writing group, has many documents of a certain type that they consider good, those documents can be used as the basis for their own standards. *Mkstand* creates the standards, which reflect the stylistic features of the input documents. Then the user can use *prose* to evaluate documents according to those standards.

*Mkstand* runs *style* on a set of documents and computes the means and standard deviations of certain *style* statistics. Then it puts these into *outfile* (*stand.out* is the default) in a format that *prose* can read. Then if *prose* is run with the command:

```
prose -x outfile textfile
```

it compares *textfile* with the standards in *outfile*. The command:

```
wbstand -x outfile
```

will display the standards in a comprehensible form.

*Mkstand* tries to produce valid standards by enforcing these requirements:

1. Input files must be at least 90 sentences or 1900 words long.
2. If an input file has *style* scores that are more than 2 standard deviations from the mean, scores for that file are excluded from the computation of the standards.

Although *mkstand* will compile standards for any number of documents (up to 75), standards will be most reliable if at least 20 documents are used.

Because *mkstand* runs *deroff*(1) on input files before computing scores, formatting header files should be included as part of the input.

Four options affect *deroff*:

- mm** eliminate *mm*(1) macros, and associated text that is not part of sentences (e.g. headings), from the analysis. This is the default.
- ms** eliminate *ms*(1) macros, and associated text that is not part of sentences, from the analysis. The **-ms** flag overrides the default, **-mm**.
- li** eliminate list items, as defined by *mm* macros, from the analysis. This is the default.
- +li** Include list items in the input text, in the analysis. This flag should be used if the texts contain lists of sentences, but not if the texts contain many lists of non-sentences.

Other options are:

- o** *outfile* put standards in *outfile* instead of the default *stand.out*.

Two options give information about the program:

- flags** print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer's Workbench version number of the command, then exit.

*Mkstand* saves the *style* scores it used in computing the standards in a file named *styl.scores*. Users should examine the scores in this file for any scores that seem unusual or invalid. If any are found, *mkstand* should be rerun without the unusual document.

**FILES**

/tmp/\$\$stat.out	temporary file containing <i>style</i> tables of input files
stand.out	default output file containing standards
styl.scores	output file containing <i>style</i> scores used in compiling standards

**SEE ALSO**

prose(1), style(1), deroff(1), wwstand(1), ww(1).

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

mkstr – create an error message file by massaging C source

**SYNOPSIS**

**mkstr** [ - ] messagefile prefix file ...

**DESCRIPTION**

*Mkstr* is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

*Mkstr* will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of *mkstr* would be

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file *mkstr* keys on the string ‘error(’ in the input stream. Each time it occurs, the C string starting at the ‘’ is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char    efilename[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
    char buf[256];
    if (efil < 0) {
        efil = open(efilename, 0);
        if (efil < 0) {
oops:
                perror(efilename);
                exit(1);
        }
    }
    if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
        goto oops;
    printf(buf, a2, a3, a4);
}
```

The optional *-* causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

**SEE ALSO**

*lseek*(2), *xstr*(1)

**AUTHORS**

William Joy and Charles Haley

**NAME**

`mm` – print out documents formatted with the MM macros

**SYNOPSIS**

`mm` [ options ] [ files ]

**DESCRIPTION**

*Mm* can be used to type out documents using *nroff*(1) and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn*(1) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff*(1) and MM are generated, depending on the options selected.

*Options* for *mm* are given below. Any other arguments or flags (e.g., **-rC3**) are passed to *nroff*(1) or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

- Tterm** Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **\$TERM** from the environment (see *profile*(5) and *environ*(7)) as the value of *term*, if **\$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.
- 12** Indicates that the document is to be produced in 12-pitch. May be used when **\$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)
- c** Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000A**, **382**, **4014**, **tek**, **1620**, and **X**.
- e** Causes *mm* to invoke *neqn*(1); also causes *neqn* to read the `/usr/pub/eqnchar` file (see *eqnchar*(7)).
- t** Causes *mm* to invoke *tbl*(1).
- E** Invokes the **-e** option of *nroff*(1)
- y** Causes *mm* to use the non-compacted version of the macros (see *mm*(7)).

As an example (assuming that the shell variable **\$TERM** is set in the environment to **450**), the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

*Mm* reads the standard input when **-** is specified instead of any file names. (Mentioning other files together with **-** leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm -
```

**HINTS**

1. *Mm* invokes *nroff*(1) with the **-h** flag. With this flag, *nroff*(1) assumes that the terminal has tabs set every 8 character positions.
2. Use the **-olist** option of *nroff*(1) to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **-e**, **-t**, and **-** options, *together* with the **-olist** option of *nroff*(1) may cause a harmless “broken pipe” diagnostic if the last page of the document is not specified in *list*.
3. If you use the **-s** option of *nroff*(1) (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **-s** option of *nroff*(1) does not work with the **-c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **-c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you’ll get (often subtle) garbage; however, if you are redirecting output into a file, use the **-T37** option, and then use the appropriate terminal filter when you actually print that file.

**SEE ALSO**

*col*(1), *env*(1), *eqn*(1), *greek*(1), *mmt*(1), *nroff*(1), *tbl*(1), *profile*(5), *mm*(7), *term*(7).

*MM—Memorandum Macros* by D. W. Smith and J. R. Mashey.

*Typing Documents with MM* by D. W. Smith and E. M. Piskorik.



**DIAGNOSTICS**

“mm: no input file” if none of the arguments is a readable file and *mm* is not used as a filter.

**NAME**

`mmt`, `mvt` – typeset documents, view graphs, and slides

**SYNOPSIS**

`mmt` [ options ] [ files ]

`mvt` [ options ] [ files ]

**DESCRIPTION**

These two commands are very similar to `mm(1)`, except that they both typeset their input via `troff(1)`, as opposed to formatting it via `nroff(1)`; `mmt` uses the MM macro package, while `mvt` uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by `tbl(1)` and/or `eqn(1)`. The proper pipelines and the required arguments and flags for `troff(1)` and for the macro packages are generated, depending on the options selected.

*Options* are given below. Any other arguments or flags (e.g., **-rC3**) are passed to `troff(1)` or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

- e** Causes these commands to invoke `eqn(1)`; also causes `eqn` to read the `/usr/pub/eqnchar` file (see `eqnchar(7)`).
- t** Causes these commands to invoke `tbl(1)`.
- Tst** Directs the output to the MH STARE facility.
- Tvp** Directs the output to a Versatec printer via the `vpr(1)` spooler; this option is not available at all UNIX sites.
- T4014** Directs the output to a Tektronix 4014 terminal via the `tc(1)` filter.
- Ttek** Same as **-T4014**.
- a** Invokes the **-a** option of `troff(1)`.
- y** Causes `mmt` to use the non-compacted version of the macros (see `mm(7)`). No effect for `mvt`.

These commands read the standard input when `-` is specified instead of any file names.

`Mvt` is just a link to `mmt`.

**HINT**

Use the `-olist` option of `troff(1)` to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the **-e**, **-t**, and `-` options, *together* with the `-olist` option of `troff(1)` may cause a harmless “broken pipe” diagnostic if the last page of the document is not specified in *list*.

**SEE ALSO**

`env(1)`, `eqn(1)`, `mm(1)`, `tbl(1)`, `tc(1)`, `troff(1)`, `profile(5)`, `environ(7)`, `mm(7)`, `mv(7)`.

*MM—Memorandum Macros* by D. W. Smith and J. R. Mashey.

*Typing Documents with MM* by D. W. Smith and E. M. Piskorik.

*A Macro Package for View Graphs and Slides* by T. A. Dolotta and D. W. Smith (in preparation).

**DIAGNOSTICS**

“m[mv]t: no input file” if none of the arguments is a readable file and the command is not used as a filter.

**NAME**

monk, monksample, monkspell, monkmerge – typeset documents and letters

**SYNOPSIS**

**monk** [ *options* | *files* ]

**monksample** [ *sample* ]

**monkspell** [ *options* ] ( .. ) [ *files* ] ( .. ).PP **monkmerge** [ *files* ]

**DESCRIPTION**

*Monk* formats the text in the named *files* for phototypesetting, using other [troff\(1\)](#) preprocessors as necessary. *Options* are given below. Any other arguments or flags (e.g., **-o1-2**) are passed to *troff*. Options can occur in any order and can be intermixed with files.

**-Acommands**

Invoke the *commands* after all preprocessors and before *troff*.

**-Bcommands**

Invoke the *commands* after *monk* and before any other preprocessor.

**-E** Invoke the **-e** option of *troff*.

**-N** Use the uncompressed monk databases. This facilitates debugging monk database entries.

**-Rfile** Use *file* as the index file for [prefer\(1\)](#).

**-Sdest**

Send output to device *dest*. Supported forms are:

**-Sapsend**

Linotronic L200P; see [apsend\(1\)](#).

**-Sd202**

Mergenthaler Linotron 202; see [d202\(A\)](#)

**-Slp** Postscript line printer; see [lp\(1\)](#) (default).

**-Sproof**

Teletype 5620 or 630 terminal; see [proof\(9\)](#)

**-Sthink**

HP ThinkJet; see [thinkblt\(9\)](#)

**-S-** Standard output.

**-Tdevice**

Prepare output for device specified as in **-T** option of [troff\(1\)](#).

**-x**

Shows the preprocessors that are being invoked.

The following options are not normally needed because monk automatically determines which preprocessors are required. However, if the commands in **-A** or **-B** options require a preprocessor, their use can be forced by the following options.

**-c** *col* postprocessor; see [column\(1\)](#). (Automatically invoked for many printing terminals.)

**-cn** [cite\(A\)](#)

**-e** [eqn\(1\)](#)

**-g** [grap\(1\)](#)

**-i** [ideal\(1\)](#)

**-ipa** [ipa\(A\)](#)

**-p** [pic\(1\)](#)

**-r** [prefer\(1\)](#)

**-t** [tbl\(1\)](#)

**-tp** [tped](#); see [ped\(9\)](#)

*Monksample* produces on the standard output a skeleton document that you can redirect into a file and edit. If no argument is given, *monksample* prints a list of the available *samples*. They are:

- acm** Association for Computing Machinery galley sheets.
- centerpb**  
Center Phone Book.
- cspress**  
Computer Science Press galley sheets.
- form1** AT&T Bell Laboratories merit review form 1.
- im** AT&T Bell Laboratories internal memorandum.
- kluwer**  
Kluwer Academic Publishers book format.
- letter** Letters with optional AT&T letterhead.
- memo**  
Internal AT&T correspondence.
- model**  
IEEE/ACM model sheets.
- research**  
AT&T Bell Laboratories bi-annual research report.
- rp** AT&T Bell Laboratories release paper.
- song** Song sheets for singing at nursing homes.
- tc** AT&T Bell Laboratories technical correspondence.
- tm** AT&T Bell Laboratories technical memorandum.

*Monkspell* looks up words from the named *files* (standard input default) in a public spelling list and in a private list. Possible misspellings—words that occur in neither and are not plausibly derivable from the former—are placed on the standard output. It ignores constructs of *monk(1)*, *troff(1)* and its standard preprocessors. It runs *demonk(1)* with all specified options and passes its output to *spell(1)*. The following options, in addition to all options for *deroff(1)*, are available:

- ddir** Use non-standard *monk* database directory *dir*.
- i** Ignore *monk* | *insert* and | *source* commands as well as *troff* .so and .nx requests.

*Monkmerge* reads each file and copies it to standard output, replacing *monk insert* commands with the file contents. It ignores *monk source* commands, which include a file without processing the *monk* commands within. If no input file is given, *monkmerge* reads from standard input.

## EXAMPLES

- monk paper* Format the file using the default typesetter fonts and output device.
- monk -Sproof paper* Format a file and proof it on a 5620 terminal.

## FILES

- `$(HOME)/cite`  
forward and backward reference preprocessor
- `$(HOME)/db`  
monk databases
- `$(HOME)/monk`  
monk compiler
- `$(HOME)/tmac.p`  
macros for *pic(1)* preprocessor
- `$(HOME)/sample`  
directory for existing samples

## SEE ALSO

*prefer(1)*, *troff(1)*, *deroff(1)*, *tex(1)*, *lp(1)*, *apsend(1)*, *d202(A)*, *thinkblt(9)* *proof(9)*  
Murrel, S. L., Kowalski, T. J., ‘Typing Documents on the UNIX System: Using Monk 0.6’, this manual,

Volume 2

**NAME**

`mp`, MetaPost – system for drawing pictures

**SYNOPSIS**

`mp [-I] [-T] [ first-line ]`

**DESCRIPTION**

*Mp* interprets the MetaPost language and produces *PostScript* pictures. The MetaPost language is similar to Knuth's Metafont with additional features for including *tex(1)* or *troff(1)* commands and accessing features of PostScript not found in Metafont. The `-T` flag selects *troff* instead of *tex*.

An argument given on the command line behaves as the first input line. That can be either a (possibly truncated) file name or a sequence MetaPost commands starting with `\` and including an `input` command. Thus `mp` processes the file `figs.mp`. The basename of `figs` becomes the *jobname*, and is used in forming output file names. If no file is named, the *jobname* becomes `mpout`. The default `.mp` extension can be overridden by specifying an extension explicitly.

There is normally one output file for each picture generated, and the output files are named *jobname.nnn*, where *nnn* is a number passed to the `beginfig` macro. The output file name is *jobname.ps* if this number is negative.

The output files can be used as figures in a TeX document by including

```
\special{psfile=jobname.nnn}
```

in the TeX document. Alternatively, one can `\input` and then use the macro `\epsfbox{jobname.nnn}` to produce a box of the appropriate size containing the figure.

***btex* TeX commands *etex***

This causes *mp* to generate a MetaPost picture expression that corresponds to the TeX commands. If the TeX commands generate more than one line of text, it must be in a `\vbox` or a `minipage` environment.

***verbatimtex* TeX commands *etex***

This is ignored by *mp* except that the TeX commands are passed on to TeX. When using LaTeX instead of TeX the input file must start with a `verbatimtex` block that gives the `\documentstyle` and `\begin{document}` commands.

Since most TeX fonts have to be downloaded as bitmaps, the *btex* feature works best when the output of `mp` is to be included in a TeX document so that *dvips(1)* can download the fonts. For self-contained PostScript output that can be used directly or included in a *troff* document, start your MetaPost input file with the command `prologues:=1` and stick to standard PostScript fonts. TeX and MetaPost use the names in the third column of the file `/usr/lib/mp/trfonts.map`.

MetaPost output can be included in a *troff* document via the *mpictures(6)* macro package. In this case *mp* should be invoked with the `-T` flag so that the commands between *btex* and *etex* or between *verbatimtex* and *etex* are interpreted as *troff* instead of TeX. (This automatically sets `prologues:=1`).

Here is a list of the environment variables affect the behavior of *mp*:

**MPINPUTS**

Search path for `\input` files. It should be colon-separated, and start with dot. Default: `./usr/lib/mp`

**MFINPUTS**

Auxiliary search path for `\input` files with `.mf` extensions. Default: `./usr/lib/mf`

**TEXTFONTS**

Search path for font metric files. Default: `./usr/lib/tex/fonts/tfm`

**MPXCOMMAND**

The name of a shell script that converts embedded typesetting commands to a form that *mp* understands. Defaults: `/usr/lib/mp/bin/makempx` for *tex* and `/usr/lib/mp/bin/troffmpx` for *troff*

**TEX** The version of TeX to use when processing `btex` and `verbatimtex` commands. Default: `tex`

**TROFF**

The *troff* pipeline for `btex` and `verbatimtex` commands. Default: `eqn -d\$\$ | troff -Tpost`

**MPMEMS**

Search path for `.mem` files. Default: `./usr/lib/mp`

**MPPOOL**

Search path for strings. Default: `./usr/lib/mp`

**MPEDITOR**

A command for invoking an editor with `%s` in place of the file name and `%d` in place of the line number. Default: `/bin/ed`

**TEXVFFONTS**

Search path for virtual fonts. Default: `/usr/lib/tex/fonts/psvf`

A `.mem` file is a binary file that permits fast loading of fonts and macro packages. *Mp* reads the default `plain.mem` unless another `.mem` file is specified at the start of the first line with an `&` just before it. There is also an `mfplain.mem` that simulates plain Metafont so that *mp* can read `.mf` fonts. (Plain Metafont is described in *The METAFONTbook*).

Experts can create `.mem` files by invoking *mp* with the `-I` switch and giving macro definitions followed by a `dump` command.

The MetaPost language is similar to Metafont, but the manual *A User's Manual for MetaPost* assumes no knowledge of Metafont. MetaPost does not have bitmap output commands or Metafont's online display mechanism. Use *dvips*(1) and *psi*(9) to see the results before printing.

**FILES**

`/usr/lib/mp/*`  
 macros, `.mem` files, and tables for handling included *tex* and *troff*

`/usr/lib/mp/bin`  
 Directory for programs that handle included *tex* and *troff*. `/usr/lib/mp/trfonts.map` table of corresponding font names for *troff*, PostScript, and TeX

`/usr/lib/tex/macros/epsf.tex`  
 The TeX input file where the `\epsfbox` macro is defined

`/usr/lib/tex/macros/doc/mpintro.tex`  
 TeX input for a document that describes the MetaPost language

`/usr/lib/mp/examples.mp`  
 The source file for the figures used in `mpintro.tex`

`/n/bowell/usr/src/cmd/tex/mp/doc/*`  
 More information on using MetaPost with *troff*.

**SEE ALSO**

[tex](#)(1), [lp](#)(1), [psi](#)(9)

Donald E. Knuth, *The METAFONTbook*, Addison Wesley, 1986,

John D. Hobby, *A User's Manual for MetaPost* AT&T Bell Labs Computing Science Technical Report 162, 1991.

**NAME**

newcsh – description of new csh features (over oldcsh)

**SYNOPSIS**

**csh** *csh-options*

**SUMMARY**

This is a summary of features new in *csh(1)* in this version of the system; an older version of *csh* is available as *oldcsh*. This newer *csh* has some new process control primitives and a few other new features. Users of *csh* must (and automatically) use the new terminal driver (summarized in *newtty(4)* and completely described with the old in *tty(4)*) which allows generation of some new interrupt signals from the keyboard which tell jobs to stop, and arbitrates access to the terminal; on CRT's the command “stty crt” is normally placed in the *.login* file to be executed at login, to set other useful modes of this terminal driver.

**Jobs.**

The most important new feature in this shell is the control of *jobs*. A job is associated with each pipeline, where a pipeline is either a simple command like “date”, or a pipeline like “who | wc”. The shell keeps a table of current jobs, and assigns them small integer numbers. When you start a job in the background, the shell prints a line which looks like:

```
[1] 1234
```

this indicating that the job which was started asynchronously with “&” is job number 1 and has one (top-level) process, whose process id is 1234. The set of current jobs is listed by the *jobs* command.

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a *stop* signal to the current job. The shell will then normally indicate that the job has been “Stopped”, and print another prompt. You can then put the job in the background with the command “bg”, or run some other commands and then return the job to the foreground with “fg”. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key ^Y which does not generate a stop signal until a program attempts to *read(2)* it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by doing “stty tostop”. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character “%” introduces a job name. If you wish to refer to job number 1, you can name it as “%1”. Just naming a job brings it to the foreground; thus “%1” is a synonym for “fg %1”, bringing job 1 back into the foreground. Similarly saying “%1 &” resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus “%ex” would normally restart a suspended *ex(1)* job, if there were only one suspended job whose name began with the string “ex”. It is also possible to say “%?string” which specifies a job whose text contains *string*, if there is only one such job.

The shell also maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a “+” and the previous job with a “-”. The abbreviation “%+” refers to the current job and “%-” refers to the previous job. For close analogy with the *history* mechanism, “%%” is also a synonym for the current job.

**Status reporting.**

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say “notify” after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that “You have stopped jobs.” You may use the “jobs” command to see what they are. If you do this or immediately try to exit again,



the shell will not warn you a second time, and the suspended jobs will be unmercifully terminated.

### New builtin commands.

#### **bg**

**bg** %job ...

Puts the current or specified jobs into the background, continuing them if they were stopped.

#### **fg**

**fg** %job ...

Brings the current or specified jobs into the foreground, continuing them if they were stopped.

#### **jobs**

**jobs** -l

Lists the active jobs; given the -l options lists process id's in addition to the normal information.

**kill** %job

**kill** -sig %job ...

**kill** pid

**kill** -sig pid ...

**kill** -l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix "SIG"). The signal names are listed by "kill -l". There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

#### **notify**

**notify** %job ...

Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. All jobs are marked "notify" if the shell variable "notify" is set.

**stop** %job ...

Stops the specified job which is executing in the background.

%job

Brings the specified job into the foreground.

%job &

Continues the specified job in the background.

### Process limitations.

The shell provides access to an experimental facility for limiting the consumption by a single process of system resources. The following commands control this facility:

**limit** *resource maximum-use*

**limit** *resource*

**limit**

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

Resources controllable currently include *cputime* (the maximum number of cpu-seconds to be used by each process), *filesize* (the largest single file which can be created), *datasize* (the maximum growth of the data+stack region via *sbrk(2)* beyond the end of the program text), *stacksize* (the maximum size of the automatically-extended stack region), and *coredumpsize* (the size of the largest core dump that will be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is "k" or "kilobytes" (1024 bytes); a scale factor of "m" or "megabytes" may also be used. For *cputime* the default scaling is "seconds", while "m" for minutes or "h" for hours, or a time of the form "mm:ss" giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

#### **unlimit resource**

#### **unlimit**

Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed.

#### **Directory stack.**

This shell now keeps track of the current directory (which is kept in the variable *cwd*) and also maintains a stack of directories, which is printed by the command *dirs*. You can change to a new directory and push down the old directory stack by using the command *pushd* which is otherwise like the *chdir* command, changing to its argument. You can pop the directory stack by saying *popd*. Saying *pushd* with no arguments exchanges the top two elements of the directory stack. The elements of the directory stack are numbered from 1 starting at the top. Saying *pushd* with a argument “+*n*” rotates the directory stack to make that entry in the stack be at the top and changes to it. Giving *popd* a “+*n*” argument eliminates that argument from the directory stack.

#### **Miscellaneous.**

This shell imports the environment variable *USER* into the variable *user*; *TERM* into *term*, and *HOME* into *home*, and exports these back into the environment whenever the normal shell variables are reset. The environment variable *PATH* is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it. (It could be set once in the *.login* except that commands over the Berknet would not see the definition.)

There are new commands *eval*, which is like the *eval* of the Bourne shell *sh*(1), and useful with *tset*(1), and *suspend* which stops a shell (as though a ^Z had stopped it; since shells normally ignore ^Z signals, this command is necessary.)

There is a new variable *cdpath*; if set, then each directory in *cdpath* will be searched for a directory named in a *chdir* command if there is no such subdirectory of the current directory.

An *unsetenv* command removing environment variables has been added.

There is a new “:” modifier “:e”, which yields the extension portion of a filename. Thus if “\$a” is “file.c”, “\$a:e” is “c”.

There are two new operators in shell expressions “!~” and “=” which are like the string operations “!=” and “=” except that the right hand side is a *pattern* (containing, e.g. “\*”s, “?”s and instances of “[...]”) against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

The form “\$<” is new, and is replaced by a line from the standard input, with no further interpretation thereafter. It may therefore be used to read from the keyboard in a shell script.

#### **SEE ALSO**

*csh*(1), *killpg*(2), *sigsys*(2), *signal*(2), *jobs*(3), *sigset*(3), *tty*(4)

#### **BUGS**

Command sequences of the form “a ; b ; c” are not handled gracefully when stopping is attempted. If you suspend “b”, the shell will then immediately execute “c”. This is especially noticeable if this expansion results from an *alias*. It suffices to place the sequence of commands in ()’s to force it to a subshell, i.e. “( a ; b ; c )”, but see the next bug.

Shell builtin functions are not stoppable/restartable.

Control over output is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

**NAME**

`newgrp` – change to a new group

**SYNOPSIS**

**newgrp** *group* [ *command* [ *arg* ... ] ]

**DESCRIPTION**

*Newgrp* runs a *command* with the (real and effective) groupid temporarily set to *group*. If no command is given, the user's shell (see [passwd\(5\)](#)) is executed.

A password is demanded if the group has a password and the user does not.

**EXAMPLES**

`exec newgrp bin` Restart the shell with a different groupid.

**FILES**

`/etc/group`

`/etc/passwd`

**SEE ALSO**

[login\(8\)](#), [getuid\(2\)](#)

**BUGS**

On other systems, *newgrp* is built into the shell. Here it will spawn a new shell unless invoked with **exec**.

**NAME**

newscheck – check to see if user has news

**SYNOPSIS**

newscheck [yne] [readnews options]

**DESCRIPTION**

*newscheck* reports to the user whether or not he has news.

**y** Reports "There is news" if the user has news to read.

**n** Reports "No news" if there isn't any news to read.

**e** Executes *readnews(1)* if there is news.

If there are no options, **y** is the default.

**FILES**

/usr/lib/news/active

Active newsgroups

~/newsrc

Options and list of previously read articles

**SEE ALSO**

*readnews(1)*, *inews(1)*

**NAME**

nice, renice, nohup – run commands at low priority or immune to hangup

**SYNOPSIS**

**nice** [ *-number* ] *command* [ *argument ...* ]

**/etc/renice** [ *-number* ] *pid ...*

**nohup** *command* [ *argument ...* ]

**DESCRIPTION**

*Nice* executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 19. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative *number*, e.g. `--10`.

*Renice* increments the scheduling priority of the processes with the named *process-ids* by *number*. The default *number* is 19, making the process least likely to run.

Only the owner of the process or the super-user may change the priority. Only the super-user may use negative increments.

*Nohup* executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5.

Any output not explicitly redirected is appended to the file `nohup.out` in the current directory.

**FILES**

default destination for standard output and standard error

**SEE ALSO**

[nice\(2\)](#)

**FILES**

`/proc` (*renice*)

**DIAGNOSTICS**

*Nice* returns the exit status of the subject command.

**BUGS**

Quoted *arguments* don't work right in all cases. The difficulty may be avoided by quoting the *command*, with arguments in inner quotes.

**NAME**

nm – name list (symbol table)

**SYNOPSIS**

**nm** [ **-agnopru** ] [ *file ...* ]

**DESCRIPTION**

*Nm* prints the name list of each object *file* in the argument list. If an argument is a library archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in `a.out` are listed.

Each symbol name is preceded by its hexadecimal value (blanks if undefined) and one of the letters

<b>U</b>	undefined
<b>A</b>	absolute
<b>T</b>	text segment symbol
<b>D</b>	data segment symbol
<b>B</b>	bss segment symbol
<b>C</b>	common symbol
<b>f</b>	source file name
<b>-</b>	extra symbols for debuggers; see <b>-a</b> below

If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

<b>-a</b>	Print all symbols; normally extra debugger symbols are excluded.
<b>-g</b>	Print only global (external) symbols.
<b>-n</b>	Sort numerically rather than alphabetically.
<b>-o</b>	Prepend file or archive element name to each output line rather than printing it once separately.
<b>-p</b>	Don't sort; print in symbol-table order.
<b>-r</b>	Sort in reverse order.
<b>-u</b>	Print only undefined symbols.

**SEE ALSO**

[ar\(1\)](#), [ar\(5\)](#), [a.out\(5\)](#), [stab\(5\)](#), [adb\(1\)](#), [pi\(9\)](#)

**NAME**

nm80 print name list

**SYNOPSIS**

**nm80** [ **-nrupgfabdth** ] [ name ]

**DESCRIPTION**

prints the symbol table from the output file of an assembler or loader run. Each symbol name is preceded by its value (blanks if undefined) and one of the letters:

**U** (undefined)

**A** (absolute)

**T** (text segment symbol)

**D** (data segment symbol)

**B** (bss segment symbol)

**F** (file name)

**C** (common symbol)

If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

If no file is given, the symbols in **80.out** are listed. Absolute symbols have their values printed in octal. Non-absolute symbols are assumed to be addresses and are printed in a "low byte:high byte" format.

Options are:

- a** list only absolute symbols
- b** list only bss symbols
- c** list only C-style external symbols, that is those beginning with underscore ‘\_’.
- d** list only data symbols those beginning with underscore ‘\_’.
- f** print only the file names.
- g** print only global (external) symbols
- n** sort by value instead of by name
- p** don't sort; print in symbol-table order
- r** sort in reverse order
- t** list only text symbols those beginning with underscore ‘\_’.
- u** print only undefined symbols.
- h** print output in hex.

**FILES**

80.out

**BUGS**

**NAME**

occ – Esterel and Lustre C code producer

**SYNOPSIS**

**occ** [ option ] ... [ file ]...

**DESCRIPTION**

The *occ* code producer takes as input one or more *oc* files and produces standard executable C code. If there is no input files, the standard input is used. Typical use is: `occ < game1.oc` or `occ game1.oc`

The following options are interpreted by *occ*.

- version** Gives the version name and terminates ignoring all others arguments.
- array** Arrays of automata are available using the **-array** option.
- nopack** With this option, there is no packaging of input, output and sensor interface procedures. This option is available for compatibility with old *occ* versions.
- s** Silent mode. No output file is generated.
- v** Verbose option: gives names of the input module.
- stat** Prints statistic informations into the standard error stream: global time and size of the process.
- size** Prints size informations into the standard error stream: how many actions and how many bytes are produced.
- memstat** Memory state after compiling.
- B name** *name* denotes the output file default base name. The suffix `.c` is added automatically (and possibly a working directory name --see the following option). If this option is omitted the output code is printed in file: `occ_out.c`. For instance, `occ -B game1 game1.oc`
- D directory**  
Specify a directory where the output file will be placed. The default is the current directory.

**FILES**

The caller of the command must have read/write permission for the directories containing the working files, and execute permission for the *occ* file.

**IDENTIFICATION**

Author: A Ressouche, INRIA,  
Sophia-Antipolis, 06600 Valbonne, FRANCE  
Revision Number: \$Revision: 1.5 \$ .

**SEE ALSO**

Esterel v3 Programming Language Manual  
Esterel v3 System Manuals.  
`strlic` (1), `iclc`(1), `lcoc` (1).

**BUGS**



**NAME**

ocdebug – Esterel and Lustre Debug code producer

**SYNOPSIS**

**ocdebug** [ option ] ... [ file ]...

**DESCRIPTION**

The *ocdebug* code producer takes as input one or more *oc* files and produces a human-readable file. If there is no input files, the standard input is used. Typical use is: `ocdebug < game1.oc` or `ocdebug game1.oc`

The following options are interpreted by *ocdebug*.

- version** Gives the version name and terminates ignoring all others arguments.
- names** With this option, *ocdebug* prints the signal name between brackets for each present signal test and between braces for each output action performed in the automaton.
- halts** With this option, *ocdebug* prints the haltset of each state after the keyword: **haltset**
- emitted** With this option, the list of output or local signals emitted in each transition is printed out after the keyword: **emitted**.
- s** Silent mode. No output file is generated.
- v** Verbose option: gives names of the input module.
- stat** Prints statistic informations into the standard error stream: global time and size of the process.
- memstat** Memory state after compiling.
- B name** *name* denotes the output file default base name. The suffix `.debug` is added automatically (and possibly a working directory name --see the following option). If this option is omitted the output code is printed in file: `ocdebug_out.debug`. For instance, `ocdebug -B game1 game1.oc`
- D directory** Specify a directory where the output file will be placed. The default is the current directory.

**FILES**

The caller of the command must have read/write permission for the directories containing the working files, and execute permission for the `occ` file.

**IDENTIFICATION**

Author: A Ressouche, INRIA,  
Sophia-Antipolis, 06600 Valbonne, FRANCE  
Revision Number: \$Revision: 1.4 \$ .

**SEE ALSO**

Esterel v3 Programming Language Manual  
Esterel v3 System Manuals.  
`strlic` (1), `iclc`(1), `lcoc` (1).

**BUGS**

**NAME**

ocr – optical character recognition

**SYNOPSIS**

**ocr** [ *option ...* ] [ *file* ]

**DESCRIPTION**

*Ocr* reads a black-and-white image of a page from *file*, and writes ASCII to the standard output. If no *file* is specified, it reads from the standard input.

The input is a *picfile(5)* image of one column of machine-printed text, normally scanned in by *cscan(1)*. Fonts, sizes, and line-spacings may vary within the column, but each line should have a constant text size and baseline. Lines should be parallel and roughly horizontal.

In the output, white space approximates the original page layout. Words that *spell(1)* are preferred, and hyphenations across lines are recombined.

The options are:

**-as** The alphabet is the union of symbol sets selected by characters in string *s*, from among:

<b>A</b>	ABCDEFGHIJKLMNOPQRSTUVWXYZ	
<b>a</b>	abcdefghijklmnopqrstuvwxyz	
<b>0</b>	0123456789	
<b>.</b>	.,-:;* "?!/&\$()[]# %	(basic punctuation)
<b>^</b>	^~‘\ { }_	(extended punct’n)
<b>+</b>	+-* /<>=.E e []	(numerical punct’n)
<b>s</b>	§ † ‡ ¢ • © ® ° ’ — — —	(selected non-ASCII)
<b>l</b>	fi fl ffi ffi r s i j	(ligatures and digraphs)
<b>g</b>	αβγδεζηθικλμνξοπρστυφχψω	(Greek lower case)
<b>G</b>	ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ	(Greek upper case)

The default is **-aAa0.+^**, the full printable-ASCII set, which may be abbreviated as **-ap**. Thus, **-apslgG** selects all of the above.

**-c** Find columns in complex nested layouts using greedy white covers algorithm.

**-ml[,r]** Trim the left and right margins of the image by *l* and *r* inches, respectively, before looking for columns. If *r* is omitted, it is assumed to equal *l*.

**-nn** Find the *n* largest columns by analysis of a single vertical projection. Each column should be compactly-printed and separated from the others by at least 2 ems of horizontal white space.

**-pn,m** Point sizes lie in the range [ *n*, *m* ]; other sizes are discarded. The default is **-p6,24**.

**-s** Defeat spelling check (but continue to favor numeric strings and good punctuation).

**-t** Write *troff(1)* format. Each column is shown on a separate page, lines at their original height, words at their original horizontal location, and characters roughly original size in Times roman. Hyphenated words are not recombined.

**-u** Unspellable words are prefixed with ‘?’ or, if **-t** is specified, printed boldface.

**-ww** Find the largest column of width *w* inches, within a single vertical projection.

**Fonts**

Trained on over 100 Latin-alphabet book fonts in various italic, bold, etc styles. Only one font of Greek, without diacriticals. Also Swedish and Tibetan, on request.

**SEE ALSO**

*bcp(1)*, *cscan(1)*, *font(6)*, *picfile(5)*, *spell(1)*, *troff(1)*

**BUGS**

For best results, use images of high-contrast, cleanly-printed original documents digitized at a resolution of 400 pixels/inch or higher. It may help to restrict the alphabet and sizes to what’s there.

**NAME**

lint – a C program verifier

**SYNOPSIS**

lint [ **-abchnpux** ] file ...

**DESCRIPTION**

*Lint* attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as `‘-lm’`, in the style of *ld(1)*.

Any number of the options in the following list may be used. The **-D**, **-U**, and **-I** options of *cc(1)* are also recognized as separate arguments.

- p**     Attempt to check portability to the *IBM* and *GCOS* dialects of C.
- h**     Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.
- b**     Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.)
- v**     Suppress complaints about unused arguments in functions.
- x**     Report variables referred to by extern declarations, but never used.
- a**     Report assignments of long values to int variables.
- c**     Complain about casts which have questionable portability.
- u**     Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).
- n**     Do not check compatibility against the standard library.

*Exit(2)* and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

`/*NOTREACHED*/`

at appropriate points stops comments about unreachable code.

`/*VARARGSn*/`

suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

`/*NOSTRICT*/`

shuts off strict type checking in the next expression.

`/*ARGSUSED*/`

turns on the `-v` option for the next function.

`/*LINTLIBRARY*/`

at the beginning of a file shuts off complaints about unused functions in this file.

**FILES**

`/usr/lib/lint/lint[12]` programs

`/usr/lib/lint/l-lib-1c` declarations for standard functions

`/usr/lib/lint/l-lib-port` declarations for portable functions

**SEE ALSO**

*cc(1)*

S. C. Johnson, *Lint, a C Program Checker*

**BUGS**

There are some things you just **can't** get lint to shut up about.

**NAME**

opr – off line print on Xerox 9700

**SYNOPSIS**

**opr** [ *option ...* ]

**DESCRIPTION**

*Pn* prints its standard input on the Xerox 9700 printer. The options include:

**-pland** Print in landscape mode, rather than the default portrait mode.

**-ystyle** Print using font *style*. Allowable styles are:

style	chr/in	lin/in	dflt lin/page		max chr/lin	
			port	land	port	land
vint	12.0	6.5	66	50	94	126
elit	12.0	6.5	66	50	94	126
pica	10.0	6.0	62	46	78	105
bell	13.6	8.5	88	66	106	132
mini	17.6	12.5	132	96	132	132
xr18	6.8	4.0	42	31	53	71
vg14	5.2	4.2	43	32	40	55
tr14	NA	4.2	43	32	NA	NA

The default style is vint. The xr, vg, and tr styles are large fonts for viewgraphs, where the third and fourth characters of the names indicate the approximate point size of the font. The tr style is a proportionally spaced font which may not align horizontally for such constructs as centered text and indented lists.

**-kmask** The *mask* is the name of an electronic form to be overlaid on each page. Possibilities are

blog	Bell Laboratories logo (top right corner).
prin3	Proprietary information message (small type).
vgraf	Bell logo for viewgraphs (bottom left corner).

**-cn** Make *n* copies of the output. Default is 1; max is 99.

**-fnohole** Print on paper without prepunched holes.

**-ln** Print *n* lines per page.

**-on** Offset the output *n* spaces from the left margin. Default is 0.

**-r** The first character of each line is taken to be carriage control (e.g. 1, +)

*Pn* queues a job to do the printing when facilities become available. A new sheet is begun for each file. Backspaces, form feeds, carriage returns, and tabs are handled. Escape characters result in a fatal error. All other control characters are passed on to the output. Tab stops are assumed every 8 spaces. The output bin, user id, and charge account are taken from /etc/passwd.

**BUGS**

The printer has a limited overprinting capability. If this is exceeded, the page will be blank. Lines longer than 132 characters are silently truncated.

**NAME**

ops5 – a rule-based production-system environment

**SYNOPSIS**

**ops5**

**DESCRIPTION**

Ops5 is a rule-based language built on Lisp. A program consists of a collection of if-then rules and a global 'working memory'. Each rule has a conditional expression, the 'LHS' and a sequence of actions, the 'RHS'. A LHS consists of one or more patterns and is 'satisfied' when every pattern matches an element in working memory.

The rule interpreter executes a 'recognize-act' cycle:

1. Match: Evaluate the LHSs of the rules to determine which are satisfied.
2. Conflict Resolution: Select one rule from among the ones with satisfied LHSs. If no LHSs is satisfied halt execution.
3. Act: Perform the operations specified in the RHS of the selected rule.

The top level commands in order of usefulness are:

<b>watch</b>	report on firings and working memory changes (watch) ;Report current watch level (watch 0) ;No report (watch 1) ;Report rule names and working memory time tags (watch 2) ;Report rule names, working memory time tags ;and changes to working memory
<b>load</b>	load working memory and rule declarations (load 'billing.1) ;Load file 'billing.1'
<b>run</b>	start the rule interpreter (run) ;Run until no rules are satisfied or halt executed (run 1) ;Run one rule firing
<b>exit</b>	exit ops5 (exit)
<b>back</b>	back up the rule interpreter (back 32) ;Back up 32 rule firings
<b>wm</b>	display working memory (wm 32) ;Display working memory element 32
<b>ppwm</b>	display parts of working memory (ppwm customer ^record bad) ;Display all customer working memory ;elements with 'bad' records
<b>pm</b>	display production or rule memory (pm good-customer) ;Display rule 'good-customer'
<b>cs</b>	print the conflict set (cs)
<b>matches</b>	print matches for condition elements of a rule (matches bad-customer) ;Display matches for rule 'bad-customer'
<b>pbreak</b>	set a break point after a production firing (pbreak bad-but-long-term-customer) ;Set break point after rule ;'bad-but-long-term-customer'
<b>make</b>	make working memory elements ;Make a customer working memory element (make customer ^name Terry ^record bad ^years 22)

**remove** remove working memory elements  
(remove \*) ;Remove all working memory elements  
(remove 17) ;Remove working memory element 17

**excise** remove rules  
;Remove 'good-customer' and 'bad-customer' rules  
(excise good-customer bad-customer)

**openfile** open a file  
;Open 'ruletrace.ops' as output  
;and associate it with traceoutput port  
(openfile traceoutput |ruletrace.ops| out)  
;Open 'answers' as input and associate it with stdin port  
(openfile stdin |answers| in)

**closefile** close a file  
(closefile traceoutput stdin) ;Close traceoutput and stdin ports

**default** change default input and output files  
(default nil trace) ;Change trace port back to default  
(default traceoutput write) ;Change write port to traceoutput  
(default stdin accept) ;Change accept port to stdin

**strategy** select rule interpreter strategy.  
(strategy) ;Report current strategy  
(strategy mea) ;Selects mea strategy  
(strategy lex) ;Selects lex strategy (default on startup)

## FILES

/usr/lib/lisp  
lisp library

## SEE ALSO

Forgy, C. L., *OPS5 User's Manual*, Department of Computer Science, Carnegie-Mellon University, July, 1981  
lisp (1)

## DIAGNOSTICS

When *ops5* stops executing for any reason, you are placed in the *lisp* top-level routine.

**NAME**

org – show the organization of a document

**SYNOPSIS**

**org** [ **-flags** ] [ **-ver** ] [file ...]

**DESCRIPTION**

*Org* copies the input text to the output, and formats it, preserving headings and paragraph boundaries, but only including the first and last sentence of each paragraph. The input text must contain standard *mm(1)* macros.

The output can be used to study the general organization of the paper, and is sometimes a good abstract.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**SEE ALSO**

*mm(1)*.

**BUGS**

The input text must contain standard *mm* macros.

*Org* will not recognize common abbreviations at the end of a sentence as the sentence end. Consequently, more than two sentences may be printed for a paragraph.

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452



**NAME**

sh – shell, the standard command programming language

**SYNOPSIS**

sh [ **-ceiknrstuvx** ] [ args ]

**DESCRIPTION**

*Sh* is a command programming language that executes commands read from a terminal or a file. See *Invocation* below for the meaning of arguments to the shell.

**Commands.**

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see [exec\(2\)](#)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see [signal\(2\)](#) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by `|`. The standard output of each command but the last is connected by a [pipe\(2\)](#) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by `;`, `&`, `&&`, or `|`, and optionally terminated by `;` or `&`. Of these four symbols, `;` and `&` have equal precedence, which is lower than that of `&&` and `|`. The symbols `&&` and `|` also have equal precedence. A semicolon (`;`) causes sequential execution of the preceding pipeline; an ampersand (`&`) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol `&&` (`|`) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

**for name [ in word ... ] do list done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in word** list. If **in word ...** is omitted, then the **for** command executes the **do list** once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

**case word in [ pattern [ | pattern ] ... ) list ;; ] ... esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* below).

**if list then list [ elif list then list ] ... [ else list ] fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else list** is executed. If no **else list** or **then list** is executed, then the **if** command returns a zero exit status.

**while list do list done**

A **while** command repeatedly executes the **while list** and, if the exit status of the last command in the list is zero, executes the **do list**; otherwise the loop terminates. If no commands in the **do list** are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

*list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

**if then else elif fi case esac for while until do done { }**

**Comments.**

A word beginning with `#` causes that word and all the following characters up to a new-line to be ignored.

**Command Substitution.**

The standard output from a command enclosed in a pair of grave accents ( ` ` ) may be used as part or all of a word; trailing new-lines are removed.

**Parameter Substitution.**

The character `$` is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

```
name=value [ name=value ] ...
```

Pattern-matching is not performed on *value*.

`${parameter}`

A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters `*`, `,`, `#`, `?`, `-`, `$`, and `!`. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is `*` or `,`, then all the positional parameters, starting with `$1`, are substituted (separated by spaces). Parameter `$0` is set from argument zero when the shell is invoked.

`${parameter:-word}`

If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

`${parameter:=word}`

If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

`${parameter:?word}`

If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message “parameter null or not set” is printed.

`${parameter:+word}`

If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-`pwd` }
```

If the colon (`:`) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- `#` The number of positional parameters in decimal.
- `-` Flags supplied to the shell on invocation or by the **set** command.
- `?` The decimal value returned by the last synchronously executed command.
- `$` The process number of this shell.
- `!` The process number of the last background command invoked.

The following parameters are used by the shell:

- HOME** The default argument (home directory) for the `cd` command.
- PATH** The search path for commands (see *Execution* below).
- MAIL** If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.
- PS1** Primary prompt string, by default “`$` ”.
- PS2** Secondary prompt string, by default “`>` ”.
- IFS** Internal field separators, normally **space**, **tab**, and **new-line**.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** is set by `login(1)`).

**Blank Interpretation.**

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (“” or `^`) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

**File Name Generation.**

Following substitution, each command *word* is scanned for the characters \*, ?, and [. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

- \* Matches any string, including the null string.
- ? Matches any single character.
- [... ] Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening `[` is a “!” then any character not enclosed is matched.

**Quoting.**

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

**; & ( ) | < > new-line space tab**

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (‘ ’), except a single quote, are quoted. Inside double quote marks (“ ”), parameter and command substitution occurs and \ quotes the characters \, \, ", and \$. "\$\*" is equivalent to "\$1 \$2 ...", whereas "\$ " is equivalent to "\$1" "\$2" ...

**Prompting.**

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

**Input/Output.**

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

- <word Use file *word* as standard input (file descriptor 0).
- >word Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
- >>word Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
- <<[–]word The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) \new-line is ignored, and \ must be used to quote the characters \, \$, \, and the first character of *word*. If – is appended to <<, then all leading tabs are stripped from *word* and from the document.
- <&digit The standard input is duplicated from file descriptor *digit* (see [dup\(2\)](#)). Similarly for the standard output using >.
- <&– The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by & then the default standard input for the command is the empty file /dev/null. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

**Environment.**

The *environment* (see [environ\(7\)](#)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On

invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd args                and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

### Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

### Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec(2)*.

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

### Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

- :** No effect; the command does nothing. A zero exit code is returned.
- . file** Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.
- break [ n ]**  
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.
- continue [ n ]**  
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.
- cd [ arg ]**  
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*.
- eval [ arg ... ]**  
The arguments are read as input to the shell and the resulting command(s) executed.
- exec [ arg ... ]**  
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- exit [ n ]**  
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

**export** [ *name* ... ]

The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is printed.

**newgrp** [ *arg* ... ]

Equivalent to **exec newgrp** *arg* ...

**read** [ *name* ... ]

One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

**readonly** [ *name* ... ]

The given *names* are marked *readonly* and the values of these *names* may not be changed by subsequent assignment. If no arguments are given, then a list of all *readonly* names is printed.

**set** [ **-ekntuvx** [ *arg* ... ] ]

- e** If the shell is non-interactive then exit immediately if a command exits with a non-zero exit status.
- k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- n** Read commands but do not execute them.
- t** Exit after reading and executing one command.
- u** Treat unset variables as an error when substituting.
- v** Print shell input lines as they are read.
- x** Print commands and their arguments as they are executed.
- Do not change any of the flags; useful in setting \$1 to -.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **\$-**. The remaining arguments are positional parameters and are assigned, in order, to **\$1**, **\$2**, ... If no arguments are given then the values of all names are printed.

**shift**

The positional parameters from **\$2** ... are renamed **\$1** ...

**test**

Evaluate conditional expressions. See [test\(1\)](#) for usage and description.

**times**

Print the accumulated user and system times for processes run from the shell.

**trap** [ *arg* ] [ *n* ] ...

*arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *n* is 0 then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

**umask** [ *nnn* ]

The user file-creation mask is set to *nnn* (see [umask\(2\)](#)). If *nnn* is omitted, the current value of the mask is printed.

**wait**

Wait for all child processes to terminate report the termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is always zero.

**Invocation.**

If the shell is invoked through [exec\(2\)](#) and the first character of argument zero is -, commands are initially read from **/etc/profile** and then from **\$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

- c** *string* If the **-c** flag is present then commands are read from *string*.
- s** If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.
- i** If the **-i** flag is present or if the shell input and output are attached to a terminal, then this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command above.

## EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

## FILES

/etc/profile  
\$HOME/.profile  
/tmp/sh\*  
/dev/null

## SEE ALSO

cd(1), env(1), login(1), newgrp(1), rsh(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(5), profile(5), environ(7).

## BUGS

The command **readonly** (without arguments) produces the same output as the command **export**.

If  $\ll$  is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh\*** is created and the shell complains about not being able to find that file by another name.

**NAME**

*p*, *pg*, *more* – paginate

**SYNOPSIS**

**p** [ *-number* ] [ *file ...* ]

**DESCRIPTION**

*P* copies its standard input, or the named files if given, to its standard output, stopping at the end of every 22nd line, and between files, to wait for a newline from the user. The page size may be set by saying (for example)

*p*

While waiting for a newline, *p* interprets some commands:

- Reprint last page. -- reprints the second last page, etc.
- ! Pass the rest of the line to the shell as a command.
- q Quit.

*Pg* and *more* are synonyms for *p*.

**BUGS**

Because of limited storage, *p* can't back up too far.

*Pg* and *more* exist only to placate old programs that call paginators.

**NAME**

pack, unpack, pcat, compress, uncompress, zcat – compress and expand files

**SYNOPSIS**

**pack** [ - ] *file* ...

**unpack** *file* ...

**pcat** [ *file* ... ]

**compress** [ *option* ... ] [ *file* ... ]

**uncompress** [ *option* ... ] [ *file* ... ]

**zcat** [ -V ] [ *file* ... ]

**DESCRIPTION**

*Pack* attempts to compress the *files* and places the results in corresponding files named *file.z* with the same access modes, dates, and owner as the originals. Successfully packed files are removed.

*Unpack* reverses the process.

*Pcat* unpacks files to the standard output.

The *.z* suffix may be omitted from the name of the input file for *unpack* or *pcat*.

*Pack* encodes individual characters in a Huffman code. Option *-* causes statistics of the encoding to be printed. The option toggles on and off at each appearance among the list of *files*.

*Compress*, *uncompress*, and *zcat* work like *pack*, *unpack*, and *pcat*, putting each compressed *file* into *file.Z*. The options are

- f** (force) Compress even when it doesn't save space.
- c** Write to the standard output; change no files. *Zcat* is identical to *uncompress -c*.
- bbits** *Compress* uses a modified Lempel-Ziv encoding. Common substrings in the file are replaced by variable-length codes up to size *bits* (default 16). Smaller limits devour less address space.
- v** Print percent reduction for each file.
- V** Print program version number.

*Compress-uncompress* pack better and are faster overall; *pack-unpack* work on smaller machines and are much more widely available.

**SEE ALSO**

T. A. Welch, 'A Technique for High Performance Data Compression,' *IEEE Computer*, 17 (1984) 8-19.

**DIAGNOSTICS**

The exit code of *pack*, *unpack*, or *pcat* is the number of files it failed to process.

The exit code of *compress*, *uncompress*, or *zcat* is 0 normally, 1 for error, 2 for ineffective compression (i.e. expansion).



**NAME**

*paper* – list input on HP2621P printer

**SYNOPSIS**

**paper** [ *file* ] (.,.).SH DESCRIPTION *paper* prints the argument files (or the standard input if there are no arguments) on the user's terminal which is assumed to be a HP2621P. A handshaking protocol is used to prevent overrunning the terminal's buffer and dropping characters.

**NAME**

pascal – language interpreter

**SYNOPSIS**

**pascal** [ **-cx** ] [ *-options* ] [ **-i** *name ...* ] [ *name.p* ] [ *obj* [ *argument ...* ] ]  
**pmerge** *name.p ...*

**DESCRIPTION**

*Pascal* translates Pascal source programs to interpretable form, executes them, or both. Under option **-c** the programs are translated but not executed. The translated code appears in file *obj*. Under option **-x** *pascal* interprets the previously translated code in file *obj* (default *Arguments* are made available through the built-ins `argc` and `argv`).

Options **-c** and **-x** must come first.

Option **-i** causes the named procedures and include files to be listed.

Other options are combined in a separate string:

<b>b</b>	Buffer the runtime file output.
<b>l</b>	Make a program listing during translation.
<b>n</b>	List each included file on a new page with a banner line.
<b>p</b>	Suppress the post-mortem control flow backtrace if an error occurs; override execution limit of 500,000 statements.
<b>s</b>	Accept standard Pascal only; non-standard constructs cause warning diagnostics.
<b>t</b>	Suppress runtime tests of subrange variables and treat assert statements as comments.
<b>u</b>	Card image mode; only the first 72 characters of input lines are used.
<b>w</b>	Suppress warning diagnostics.
<b>z</b>	Cause the interpreter to gather profiling data for later analysis by <i>pxp</i> (A)

*Pmerge* combines the named source files into a single source file on the standard output.

**FILES**

*\*.p* source  
*\*.i* include files  
*/usr/lib/pascal/\**  
*obj*  
*/tmp/pix\**  
*obj* for compile-and-go  
*pmon.out*  
 profile data file

**SEE ALSO**

[pc](#)(1), [pxp](#)(A)

W. N. Joy, Susan L. Graham, C. B. Haley, ‘Berkeley Pascal User’s Manual’, in *Unix Programmer’s Manual, Seventh Edition, Virtual VAX-11 Version*, 1980, Vol 2C (Berkeley). There *pascal* is called *pi*, *px*, and *pix*.

**DIAGNOSTICS**

The first character of an error message indicates its class:

<b>E</b>	Fatal error; no code will be generated.
<b>e</b>	Non-fatal error.
<b>w</b>	Warning – a potential problem.
<b>s</b>	Warning – nonstandard Pascal construct.

**BUGS**

The keyword `packed` is recognized but has no effect.

Diagnostics for an included file may appear in the listing of the next one.

A dummy *obj* must be given if both source and *arguments* are present.

**NAME**

passwd – change login password

**SYNOPSIS**

**passwd** [ **-an** ] [ *name* ]

**DESCRIPTION**

This command changes a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monospace. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

If the **-a** option is given, *passwd* prompts for new values of certain fields of the password file entry.

The super-user may use the **-n** option to install new users. The prompts are self-explanatory, and most of the defaults obvious. A null response to the **UID:** prompt assigns a numeric userid one greater than the largest one previously in **A** null response to **Directory:** assigns a home directory in If the first character of the response to this prompt is an asterisk, the remaining characters are taken as the name of the new user's home directory, and a symbolic link to this directory is placed in

If **/etc/stdprofile** exists, each new user's home directory starts with a file name which is a copy of **/etc/stdprofile** with **\N** replaced by the user's name, and **\D** replaced by the name of the home directory.

**FILES**

**/etc/passwd**  
**/etc/stdprofile**

**SEE ALSO**

[crypt\(3\)](#) [passwd\(5\)](#)

Robert Morris and Ken Thompson, 'UNIX password security,' *AT&T Bell Laboratories Technical Journal* 63 (1984) 1649-1672

**BUGS**

The password file information should be kept in a different data structure allowing indexed access.

**NAME**

`patch` – a program for applying a diff file to an original

**SYNOPSIS**

`patch` [options] orig diff [+ [options] orig]

**DESCRIPTION**

*Patch* will take a patch file containing any of the three forms of difference listing produced by the *diff* program and apply those differences to an original file, producing a patched version. By default, the patched version is put in place of the original, with the original file backed up to the same name with the extension “.orig”, or as specified by the **-b** switch. You may also specify where you want the output to go with a **-o** switch. If *diff* is omitted, or is a hyphen, the patch will be read from standard input.

Upon startup, *patch* will attempt to determine the type of the diff file, unless over-ruled by a **-c**, **-e**, or **-n** switch. Context diffs and normal diffs are applied by the *patch* program itself, while *ed* diffs are simply fed to the *ed* editor via a pipe.

*Patch* will try to skip any leading garbage, apply the diff, and then skip any trailing garbage. Thus you could feed an article or message containing a context or normal diff to *patch*, and it should work. If the entire diff is indented by a consistent amount, this will be taken into account.

With context diffs, and to a lesser extent with normal diffs, *patch* can detect when the line numbers mentioned in the patch are incorrect, and will attempt to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned for the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, *patch* will scan both forwards and backwards for a set of lines matching the context given in the hunk. All lines of the context must match. If *patch* cannot find a place to install that hunk of the patch, it will put the hunk out to a reject file, which normally is the name of the output file plus “.rej”. (Note that the rejected hunk will come out in context diff form whether the input patch was a context diff or a normal diff. If the input was a normal diff, many of the contexts will simply be null.)

If no original file is specified on the command line, *patch* will try to figure out from the leading garbage what the name of the file to edit is. In the header of a context diff, the filename is found from lines beginning with “\*\*\*” or “---”, with the shortest name of an existing file winning. Only context diffs have lines like that, but if there is an “Index:” line in the leading garbage, *patch* will try to use the filename from that line. The context diff header takes precedence over an Index line. If no filename can be intuited from the leading garbage, you will be asked for the name of the file to patch.

(If the original file cannot be found, but a suitable SCCS or RCS file is handy, *patch* will attempt to get or check out the file.)

Additionally, if the leading garbage contains a “Prereq:” line, *patch* will take the first word from the prerequisites line (normally a version number) and check the input file to see if that word can be found. If not, *patch* will ask for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, the following:

```
| patch -d /usr/src/local/blurfl
```

and patch a file in the blurfl directory directly from the article containing the patch.

If the patch file contains more than one patch, *patch* will try to apply each of them as if they came from separate patch files. This means, among other things, that it is assumed that separate patches will apply to separate files, and that the garbage before each patch will be examined for interesting things such as filenames and revision level, as mentioned previously. You can give switches (and another original file name) for the second and subsequent patches by separating the corresponding argument lists by a ‘+’. The argument list for a second or subsequent patch may not specify a new patch file, however.

*Patch* recognizes the following switches:

- b** causes the next argument to be interpreted as the backup extension, to be used in place of “.orig”.
- c** forces *patch* to interpret the patch file as a context diff.

- d** causes *patch* to interpret the next argument as a directory, and `cd` to it before doing anything else.
- D** causes *patch* to use the "#ifdef...#endif" construct to mark changes. The argument following will be used as the differentiating symbol. Note that, unlike the C compiler, there must be a space between the **-D** and the argument.
- e** forces *patch* to interpret the patch file as an ed script.
- l** causes the pattern matching to be done loosely, in case the tabs and spaces have been munged in your input file. Any sequence of whitespace in the pattern line will match any sequence in the input file. Normal characters must still match exactly. Each line of the context must still match a line in the input file.
- n** forces *patch* to interpret the patch file as a normal diff.
- N** forces *patch* to not try and reverse the diffs if it thinks that they may have been swapped. See the **-R** option below.
- o** causes the next argument to be interpreted as the output file name.
- p** causes leading pathnames to be kept. If the diff is of the file "b/a.c", *patch* will look for "a.c" in the "b" directory, instead of the current directory. This probably won't work if the diff has rooted pathnames.
- r** causes the next argument to be interpreted as the reject file name.
- R** tells *patch* that this patch was created with the old and new files swapped. (Yes, I'm afraid that does happen occasionally, human nature being what it is.) *Patch* will attempt to swap each hunk around before applying it. Rejects will come out in the swapped format. The **-R** switch will not work with ed diff scripts because there is too little information to reconstruct the reverse operation.  
  
If the first hunk of a patch fails, *patch* will reverse the hunk to see if it can be applied that way unless the **-N** option is supplied. If it can, the **-R** switch will be set automatically. If it can't, the patch will continue to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed. Luckily, most patches add lines rather than delete them, so most reversed normal diffs will begin with a delete, which will fail, triggering the heuristic.)
- s** makes *patch* do its work silently, unless an error occurs.
- x<number>**  
sets internal debugging flags, and is of interest only to *patch* patchers.

## ENVIRONMENT

No environment variables are used by *patch*.

## FILES

/tmp/patch\*

## SEE ALSO

diff(1)

## DIAGNOSTICS

Too many to list here, but generally indicative that *patch* couldn't parse your patch file.

The message "Hmm..." indicates that there is unprocessed text in the patch file and that *patch* is attempting to intuit whether there is a patch in that text and, if so, what kind of patch it is.

## CAVEATS

*Patch* cannot tell if the line numbers are off in an ed script, and can only detect bad line numbers in a normal diff when it finds a "change" command. Until a suitable interactive interface is added, you should probably do a context diff in these cases to see if the changes made sense. Of course, compiling without errors is a pretty good indication that it worked, but not always.

*Patch* usually produces the correct results, even when it has to do a lot of guessing. However, the results are guaranteed to be correct only when the patch is applied to exactly the same version of the file that the patch was generated from.

**BUGS**

Could be smarter about partial matches, excessively deviant offsets and swapped code, but that would take an extra pass.

If code has been duplicated (for instance with `#ifdef OLDCODE ... #else ... #endif`), *patch* is incapable of patching both versions, and, if it works at all, will likely patch the wrong one, and tell you it succeeded to boot.

If you apply a patch you've already applied, *patch* will think it is a reversed patch, and un-apply the patch. This could be construed as a feature.

**NAME**

`pc` – pascal language compiler

**SYNOPSIS**

`pc` [ *option* ] [ `-i name ...` ] *name ...*

**DESCRIPTION**

*Pc* compiles the Pascal source file *name.p* into an executable file called, by default, *a.out*.

Multiple **.p** files are compiled into object files suffixed **.o** in place of **.p**. Object files may be combined by *ld(1)* into an executable *a.out* file. Exactly one object file must supply a **program** statement. The other files contain declarations which logically nest within the program. Objects shared between separately compiled files must be declared in **included** header files, whose names must end with **.h**. An **external** directive, similar to **forward**, declares **functions** and **procedures** in **.h** files.

These options have the same meaning as in *cc(1)*: **-c -g -w -p -O -S -o**. The following options are peculiar to *pc*.

- C** Compile code to perform runtime checks, verify **assert** statements, and initialize variables to zero as in *pascal(1)*.
- b** Block buffer the file *output*.
- i** Produce a listing for the specified procedures, functions and **include** files.
- l** Make a program listing during translation.
- s** Accept standard Pascal only; non-standard constructs cause warning diagnostics.
- z** Allow execution profiling with *pxp(A)* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

Other arguments are taken to be loader option arguments, perhaps libraries of *pc*-compatible routines; see *ld(1)*. Certain options can also be controlled in comments within the program as described in the *Berkeley Pascal User's Manual*.

**FILES**

- file.p** pascal source files
- /usr/lib/pc0**  
compiler
- /lib/f1** code generator
- /usr/lib/pc2**  
runtime integrator (inline expander)
- /lib/c2**  
peephole optimizer
- /usr/lib/pc3**  
separate compilation consistency checker
- /usr/lib/pc2.0strings**  
text of the error messages
- /usr/lib/how\_pc**  
basic usage explanation
- /usr/lib/libpc.a**  
intrinsic functions and I/O library
- /usr/lib/libm.a**  
math library
- /lib/libc.a**  
standard library, see *intro(3)*

**SEE ALSO**

*pascal(1)*, *pxp(A)*, *cc(1)*, *ld(1)*, *adb(1)*, *sdb(1)*, *prof(1)*

W. N. Joy, Susan L. Graham, C. B. Haley, 'Berkeley Pascal User's Manual', in Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version, 1980, Vol 2C (Berkeley).

**DIAGNOSTICS**

See [pascal\(1\)](#) for an explanation of the error message format. Internal errors cause messages containing the word 'SNARK'.

**BUGS**

The keyword **packed** is recognized but has no effect.

The binder is not as strict as it might be.

The **-z** flag doesn't work for separately compiled files.

Because **-s** is used by *pc*, it can't be passed to the loader.



**NAME**

pic, tpic – troff and tex preprocessors for drawing pictures

**SYNOPSIS**

**pic** [ *files* ]

**tpic** [ *files* ]

**DESCRIPTION**

*Pic* is a *troff*(1) preprocessor for drawing figures on a typesetter. *Pic* code is contained between **.PS** and **.PE** lines:

```
.PS optional-width optional-height
element-list
.PE
```

If *optional-width* is present, the picture is made that many inches wide, regardless of any dimensions used internally. The height is scaled in the same proportion unless *optional-height* is present. If **.PF** is used instead of **.PE**, the typesetting position after printing is restored to what it was upon entry.

A line of the form

**.PS<file** causes *pic* to treat the the named file as if it stood in place of the **.PS** line.

An *element-list* is a list of elements:

```
primitive attribute-list
placename : element
placename : position
var = expr
direction
{ element-list }
[ element-list ]
for var = expr to expr by expr do { anything }
if expr then { anything } else { anything }
copy file, copy thru macro, copy file thru macro
sh { commandline }
print expr
reset optional var-list
troff-command
```

Elements are separated by newlines or semicolons; a long element may be continued by ending the line with a backslash. Comments are introduced by a # and terminated by a newline. Variable names begin with a lower case letter; place names begin with upper case. Place and variable names retain their values from one picture to the next.

After each primitive the current position moves in the current direction (**up,down, left,right** (default)) by the size of the primitive. The current position and direction are saved upon entry to a {...} block and restored upon exit. Elements within a block enclosed in [...] are treated as a unit; the dimensions are determined by the extreme points of the contained objects. Names, variables, and direction of motion within a block are local to that block.

*troff-command* is any line that begins with a period. Such a line is assumed to make sense in the context where it appears; generally, this means only size and font changes. Changes to vertical spacing will produce broken pictures.

The *primitive* objects are:

```
box circle ellipse arc line arrow spline move text-list
arrow is a synonym for line ->.
```

An *attribute-list* is a sequence of zero or more attributes; each attribute consists of a keyword, perhaps followed by a value.

```
h(eigh)t expr          wid(th) expr
rad(ius) expr          diam(eter) expr
up opt-expr           down opt-expr
```

right <i>opt-expr</i>	left <i>opt-expr</i>
from <i>position</i>	to <i>position</i>
at <i>position</i>	with <i>corner</i>
by <i>expr, expr</i>	then
dotted <i>opt-expr</i>	dashed <i>opt-expr</i>
chop <i>opt-expr</i>	-> <- <->
invis	same
<i>text-list</i>	<i>expr</i>

Missing attributes and values are filled in from defaults. Not all attributes make sense for all primitives; irrelevant ones are silently ignored. The attribute *at* causes the geometrical center to be put at the specified place; *with* causes the position on the object to be put at the specified place. For lines, splines and arcs, *height* and *width* refer to arrowhead size. A bare *expr* implies motion in the current direction.

Text is normally an attribute of some primitive; by default it is placed at the geometrical center of the object. Stand-alone text is also permitted. A text list is a list of text items:

```
text-item:
    "... " positioning ...
    sprintf("format", expr, ...) positioning ...
positioning:
    center ljust rjust above below
```

If there are multiple text items for some primitive, they are arranged vertically and centered except as qualified. Positioning requests apply to each item independently. Text items may contain in-line *troff* commands for size and font changes, local motions, etc., but make sure that these are balanced so that the entering state is restored before exiting.

A position is ultimately an *x,y* coordinate pair, but it may be specified in other ways.

```
position:
    expr, expr
    place ± expr, expr
    place ± ( expr, expr )
    ( position, position ) x from one, y the other
    expr [of the way] between position and position
    expr < position , position >
    ( position )

place:
    placename optional-corner
    corner of placename
    nth primitive optional-corner
    corner of nth primitive
    Here
```

An *optional-corner* is one of the eight compass points or the center or the start or end of a primitive.

```
optional-corner:
    .n .e .w .s .ne .se .nw .sw .c .start .end

corner:
    top bot left right start end
```

Each object in a picture has an ordinal number; *nth* refers to this.

```
nth:
    nth,      nth last
```

The built-in variables and their default values are:

boxwid = 0.75	boxht = 0.5
circlerad = 0.25	arcrad = 0.25
ellipsoidwid = 0.75	ellipseht = 0.5
linewid = 0.5	lineht = 0.5
movewid = 0.5	moveht = 0.5
textwid = 0	textht = 0
arrowwid = 0.05	arrowht = 0.1
dashwid = 0.1	arrowhead = 2

```
scale = 1
```

These may be changed at any time, and the new values remain in force from picture to picture until changed again or reset by a `reset` statement. Variables changed within [ and ] revert to their previous value upon exit from the block. Dimensions are divided by **scale** during output.

Expressions in *pic* are evaluated in floating point. All numbers representing dimensions are taken to be in inches.

```
expr:
  expr op expr
  - expr
  ! expr
  ( expr )
  variable
  number
  place .x place .y place .ht place .wid place .rad
  sin(expr) cos(expr) atan2(expr,expr) log(expr) exp(expr)
  sqrt(expr) max(expr,expr) min(expr,expr) int(expr) rand()
op:
  + - * / % < <= > >= == != && ||
```

The **define** and **undef** statements are not part of the grammar.

```
define name { replacement text }
undef name
```

Occurrences of **\$1**, **\$2**, etc., in the replacement text will be replaced by the corresponding arguments if *name* is invoked as

```
name(arg1, arg2, ...)
```

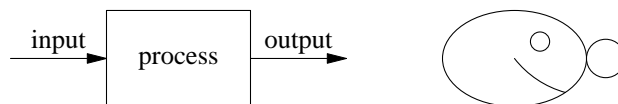
Non-existent arguments are replaced by null strings. Replacement text may contain newlines. The **undef** statement removes the definition of a macro.

*Tpic* is a [tex\(1\)](#) preprocessor that accepts *pic* language. It produces Tex commands that define a box called **\graph**, which contains the picture. The box may be output this way:

```
\centerline{\box\graph}
```

## EXAMPLES

```
arrow "input" above; box "process"; arrow "output" above
move
A: ellipse
  circle rad .1 with .w at A.e
  circle rad .05 at 0.5 <A.c, A.ne>
  arc from A.c to A.se rad 0.5
```



## SEE ALSO

[cip\(9\)](#) [ideal\(1\)](#), [ped\(9\)](#) [grap\(1\)](#), [dag\(1\)](#), [doctype\(1\)](#), [troff\(1\)](#)

B. W. Kernighan, 'PIC—a Graphics Language for Typesetting', this manual, Volume 2

**NAME**

picasso – a line drawing program

**SYNOPSIS**

**picasso** [ **-b***size* **-F***path* **-I***path* **-l***n* **-M***n* **-m***margin* **-p***m***x***n* **-t** **-x** ] [ **-** ] [ *file\_name(s)* ]

**DESCRIPTION**

**Picasso** is a processor for a PIC-like drawing language that produces PostScript output. By default, this output is scaled to fit an 8 by 10 inch print area, and centered on the page.

**-b** *size* specifies a buffer of *size* objects accumulated before translation into PostScript. By default, an entire picture is buffered; on machines with small memories, a buffer of a few thousand objects can prevent thrashing when processing a very large picture. This option is for exceptional cases and is not often needed.

**-I** *path* overrides the standard path for searching for the PostScript prologue and font descriptions (not needed in normal use).

**-F** *path* overrides the standard path for font width tables. The default is to use the `troff` tables.

**-l** *n* processes layer *n* only, as specified by `curlayer=n`.

**-M** *n* magnifies the output image by *n* (shrinks if  $0 < n < 1$ ).

**-p** *m***x***n* specifies output device size in inches (8.5x11 default).

**-t** packages the PostScript with surrounding troff input so that the output file may be passed down a pipeline to `troff(1)`. The Drechsler/Wilks **mpictures** macro package can be used with **troff** to insert the pictures appropriately within the document. Without the flag **picasso** outputs *only* PostScript, dropping any text outside the markers (.PS and .PE) delimiting each picture.

**-m** *margin* specifies an empty border, in printer's points, that **picasso** will place around each picture. This may be useful at times to prevent too tight clipping against adjacent text or the edge of the paper. By default no margin is supplied; to cause a 1/8" (9 point) margin, for example, specify **-m9**.

**-x** suppresses the default scaling and centering.

The **picasso** picture description language is object oriented, the basic objects being **arrow**, **arc**, **box**, **circle**, **ellipse**, **line**, **sector**, **spline**, and (quoted) text. These can be combined, hierarchically, into **blocks**. Primitive objects can be drawn with **solid**, **dashed**, **dotted**, or **invisible** edges. These edges may be of varying **weight** (thickness) and of any shade of gray (from black = 0 to white = 1) or color. The predefined colors are **black**, **white**, **red**, **green**, **blue**, **cyan**, **magenta**, and **yellow**.

Objects may be named and referred to by name or by anonymous references such as 1st box, 4th object, or 2nd last circle. Object names require an initial upper case letter; names beginning with lower case or an underscore are numeric variables. There are a number of predefined variables such as **circlearad**, **boxwid**, **linecolor**. **Picasso** provides a limited set of programming language constructs (loops, if statements, macros, some arithmetic) for combining simple objects into relatively complex pictures.

By default, objects are placed on the page adjacent to each other and from left to right. The default direction may be changed, and any object can be placed **at** a specific position, given either in absolute coordinates or by reference to other objects and points of interest. Any object has a **top**, **bottom**, **left**, and **right** point; these points may also be referred to directionally as **north**, **south**, **west**, and **east** (or **n**, **s**, **w**, and **e**). The "corner" points may also be specified, *e.g.*, **northwest** or **nw**. Lines have **start** and **end** points; you may also refer to **1st**, **2nd**... **nth** points along a line. Boxes, circles, and ellipses have eight predefined points corresponding to the directional references mentioned above, the first point being in the eastern direction and the 8th point towards the southeast. For any object, the "corner" points really lie on the corners of a box surrounding the object while the "counted" points lie on the object itself. This distinction is normally relevant only for circles and ellipses, but since an object can be rotated or otherwise transformed it occasionally has significance for other objects as well.

**EXAMPLE**

The following is a simple no-smoking sign described in the **picasso** language.

```
.PS
d = 0.5
[ box ht d wid 3.5 weight d/20
```

```
    box ht d wid d/2 filled 0.5 noedge
    spline weight 0.2 edge .75 right d then up d \
        then right d then up d
]
linecolor = red; linewidth = 0.375
circle rad 3 at last block
line from last circle .4th to last circle .8th
.PE
```

If this is used in a **troff** document and processed through **picasso** with the **-t** flag, the **.PS** marking the start of the picture can specify the size and placement of the picture at that point in your document. For example, to place the no smoking sign centered on the page in a 3 inch square area, flag the start of the picture with **.PS 3 3 c**.

## SEE ALSO

[troff\(1\)](#), [troff\(5\)](#)

## REFERENCE

R. L. Drechsler and A. R. Wilks, *PostScript pictures in troff documents*.  
B. W. Kernighan, *PIC — A Crude Graphics Language for Typesetting*  
N-P. Nelson, M. L. Siemon, *Picasso 1.0, An OPEN LOOK Drawing Program*

## BUGS

**Picasso** is not completely compatible with [pic\(1\)](#). Besides having a number of new keywords and predefined variable names, **picasso** also centers pictures on a page rather than placing them at upper left.

The interactive version is unable to generate many elements of the language, nor will it preserve such elements (e.g., loops) if they are read in then written out.

**NAME**

pico – graphics editor

**SYNOPSIS**

**pico** [ **-mfto** ] [ **-wN -hN** ] [ *files* ]

**DESCRIPTION**

*Pico* is an interactive editor for grey-scale and color images. Editing operations are expressed in a C-like style. The options are

**-mn** Display on a Metheus frame buffer, **/dev/omn**. A missing *n* is taken to be 0.

**-f** Display on an Itoh frame buffer, **/dev/iti0**.

**-t** Show parse trees for expressions; toggled by the interactive command **tree**.

**-o** Turn off the optimizer; toggled by **optim**.

Files are referred to in expressions as **\$n**, where *n* is the basename or an integer, see **f** below. Otherwise file names are given as strings in double quotes, which may be elided from names that do not contain **/**.

In general, the result of the previous edit operation is available under the name **old**. The destination of the current operation is called **new**.

*Pico* handles images with coordinates (0,0) in the upper left hand corner and (**X**,**Y**) in the lower right. Brightnesses range from 0 (black) to **Z** (white, **Z**=255). The quantities **X**,**Y**,**Z** may be used in expressions and set by options:

**-w n** Set the width **X** of the work area to *n* pixels, default 511.

**-h n** Set the height **Y**, default 511.

*Pico* reads commands from the standard input:

**help** Give a synopsis of commands and functions.

**a file**

**a x y w d file**

Attach a new file. Optional parameters *x* and *y* give the origin of a subrectangle in the work buffer; *w* and *d* define width and depth of the image as stored in the file.

**d file**

**d \$n** Delete (close) the file.

**h file** Read header information from the file.

**r file** Read commands from *file* as if they were typed on the terminal. Can not be done recursively.

**w file**

**w - file**

Write the file, restricted to the current window (see below). Use *pico* format by default. With a minus flag, write a headerless image (red channel only, if picture is colored); see also [picfile\(5\)](#).

**nocolor**

**color** Set the number of channels updated in the work buffer to 1 (black and white) or 3 (red, green, blue).

**window x y w d**

Restrict the work area to a portion of the work buffer with the upper left corner at (*x*,*y*), and the lower right at (*x*+*w*,*y*+*d*).

**get file**

**get \$n** The picture file is (re)opened and read into the work area.

**f** Show names, sizes, and file numbers of open files.

**faster**

**slower**

In slow display the screen is updated once per pixel computed; in fast display (default), once per line of pixels.

**show name**

Show symbol table information, such as the current value of variables. If *name* is omitted, the whole symbol table is shown.

**functions**

Print information on all user defined and builtin functions.

**def name ( args ) { program }**

Define a function, with optional arguments. Variables are declared in these styles:

```
int var;
global int var;
array var[N];
global array var[N];
```

**x expr** Execute the expression in a default loop over all pixels in the current window.

**x { program }**

Execute the *program*. The program must define its own control flow.

**q** Quit.

**EXAMPLES**

```
pico -w1280 -h1024 -m5
```

Get a work buffer that exactly fills a Metheus screen.

a "/tmp/images/rob" Make a file accessible. It will be known henceforth as \$rob.

```
a
get
```

Direct attention to a 512×512 subrectangle in the middle of a 3072×512 image stored in a file named junk, and read it into the workspace.

```
x new = Z - old
x new[x,y] = Z - old[x,y]
x {for(x=0; x<=X; x++) for(y=0; y<=Y; y++) new[x,y] = Z-old[x,y];}
```

Three ways to make a negative image. Note the defaults on control flow and array indexing.

```
window 0 0 256 256
```

```
x new = $1[xclamp(x*2), yclamp(y*2)]
```

Scale a 512×512 image to one quarter of the screen. The built-in functions `xclamp` and `yclamp` guard against indexing out of range.

```
x { printf("current value of %s[%d]:\t%d\n", "histo", 128, hist[128]); }
```

Turn off the default control flow (curly braces) and use the builtin function `printf` to check the value of an array element.

**SEE ALSO**

[bcp\(1\)](#), [imscan\(1\)](#), [flicks\(9\)](#) [rebecca\(9\)](#) [picfile\(5\)](#), [flickfile\(9\)](#)

G. J. Holzmann, 'PICO Tutorial', this manual, Volume 2

G. J. Holzmann, *Beyond Photography—the Digital Darkroom*, Prentice-Hall, 1988

**NAME**

`pl` – print share information for designated users

**SYNOPSIS**

`pl [-a[g]] [-n[v]] [-pfilename] [-u uid[-uid] ...] [login-name ...]`

**DESCRIPTION**

`pl` prints the share information for the given list of login names. The optional flags affect the default behaviour as follows:-

- a[g]** This flag causes information on all currently active users to be printed. The optional flag **g** restricts the selection to real users (ie: doesn't print *groups*).
- n** The normal output is one item per line, this flag puts all items for a user on the same line.
- p file** Directs `pl` to use an alternate shares file, whose path name is *file*.
- u** The list is assumed to be user IDs. If any two user IDs are separated by a minus, then an inclusive range is assumed.
- v** The normal output includes descriptions of each item, this flag turns off verbose mode.

If no arguments are given then `pl` will use the login name of the person that executed the command.

**FILES**

`/etc/shares` The shares file.  
`/etc/passwd` Information on user names and IDs.

**SEE ALSO**

`lnode(5)`, `passwd(5)`, `share(5)`.



**NAME**

plot – graphics filters

**SYNOPSIS**

**plot** [ **-Tterminal** ]

**DESCRIPTION**

These filters read plotting instructions (see [plot\(5\)](#)) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter TERM (see [environ\(5\)](#)) is used. Known *terminals* are:

**2621**

**hp** Hewlett-Packard screen terminal.

**4014**

**tek** Tektronix 4014 storage scope.

**troff** Convert to [troff\(1\)](#) input.

**5620**

**jerq** Teletype DMD display under [mux\(9\)](#) The filter persists after plotting, to make a layer that skips downloading on subsequent plots.

**750**

**pen** Hewlett-Packard pen plotter.

**FILES**

/usr/lib/plot/hpplot  
/usr/lib/plot/trplot  
/usr/lib/plot/penplot  
/usr/lib/plot/tek  
/usr/jerq/bin/jplot

**SEE ALSO**

[plot\(3\)](#), [plot\(5\)](#)

**BUGS**

A persistent *plot* layer on a 5620 terminal imitates `term 33`, it loses most abilities of a [mux\(9\)](#) layer; see [term\(9\)](#)

Which plotters are known depends on which computer you are on.

**NAME**

post – mail and directory service by name

**SYNOPSIS**

**post** [ *option ...* ] [ *person ...* ]

**postext** [ **-c** *directory* ] *extension ...*

**postorg** [ **-c** *directory* ] *organization*

**postorg** [ **-h** *directory* ]

**postsx** [ **-c** *directory* ] *person ...*

**DESCRIPTION**

Post mail and gives directory assistance based on a corporate database. Aside from addressing, it reads and sends mail in the manner of *mail(1)*.

Mail may be sent to a *person* by name:

*first.middle.lastname:dir.org:loc:paper*

where partial fields are valid and every field except *lastname* may be omitted. *Dir* is a phone book, such as *attbl* or *attc*; *loc* is a location code such as *ih*, and *org* is an organization number such as 45 or 11271; *paper* sends paper mail. This happens automatically for persons with no electronic mail address. If a *person* argument is ambiguous, *post* gives a list of possibilities to choose from.

Mail may also be sent to a *person* specified as *system!userid* for electronic mail.

Options are:

**-c** *directory*

Search for *persons* only in the given *directory*.

**-p** Send paper mail.

**-w** Directory assistance. Give the full directory entry for each *person*. The answer comes by mail if your machine is a remote post office.

The following options are not to be accompanied by *persons* or *addresses*.

**-A** *dafile*

Send mail to all names in *dafile* where *dafile* is output from a **-w** request.

**-B** [ *mbox* ]

Read mail from a specified *mbox* file or from the default *\$HOME/mbox*.

**-D** [ *directory* ]

Give modification times for the *directory* or for all *post* directories if no directory name is given.

**-S** Start an interactive session to send change-of-address notices to an update site.

**-v** Give the current version of *post* and tell whether the local post office is general, *g*, (has a directory) or remote, *r*, (no directory).

**-X** Start an interactive set-up session. Should be used by new users.

To maintain mailing lists or to avoid typing long addresses, you may keep an address book in *\$HOME/.mailrc*. Each line in this file begins with the word *alias* followed by the alias and the *persons* or previous aliases that it stands for.

If environment variable **POSTETC=ON**, the *passwd(5)* file is treated as a directory so that login names may be used as *persons*. In any case a login name on the local system can be written *!userid*.

*Post* will ask for verification of recipient addresses if environment variable **CONFIRM=ON**.

*Post* searches directories sequentially. The directory search path may be altered by setting the environment variable **DIRPATH** similarly to **PATH** in *sh(1)*.

*Postext* is like *post -w*, but retrieves by phone number instead of by name:

*number:dir:loc:org*

*Postorg* retrieves by organization:

*org:dir:loc:oclevel*

where *oclevel* is a (possibly partial) occupation level. Option **-h** lists occupation levels.

*Postsx* retrieves names phonetically similar to *persons*.

## EXAMPLES

**post s.j.griesmer**

Send mail.

**post -c attc -w smith**

Who are all the smiths in attc?

**post -w jackson:452:ih >dafile; post -A dafile**

## FILES

`/usr/mail/*`

mailboxes

`dead.letter`

unmailable text

`$HOME/mail`

default directory for secure saving of mail

`$HOME/.mailrc`

Mail options

`$HOME/mbox`

default mail repository

`/usr/post/info/post/*db*`

LLA database components

`/usr/post/info/post/postsys`

post administrative address

`/usr/post/postlib/postoff`

local and remote directories

`/usr/post/postlib/postversion`

version number

`/usr/post/tmp/post/postlog`

log of command executions

## SEE ALSO

[mail\(1\)](#)

## DIAGNOSTICS

Exit status is 0 on total success, 1 on lookup failure, 2 on partial failure (e.g. ambiguous entries).

**NAME**

`pp` – C program pretty printer

**SYNOPSIS**

`pp` [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Pp* formats the named C source files, or its standard input if none are given, and writes a ‘publication format’ on the standard output for printing on a typesetter with filters like *lp(1)*. The options are

- Tdev** Prepare output for devices named as in *troff(1)*; default is **202**.
- b** Use bold fonts suffixed  $\mathcal{K}$  (demi-bold) rather than  $\mathcal{B}$  (bold).
- f font** Set the main font; the default is MM, Memphis Medium.
- k file** Cause words in the named file, one per line, to be recognized as keywords; the file will be looked up in `/usr/lib/pp` if it is not in the current directory.
- ttitle** Generate a title page with the title specified and a date stamp.

**SEE ALSO**

*pr(1)*, *troff(1)*, *d202(1)*, *lp(1)*, *font(6)*

**DIAGNOSTICS**

*Pp* complains and exits if it cannot find a required font. If this happens, take the name of the missing font to a local font guru.

**BUGS**

The default device for *pp* should be **-Tpost**.

**NAME**

`pr` – print file

**SYNOPSIS**

`pr` [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

`Pr` produces a printed listing of one or more *files*. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. For no file arguments, or for a file argument `-`, `pr` prints its standard input.

Options apply to all following files but may be reset between files:

- `-n` Produce *n*-column output.
- `+n` Begin printing with page *n*.
- `-b` Balance columns on last page, in case of multi-column output.
- `-d` Double space.
- `-h` Take the next argument as a page header (*file* by default).
- `-f` Use formfeeds to separate pages. Pause, ring bell, and wait for newline before beginning.
- `-ln` Take the length of the page to be *n* lines instead of the default 66.
- `-m` Print all *files* simultaneously, each in one column.
- `-n` Number the lines of each *file*.
- `-on` Offset the left margin *n* character positions.
- `-p` Between pages pause, ring bell, and wait for newline.
- `-sc` Separate columns by the single character *c* instead of aligning them with white space. A missing *c* is taken to be a tab.
- `-t` Do not print the 5-line header or the 5-line trailer normally supplied for each page.
- `-wn` For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.

Inter-terminal messages via [write\(1\)](#) are forbidden during a `pr`.

**FILES**

`/dev/tty` to suspend messages

**SEE ALSO**

[cat\(1\)](#), [lp\(1\)](#), [thinkblt\(9\)](#)

**DIAGNOSTICS**

There are no diagnostics when `pr` is printing on a terminal.

**NAME**

prefer, pinvert, penter, plook, pconvert – maintain and use bibliographic references

**SYNOPSIS**

**prefer** [ *option ...* ]

**penter** [ *outfile* ]

**pinvert** [ *option ...* ] [ *file ...* ]

**plook** [ **-p** *dbfile* ] [ *keyword ...* ]

**pconvert** [ **-d** ] *file*

**DESCRIPTION**

*Prefer* is a *troff*(1) preprocessor for bibliographic references. It copies a document from the standard input to the standard output, using a bibliographic database to change symbolic references into full references ready for typesetting by *troff*(1). Although symbolic references are in the style of *monk*(1), *prefer* does not depend on *monk*. The options are:

**-n** Format for *nroff*.

**-o***sortkey*

Under the **|reference\_list** command, sort according to *sortkey*, any combinations of the letters a (author) d (date), and t (title), rather than in database sequence. If *sortkey* is `sort`, sort according to the current style.

**-p***dbfile*

Use *dbfile* as the bibliographic database (default)

**-r** Format as a released paper (technical memorandum default).

**-s***style*

Set the formatting *style*, one of `att` (default), `acm`, `apa`, `ieee`, `lsa`, `pami`, `spectrum`.

*Prefer* recognizes the following commands, which may appear anywhere in a document. Parentheses () in the commands may be replaced by any of {} [] <> .

**|reference\_style**( *style arg ...* )

Switch to a new formatting style. All previous references are forgotten and a new list of references is begun. If *style* is **same** the current style remains (but all previous references are forgotten). Optional *args* are:

**tm** Format as a technical memorandum.

**rp** Format as a released paper.

**nroff** Format for *nroff*.

**troff** Format for *troff*.

**sort** Print a **|reference\_list** in an order appropriate for the current style.

**sequence**

Print a **|reference\_list** in database sequence.

*sortkey*

Print a **|reference\_list** according to the *sortkey*, any combination of the letters a, d, t as above.

**|reference**(*keywords*

*%ref\_fields %flags*)

Insert a citation mark in the current style (e.g. [7], <sup>3</sup>, (Knuth, 1975)). One or more *keywords* cause selection from the bibliographic database. Each **%** argument must begin a new line. *%ref\_field* lines override information from the database; with no *keywords* a complete reference may be given. For the form of reference fields, see the output of *penter* or the paper in Volume 2. The following *%flags* may modify the citation.

**%no\_author**

Exclude author information.

**%no\_date**

Exclude date from the citation mark.

**%no\_cite**

Omit the entire citation, but include the entry in the final reference list.

**%pre\_text *string***

Insert *string* before the citation mark.

**%post\_text *string***

Insert *string* after the citation mark

**|reference\_include( *dbfile* ...)**

Include the contents of the database(s) *dbfile(s)* in the list of references, treating them as **%no\_cite** entries.

**|reference\_placement**

Produce a list of all references specified in **|reference** or **|reference\_include** commands since the beginning of the document or the last **|reference\_style** or **|reference\_placement**.

**|reference\_list( *dbfile* ...)**

Format the contents of the database(s) *dbfile*.

**|reference\_database( *dbfile* )**

Switch to database *dbfile*

*Penter* helps build *prefer* bibliographic databases. It prompts for a reference type, and then for admissible attributes, such as author, date, etc. A default value proposed in brackets [] may be accepted by typing a newline, skipped by typing spaces before the newline, or overridden by typing a new value. The character & appended to an attribute causes *penter* to prompt for the attribute again (to enter multiple authors, for example).

The answer ? to the initial prompt gets a list of all reference types. The answer help gets a subprompt for a reference type whose pertinent attributes will then be listed. The answer ? to the subprompt gets attributes for every type.

The attribute **also** permits one entry to refer to another by naming keywords for the other reference. An entire 'also' citation may be included within a **|reference** thus:

```
%also_begin text
%ref_fields
%also_end
```

The attribute **keywords** prompts for distinguishing keys for the current entry, in addition to those already occurring within author, title, etc.

The 'reference type' **quit** causes *penter* to exit, first appending the collected database information to *out-file* by default).

The 'attribute' **e** permits editing of the current reference with the editor specified by environment variable **EDITOR**, *ed(1)* by default; **v** gets the editor **VISUAL**, *vi(1)* by default.

*Pinvert* creates an inverted index to one or more bibliographic database *files*. The index is placed in *file.i*, where *file* is the first input file. An associated *file.h* contains the names of the input files. The options are:

**-ccommon**

Do not index words listed in file *common* (default)

**-ignore**

Do not index information about attributes listed in file *ignore*. (The default */usr/lib/prefer/ignore* lists **%volume**, **%number**, **%part**, **%pages**, **%X** (location status), **%Y** (read status), **%Z** (comment).)

- ki** Maximum number of keys kept per record (default 100).
- li** Maximum length of keywords (default 6, none is less than 3).
- pfile** The basename of the index is *file*. Prefer will write the index to *file.i*.
- v** Verbose. Print statistics.

*Plook* uses the inverted index to retrieve bibliographic records by *keywords* from the command line or the standard input. Records that contain all the keywords in the request are sent to the standard output. Option **-p** is the same as for *pinvert*.

*Pconvert* converts a [refer\(1\)](#) database to *prefer* style. Under option **-d** it converts *refer*-style commands in a document to *prefer* style.

## FILES

```
prefer.out
    default database

prefer.out.i
    default index file

prefer.out.h
    default header file containing names of databases

/usr/lib/eign
    default list of common words

/usr/lib/prefer/ignore
    default list of %ref_fields to ignore for indexing

/usr/lib/prefer/styles/*
    awk scripts of formatting instructions for each style

/tmp/prefer*
    scratch file

/usr/lib/prefer/ptemplate
    reference type definitions, self-describing

/usr/lib/prefer/mypubenter
    program executed by penter
```

## SEE ALSO

M. A. Derr, ‘Formatting References with Prefer’, this manual, Volume 2  
[refer\(1\)](#), [monk\(1\)](#), [troff\(1\)](#)

## BUGS

*Prefer* commands don’t work immediately after certain formatting macros, e.g. .SM, .I, .B.  
*Plook* complains if the first key matches more references than it can store. Try rearranging your request so a less common word comes first.  
*Pinvert* does not record options **-c** and **-I**. If you use them with *pinvert*, you will have to supply them for *prefer* and *plook* as well.



**NAME**

printenv – print environment

**SYNOPSIS**

**printenv** [ *name* ]

**DESCRIPTION**

With no arguments, *printenv* places the strings of the environment, described in [environ\(5\)](#), on the standard output one per line.

If a *name* is specified, its value is retrieved from the environment and printed.

**SEE ALSO**

[sh\(1\)](#), [rc\(1\)](#), [environ\(5\)](#), [getenv\(3\)](#)

**DIAGNOSTICS**

Exit status 1 is returned when a specified *name* is not present in the environment.

**BUGS**

The *name* feature cannot handle functions.

**NAME**

prof – display profile data

**SYNOPSIS**

**prof** [ *option ...* ] [ *a.out* [ *mon.out ...* ] ]

**DESCRIPTION**

*Prof* interprets files produced by [monitor\(3\)](#) or the **-p** option of *cc* or *f77*. The symbol table in the named object file by default) is read and correlated with the profile file by default). For each symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the time spent there and the number of times that routine was called. If more than one profile file is specified, the output represents the sum of the profiles.

Zero call counts are tallied for subroutines not compiled with option **-p**. The flag **-p** must be passed to the loader to get the profiling output written.

Options are:

- l** Sort the output by symbol value.
- n** Sort the output by number of calls.
- s** Produce a summary profile file in
- v -low -high**  
Produce a graphic profile on the standard output for display by the [plot\(1\)](#) filters. Optional numbers *low* and *high*, by default 0 and 100, select a percentage of the profile to be plotted.
- z** Include routines with zero usage in the output.

**FILES**

*mon.out*  
for profile

*a.out*  
for namelist

*mon.sum*  
for summary profile

**SEE ALSO**

[time\(1\)](#), [lcomp\(1\)](#), [monitor\(3\)](#), [getopt\(3\)](#), [profil\(2\)](#), [plot\(1\)](#), [cc\(1\)](#), [f77\(1\)](#)

**BUGS**

Beware of quantization errors.

*Prof* is confused by *f77*, which puts the entry points at the bottom of subroutines and functions.

Option **-v** has been disabled.

**NAME**

proofr – automatic proofreader  
 proofer – alternative command-name for proofr

**SYNOPSIS**

**proofr** [ **-s** ][ **-flags** ][ **-ver** ] file ...

**DESCRIPTION**

*Proofr* is an automatic proofreading system that runs modified versions of 5 programs:

- spellwwb*(1) - checks for misspelled words.
- punct*(1) - checks for rudimentary punctuation errors.
- double*(1) - searches for consecutive occurrences of the same word.
- dictplus*(1) - locates wordy and/or misused phrases and suggests alternatives.
- splitinf*(1) - searches for split infinitives.

*Proofr* is one of the programs run under the *wwb*(1) command.

Options are:

- s** produce a short summary version of *proofr*.

Two options give information about the program:

- flags** print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer's Workbench version number of the command, then exit.

**NOTE**

If the user has a file called *\$HOME/lib/ddict*, *proofr* will run *dictplus* so that phrases in *ddict* are located or ignored, as specified. See *diction*(1), *dictadd*(1), *dictplus*(1) for more information.

If the user has a file called *\$HOME/lib/spelldict*, *proofr* will run *spellwwb* so that words in *spelldict* are not listed as errors. See *spellwwb*(1) and *spelladd*(1) for more information.

**FILES**

/tmp/\$\$\* temporary files

**SEE ALSO**

*spellwwb*(1), *punct*(1), *double*(1), *splitinf*(1), *diction*(1), *wwb*(1), *worduse*(1), *spelltell*(1), *deroff*(1).

**BUGS**

See other manual pages for bugs in individual programs.

**SUPPORT**

**COMPONENT NAME:** Writer's Workbench

**APPROVAL AUTHORITY:** Div 452

**STATUS:** Standard

**SUPPLIER:** Dept 45271

**USER INTERFACE:** Stacey Keenan, Dept 45271, PY x3733

**SUPPORT LEVEL:** Class B - unqualified support other than Div 452

**NAME**

*prose* – describe style characteristics of text

**SYNOPSIS**

**prose** [ **-flags** ] [ **-ver** ] [ **-tm** | **-c** | **-t** | **-x standards-file** ] [ **-mm** | **-ms** ] [ **-li** | **+li** ] [ **-s** ] [ **-f style-file** | **file ...** ]

**DESCRIPTION**

*Prose* describes the writing style of a document as determined by *style(1)*, but the output is in prose form. The output describes readability, word and sentence lengths, sentence structure and variation.

The program checks that a document's scores on certain *style* variables fall within the average range for documents of a specified type. Whenever the score for a variable is outside the average range, a warning message is printed with information about the variable, and commands that can be run to get further information.

*Prose* creates a file called *styl.tmp* that contains the table produced by *style*.

*Prose* compares a document with standards for one of several document types, according to the following flags:

- tm** Compare input text to good Bell Laboratories TM's. (This is the default.)
- c** Evaluate input text for craft suitability.
- t** Compare input text with good training documents.
- x standards-file** Compare input text with standards contained in user-specified *standards-file*. See [mkstand\(1\)](#) to set up the *standards-file*.

Because *prose* runs [deroff\(1\)](#) before looking at the text, formatting header files should be included as part of the input.

Options affecting [deroff\(1\)](#) are:

- mm** eliminate [mm\(1\)](#) macros, and associated text that is not part of sentences (e.g. headings), from the analysis. This is the default.
- ms** eliminate [ms\(1\)](#) macros, and associated text that is not part of sentences, from the analysis. The **-ms** flag overrides the default, **-mm**.
- li** eliminate list items, as defined by *mm* macros, from the analysis. This is the default.
- +li** Include list items in the input text, in the analysis. This flag should be used if the text contains lists of sentences, but not if the text contains many lists of non-sentences.

Other options are:

- s** Produce a short (10 line) summary version of *prose*.
- f style-file** If a file containing the *style* table exists as output from the *style* program, or from a previous *prose* run, it may be specified so that *prose* need not run *style* again. *Styl.tmp* can be used as the *style-file*. The input text file should not be used with the **-f** flag.

Two options give information about the program:

- flags** print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer's Workbench version number of the command, then exit.

*Prose* is one of the programs run under the [wwb\(1\)](#) command.

**EXAMPLES**

The command:

**prose -t +li filename**

will describe how the style characteristics of *filename* compare with standards for training documents. Lists will be included in the analysis. The *style(1)* table will be left in the file *styl.tmp*. Then the command:

**prose -x standards-file -f styl.tmp**

will use the style statistics already gathered for *filename*, and describe how they compare with the user-defined standards contained in *standards-file*.

## FILES

styl.tmp	contains <i>style</i> table
wwb/lib/prosedoc	contains all standards used for comparison, and stored <i>prose</i> output text files

## SEE ALSO

*style(1)*, *wwb(1)*, *deroff(1)*, *match(1)*, *wwbstand(1)*, *mkstand(1)*, *worduse(1)*.

## SUPPORT

**COMPONENT NAME:** Writer's Workbench

**APPROVAL AUTHORITY:** Div 452

**STATUS:** Standard

**SUPPLIER:** Dept 45271

**USER INTERFACE:** Stacey Keenan, Dept 45271, PY x3733

**SUPPORT LEVEL:** Class B - unqualified support other than Div 452

**NAME**

`ps` – process status

**SYNOPSIS**

`ps option ...`

**DESCRIPTION**

*Ps* prints information about processes.

For each process reported, the process id, control terminal, status, cpu time, and command name are printed. Status is at least one of the following letters:

<b>R</b>	Runnable.
<b>S</b>	Asleep for less than 20 seconds.
<b>I</b>	Asleep for 20 seconds or more.
<b>P</b>	Waiting for memory to be paged in.
<b>T</b>	Stopped by a debugger.
<b>W</b>	Swapped out of memory.
<b>N</b>	Positive scheduling priority; see <i>nice(2)</i> .

These options modify the report for each process:

<b>f</b>	Print additional lines listing each open file in use by the process.
<b>ff</b>	Print open files, but omit the process id at the beginning of each line.
<b>h</b>	Print column headers.
<b>l</b>	Also print virtual size and current resident size in kilobytes, parent process id, and wait channel.
<b>n</b>	Don't sort the output.
<b>u</b>	Also print effective userid and recent cpu share; sort by cpu share rather than by process id.

By default, processes running under the current real userid that don't appear to be shells are reported. These options pick different processes:

<b>a</b>	Report processes running under any userid.
<b>F file</b>	Report processes using the named <i>file</i> .
<b>r</b>	Report processes with real or effective userid matching the current real userid.
<b>ts</b>	Report processes with controlling terminal <i>s</i> . <i>S</i> may be <code>.</code> (the current controlling terminal) or one of the abbreviations printed by <i>ps</i> , e.g. <code>03</code> for <code>dk26</code> for or <code>?</code> for processes with no control terminal.
<b>x</b>	Include processes that appear to be shells.
<b>num</b>	Report the process with process id <i>num</i> .

Multiple **F**, **t**, and *num* options are allowed; the union of all selections is printed.

By default, *ps* looks for process data in the process file system *proc(4)*, but reads `/dev/drum` for information about swapped processes (to avoid swapping them in just to look at them) and `/dev/kmem` for information about open files. These options cause it to gather information differently:

<b>o</b>	Ignore <i>proc(4)</i> ; read directly from <code>/dev/mem</code> and the swap area. Useful mostly in single-user mode or when examining a crash dump.
<b>Mmem</b>	Read memory data from <i>mem</i> instead of <code>/dev/mem</code> or
<b>Dswap</b>	Read swap data from <i>swap</i> instead of
<b>Nname</b>	Read symbols from <i>name</i> instead of This matters only under option <b>o</b> .

To examine a crash dump, use `ps oMdumpfile`. Option **M** changes the default swap device to

**FILES**

<code>/proc</code>	process images
<code>/dev/drum</code>	swap device

`/dev/kmem`  
kernel memory

`/dev/mem`  
physical memory

`/lib/ttydevs`  
searched to find tty names

`/etc/fstab`  
searched to find local file system names

**SEE ALSO**

[kill\(1\)](#), [proc\(4\)](#), [load\(1\)](#), [pstat\(8\)](#)

**BUGS**

Things can change while *ps* is running.

Since *ps* is usually set-userid, filename arguments like that to `-M` are potential security botches.

**NAME**

psifile, mhssend— postscript interpreter/fax sender

**SYNOPSIS**

**psifile** [ *option ...* ] [ *file* ]  
**mhssend** *phone\_number file*

**DESCRIPTION**

*Psifile* reads Postscript input from *file* or from standard input and produces a file containing an image of the page. The format of the output file is specified by the following options:

- fax** runs at 200 dpi and produces g31 fax in the multipage fax format called **mhs**, putting its output in file **fax\$\$mhs** by default. If a phone number is supplied, the output file is pushed to **/tmp** on fama and **mhssend** is run to send the fax.
- P** *phone\_no* specifies the destination phone number for **-fax**.
- g4** runs at 300 dpi and produces a fax g4 file called **psi.out.g4** by default that can be displayed on the gnots with *rbits*.
- bm** produces *bitfile(9)* output in file **psi.out** by default. *-bm* is useful for debugging postscript programs because it has better diagnostics than the printers.

Other options are

- s** assumes the file is in **mhs** format and sends it to the phone number provided with **-P** above.
- o** *name* use *name* as the basename of the output file.
- p** *page* only output postscript page number *page* as determined by **%%Page** comments in the file.

Fonts are implemented with 24 point bitmap fonts. Those available are Symbol, Courier, Times-Roman, Times-Italic, Times-Bold, Times-BoldItalic, Helvetica, Helvetica-Oblique, Helvetica-Bold, Helvetica-BoldOblique. Fonts Courier-Bold, Courier-Oblique, and Courier-BoldOblique are mapped to Courier. Postscript type 1 fonts are implemented and work if supplied with the input.

For best results with TeX documents, run **dvips** with the **-Tfax** or **-D 200** option to get fonts of the proper resolution.

**EXAMPLES**

```
troff -ms memo | lp -dstout -H | psifile -fax -P 4223
```

troff -ms memo | dpost | psifile -fax -P 4223 Two equivalent ways to format a memo, convert it to PostScript, and produce a fax file.

**FILES**

fax\$\$mhs  
 default **-fax** output file

psi.out.g4  
 default **-g4** output file

psi.out  
 default **-bm** output file

**/usr/lib/psi/psifax**  
 postscript→mhs format program

**/usr/lib/psi/psifaxg4**  
 postscript→fax g4 program

**/usr/lib/psi/psibm**  
 postscript→bitfile program



**SEE ALSO**

*psi(9) lp(1), dvips(1), postscript(8), proof(9) bcp(1)*

**DIAGNOSTICS**

Symbols that lack bitmaps are replaced by ‘?’ and an error is reported.

**BUGS**

Unimplemented PostScript features are rotated images and half tone screens. Imagemasks may only be rotated by multiples of 90 degrees, not by arbitrary angles.

**NAME**

psix – postscript interpreter

**SYNOPSIS**

**psix** [ *option ...* ] [ *file* ]

**DESCRIPTION**

*Psix* reads Postscript input from *file* or from standard input and simulates the resulting pages in a window under X Windows. If the large window it brings up is too big for your screen, you can use *-geometry* to change its size. You may also want to use the *-a* option described below.

The options are

- pn**     Display page *n*, where *n* is determined from the **%%Page** comments in the file. If these are not present, page selection will not work.
- R**     Pages in the file are in reverse order. This flag must be used on such files for the *-p* option to work.
- r**     Display the image at full scale, with the bottom left corner positioned at the bottom left corner of the window. (By default, the image is scaled to fit the window, maintaining the aspect ratio of a printer.)
- a x y**     Display the image at full scale with position *x,y* of the image placed at the bottom left corner of the window.

Fonts are implemented with size-24 bitmap fonts. Those available are Symbol, Courier, Times-Roman, Times-Italic, Times-Bold, Times-BoldItalic, Helvetica, Helvetica-Oblique, Helvetica-Bold, Helvetica-BoldOblique. Fonts Courier-Bold, Courier-Oblique, and Courier-BoldOblique are mapped to Courier. Other postscript fonts, including type1, may be used if they are supplied before they're referenced.

When the 'cherries' icon is displayed, you can move forward by typing return or you can use mouse button 3 to move forward (**more**), to a particular page (**page**), or quit (**done**).

**EXAMPLES**

```
troff -ms memo | lp -dstout -H | psi
```

troff -ms memo | dpost | psi Two equivalent ways to format a memo, convert it to PostScript, and display it.

For best results with TeX documents, use **dvips** with the **-Tjrq**, **-Tgnot**, or **-D 100** option to get fonts of the proper resolution and run *psi* with the *-r* or *-a* flag to prevent *psi* from scaling.

**FILES**

psi.err

error messages

**SEE ALSO**

[lp\(1\)](#), [dvips\(1\)](#), [postscript\(8\)](#), [proof\(9\)](#) [psifile\(1\)](#), [psi\(9\)](#)

**DIAGNOSTICS**

A 'dead mouse' icon signals an error; error comments are placed on file

Symbols that lack bitmaps are replaced by '?' and an error is reported.

**BUGS**

Unimplemented PostScript features are rotated images and half tone screens. Imagemasks may only be rotated by multiples of 90 degrees, not by arbitrary angles.

Skipping pages may cause operators to be undefined.

**NAME**

`ptx` – permuted index

**SYNOPSIS**

`ptx` [ *option ...* ] [ *input* [ *output* ] ]

**DESCRIPTION**

*Ptx* generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output exemplified by:

```
.xx "tail" "before" "keyword and after" "head"
```

where `.xx` may be defined as a [troff\(1\)](#) macro for user-defined formatting. The *before* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, `/` marks the spot.

The following options can be applied:

- f**      Fold upper and lower case letters for sorting.
- t**      Prepare the output for the phototypesetter; the default line length is 100 characters.
- w *n***    Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.
- g *n***    Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.
- o *only***  
          Use as keywords only the words given in the *only* file.
- i *ignore***  
          Do not use as keywords any words given in the *ignore* file. If the **-i** and **-o** options are missing, use `/usr/lib/eign` as the *ignore* file.
- b *break***  
          Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.
- r**      Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

**FILES**

`/usr/lib/eign`

**BUGS**

Line length counts do not account for overstriking or proportional spacing.

**NAME**

punct – punctuation checker

**SYNOPSIS**

`/usr/bin/WWB/punct` [ file ... ]

**DESCRIPTION**

*Punct* scans English text for punctuation errors and doubled words. When it finds an error, it places the error on the standard output together with line number and suggested repunctuation.

**FILES**

`/tmp/$$*` temporary files

**SEE ALSO**

`style(1)`, `diction(1)`, `wwb(1)`

**BUGS**

*Punct* will consider unfamiliar abbreviations ending with a period (except initials) to be the end of the sentence, consequently, it will capitalize the next word.

**NAME**

push, pull, npush, npull – datakit remote file copy

**SYNOPSIS**

**push** [ -v ] *machine file ... remotedir*

**pull** [ -v ] *machine file ... localdir*

**npush** [ -v ] *machine file ... remotedir*

**npull** [ -v ] *machine file ... localdir*

**DESCRIPTION**

*Push* and *pull* copy files between machines over Datakit. *Push* copies *files* from the local machine to the directory *remotedir* on the named *machine*. *Pull* copies *files* from the named *machine* to the directory *localdir* on the local machine. The last component of the name of a copy is the same as that of the original. If one of the *files* is a directory, a corresponding directory is created and the directory's files are copied, recursively.

Option **-v** announces each file as it is copied.

Pushing and pulling involve two programs running in different contexts on different machines. In particular, pulling to directory `.` puts files in the local current directory, but pushing to `.` puts files in the remote home directory. Shell metacharacters which are to be interpreted on the remote machine must be quoted.

*Npush* and *npull* behave exactly like *push* and *pull*, but use a different protocol, necessary for communicating to some other Datakit clusters.

**FILES**

`/usr/lib/Rpull`

link to `/usr/bin/pull` for remote end of transaction

`/usr/lib/Rpush`

link to `/usr/bin/push`

**SEE ALSO**

*con(1)*, *cp(1)*, *rcp(1)*, *cu(1)*, *uucp(1)*

**DIAGNOSTICS**

Messages marked ( `remote` ) are from the sister process running on the remote machine.

**NAME**

`pwd`, `where` – machine name and working directory

**SYNOPSIS**

`pwd`

`where`

**DESCRIPTION**

*Pwd* prints the pathname of the working (current) directory.

*Where* prints the name of the machine and the pathname of the current directory in the form

*machine!fullpathname*

**SEE ALSO**

*cd* in [sh\(1\)](#)

**NAME**

pwintf – print selected limits file entries using printf formats

**SYNOPSIS**

**pwintf** [-p shares\_file] expression format [identifier..]

**DESCRIPTION**

For each entry in the system shares file the expression argument is evaluated. If the result is non zero the remaining arguments are treated as if they were arguments to *printf* (see *printf(3S)*). An alternative shares file may be specified with the *-p* option.

Expression elements are:

	Binary or. Non zero if the left hand side or the right hand side evaluates to a non zero value.
&&	Binary and. Non zero if the left hand side and the right hand side are both non zero.
== !=	Equal/not equal to. Non zero if the left hand side and the right hand side are equal/not equal.
> <	Greater/less than. Non zero if the left hand side is greater/less than the right hand side.
>= <=	Greater/less than or equal to. Non zero if the left hand side is greater/less than or equal to the right hand side.
	Regular expression matching. Non zero if the string on the left hand side matches the regular expression given by the string on the right hand side. Regular expressions are given in the style of <i>ed</i> (see <i>ed(1)</i> ).
!	Unary not. Non zero if the right hand side evaluates to zero.
".."	A string of characters.
{..}	A date. Date specifications are in the style: {[[[[[yy]mm]dd]hh]mm][.ss]}. For example {01271200} would be noon on the 27th of January in the current year.
(..)	A sub-expression.
identifier	Any one of the identifiers described below.
number	A decimal digit string.

Note that the expression may have to be quoted to stop the shell from interpreting symbols such as **&** as symbols having special meaning.

**FORMATS**

All *printf* format specifications and modifiers are allowed except the '\*' modifier. To facilitate the printing of dates which are stored as the number of seconds since 1st January 1970, **%t** may be used. This will cause the corresponding integral argument to be interpreted as a time and given in the style of *ctime* (see *ctime(3C)*). All modifiers will be ignored in such a time specification.

**IDENTIFIERS**

In the following list words printed in **this font** are as defined in the include files <shares.h> and <sys/lnode.h>.

activenode	1 if the account has the <b>ACTIVELNODE</b> flag set.
changed	1 if the account has the <b>CHANGED</b> flag set.
charge	The long term accumulated costs of the account as a floating point number.
deadgroup	1 if the account has the <b>DEADGROUP</b> flag set.
dirpath	The initial directory of the account as a string.
flags	A string containing the names of the flags set for this account.
gecos	The "gecos" field of the account as a string.
gid	The integral gid of this account.
lastref	1 if the account has the <b>LASTREF</b> flag set.
lastused	The time the account last did anything.
lname	The login name of the account as a string.

notshared	1 if the account has a <b>NOTSHARED</b> flag.
now	The current time.
pwd	The encrypted password of the account as a string.
sgr	The uid of the scheduling group of this account.
sgrname	The lname of the scheduling group of this account.
shares	The integral number of shares the account has.
shellpath	The initial shell of the account as a string.
usage	The usage of the account as a floating point number.
uid	The integral uid of this account.

**FILES**

/etc/passwd  
/etc/shares

**SEE ALSO**

printf(3S), ctime(3C), shares(5).

**DIAGNOSTICS**

Yes. A summary of usage is given when pwintf is invoked with no arguments.



**NAME**

pxp, pxref – pascal printer, profiler, and cross-reference lister

**SYNOPSIS**

**pxp** [ **-acdefjnstuw\_** ] [ **-23456789** ] [ **-z** [ *name ...* ] ] *name.p*

**pxref** [ **-** ] *name.p*

**DESCRIPTION**

*Pxp* prints the Pascal program *name.p* in a standard ‘pretty’ form. Under option **-z** the listing is annotated with statement execution counts from a previous *pascal(1)* run.

- a** Print the bodies of all procedures and functions in the profile; even those which were never executed.
- d** Include declaration parts in a profile.
- e** Eliminate **include** directives when reformatting a file; the **include** is replaced by the reformatted contents of the specified file.
- f** Fully parenthesize expressions.
- j** Left justify all procedures and functions.
- n** Eject a page as each file is included; in profiles, print a blank line at the top of the page.
- s** Strip comments from the input text.
- t** Print only a table of counts of procedure and function calls.
- u** Card image mode; only the first 72 characters of input lines are used.
- w** Suppress warning diagnostics.
- z** Generate an execution profile. The presence of any *names* causes the profile to be restricted to the named procedures, functions, and include files.
- \_** Underline keywords.
- d** With *d* a digit, use *d* spaces as the indenting unit. The default is 4.

*Pxref* makes a line-numbered listing and cross-reference index for *name.p*. The optional **-** argument suppresses the listing.

**FILES**

- \*.p* input files
- \*.i* include files
- pmon.out*  
profile data
- /usr/lib/how\_pxp*  
information on basic usage

**SEE ALSO**

Berkeley Pascal User’s Manual  
*pascal(1)*

**BUGS**

*Pxref* trims identifiers to 10 characters and pads lines with blanks.

**NAME**

qed – multi-file text editor

**SYNOPSIS**

**qed** [ - ] [ -i ] [ -q ] [ -e ] [ -x startupfile ] [ filename1 filename2 ... ]

**DESCRIPTION**

*Qed* is a multiple-file programmable text editor based on *ed*.

*Qed* operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a *w* or *W* (write) command is given. The copy of the text being edited resides in a scratch area called a *buffer*. There are 56 buffers, labeled by alphabetic 'a' to 'z' and 'A' to 'Z', and the characters '{', '|', '}' and ' ' (the four ASCII characters following 'z'). These 56 characters are called, for notational efficiency, *bnames*. The buffers can contain any ASCII character except NUL.

If *file* arguments are given, *qed* simulates an *r* command (see below) on each of the named files; that is to say, the files are read into *qed*'s buffers so that they can be edited. The first is read into buffer 'a', the second into buffer 'b', through 'z', then from 'A' to 'Z', up to a maximum of 52 files. The optional - puts *qed* in non-*verbose* mode (described with the *o* command). The -q, -e and -i are equivalent to performing an initial 'oqs', 'oes' or 'ois' command (see the *o* command below).

When *qed* starts up, the file named by the environment variable **QEDFILE** is read into buffer ' ' and executed (i.e. read as command input), before reading in files and accepting commands from the terminal. The argument *filenames* are set in the buffers before the startup file is executed, so the startup file can treat the *filenames* as arguments. The default startup file may be overridden with the -x option.

Input to *qed* can be redirected, at any time, to come from storage such as a buffer by use of a *special character* such as "\b". All the *qed special character* sequences are discussed in detail below; they all begin with a backslash '\

*Qed* has a *truth flag* which is set according to the success of certain commands and which can be tested for conditional execution, and a *count* which is set to such values as the number of successful substitutions performed in an *s* command. Each buffer has associated with it a (possibly null) filename and a *changed* flag, which is set if the contents of the buffer are known to differ from the contents of the named file in that buffer.

Commands to *qed* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Every command which requires addresses has default addresses, so that the addresses can often be omitted.

In general, any number of commands can appear on a line. Some commands require that the character following the command be a separator, such as blank, tab or newline. Usually, a *display character*, *p*, *P*, *l*, or *L* may precede the separator, causing the resulting line to be displayed in the specified format after the command. Certain commands allow the input of text for placement in the buffer. This text can be supplied in two forms: either on the same line, after the command, or on lines following the command, terminated by a line containing only a period '.'. If the text is on the command line, it is separated from the command by a space or a tab. If the tab is used, it is considered part of the text.

*Qed* supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. Regular expressions in *qed* are delimited by enclosing them in a pair of identical characters, frequently slashes '/'. In the following specification for regular expressions the word 'character' means any character but newline. Note that special character interpretation always occurs *before* executing a command. Thus, the backslashes mentioned below are those present after special characters have been interpreted.

1. Any character except a metacharacter matches itself. Metacharacters are the regular expression delimiter plus < [ . and \ | > ^ \* + \$ when another rule gives them a meaning.
2. A . matches any character.
3. A backslash \ followed by any metacharacter in the list given in rule 1 is a regular expression and matches that character. A backslash followed by one of ! \_ { } ( ) or a non-zero digit has a special meaning discussed below; otherwise, backslashes have literal meaning in regular expressions.

4. The metacharacter `\!` matches any control character except tab or newline.
5. A non-empty string *s* enclosed in square brackets `[s]` (or `[^s]`) matches any character in (or not in) *s*. In *s*, `\` has no special meaning, and `]` may only appear as the first character. A substring *a-b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
6. A regular expression, of the form `<xI>` or `<xI|x2|...|xm>`, where the *x*'s are regular expressions of form 1-12, matches what the leftmost successful *x* matches.
7. A backslash followed by a non-zero digit *n* matches a copy of the string that the bracketed regular expression (see rule 11) beginning with the *n*th `\(` matched.
8. A regular expression of form 1-7 followed by `*` (`+`) matches a sequence of zero (one) or more matches of the regular expression.
9. The metacharacter `\_` matches a non-empty maximal-length sequence of blanks and tabs.
10. The metacharacter `\{ (\}`) matches the empty string at the beginning (end) of an identifier. An identifier is defined to be an underscore `_` or alphabetic followed by zero or more underscores, alphabetic or digits.
11. A regular expression, *x*, of form 1-12, bracketed `\(x\)` matches what *x* matches. The nesting of these brackets in each regular expression of an alternation (rule 6) must be identical. An alternation with these brackets may not be iterated (rule 8).
12. A regular expression of form 1-12, *x*, followed by a regular expression of form 1-11, *y*, matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.
13. A regular expression of form 1-12 preceded by `^` (followed by `$`) is constrained to matches that begin at the left (end at the right) end of a line.
14. A regular expression of form 1-13 picks out the longest among the leftmost matches in a line.
15. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses and the *g* and *v* commands to specify lines, in the *s* command to specify a portion of a line which is to be replaced, in the *G* and *V* commands to refer to buffers in which to perform commands, and in general whenever text is being specified.

To understand addressing in *qed* it is necessary to know that at any time there is a *current buffer* and a *current line*. When *qed* is invoked, the current buffer is buffer 'a', but may be changed at any time by a *b* (change buffer) command. All addresses refer to lines in the current buffer, except for a special case described under the *m* (move) command.

Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character `'.'` addresses the current line.
2. The character `'$'` addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. `'x'` addresses the line marked with the mark name character *x*, which must be a bname. Lines are marked with the *k* command described below. It is an error for the marked line to be outside of the current buffer.
5. A regular expression enclosed in slashes `'/'` addresses the first matching line found by searching forwards from the line after the current line. If necessary, the search wraps around to the beginning of the buffer. If the trailing `'/'` would be followed by a newline, it may be omitted.
6. A regular expression enclosed in queries `'?'` addresses the first matching line found by searching backwards from the line before the current line. If necessary the search wraps around to the end of the buffer. If the trailing `'?'` would be followed by a newline, it may be omitted.
7. An address followed by a plus sign `'+'` or a minus sign `'-'` followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

8. An address followed by '+' or '-' followed by a regular expression enclosed in slashes specifies the first matching line following (resp. preceding) that address. The search wraps around if necessary. The '+' may be omitted. Enclosing the regular expression in '?' reverses the search direction.
9. If an address begins with '+' or '-' the addition or subtraction is taken with respect to the current line; e.g. '-5' is understood to mean '-5'.
10. If an address ends with a '+' (or '-') 1 is added (resp. subtracted). As a consequence of this rule and rule 9, the address '-' refers to the line before the current line. Moreover, trailing '+' and '-' characters have cumulative effect, so '--' refers to the current line less 2.
11. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is entirely equivalent to '-'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when none is given. If more addresses are given than the command requires, the last one or two (depending on what is accepted) are used. The last addressed line must not precede the second-last addressed line.

Typically, addresses are separated from each other by a comma ','. They may instead be separated by a semicolon ';' in which case the current line '.' is set to the first address before the second address is interpreted. The second of two separated addresses may not be a line earlier in the buffer than the first. If the address on the left (right) side of a comma or semicolon is absent, it defaults to the first (resp. last) line.

Filename operands of commands may be made up of printing characters only. However, when the filename appears as the argument to the invocation of *qed*, non-printing characters may be included. When a filename is specified for a command, it is terminated at the first blank, tab or newline.

In the following list of *qed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

(.)a <text>

The append command accepts input text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

b<bname>

The change buffer command sets the current buffer to be that named. '.', '\$' and the remembered *filename* are set to those of the new buffer; upon return to a previously used buffer, '.' will be set to its value when the buffer was last used.

(.)b[+-.]<pagesize><display character>

The browse command provides page-oriented printing. The optional '+', '-', or '.' specifies whether the next, previous, or surrounding page is to be printed; if absent, '+' is assumed. *b.* also prints several carets '^'^'^'^' immediately below the current line. If a *pagesize* is given, it is used for the current browse command and remembered as the default. The *pagesize* is initially 22 lines. If a *display character* is given, the lines are printed in the specified format, and the format is remembered as the default. Initially, 'p' is the default. For *b+* and *b-*, '.' is left at the last line displayed; for *b.*, it is unchanged. NOTE: The browse and change buffer commands are the same character! The two commands can be syntactically distinguished in all cases except for 'b<display char>'; this ambiguity may be resolved by typing the (implicit) '+' after the 'b'.

(.,.)c <text>

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines. If an interrupt signal (usually ASCII DEL) is received during a change command, the old lines are not deleted.

(.,.)d

The delete command deletes the addressed lines from the buffer. The line after the deleted section becomes the current line; if the deleted lines were originally at the end, the new last line becomes the current line. The character after the 'd' can only be one of a blank, newline, tab, or display

character. Line 0 is a valid address for deletion; deleting line 0 has no affect on any lines in the buffer.

#### e filename

The edit command causes the entire contents of the current buffer to be deleted, and then the named file to be read in. ‘.’ is set to the last line of the buffer. The number of characters read is typed if *qed* is in *verbose* mode. The *filename* is remembered for possible use as a default file name in a subsequent *f*, *r*, *w*, or *W* command.

#### E filename

The *E* command is like *e*, except that *qed* does not check to see if the buffer has been modified since the last *w* command.

#### f filename

The filename command prints information about the current buffer, in the format used by the *n* command. If *filename* is given, the currently remembered file name is changed to *filename*. If *qed* is not in *verbose* mode, the information is only printed if the *filename* is not specified. If it is not desired to set the *filename*, the character immediately after the *f* must be a newline. Otherwise, the first token (which may be the null string) on the line, after a mandatory non-empty sequence of blanks and tabs, is taken to be the *filename*. These rules apply to all *filename*-using commands, *e*, *f*, *r*, *R*, *S*, *w* and *W*, although some regard specification of an explicitly null *filename* as an error.

#### ( 1 , \$ ) g/regular expression/command list

In the global command, the first step is to mark every line in the range which matches the regular expression. Then for every such line, the command list is executed with ‘.’ initially set to that line. Any embedded newlines in the command list must be escaped with a backslash. The *a*, *i*, and *c* commands and associated input are permitted; the ‘.’ terminating input mode may be omitted if it would be on the last line of the command list. The commands *g* and *v* are not permitted in the command list. If the command list is empty, ‘.p’ is assumed. The regular expression may be delimited by any character other than newline.

#### G/regular expression/command list

In the globuf command, the first step is to mark every active buffer whose output from an *f* command (with the *filename* printed literally) would match the regular expression. (An active buffer is one which has either some text or a remembered file name.) Then for every such buffer, the command list is executed with the current buffer set to that buffer. In other respects it is like the global command, except that only the commands *G* and *V* are not permitted in the command list. If the command list is empty, ‘f’ is assumed.

#### h<option> command list

The until command provides a simple looping mechanism. The command list is a newline-terminated command sequence which forms the body of the loop; embedded newlines must be escaped with a backslash. The option specifies the exit condition for the loop, and is specified by the character(s) immediately following the ‘h’:

h[*N*]t The loop is executed until the truth flag is true.

h[*N*]f The loop is executed until the truth flag is false.

h[*N*] The loop is executed indefinitely.

The loop condition is tested *after* execution, so the ‘ht’ and ‘hf’ forms execute at least once. *N* denotes an optional non-negative number which indicates the maximum number of times to execute the loop.

#### ( . ) i <text>

The insert command accepts input text and inserts it before the addressed line. ‘.’ is left at the last line input; if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

#### ( .-1 , . ) j

#### ( .-1 , . ) j/replacement/

The join command collapses all addressed lines into a single line by deleting intermediate newlines. The *replacement* (if any) is placed between joined lines. Newlines, backslashes ‘\’, and slashes ‘/’ within *replacement* must be preceded by a backslash. Only slashes may delimit *replacement*. ‘.’ is

left at the resulting line. NOTE: The join command in *qed* has a different default addressing from that in *ed*.

(.)k<bname>

The mark command marks the addressed line with the given *bname*. (The *bname* used in the mark has no relation to any buffer; it is just a label.) The address form ‘<bname>’ then addresses this line. ‘.’ is not changed. The marks are global to *qed*; marking a line ‘x’ erases any previous mark ‘x’ in any buffer.

(.,.)l

The list command prints the addressed lines in an unambiguous way: a tab is printed as ‘\t’, a back-space as ‘\b’, a backslash as ‘\\’, a non-printing character is printed as a backslash followed by three octal digits, and a long line is folded, with the second and subsequent sub-lines indented one tab stop. If the last character in the line is a blank, it is followed by ‘\n’.

(.,.)L

The *L* command is similar to the *l* command, but each line displayed is preceded by its line number, any marks it has (which appear as ‘x’), and a tab.

(.,.)ma

The move command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line. The address *a* can also be of the form <bname>address, in which case the text is moved after the address in the named buffer. The buffer to which the text was moved becomes the current buffer. The original buffer (if different) has ‘.’ left at the line before the moved lines.

n The names command displays the *bname*, dollar and *filename* (in ‘l’ format) of the current buffer and all active buffers. If the buffer’s changed flag is set, an apostrophe ‘’ is printed after the *bname*. The current buffer is indicated by a period ‘.’ before the dollar value. If present, the *filename* is preceded by a tab.

N The *N* command is similar to the *n* command, but the display is only given for those buffers which have a *filename* and for which the changed flag is set.

ops The option command allows various options to be set. The first argument, *p*, specifies which option is being set. The rest of the command, *s*, specifies the setting. Most options can be either enabled or disabled; *s* is ‘s’ to set the option, or ‘r’ to reset it. The following table describes the available options. The default setting is shown after the option’s letter.

b22p Set the length and format of the page printed by the browse command. Either the length or the format may be omitted.

B<null string>

Set the default command sequence to be performed when a newline command is typed at the terminal. The command sequence is set by following the ‘B’ with a newline-terminated string. If the string is null, the newline command resumes its default behaviour.

cr Set the changed flag of the current buffer.

dr Dualcase search mode affects rule one of regular expression construction so that a letter is matched without regard to its case.

er Error exit mode causes *qed* to exit if an error occurs (see the DIAGNOSTICS section). This option is mainly intended for use of *qed* in shell files.

ir Interrupt catching mode causes *qed* to exit when interrupted. (This includes removing the temporary file).

pr Prompting mode causes ‘\*’ to be typed immediately before a command (as opposed to text) is read from the terminal.

qr Quit catching mode causes *qed* to dump core, leaving the temporary file intact, when a QUIT signal is received.

Tr Tracing mode causes all commands not typed directly by the user to be echoed on the terminal. When a special character (other than ‘\B or ‘\N’) is encountered, a ‘[’ is typed, followed by a code specifying the character — ‘za’ for register ‘a’, ‘g’ for global command list, ‘l’ for ‘\l’, ‘B’ for browse pseudo-register, etc. Then, an ‘=’ is typed, followed by the interpretation of the special character, followed by a ‘]’.

- us Uppercase conversion mode enables case transformation in substitute commands. If the ‘u’ flag is set, the character caret (‘^’) becomes non-literal in the replacement text of a substitution. It behaves just like ‘&’, but with case switching of alphabetic characters in the replaced text. If the flag is ‘u’, all alphabetic characters are mapped to upper case; if ‘l’, lower case; and if ‘s’, the case is switched.
- vs Verbose mode causes character counts to be typed after *e*, *r*, *w*, *R*, *S*, and *W* commands. It also causes ‘!’ to be typed upon completion of the *!*, *<*, */* and *>* commands.
- ?c *c* must be one of ‘c’, ‘d’, ‘i’, ‘p’, ‘T’ or ‘v’. The value of the corresponding flag is stored in the truth.

(.,.)p

The print command prints the addressed lines. ‘.’ is left at the last line printed.

(.,.)P

The PRINT command is similar to the print command, but each line displayed is preceded by its line number, and marks it has (which appear as ‘x’), and a tab.

q The quit command causes *qed* to exit. No automatic write of a file is done. If the changed flag is set in any buffer, *qed* prints ‘?q’ and refuses to quit. A second *q* or a *Q* will get out regardless, as will an end-of-file on the standard input.

Q Like *q*, but changed flags are not checked.

( \$ )r filename

The read command reads in the given file after the addressed line. If no *filename* is given, the remembered *filename* is used (see *e* and *f* commands). The *filename* is remembered if there was not already a remembered *filename* in the current buffer. Address ‘0’ is legal for *r* and causes the file to be read at the beginning of the buffer. If *qed* is in *verbose* mode and the read is successful, the number of characters read is typed, except while *qed* is starting up, in which case an *f* command is performed. ‘.’ is left at the last line read in from the file.

R filename

The restore command restores an environment saved by a save (*S*) command. The changed flag in each buffer is restored from the files; all other flags are unaffected. The input stack is reset to the top (teletype input) level, and the current buffer becomes ‘a’. ‘.’ is left at the saved value of ‘.’ in buffer ‘a’. If the *filename* is not specified, ‘q’ is used.

(.,.)sn/regular expression/replacement/

(.,.)sn/regular expression/replacement/g

The substitute command searches each addressed line for occurrences of the specified regular expression. The decimal number *n* defaults to 1 if missing. On each line in which *n* matches are found, the *n*th matched string is replaced with *replacement*. If the global replacement indicator ‘g’ follows the command, all subsequent matches on the line are also replaced. Within a line, a search starts from the character following the last match, unless the last match was an empty string, in which case the search starts at the second character following the empty string (to ensure a match is not repeated). It is an error for the substitution to fail on all addressed lines unless it is in a global command. ‘.’ is left at the last line substituted.

Any character other than newline or a numeral may be used instead of ‘/’ to delimit the regular expression and *replacement*. If the trailing delimiter is missing (i.e., an unescaped newline in the *replacement*), its presence is assumed, and the last line affected is printed, as if the substitute was followed by a *p* command. If delimiter following the expression is omitted as well, an empty *replacement* is assumed.

An ampersand ‘&’ appearing in *replacement* is replaced by the string matching the regular expression. As a more general feature, the characters ‘\n’, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between ‘\(' and ‘\)’. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of ‘\(' starting from the left.

A caret ‘^’ appearing in *replacement* behaves much like an ampersand, but provides a mechanism for case switching of alphabetic characters, as discussed under the *o* command. To include an ampersand ‘&’, caret ‘^’, backslash ‘\’, newline, or the delimiter literally in *replacement*, the character must

be preceded by a backslash. Lines may be split by substituting newline characters into them.

#### S filename

The save command saves the full buffer and register information in two files called ‘filename:aq’ and ‘filename:bq’. If the filename is absent, ‘q’ is used. If the filename has more than 12 characters after the last slash ‘/’, it is truncated to 12 characters to avoid overwriting the file.

#### (.,.)ta

The copy command acts just like the move *m* command except that a copy of the addressed lines is placed after address *a*. ‘.’ is left on the last line of the copy. The buffer to which the text was copied becomes the current buffer.

u The undo command restores the last line changed by a *s*, *u*, or *x* command. Any new lines created by splitting the original are left. It is an error if the line is not in the current buffer. ‘.’ is left at the restored line.

#### (1,\$)v/regular expression/command list

This command is the same as the global command except that the command list is executed with ‘.’ initially set to every line *except* those matching the regular expression.

#### V/regular expression/command list

This command is the same as the globuf command except that the command list is executed with the current buffer initially set to every active buffer *except* those matching the regular expression.

#### (1,\$)w filename

The write command writes the addressed lines onto the given file. If the file does not exist, it is created. The filename is remembered if there was not already a remembered file name in the current buffer. If no file name is given, the remembered file name is used. ‘.’ is unchanged. If *qed* is in *verbose* mode and the command is successful, the number of characters written is typed.

#### (1,\$)W

The *W* command is the same as the *w* command except that the addressed lines are appended to the file.

#### (.,.)x

The xform command allows one line at a time to be modified according to graphical requests. The line to be modified is typed out, and then the modify request is read from the terminal (even if the xform command is in a global command or other nested input source). Generally each character in the request specifies how to modify the character immediately above it, in the original line, as described in the following table.

#	Delete the above character.
%	Replace the above character with a space.
^	Insert the rest of the request line before the above character. If the rest of the request line is empty, insert a newline character.
\$	Delete the characters in the above line from this position on; replace them with the rest of the request line.

space or tab:

Leave above character(s) unchanged.

any other:

This character replaces the one above it.

If the request line is longer than the line to be modified, the overhang is added to the end of the line without interpretation, that is, without treating ‘#’, ‘%’, ‘^’ or ‘\$’ specially. Any characters after a ‘^’ or ‘\$’ request are not interpreted either.

Xform will not process control characters other than tab and newline, except in contexts where it need not know their width (that is, after a ‘^’ or ‘\$’ request, or in the part of either the request or the line that overhangs the other). Remember that the ERASE character (processed by the system) erases the last character typed, not the last column.

Some characters take more than one column of the terminal to enter or display. For example, entering the ERASE or KILL characters literally takes two columns because they must be escaped. To delete a multi-column character, one must type ‘#’ under all its columns. To replace a multicolumn



character, the replacement must be typed under the first column of the character. Similarly, if a replacement character is multi-columned, it replaces the character in its first column.

The tab character prints as a sequence of spaces, and may be modified as if it were that sequence. As long as the last space is unmodified, it and the remaining contiguous spaces will represent a tab.

The modification process is repeated until the request is empty. Only a newline may immediately follow the ‘x’.

#### y<condition><type>

The jump command controls execution nested input sources. The condition is compared to the truth flag to see if the jump should be performed; if a ‘t’, the jump is performed if the truth flag is true, if an ‘f’, the jump is performed if the truth flag is false, if absent the jump is always performed. Several types of jumps exist:

y[tf]o Jump out of the current input source. If the current input source is the command line for a *g*, *G*, *v*, *V* or *h* command, the command is terminated.

y[tf]N Control is transferred to absolute line *N* (an integer) in the executing buffer. The current input source must be a buffer.

y[tf]’<label>

Control is transferred to the first line found, searching forward in the buffer, that begins with a comment “<label>”. The match of the labels must be exact; regular expressions are not used to define the control label. (A tab, blank or newline after the double quote specifies a null label: a line beginning “ LAB” cannot be transferred to by this form of jump.)

If no such label is found, control resumes at the character after the label in the jump command. The current input source must be a buffer.

y[tf]`<label>

Similar to ‘y’<label>’, but the search is in the opposite (reverse) direction.

y[tf] If no recognized type is given, input is skipped up to the next newline.

It is an error if reading the label or line number for a jump command causes the current input source (i.e. buffer) to be ‘popped.’ This can happen if the label is the last word in the buffer, but can be circumvented by putting an extra blank or newline after the jump command.

#### (.,.)zXc

*Qed* has 56 registers labeled by bnames. Three of these, registers ‘T’, ‘C’, and ‘U’, are reserved: ‘T’ is the truth flag, ‘C’ is the count, ‘U’ contains the UNIX command from the most recent bang, crunch, zap, or pipe command. The contents of register *X*, where *X* is a bname, can be inserted into the input stream with the special character “\zX”. The command “zX” specifies register *X* as the argument to the operation character (signified above by *c*) that follows it. In the description below, *N* stands for a possibly signed decimal integer and *S* stands for a newline-terminated string. Newlines may be embedded in registers by escaping them with a backslash. Although some of the register commands refer to addressed lines, ‘.’ is unaffected by a *z* command. The operations are as follows:

p Print the contents of the register in ‘p’ format.

l Print the contents of the register in ‘l’ format.

. Set the register to the contents of the addressed line.

/reg-exp/Set the register to the portion of current line that matches the regular expression in slashes. If no such pattern is found, the register is cleared. The truth flag is set according to whether a match was found.

:S Set register to the string following the colon.

Y Make a direct copy of register *Y* in register *X*, without interpreting special characters. *Y* is any register bname.

+N Increment by *N* the ASCII value of each character in the register. Similarly, a ‘-’ decrements each character.

=S (Or ‘<’ or ‘>’ or ‘!=’ or ‘!<’ or ‘!>’.) Set truth flag to the result of the lexical comparison of the register and the string *S*.

n Set the count to the length of the register.

*N* (Or *'(.)* 'Take' the first *N* characters of the register, i.e. truncate at the *N*+1'th character. *'(* ('drop') is the complementary operator; it deletes the first *N* characters from the register. If *N* is negative, the break point is  $|N|$  from the end.

*/reg-exp/*

Set the count to the starting index of the regular expression in the register. Set the truth to whether the expression matches any of the register.

*sn/reg-exp/replacement/*

*sn/reg-exp/replacement/g*

Perform a substitute command with semantics identical to the *s* command, but in the text of the register, not a line of the buffer.

*C* 'Clean' the register: collapse each occurrence of 'white space' in the register to a single blank, and delete initial and trailing white space.

*{ S* Set the register to the value of the shell environment variable *S*, whose name may be terminated by a space, tab, newline or *'}*'.

The registers can also be manipulated as decimal numbers. Numerical operations are indicated by a number sign '#' after the register name: e.g. *'zx#+2'*. It is an error to attempt to perform arithmetic on a register containing non-numeric text other than a leading minus sign. The numerical operations are:

*a* Set the value of the register to be the value of the address given to the command; e.g. *'\$za#a'* sets register 'a' to the number of lines in the buffer.

*r* Set register *X* to be the first address given the command, and *X*+1 to be the second. If *X* is *' '*, an error is generated. For example, *'5,\$zi#r'* sets register 'i' to 5, and register 'j' to the value of '\$'. *'.'* is unchanged. This command is usually used to pass addresses to a command buffer.

*n* Set register to the length of the addressed line.

*:N* Set register to *N*. Scanning of the number stops at the first non-numeric character, not at the end of the line.

*+N* Increment register by *N*. *'-'*, *'\*'*, *'/'*, and *'%'* decrement, multiply, divide, or modulo the register by *N*.

*P* Set register to the decimal value of the process id of *qed*.

*=N* (Or *'<'* or *'>'* or *'!='* or *'!<'* or *'!>'*.) Set truth flag to the result of the numeric comparison of the register and the number *N*.

Several numerical operations may be combined in one command (and it is more efficient to do so when possible.) For example, *'\$zd#a-3'* sets register 'd' to three less than the value of '\$'.

*Z* The zero command clears the current buffer. The contents, filename and all flags for the buffer are zeroed. The character after the 'Z' must be a blank, tab or newline.

*( \$ ) =*

The line number of the addressed line is typed. *'.'* is unchanged.

*!<UNIX command>*

The bang command sends the command line after the *'!'* to the UNIX shell to be interpreted as a command. Embedded newlines must be preceded by a backslash. The signals INTR, QUIT, and HUP are enabled or disabled as on entry to *qed*. At the completion of the command, if *qed* is in *verbose* mode, an *'!'* is typed. The return status of the command is stored in the truth flag. *'.'* is unchanged.

The command line is stored in register 'U'. If a second *'!'* immediately follows the first, it is replaced with the uninterpreted contents of this register. Thus *'!!'* repeats the most recent bang command, and *'!! | wc'* repeats the command with an additional pipeline element added.

*( 1 , \$ ) > <UNIX command>*

The zap command is similar to the bang command, but the addressed lines become the default standard input of the command. The command is stored in register 'U', as for bang; *'>>'* corresponds to *'!!'*.

( \$ ) <<UNIX command>>

The crunch command is similar to the bang command, but the standard output of the command is appended to the current buffer after the addressed line, as though read with an *r* command from a temporary file. The command is stored in register ‘U’ as for bang; ‘<<’ corresponds to ‘!!’. ‘.’ is left at the last line read.

( 1 , \$ ) | <UNIX command>

The pipe command is similar to the bang command, but the addressed lines become the default standard input of the command, and are replaced by the standard output of the command. The command is stored in register ‘U’ as for bang; ‘|’ corresponds to ‘!!’. If the command returns non-zero status, the original lines are not deleted. ‘.’ is left at the last line read.

( . ) "

The comment command sets dot to the addressed line, and ignores the rest of the line up to the first following double quote or newline. If, however, the character immediately after the double quote is a second double quote (i.e. the command is “""”), the text which would normally be ignored is typed on the standard output. Special characters in the text will be interpreted, whether or not the text is printed, so to print a message such as “Type \bx” requires the command “"" Type \cbx”. Commented lines are used as labels by the *y* (jump) command.

% The register print command displays the name and value of all defined registers, followed by the \p (‘P’) and \r (‘R’) pseudo-registers, and the browse (‘B’) pseudo-register, if defined.

# The numeric register print command displays the name and value of all defined registers with numeric values.

( .+1 , .+1 ) <newline>

An address or addresses alone on a line cause the addressed lines to be printed. If the last address separator before the newline was ‘;’, only the final addressed line is printed. A blank line alone causes the contents of the browse pseudo-register (described with the *o* command) to be executed. If the register is null, as it is initially, the newline command behaves as though the register contains ‘.+1p’.

### Special Characters

*Qed* has some special character sequences with non-literal interpretations. These sequences are processed at the *lowest* level of input, so their interpretation is completely transparent to the actual commands. Whenever input from the user is expected, a special character can appear and will be processed. Special characters can be nested in the sense that, for example, a buffer invoked by ‘\b’ can contain a register invocation ‘\z’. Backslashed escape sequences such as ‘\(' in regular expressions are *not* special characters, so are not interpreted at input. The sequence ‘\(' is left untouched by the input mechanism of *qed*; any special meaning it receives is given it during regular expression processing. The special characters are:

\b<bname>

The ‘b’ must be followed by a bname. When ‘\bX’ is typed, the contents of buffer X, up to but *not including* the last newline, are read as if they were entered from the keyboard. Typically, the missing newline is replaced by the newline which appears after the buffer invocation. Changing the contents of an executing buffer may have bizarre effects.

\B Equivalent to current buffer’s bname.

\c The sequence \c is replaced by a single backslash, which is not re-scanned. The effect of the ‘c’ is to delay interpretation of a special character.

\f Equivalent to current buffer’s file name.

\F<bname>

Equivalent to the file name in the named buffer.

\l One line is read from the standard input up to, but *not including* the terminal newline, which is discarded. Note that the first invocation will read the remainder of the last line entered from the keyboard. For example, if a buffer is invoked by typing the line:

\bxjunk

the first \l in buffer ‘x’ will return the string ‘junk’.

- `\N` Equivalent to a newline. Primarily useful when delayed.
- `\p` Equivalent to the most recent regular expression used.
- `\r` Equivalent to the replacement text of the most recent substitute or join command.
- `\z<bname>`  
Equivalent to the contents of register `\zX`. If the register changes during execution, the changes appear immediately and affect execution. If a '+' ('-') appears between the 'z' and the bname, the ASCII values of the characters in the register are incremented (decremented) by one before interpretation. If a '#' precedes the '+' ('-') the contents of the register are numerically incremented (decremented).
- `\"` The sequence `\"` means 'no character at all'! It is primarily used to delay interpretation of a period that terminates an append, until the second or third time it is read (e.g. in loading execution buffers): the sequence `\c"` at the beginning of a line puts a period on the line which will terminate an append the second time it is read.
- `\[bflprz]`  
If an apostrophe appears between the backslash and the identifying character for one of the special characters `\b`, `\f`, `\F`, `\l`, `\p`, `\r` or `\z`, interpretation is as usual except that any further special characters *embedded* in the buffer, register, etc. are *not* interpreted. Actually, any special character may be quoted, but in forms such as `\`B`, the quote has no effect.

A special character is interpreted immediately when it appears in the input stream, whether it is currently coming from the teletype, a buffer, a register, etc. (This includes characters read when typing a special character: `\b\za`, with register 'a' containing the character 'X', invokes the literal contents of buffer 'X'.) Thus, interpretation is recursive unless the special character is `\c`. Special characters appearing in text processed in a command such as move, read or write, are *not* interpreted. If the backslash-character pair is not a special character from the above list, it is passed unchanged. Interpretation may be delayed using `\c`; for example, if a `\bx` is to be appended to a buffer for later interpretation, the user must type `\cbx`. To delay interpretation *n* times, *n* `c`'s must be placed between the backslash and the identifying character. In regular expressions and substitutes, a backslash preceding a metacharacter turns off its special meaning. Even in these cases, a backslash preceding an ordinary character is not deleted, unlike in *ed*. For example, since the 'g' command must read its entire line, a `\zx` in a substitute driven by a global must be delayed if the contents of the register are to be different for each line, but since `\&` is not a special character except to the substitute, its interpretation need not be delayed:

```
zA#:1
g/\$/ s/\.xyz/\czA \&/p zA#+1
```

globally searches for lines with a literal currency sign, and on each one substitutes for `.xyz` the contents of register 'A' at the time of substitution, followed by a space and a literal ampersand, prints the result and increments register 'A'. As a second example, the substitute

```
s/xyz/\\N&/
```

replaces `xyz` with a newline followed by `xyz`. Note that the `\\N` is interpreted as 'backslash followed by newline,' as the sequence `\\` has no special meaning in *qed* outside of regular expressions and replacement text. However, to match, say, `\\z` using a regular expression, it must be entered as `\\c\\z`.

If an interrupt signal (ASCII DEL) is sent, *qed* prints '??' and returns to its command level. If a hangup signal is received, *qed* executes the command `S qed.hup`.

Some size limitations: 512 characters per line, 256 characters per global command list, 1024 characters of string storage area, used for storing registers, file names and regular expressions, 16 levels of input nesting, and 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

## FILES

/tmp/q#, temporary; '#' is the process number (six decimal digits).

## DIAGNOSTICS

Diagnostics are in the form of ‘?’ followed by a single letter code. If the diagnostic is because of an inaccessible file, the offending file name is also displayed. If input is not from the highest level (i.e. the standard input, usually the terminal), a traceback is printed, starting with the lowest level. The elements of the traceback are of the form ?bXM.N or ?zXN, where X is the buffer or register being executed when the error was encountered, M is the line number in the buffer and N is the character number in the line or register. The possible errors are:

0	non-zero status return in / command
F	bad bname for \F
G	nested globuf commands
N	last line of input did not end with newline
O	unknown option in the o?c command
R	restore (R) command failed (file not found or bad format)
T	I/O error or overflow in tempfile
Z	out of string space; clear a few registers or file names
a	address syntax
b	bad bname in a b command or for \b
c	ran out of core
f	filename syntax error
g	nested global commands
i	more than 52 files in initialization argument list
k	bad bname in k command
l	an internal table length was exceeded
m	tried to move to an illegal place (e.g. 1,6m4)
o	error opening or creating a file
p	bad regular expression (pattern) syntax
q	e with the current changed flag set, or q with any changed flag set
r	read error from file
s	no substitutions found
t	bad x command data or single-case terminal
u	no line for u command to undo
x	command syntax error
w	write error on file
y	bad jump command (including popping the input buffer while scanning the label)
z	bad register bname
	failure to create pipe for <, / or > command
#	bad numeric register operation
\$	line address out of range
?	interrupt
/	line search failed
[	bad index in a register take or drop command
\	attempt to recursively append a buffer
!	jackpot — you found a bug in regular expression matching

## SEE ALSO

qedbufs(1)  
 A Tutorial Introduction to the ED Text Editor (B. W. Kernighan)  
 Programming in Qed: a Tutorial (Robert Pike)  
 ed(1)

## U of T INFO

Written at U of T, based on several incarnations of *ed*, with contributions from Tom Duff, Robert Pike, Hugh Redelmeier and David Tilbrook.

## BUGS

The changed flag is not omniscient; changing the contents of the file outside of *qed* will fool it. Xform *could* work on single-case terminals, but backslashes become very confusing for the user. On the PDP-11, numeric registers are 16-bit integers, but the count is a 32-bit integer.

**NAME**

qsnap – high resolution digital film printer

**SYNOPSIS**

qsnap [ -bfmrXRXY ] [ *N* ... ] *file*

**DESCRIPTION**

*Qsnap* produces images on a QCR digital film printer. The input *file* should be in the form of *picfile(5)*, 3-channel for color or 1-channel for black and white.

Option letters appear in one string. Certain options require a numeric argument, *N*, which follows the string as a separate argument.

- b N** Set the brightness of the image to *N*,  $0 \leq N \leq 8$ ; 0 is brightest. The default brightness value is 2 in high resolution mode (4K×4K pixels), and 0 in low resolution mode (2K×2K pixels) as set by option **R**.
- f N** Correct the exposure for a given type of film. Valid film types are: 7, 8, and 0 to 5 inclusive.
  - 0 = (default) Polaroid Type 52 (4×5 inch).
  - 1 = linear correction table.
  - 2 = Polaroid Type 559 color pack film (4×5 inch), 2K mode.
  - 3 = Ektachrome 100 color film (35mm), 2K mode.
  - 4 = Ektachrome 100 color film (35mm), 4K mode.
  - 5 = Polaroid Type 559 color pack film (4×5 inch), 4K mode.
  - 7 = Polaroid Type 809 color film (8×10 inch), 4K mode.
  - 8 = Tmax-100 black&white film (35mm), 2K or 4K mode.
- m N** Set the enlargement factor. The image is enlarged with a simple box filter. Default values are *N*=3 for 2K resolution and *N*=6 for 4K.
- r** Expose the red channel of a color image only (for multiple red overlays).
- x N** Expose the image *N* times. (Useful if the maximum brightness value is not bright enough.)
- R N** Set the resolution, where *N* is either 2 or 4, to select low (2K×2K) or high (4K×4K) resolution, respectively. The default it to leave the resolution unchanged.
- X N** Offset the image along the x-coordinate by *N* pixels. The 35mm camera in 4K mode may require an X- and/or a Y-offset. The offset in each direction is multiplied by the enlargement factor.
- Y N** Offset the image along the y-coordinate by *N* pixels.

The imaging resolution for 35mm film is 114 pixels/mm (2895 dots/inch) in high resolution mode, and 57 pixels/mm in low resolution mode. The maximum size image that fits a 35mm negative is 3840x3072 pixels. It takes about 2 minutes to render such an image in black and white, or 6 minutes in color. Since the color film is usually less sensitive to red, it is good practice to expose the red channel of a color image twice, using option **r**. Kodak Ektachrome color film, 100 ASA, or Kodak Tmax-100 black and white film are recommended. For 100 ASA film, imaging at brightness level 2 produces the best results.

With the 4×5 inch module, the imaging resolution is 34 pixels/mm (864 dots/inch) in high resolution mode, 17 pixels/mm in low resolution. Polaroid Type 559 color film or Polaroid Type 52 black and white film is recommended.

With the 8×10 inch module, the imaging resolution is 17 pixels/mm. This module can only be used in high resolution mode. Polaroid Type 809 color film is recommended.

**SEE ALSO**

*pico(1)*, *bcp(1)*, *cscan(1)*, *imscan(1)*, *picfile(5)*

**NAME**

random, fortune – sample lines from a file, return cookies

**SYNOPSIS**

**random** [ **-e** ] [ *n* ]

**/usr/games/fortune** [ *file* ]

**DESCRIPTION**

*Random* reads the standard input and copies each line to the standard output with probability  $1/n$ . The default value of *n* is 2.

Option **-e** writes no output and returns a random exit code in the range  $[0, n-1]$ .

*Fortune* prints a one-line aphorism chosen at random. If a *file* is specified, the saying is taken from that file; otherwise it is selected from

**FILES**

`/usr/games/lib/fortunes`

`/usr/games/lib/fortunes.index` fast lookup table, maintained automatically

**BUGS**

Successive results of option **-e** are highly correlated if *random* is called more than once per second.

**NAME**

rates – show system share scheduling rates by scheduling group

**SYNOPSIS**

rates [-Kn] [-cn] [-sn] [-u]

**DESCRIPTION**

*Rates* prints a table of rates by scheduling group, which purports to show the share of the system being allocated to each group of users. In particular, it is possible to compare the actual working rate with the intended and effective share.

The table contains ‘#’s to indicate rate of consumption of resources for each group, an ‘I’ indicating the intended share, and (if different) an ‘!’ indicating the effective share. Any difference between intended and effective share is caused by the recent history of usage.

Invoked without arguments, *Rates* will print the table and exit. The flags affect operation as follows:-

<i>flag</i>	<i>meaning</i>
<b>-Kn</b>	Set the <i>half-life</i> for decaying the displayed rate to <b>n</b> seconds [default 4].
<b>-cn</b>	Continuous operation, where <b>n</b> , if present, limits the number of cycles.
<b>-sn</b>	Delay time between updates becomes <b>n</b> seconds [default 4].
<b>-u</b>	The display will show users, instead of scheduling groups.

**EXAMPLES**

rates -c | dis

**SEE ALSO**

dis(1), shstats(1), ustats(1), share(5).



**NAME**

ratfor – rational Fortran dialect

**SYNOPSIS**

**ratfor** [ *option ...* ] [ *filename ...* ]

**DESCRIPTION**

*Ratfor* converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
```

```
switch (integer value) {
```

```
    case integer:    statement
```

```
    ...
```

```
    [ default: ]    statement
```

```
}
```

loops: while (condition) statement

```
for (expression; condition; expression) statement
```

```
do limits statement
```

```
repeat statement [ until (condition) ]
```

```
break
```

```
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:

```
multiple statements/line; automatic continuation
```

comments:

```
# this is a comment
```

translation of relationals:

```
>, >=, etc., become .GT., .GE., etc.
```

return (expression)

```
returns expression to caller from function
```

define:

```
define name replacement
```

include:

```
include filename
```

*Ratfor* is best used with [f77\(1\)](#).

**SEE ALSO**

[efl\(1\)](#), [f77\(1\)](#), [struct\(1\)](#)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

**NAME**

rc, cd, wait, whatis – command language

**SYNOPSIS**

```
rc [ -dilxepv ] [ -c command ] [ file [ arg ... ] ]
```

**DESCRIPTION**

*Rc* is the Plan 9 shell. It executes command lines read from a terminal or a file or, with the **-c** flag, from *rc*'s argument list.

**Command Lines**

A command line is a sequence of commands, separated by ampersands or semicolons (& or ;) and terminated by a newline. The commands are executed in sequence from left to right. *Rc* does not wait for a command followed by & to finish executing before starting the following command. Whenever a command followed by & is executed, its process id is assigned to the *rc* variable **\$apid**. Whenever a command *not* followed by & exits normally, the low 8 bits of the argument to its *exit(2)* call are assigned to the *rc* variable **\$status**. If it terminates abnormally, **\$status** has value 1000 plus the termination status provided to wait(2).

A long command line may be continued on subsequent lines by typing a backslash (\) followed by a newline. This sequence is treated as though it were a blank. Backslash is not otherwise a special character.

A number-sign (#) and any following characters up to (but not including) the next newline are ignored, except in quotation marks.

**Simple Commands**

A simple command is a sequence of arguments interspersed with I/O redirections. If the first argument is the name of an *rc* function or of one of *rc*'s built-in commands, it is executed by *rc*. Otherwise if the name contains a slash (/), it must be the pathname of the program to be executed. Names containing no slash are searched for in a list of directory names stored in **\$path**. The first executable file of the given name found in a directory in **\$path** is the program to be executed.

The first word of a simple command cannot be a keyword unless it is quoted or otherwise disguised. The keywords are

```
for in while if not switch fn ! @
```

**Arguments and Variables**

A number of constructions may be used where *rc*'s syntax requires an argument to appear. In many cases a construction's value will be a list of arguments rather than a single string.

The simplest kind of argument is the unquoted word: a sequence of one or more characters none of which is a blank, tab, newline or any of the following:

```
# ; & | ^ $ = \ ' { } ( ) < >
```

An unquoted word that contains any of the characters \* ? or [, is a pattern for matching against file names. The character \* matches any sequence of characters, ? matches any single character and [*class*] matches any character in the *class*. If the first character of *class* is ^, the class is complemented. The *class* may also contain pairs of characters separated by -, standing for all characters lexically between the two. The character / must appear explicitly in a pattern, as must the first character of the pathname components . and ... A pattern is replaced by a list of arguments, one for each pathname matched, except that a pattern matching no names is not replaced by the empty list, but rather stands for itself. Pattern matching is done after all other operations. Thus,

```
x=/tmp echo $x^/*.c
```

matches **/tmp/\*.c**, rather than matching **/\*.c** and then prepending **/tmp**.

A quoted word is a sequence of characters surrounded by single quotes ('). A single quote is represented in a quoted word by a pair of quotes (").

Each of the following is an argument.

(arguments)

The value of a sequence of arguments enclosed in parentheses is a list comprising the members of each element of the sequence. Argument lists have no recursive structure, although their syntax may suggest it. The following are entirely equivalent:

```
echo hi there everybody
((echo) (hi there) everybody)
```

***\$argument******\$argument(subscript)***

The *argument* after the **\$** is the name of a variable whose value is substituted. Multiple levels of indirection are possible, but of questionable utility. Variable values are lists of strings. If *argument* is a number *n*, the value is the *n*th element of **\$\***, unless **\$\*** doesn't have *n* elements, in which case the value is empty. If *argument* is followed by a parenthesized list of subscripts, the value substituted is a list composed of the requested elements (origin 1). The parenthesis must follow the variable name with no spaces. Assignments to variables are described below.

 ***\$#argument***

The value is the number of elements in the named variable. A variable never assigned a value has zero elements.

***{command}***

*rc* executes the *command* and reads its standard output, splitting it into a list of arguments, using characters in **\$ifs** as separators. If **\$ifs** is not otherwise set, its value is **'\t\n'**.

***<{command}******>{command}***

The *command* is executed asynchronously with its standard output or standard input connected to a pipe. The value of the argument is the name of a file referring to the other end of the pipe. This allows the construction of non-linear pipelines. For example, the following runs two commands **old** and **new** and uses **cmp** to compare their outputs

```
cmp <{old} <{new}
```

This feature does not work on systems that do not support **/dev/fd** or its equivalent, e.g. on Sunos and System V.

***argument^argument***

The **^** operator concatenates its two operands. If the two operands have the same number of components, they are concatenated pairwise. If not, then one operand must have one component, and the other must be non-empty, and concatenation is distributive.

**Free Carets**

In a most circumstances, *rc* will insert the **^** operator automatically between words that are not separated by white space. Whenever one of **\$ ' {** follows a quoted or unquoted word or an unquoted word follows a quoted word with no intervening blanks or tabs, a **^** is inserted between the two. If an unquoted word immediately follows a **\$** and contains a character other than an alphanumeric, underscore, or **\***, a **^** is inserted before the first such character. Thus

```
cc -$flags $stem.c
```

is equivalent to

```
cc -^flags $stem^.c
```

**I/O Redirections**

The sequence **>file** redirects the standard output file (file descriptor 1, normally the terminal) to the named *file*; **>>file** appends standard output to the file. The standard input file (file descriptor 0, also normally the terminal) may be redirected from a file by the sequence **<file**, or from an inline 'here document' by the sequence **<<eof-marker**. The contents of a here document are lines of text taken from the command input stream up to a line containing nothing but the *eof-marker*, which may be either a quoted or unquoted word. If *eof-marker* is unquoted, variable names of the form **\$word** have their values substituted from *rc*'s environment. If **\$word** is followed by a caret (**^**), the caret is deleted. If *eof-marker* is quoted, no substitution occurs.

Redirections may be applied to a file-descriptor other than standard input or output by qualifying the redirection operator with a number in square brackets. For example, the diagnostic output (file descriptor 2) may be redirected by writing **cc junk.c >[2]junk**.

A file descriptor may be redirected to an already open descriptor by writing **>[fd0=fd1]** or **<[fd0=fd1]**. *fd1* is a previously opened file descriptor, and *fd0* becomes a new copy (in the sense of *dup(2)*) of it. A file descriptor may be closed by writing **>[fd0=]** or **<[fd0=]**.

Redirections are executed from left to right. Therefore, **cc junk.c >/dev/null >[2=1]** and **cc junk.c >[2=1] >/dev/null** have different effects – the first puts standard output in **/dev/null**, and then puts diagnostic output in the same place, where the second directs diagnostic output to the terminal and sends

standard output to **/dev/null**.

### Compound Commands

A pair of commands separated by a pipe operator (**|**) is a command. The standard output of the left command is sent through a pipe to the standard input of the right command. The pipe operator may be decorated to use different file descriptors. **[[fd]** connects the output end of the pipe to file descriptor *fd* rather than 1. **[[fd0=fd1]** connects output to *fd0* of the left command and input to *fd1* of the right command.

A pair of commands separated by **&&** or **||** is a command. In either case, the left command is executed and its exit status examined. If the operator is **&&** the right command is executed if the left command's status is zero. **||** causes the right command to be executed if the left command's status is non-zero.

The exit status of a command may be inverted (non-zero is changed to zero, zero is changed to one) by preceding it with a **!**.

The **|** operator has highest precedence, and is left-associative (i.e. binds tighter to the left than the right.) **!** has intermediate precedence, and **&&** and **||** have the lowest precedence.

The unary **@** operator, with precedence equal to **!**, causes its operand to be executed in a subshell.

Each of the following is a command.

#### **if ( list ) command**

A *list* is a sequence of commands, separated by **&**, **;** or newline. It is executed and if its exit status is zero, the *command* is executed.

#### **if not command**

The immediately preceding command must have been **if ( list ) command**. If its condition was non-zero, the *command* is executed.

#### **for ( name in arguments ) command**

#### **for ( name ) command**

The *command* is executed once for each *argument* with that argument assigned to *name*. If the argument list is omitted, **\$\*** is used.

#### **while ( list ) command**

The *list* is executed repeatedly until its exit status is non-zero. Each time it returns zero status, the *command* is executed. The empty *list* always yields zero status.

#### **switch(argument){list}**

The *list* is searched for simple commands beginning with the word **case**. (The search is only at the 'top level' of the *list*. That is, **cases** in nested constructs are not found.) *Argument* is matched against each word following **case** using the pattern-matching algorithm described above, except that **/** and the first characters of **.** and **..** need not be matched explicitly. When a match is found, commands in the list are executed up to the next following **case** command (at the top level) or the closing parenthesis.

#### **{list}**

Braces serve to alter the grouping of commands implied by operator priorities. The *body* is a sequence of commands separated by **&**, **;** or newline.

#### **fn name{list}**

#### **fn name**

The first form defines a function with the given *name*. Subsequently, whenever a command whose first argument is *name* is encountered, the current value of the remainder of the command's argument list will be assigned to **\$\***, after saving its current value, and *rc* will execute the *list*. The second form removes *name*'s function definition.

#### **fn signal{list}**

#### **fn signal**

A function with the name of a signal, in lower case, is defined in the usual way, but called when *rc* receives that signal; see [signal\(2\)](#). By default *rc* exits on receiving any signal, except when run interactively, in which case interrupts and quits normally cause *rc* to stop whatever it's doing and start reading a new command. The second form causes *rc* to handle a signal in the default manner. *Rc* recognizes an artificial signal, **sigexit**, which occurs when *rc* is about to finish executing.

#### **name=argument command**

Any command may be preceded by a sequence of assignments interspersed with redirections. The assignments remain in effect until the end of the command, unless the command is empty

(i.e. the assignments stand alone), in which case they are effective until rescinded by later assignments.

### Built-in Commands

These commands are executed internally by *rc*, usually because their execution changes or depends on *rc*'s internal state.

**.file ...**

Execute commands from *file*. **\$\*** is set for the duration to the remainder of the argument list following *file*. *File* is searched for using **\$path**.

**builtin command ...**

Execute *command* as usual except that any function named *command* is ignored.

**cd [dir]**

Change the current directory to *dir*. The default argument is **\$home**. *dir* is searched for in each of the directories mentioned in **\$cdpath**.

**eval [arg ...]**

The arguments are concatenated separated by spaces into a single string, read as input to *rc*, and executed.

**exec [command ...]**

*Rc* replaces itself with the given (non-built-in) *command*.

**exit [status]**

Exit with the given exit status. If none is given, the current value of **\$status** is used.

**shift [n]**

Delete the first *n* (default 1) elements of **\$\***.

**umask [octal]**

Set *rc*'s file-creation mask (see [umask\(2\)](#)) to the given octal value. If no value is given, the current mask value is printed.

**wait [pid]**

Wait for the process with the given *pid* to exit. If no *pid* is given, all outstanding processes are waited for.

**whatis name ...**

Print the value of each *name* in a form suitable for input to *rc*. The output is an assignment to any variable, the definition of any function, a call to **builtin** for any built-in command, or the completed path name of any executable file.

**subject pattern ...**

The *subject* is matched against each *pattern* in sequence. If it matches any pattern, **\$status** is set to zero. Otherwise, **\$status** is set to one. Patterns are the same as for file name matching, except that */* and the first character of *.* and *..* need not be matched explicitly. The *patterns* are not subjected to file name matching before the *command* is executed, so they need not be enclosed in quotation marks.

### Environment

The *environment* is a list of strings made available to executing binaries. *Rc* creates an environment entry for each variable whose value is non-empty, and for each function. The string for a variable entry has the variable's name followed by = and its value. If the value has more than one component, these are separated by ctrl-a ('\001') characters. The string for a function is just the *rc* input that defines the function.

When *rc* starts executing it reads variable and function definitions from its environment.

### Special Variables

The following variables are set or used by *rc*.

**\$\*** Set to *rc*'s argument list during initialization. Whenever a *.* command or a function is executed, the current value is saved and **\$\*** receives the new argument list. The saved value is restored on completion of the *.* or function.

**\$apid** Whenever a process is started asynchronously with **&**, **\$apid** is set to its process id.

**\$home**

The default directory for **cd**. Initially, if **\$home** is not set and **\$HOME** is, then **\$home** is set to the value of **\$HOME**.

- \$ifs** The input field separators used in backquote substitutions. If **\$ifs** is not set in *rc*'s environment, it is initialized to blank, tab and newline.
- \$path** The search path used to find commands and input files for the `.` command. If not set in the environment, it is initialized by **path=(. /bin /usr/bin)**.
- \$pid** Set during initialization to *rc*'s process id.
- \$prompt**  
When *rc* is run interactively, the first component of **\$prompt** is printed before reading each command. The second component is printed whenever a newline is typed and more lines are required to complete the command. If not set in the environment, it is initialized by **prompt=( '% ' ' ' )**.
- \$status**  
Set to the low 8 bits of the *exit(2)* argument of a normally terminating binary (unless started with **&**), or to 1000 plus the termination status on abnormal termination. **!** and `&` also change **\$status**. Its value is used to control execution in **&&**, `||`, **if** and **while** commands. When *rc* exits at end-of-file of its input or on executing an **exit** command with no argument, **\$status** is its exit status.

### Invocation

If *rc* is started with no arguments it reads commands from standard input. Otherwise its first non-flag argument is the name of a file from which to read commands (but see **-c** below). Subsequent arguments become the initial value of **\$\***. *Rc* accepts the following command-line flags.

- c string**  
Commands are read from *string*.
- d** Debugging flag, causes *rc* only to catch **SIGINT**, so that **SIGQUIT** will cause it to dump core.
- e** Exit if **\$status** is non-zero after executing a simple command.
- i** If **-i** is present, or *rc* is given no arguments and its standard input is a terminal, it runs interactively. Commands are prompted for using **\$prompt** and **SIGINT** and **SIGQUIT** are caught and sloughed off.
- l** If **-l** is given or the first character of argument zero is **-**, *rc* reads commands from **\$home/.rcrec**, if it exists, before reading its normal input.
- p** A no-op.
- v** Echo input on file descriptor 2 as it is read.
- x** Print each simple command before executing it.

### BUGS

It's too slow and too big.  
There should be away to match patterns against whole lists rather than just single strings.  
Using `&` to check the value of **\$status** changes **\$status**.  
Functions that use here documents don't work.  
Environment entries for variables are kludgy for UNIX compatibility. Woe betide the imported variable whose value contains a ctrl-a.

**NAME**

rcp – remote file copy

**SYNOPSIS**

**rcp** *filename1 filename2*

**rcp** [ **-r** ] *filename ... directory*

**DESCRIPTION**

*Rcp* copies files across TCP/IP connections. Each *filename* or *directory* argument is either a remote file name of the form:

*hostname:path*

or a local file name (containing no `:` unless preceded by `/`).

If a *filename* is not a full path name, it is interpreted relative to your home directory on machine *hostname*. A *path* on a remote host may be quoted to cause metacharacters to be interpreted remotely.

Your current local user name must exist on *hostname* and allow remote command execution by *rsh*; see [con\(1\)](#).

*Rcp* handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

*username@hostname:filename*

to use *username* rather than your current local user name as the user name on the remote host. In this usage, *hostname* may be a full internet domain name.

The option is

**-r** Copy each subtree rooted at *filename*; in this case the destination must be a directory.

**FILES**

.cshrc

.login

.profile

**SEE ALSO**

[con\(1\)](#), [cu\(1\)](#), [push\(1\)](#), [uucp\(1\)](#)

**BUGS**

There is no check against copying a file onto itself.

Certain cases where a file name is given when a directory is required are not diagnosed.

**NAME**

readslow – watch a growing file

**SYNOPSIS**

**readslow** [ **-e** ] [ file ]

**DESCRIPTION**

*Readslow* copies the contents of the *file* (standard input by default) to the standard output. Upon reaching an apparent end of file, it tries periodically to read further, thus letting you watch the progress of a file that another process is writing into. Option **-e** causes *readslow* to begin at the present end of the *file*.

**SEE ALSO**

tail(1)



**NAME**

refer, lookbib, pubindex – maintain and use bibliographic references

**SYNOPSIS**

**refer** [ *option ...* ] [ *file ...* ]

**lookbib** [ *file ...* ]

**pubindex** *file ...*

**DESCRIPTION**

*Refer* is a preprocessor for *nroff* or *troff(1)* that finds and formats references. The input files (standard input default) are copied to the standard output, except for lines between `. [` and `. ]` which are assumed to contain keywords and are replaced by information from the bibliographic data base. The user may avoid the search, override fields from it, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms(6)* print the finished reference text from these strings. A flag is placed in the text at the point of reference; by default the references are indicated by numbers.

The following options are available:

- ar** Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted all author names are reversed.
- b** Bare mode: do not put any flags in text (neither numbers nor labels).
- cstring** Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *string*.
- e** Instead of leaving the references where encountered, accumulate them until a sequence of the form
 

```
. [
$LIST$
. ]
```

 is encountered, and then write out all references collected so far. Collapse references to the same source.
- kx** Instead of numbering references, use labels as specified in a reference data line beginning `%x`; by default *x* is **L**.
- lm,n** Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted the entire name or date respectively is used.
- p** Take the next argument as a file of references to be searched. The default file is searched last.
- n** Do not search the default file.
- skeys** Sort references by fields whose key-letters are in the *keys* string; permute reference numbers in text accordingly. Implies **-e**. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title use **-sA+T**.

A bibliographic reference in a **-p** file is a set of lines that contain bibliographic information fields. Empty lines separate references. Each field starts on a line beginning with `%`, followed by a key-letter, followed by a blank, and followed by the contents of the field, which continues until the next line starting with `%`. The most common key-letters and the corresponding fields are:

A	Author name
B	Title of book containing article referenced
C	City
D	Date
d	Alternate date

E	Editor of book containing article referenced
G	Government (CFSTI) order number
I	Issuer (publisher)
J	Journal
K	Other keywords to use in locating reference
M	Technical memorandum number
N	Issue number within volume
O	Other commentary to be printed at end of reference
P	Page numbers
R	Report number
r	Alternate report number
T	Title of article, book, etc.
V	Volume number
X	Commentary unused by <i>pubindex</i>

Except for A, each field should only be given once. Only relevant fields should be supplied. When *refer* is used with *eqn*, *neqn* or *tbl(1)*, *refer* should be first, to minimize the volume of data passed through pipes.

*Lookbib* accepts keywords from the standard input and searches a bibliographic data base for references that contain those keywords anywhere in the title, author, journal name, etc. Matching references are printed on the standard output. Blank lines are taken as delimiters between queries.

*Pubindex* makes a hashed inverted index to the named bibliographic *files* for use by *refer*.

## EXAMPLES

```
%T 5-by-5 Palindromic Word Squares
%A M. D. McIlroy
%J Word Ways
%V 9
%P 199-202
%D 1976
```

## FILES

```
/usr/dict/papers
    directory of default publication lists and indexes
```

```
/usr/lib/refer
    directory of programs
```

***x.ia***, ***x.ib***, ***x.ic***  
 where *x* is the first argument to *pubindex*

## SEE ALSO

M. E. Lesk, 'Some Applications of Inverted Indexes on UNIX' in AT&T Bell Laboratories, *UNIX Programmer's Manual, Volume 2*, Holt-Rinehart (1984)  
[troff\(1\)](#), [doctype\(1\)](#), [prefer\(1\)](#)

## BUGS

*Refer* is unmaintained; better use [prefer\(1\)](#).

**NAME**

remshent – remove an entry from the shares data-base

**SYNOPSIS**

**remshent** [*name/uid ...*]

**DESCRIPTION**

*Remshent* removes entries from the shares data-base in the file */etc/shares*. Entries are specified as user names, or as uids. If no arguments are given, *remshent* reads the names from standard input, one per line.

**FILES**

*/etc/shares*           for share details.  
*/etc/passwd*         for user names and IDs.

**SEE ALSO**

*lim(1)*, *pl(1)*, *shares(5)*.

**NAME**

rev, revpag – reverse lines or pages

**SYNOPSIS**

**rev** [*file ...* ]

**revpag** [*option ...* ] [*file ...* ]

**DESCRIPTION**

*Rev* copies the standard input or the named files to the standard output, reversing the order of characters in every line.

*Revpag* copies the standard input or the named files to the standard output, reversing the order of the pages. (The name *rev* means the standard input.) Options define what constitutes a ‘page’:

- d** The input is *troff(1)* output; page breaks are encoded in it.
- f** Append a new-page character (014) to the last input page (which is the first page on the output), if this page is not of the declared length.
- l *n*** Set the number of lines per page in ordinary ASCII input (66 by default). A new-page character (014) is also recognized as a page break.
- o *list*** Output only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A ‘page number’ means the ordinal position of a page in the input. A range *n-m* means pages *n* through *m*. In a range, a missing *m* means the beginning; a missing *n* means the end.

**EXAMPLES**

**rev <webster | sort | rev >walker**

From a standard Webster’s dictionary, produce Walker’s rhyming dictionary, which is alphabetized from right to left.

**tail -r <forward >backward**

Reverse the order of lines in a file; see *tail(1)*.

**NAME**

`rm` – remove (unlink) files

**SYNOPSIS**

`rm [ -fri ] file ...`

**DESCRIPTION**

`Rm` removes directory entries. If an entry was the last link to a file, the file is destroyed. If an entry is a directory it is removed only if empty. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file lacks write permission and the standard input is a terminal, a query is written to the standard output and a line is read from the standard input. If that line begins with `y` the file is deleted, otherwise the file remains. The options are

- f** (force) Ask no questions about unwritable files and report no errors.
- r** Recursively delete the entire contents of a directory and the directory itself.
- i** (interactive) Ask whether to delete each file, and, under **-r**, whether to examine each directory. If the first character of the response is `y`, the answer is yes; otherwise the answer is no.

**SEE ALSO**

[unlink\(2\)](#)

**DIAGNOSTICS**

It is forbidden to remove the file `..` merely to avoid the antisocial consequences of inadvertently doing something like `rm -r .*`.

**NAME**

rscan, pix – scan page on ricoh scanner and display on 5620

**SYNOPSIS**

**rscan** [ **-D** ] [ **-r** *n* ] [ **name** ]

**DESCRIPTION**

*Rscan* reads data from the Ricoh scanner and writes it in file *name* with a header as described in *pictures*(5). If no *name* is given, it calls the file **Junk**. The flag **-D** turns on dither mode. The flag **-r** selects the scanner resolution. Values of *n* are 0 for 200×200 dots per inch, 1 for 200×100 DPI, 2 for 300×300 DPI (default), and 3 for 240×240 DPI. The data from the scanner are bytes with the low order bit corresponding to the left edge of the paper. There are 324 bytes/scan line at the default resolution.

*Pix* displays a scanned page in a window on the 5620 terminal. Button 3 displays a menu for moving around in the data.

**SEE ALSO**

[imscan\(1\)](#), [pictures\(5\)](#)

**NAME**

sdb – symbolic debugger

**SYNOPSIS**

**sdb** [ *objfil* [ *corfil* [ *directory* ] ] ]

**DESCRIPTION**

*Sdb* is a symbolic debugger. It may examine source files, object files, and running or stopped core images.

*Objfil* is an executable program (default **a.out**) compiled with option **-g** of *cc* or *f77(1)*. *Corfil* is a core image (default **core**) resulting from the execution of *objfil*. *Directory* is the home of source files, **.** by default.

*Sdb* maintains a ‘current line’ and a ‘current file’, initially the line number and source file name where *corfil* stopped executing, or the first line in function **main** if there is no core image.

Variables are referred to by name, by structure or array reference, or by a combination thereof. Variables not in scope at the current line are referred to as *procedure:variable* (*procedure* may be a Fortran common block). In the syntax below, a ‘variable’ may also be an integer constant designating a storage location, or a variable plus a constant designating a storage offset.

Line numbers in the source program are referred to as *filename:number*, *procedure:number*, or *number* (in the current file). The number is always relative to the beginning of the file. A missing number is taken as the first line of the procedure or file.

The commands for examining data are:

**t** Print a stack trace of the terminated or stopped program.

**T** Print the top line of the stack trace.

*variablellm*

*variable?lm*

Print the value of the variable according to (an optional) length *l* and format *m*. By default *l* and *m* are taken from the variable’s declaration. The length specifier for formats **duox** is **b** (1 byte), **h** (2 bytes), or **I** (4 bytes, default). For formats **s** and **a** it is an integer string length. Punctuation */* designates variables in data segments, **?** in the text segment. Legal values for format *m* are:

<b>c</b>	character
<b>d</b>	decimal
<b>u</b>	decimal, unsigned
<b>o</b>	octal
<b>x</b>	hexadecimal
<b>f</b>	32 bit single precision floating point
<b>g</b>	64 bit double precision floating point
<b>s</b>	Assume variable is a string pointer and print characters until a null is reached.
<b>a</b>	Print characters starting at the variable’s address until a null is reached.
<b>p</b>	pointer to procedure
<b>i</b>	machine instruction

*variable=lm*

*linenumber=lm*

*number=lm*

Print the address of the variable or line number or the value of the number in the specified format, **lx** by default. The last variant may be used to convert number bases.

*variable!value*

Set the variable to the given value. The value may be an integer or character constant, a variable, or a floating-point constant (if *variable* is float or double).

The commands for examining source files are

**e** *procedure*

**e** *filename.c*

Set the current line and file. If no name is given, report the current procedure and file.

*/regular expression/*

Search forward as in *ed(1)*.

*?regular expression?*

Search backward.

**p** Print the current line.

**z** Print the current line and 9 more; set the current line to the last one printed.

control-D

Print the next 10 lines; set the current line to the last one printed.

**w** Window. Print the 10 lines around the current line.

*number*

Set the current line and print it.

*count* + Advance the current line by *count* lines. Print the new current line.

*count* - Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

*count* **r** *args*

*count* **R**

Run the program with the given arguments. The **r** command with no arguments reuses the previous arguments to the program while the **R** command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output respectively. *Count*, if given, specifies a number of breakpoints to be ignored.

*linenumber* **c** *count*

*linenumber* **C** *count*

Continue after a breakpoint or interrupt. *Count* is as for **r**. **C** continues with the signal which caused the program to stop and **c** ignores it. If a *linenumber* is given, a temporary breakpoint is placed there and is deleted when the command finishes.

*count* **s** Single step. Run the program through *count* lines, one line by default.

*count* **S** Single step, but step through subroutine calls.

**k** Kill the debugged program.

*procedure*(*arg1,arg2,...*)/*m*

Execute the named procedure with the given arguments. Arguments can be variables in scope or integer, character or string constants. If a format, */m*, is given, print the result in that form, otherwise *d*.

*linenumber* **b** *commands*

Set a breakpoint at the given line. If a procedure name without a line number is given (e.g. `proc:`), a breakpoint is placed at the first line in the procedure. If no *linenumber* is given, a breakpoint is placed at the current line.

If no *commands* are given, stop execution just before the breakpoint. Otherwise, when the breakpoint is encountered perform the semicolon-separated *commands* and then continue execution.

*linenumber* **d**

Delete a breakpoint at the given line. If no *linenumber* is given, each breakpoint location is printed and a line is read from the standard input. Answer *y* or *d* to delete it.

**B** Print a list of the currently active breakpoints.

**D** Delete all breakpoints.



**l** Print the last executed line.

*linenumber a*

Announce. If *linenumber* is of the form *proc:number* the command does `linenumber b 1` (print the line each time it's reached). If *linenumber* is of the form `proc:`, the command does `proc: b T` (print the stack frame).

Miscellaneous commands.

**!** *command*

The command is interpreted by [sh\(1\)](#).

**n** *newline*

Advance to the next storage location or source line, depending on which was last printed, and display it.

**"** *string*

Print the given string.

**q** Exit the debugger.

The following commands are intended for debugging the debugger.

**V** Print the version number.

**X** Print a list of procedures and files being debugged.

**Y** Toggle debug output.

## FILES

`a.out`  
`core`

## SEE ALSO

[adb\(1\)](#), [pi\(9\)](#) [cin\(1\)](#), [nm\(1\)](#), [a.out\(5\)](#), [bigcore\(1\)](#), [cc\(1\)](#), [f77\(1\)](#)

## BUGS

*Sdb* is old and unmaintained.

If a procedure is called when the program is not stopped at a breakpoint, a fresh core image results. Thus a procedure can't be used to extract data from a dump.

*Sdb* doesn't know Fortran: arrays are singly dimensioned and 0-indexed; scalar arguments are reported as pointers.

The default type for printing Fortran parameters is incorrect: address instead of value.

Tracebacks containing Fortran subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The meaning of control-D is nonstandard.

**NAME**

sed – stream editor

**SYNOPSIS**

**sed** [ **-n** ] *script* [ *file ...* ]

**sed** [ **-e script** ] [ **-f sfile** ] [ *file ...* ]

**DESCRIPTION**

*Sed* copies the named *files* (standard input default) to the standard output, edited according to a command script. Script options accumulate.

**-e script**

Script is given literally in command line.

**-f sfile** Script is given in file *sfile*.

**-n** Suppress the default output.

A script consists of editing commands, usually one per line. If a command ends with *;*, *{*, or *}*, the next command begins immediately thereafter. Empty commands are ignored. Commands have the form

```
[address [, address] ] function [argument ...] [;]
```

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a *D* command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a *\$* that addresses the last line of input, or a context address, */regular-expression/*, in the style of *ed(1)*, with the added convention that *\n* matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address. (Address 0 is never matched.)

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied to non-selected pattern spaces by use of the negation function *!* (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

A *text* argument consists of one or more lines, all but the last of which end with *\* to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an *s* command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 120 distinct *wfile* arguments.

**(1) a\**

*text* Append. Place *text* on the output before reading the next input line.

**(2) b label**

Branch to the *:* command bearing the *label*. If *label* is empty, branch to the end of the script.

**(2) c\**

*text* Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

**(2) d** Delete the pattern space. Start the next cycle.

- (2)**D** Delete the initial segment of the pattern space through the first newline. Start the next cycle.
- (2)**g** Replace the contents of the pattern space by the contents of the hold space.
- (2)**G** Append the contents of the hold space to the pattern space.
- (2)**h** Replace the contents of the hold space by the contents of the pattern space.
- (2)**H** Append the contents of the pattern space to the hold space.
- (1)**i**  
*text* Insert. Place *text* on the standard output.
- (2)**I** Literal. Place an unambiguous image of the pattern space on the standard output, using C escape sequences. Break long lines, indicating the breakpoint by a single backslash. Append **\n** if pattern space ends with space or newline.
- (2)**n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
- (2)**N** Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)
- (2)**p** Print. Copy the pattern space to the standard output.
- (2)**P** Copy the initial segment of the pattern space through the first newline to the standard output.
- (1)**q** Quit. Branch to the end of the script. Do not start a new cycle.
- (2)**r** *rfile*  
Read the contents of *rfile*. Place them on the output before reading the next input line.
- (2)**s**/*regular-expression/replacement/flags*  
Substitute the *replacement* string for instances of the *regular-expression* in the pattern space. Any character may be used instead of /. For a fuller description see [ed\(1\)](#); although unlike *ed*, the trailing / *must* be supplied. *Flags* is zero or more of
  - g** Global. Substitute for all non-overlapping instances of the *regular expression* rather than just the first one.
  - p** Print the pattern space if a replacement was made.
- w** *wfile*  
Write. Append the pattern space to *wfile* if a replacement was made.
- (2)**t** *label*  
Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.
- (2)**w** *wfile*  
Write. Append the pattern space to *wfile*.
- (2)**x** Exchange the contents of the pattern and hold spaces.
- (2)**y**/*string1/string2*  
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.
- (2)**!** *function*  
Don't. Apply the *function* (or group, if *function* is { }) only to lines *not* selected by the address(es).
- (0)**#** Comment. Ignore the rest of the line.
- (0)**:** *label*  
This command does nothing; it bears a *label* for **b** and **t** commands to branch to.
- (1)**=** Place the current line number on the standard output as a line.

- (2){ Execute the following commands through a matching } only when the pattern space is selected.  
 (0) An empty command is ignored.

## EXAMPLES

### sed 10q file

Print the first 10 lines of the file.

### sed '/\$/d'

Delete empty lines from standard input.

### sed 's/UNIX/& system/g'

Replace every instance of UNIX by UNIX system.

```
sed 's/ *$//drop trailing blanks
/^$/d          drop empty lines
s/ */\        replace blanks by newlines
/g
/^$/d' chapter*
```

Print the files **chapter1**, **chapter2**, etc. one word to a line.

```
nroff -ms manuscript | sed '
${
    /^$/p          if last line of file is empty, print it
}
//N              if current line is empty, append next line
/^\n$/D'         if two lines are empty, delete the first
```

Delete all but one of each group of empty lines from a formatted manuscript.

```
ls /usr/* | sed '
/^$/d          delete empty lines
/^[/[.]*:$/{
    s/:$/\//    replace : by /
    h           hold directory name
    d           don't print; get next line
}
G              append held directory name
s/\(.*\)n\(.*\)\/\2\1/' exchange file and directory
```

List all files in user directories, as **ls -d /usr/\*/\*** would do if it didn't cause argument list overflow.

## SEE ALSO

[ed\(1\)](#), [gre\(1\)](#), [awk\(1\)](#), [lex\(1\)](#), [cut\(1\)](#), [split\(1\)](#), [sam\(9\)](#)

L. E. McMahon, 'SED — A Non-interactive Text Editor', this manual, Volume 2.

## BUGS

If input is from a pipe, buffering may consume characters beyond a line on which a **q** command is executed.

**NAME**

sendnews – send news articles via mail

**SYNOPSIS**

sendnews [-o] [-a] [-b] [-n newsgroups] destination

**DESCRIPTION**

*sendnews* reads an article from its standard input, performs a set of changes to it, and gives it to the mail program to mail it to *destination*.

An 'N' is prepended to each line for decoding by *uurec(8)*.

The **-o** flag handles old format articles.

The **-a** flag is used for sending articles via the **ARPANET**. It maps the article's path from *uucphost!xxx* to *xxx@arpahost*.

The **-b** flag is used for sending articles via the **Berknet**. It maps the article's path from *uucphost!xxx* to *berkhost:xxx*.

The **-n** flag changes the article's newsgroup to the specified *newsgroup*.

**SEE ALSO**

*inews(1)*, *uurec(1)*, *recnews(1)*, *readnews(1)*, *newscheck(1)*

**NAME**

seq – print sequences of numbers

**SYNOPSIS**

seq [ **-w** ] [ **-f***format* ] [ *first* [ *incr* ] ] *last*

**DESCRIPTION**

*Seq* prints a sequence of numbers, one per line, from *first* (default 1) to as near *last* as possible, in increments of *incr* (default 1). The numbers are interpreted as floating point.

Normally integer values are printed as decimal integers. The options are

**-f***format*

Use the *printf(3)*-style *format* for printing each (floating point) number. The default is %g.

**-w**

Equalize the widths of all numbers by padding with leading zeros as necessary. Not effective with option **-f**, nor with numbers in exponential notation.

**EXAMPLES**

seq 0 .05 .1 Print **0 0.05 0.1** (on separate lines).

seq -w 0 .05 .1 Print **0.00 0.05 0.10**.

**BUGS**

Option **-w** always surveys every value in advance, although that's not necessary for integers. Thus seq -w 1000000000 is a hopeless way to get an 'infinite' sequence.

**NAME**

`server` – run anonymous command on another machine

**SYNOPSIS**

`server machine command`

**DESCRIPTION**

*Server* uses Datakit to run *command* on the named *machine*. If it has set-userid mode, the owner of *server* will log in on the remote machine. In this way, a user can execute commands on the remote machine without having a login there.

*Server* typically accepts only a small set of commands, such as *who*, *ps*, *cat*, and *date*; the list is at the discretion of the remote machine. *Server* also checks for suspicious argument characters.

**FILES**

`/etc/server` list of legal commands

**SEE ALSO**

[con\(1\)](#), [push\(1\)](#), [dcon\(1\)](#)

**NAME**

`sexist` – print sexist terms and suggest alternatives

**SYNOPSIS**

`sexist` [ **-flags** ] [ **-ver** ] [ **-f** *pfile* ] [ *file* ... ]

**DESCRIPTION**

*Sexist* locates and prints all sentences in a document that contain possibly sexist words or phrases. The word or phrase in each sentence is surrounded with stars and brackets, e.g. \*[girl]\*. The line numbers of the sentences are also printed.

Following that, *sexist* prints alternatives to the sexist phrases found in the document. The alternatives encourage:

1. describing men and women in parallel terms. For example, "men and girls" should probably be changed to "men and women."
2. using neutral terms instead of sex-related terms. For example "business executive" could easily replace the stereotypic "businessman."
3. using new, non-sexist Bell System job titles.

If the user has a file named *\$HOME/lib/sexdict*, *sexist* locates or ignores phrases contained in that file. *Dictadd*(1) can be used to set up *\$HOME/lib/sexdict*. *Dictadd* gives instructions on the necessary format for phrases to be located or ignored by *sexist*.

Options are:

**-f** *pfile* Use the user's phrase file, *pfile*, in addition to the default file of sexist phrases. When the **-f** *pfile* option is used, *sexist* does not check *\$HOME/lib/sexdict* for phrases. The format instructions given by *dictadd* should be followed in setting up *pfile*.

Two options give information about the program:

**-flags** print the command synopsis line (see above) showing command flags and options, then exit.

**-ver** print the Writer's Workbench version number of the command, then exit.

**EXAMPLE**

1. The command:

**sexist -f patfile filename**

will print sentences from *filename* that contain sexist words or phrases, including or suppressing phrases as specified in *patfile*. Suggested replacements for sexist phrases will also be printed. *Sexist* will not locate or ignore phrases in *\$HOME/lib/sexdict* when the **-f** option is used.

**FILES**

*/tmp/\$\$\** temporary files

**SEE ALSO**

*dictadd*(1)

**BUGS**

Because *sexist* does not consider context, it may bracket phrases that are used appropriately and may recommend inappropriate alternatives. It is up to the user to determine which changes should be made.

If formatting macros are included in the input text, the beginning line number of a sentence containing a sexist phrase may be a line containing a macro preceding the sentence.

*Sexist* makes no attempt to segment text into sentences accurately, e.g., it does not know any abbreviations.

*Sexist* will not find the second of two consecutive sexist phrases, nor will it find a sexist phrase at the end of a sentence.



**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

sh, cd, wait, whatis – shell, the standard command programming language

**SYNOPSIS**

**sh** [ **-acefknpstuvx** ] [ *args* ]

**DESCRIPTION**

*Sh* is a command programming language that executes commands read from a terminal or a file. See ‘Invocation’ below for the meaning of arguments to the shell.

**Definitions**

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters \*, @, #, ?, -, \$, and !. A *word* is a sequence of characters and quoted strings set off by operators, blanks, or newlines; see ‘Quoting’.

**Commands**

A *simple-command* is a sequence of *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0; see [exec\(2\)](#). The *value* of a simple-command is its exit status if it terminates normally, or **0200**+status if it terminates abnormally; see [signal\(2\)](#) for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by |. If there is more than one command, each is run in a subshell; | denotes a [pipe\(2\)](#) connecting the standard output of one command to the standard input of the next. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the pipeline; the shell does not wait and proceeds as if the pipeline had returned zero exit status. The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. One or more newlines may follow any sequencing operator (; & && ||).

One or more newlines may always be used in place of a single semicolon, and newlines may be freely inserted after any of | ; & && || ; ; **if do then elif else fi done while until**.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [ in *word* ... ; ] do *list* ; done

A for command executes a *list* of commands once for each *word*, with *name* set to each *word* in turn. If **in word ... ;** is omitted or replaced by newlines, then the *list* is executed once for each positional parameter that is set; see ‘Parameter Substitution’.

case *word* in [ *pattern* [ | *pattern* ] ... ) *list* ;; ] ... esac

A case command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see ‘File Name Generation’) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly. Newlines may precede each *pattern* and replace the last ;; before **esac**.

if *list* then *list* [ elif *list* then *list* ] ... [ else *list* ] fi

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *lists* in **elif** clauses are executed in turn until one returns zero status; then the *list* following the next **then** is executed. Otherwise, the **else list** is executed. If no **else list** or **then list** is executed, then the **if** command returns a zero exit status.

while *list* do *list* done

A while command repeatedly executes the **while list** and, if the exit status of the last command in the *list* is zero, executes the **do list**; otherwise the loop terminates. If no commands in the **do list** are executed, then the while command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*}

*list* is simply executed.

*name* () *command*

Define a function which is referenced by *name*. The body of the function is the *command*. The most useful form of *command* is a sequence of commands enclosed in braces { }. Execution of functions is described under *Execution* below.

These words are only recognized as the first word of a command and when not quoted: **if then else elif fi case esac for while until do done**.

### Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

### Command Substitution

The standard output from a command enclosed in a pair of grave accents ‘ ‘ may be used as part or all of a word; trailing newlines are removed.

### Parameter Substitution

The character \$ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

```
name=value [ name=value ] . . .
```

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

\${*parameter*}

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is \* or @, all the positional parameters, starting with \$1, are substituted (separated by spaces). Parameter \$0 is set from argument zero when the shell is invoked.

\${*parameter*:-*word*}

If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

\${*parameter*:=*word*}

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

\${*parameter*?:*word*}

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message “parameter null or not set” is printed.

\${*parameter*+*word*}

If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (: ) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- # The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
- ? The decimal value returned by the last synchronously executed command; see [exit\(2\)](#).
- \$ The process number of this shell.
- ! The process number of the last background command invoked.

The following parameters are used by the shell:

**HOME**

The default argument (home directory) for the *cd* command.

**PATH** The search path for commands; see 'Execution'.

**CDPATH**

The search path for the *cd* command.

**MAIL** If this parameter is set to the name of a mail file the shell informs the user of the arrival of mail in the specified file. The file is inspected every three minutes.

**HISTORY**

If this parameter is set to the name of a writable file, the shell appends interactive input to the file, for use by the command *=(1)*.

**PS1** Primary prompt string, by default \$.

**PS2** Secondary prompt string, by default >.

**IFS** Internal field separators, normally space, tab, and newline.

The shell gives default values to PATH, PS1, PS2 and IFS. **HOME** is set by *login(8)*.

**Blank Interpretation**

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (" or `) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

**File Name Generation**

Following substitution, each command *word* is scanned for the characters \*, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The directories . and .. (initially or after a /) are only matched by patterns beginning with an explicit period. The character / itself must be matched explicitly.

\* Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening [ is a ^ any character not enclosed is matched.

**Quoting**

These characters have a special meaning to the shell and terminate a word unless quoted:

; & ( ) | < > { } newline space tab

(The characters { and } need not be quoted inside a \${ } construction.) A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \newline is ignored. All characters enclosed between a pair of single quote marks ` (except a single quote) are quoted. Inside double quote marks "" parameter and command substitution occurs and \ quotes the characters \, `, ", and \$. "\$\*" is equivalent to "\$1 \$2 ...", whereas "\$@" is equivalent to "\$1" "\$2" ....

**Prompting**

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

**Input/Output**

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word* Use file *word* as standard input (file descriptor 0).

>*word* Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>*word* Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

`<<word` The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) `\newline` is ignored, and `\` must be used to quote the characters `\`, `$`, ```, and the first character of *word*.

`<&digit` Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using `>&digit`.

`<&-` The standard input is closed. Similarly for the standard output using `>&-`.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file `xxx`, then associates file descriptor 2 with the same file as descriptor 1, namely `xxx`, while

```
... 2>&1 1>xxx
```

associates file descriptor 2 with the current value of file descriptor 1 (typically the terminal) and file descriptor 1 with `xxx`.

If a command is followed by `&`, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

## Environment

The *environment* is a list of strings, conventionally function definitions and name-value pairs, that is passed to an executed program in the same way as a normal argument list; see [environ\(5\)](#). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter or function for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the `export` command is used to bind the shell's parameter to the environment; see also `set -a`. A parameter may be removed from the environment with the `unset` command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by `unset`, plus any modifications or additions, all of which must be noted in `export` commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters (but not functions). Thus `tabs` gets the same environment in both lines below, but the shell has one less variable in the second.

```
(export TERM; TERM=450; tabs)
TERM=450 tabs
```

If the `-k` flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name.

## Signals

`SIGINT` and `SIGQUIT` (see [signal\(2\)](#)) for an invoked command are ignored if the command is followed by `&`; otherwise signals have the values inherited by the shell from its parent (but see also the `trap` command below).

## Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches the name of a defined function, the function is executed in the shell process. (Note how this differs from calling a shell script.) The positional parameters `$1`, `$2`, ... are set to the arguments of the function. If the command name does not match a function, but matches one of the builtin commands listed below, it is executed in the shell process. If the command name matches neither a builtin command nor the name of a

defined function, a new process is created and an attempt is made to execute the command via *exec(2)*.

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not executable by *exec(2)*, it is assumed to be a ‘shell script’, a file of shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

## Builtin Commands

Input/output redirection is permitted for these commands. File descriptor 1 is the default output location.

- :** No effect; the command does nothing. A zero exit code is returned.
- .file** Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.
- builtin [ command ]**  
Execute the builtin *command* (such as **break**) regardless of functions defined with the same name.
- break [ n ]**  
Exit from the enclosing for or **while** loop, if any. If *n* is specified break *n* levels.
- continue [ n ]**  
Resume the next iteration of the enclosing for or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.
- cd [ arg ]**  
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The current directory (default) is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*.
- eval [ arg ... ]**  
The arguments are read as input to the shell and the resulting command(s) executed.
- exec [ arg ... ]**  
The non-builtin command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- exit [ n ]**  
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)
- export [ name ... ]**  
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed.
- read [ name ... ]**  
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.
- return [ n ]**  
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.
- set [ --aehknptuvx [ arg ... ] ]**
  - a** Mark variables which are modified or created for export.
  - e** Exit immediately if a command exits with a non-zero exit status.
  - f** Disable file name generation
  - k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.

- n** Read commands but do not execute them.
- p** Remove the definitions for all functions imported from the environment, and set **IFS** to blank, tab and newline.
- t** Exit after reading and executing one command.
- u** Treat unset variables as an error when substituting.
- v** Print shell input lines as they are read.
- x** Print commands and their arguments as they are executed.
- Do not change any of the flags; useful in setting **\$1** to **-**.

Using **+** rather than **-** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **\$-**. The remaining arguments are positional parameters and are assigned, in order, to **\$1**, **\$2**, **...**. If no arguments are given the values of all names are printed.

shift [ *n* ]

The positional parameters from **\$n+1** **...** are renamed **\$1** **...**. If *n* is not given, it is assumed to be 1.

times

Print the accumulated user and system times for processes run from the shell.

trap [ *arg* ] [ *n* ] **...**

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is absent all traps *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

umask [ *nnn* ]

The user file-creation mask is set to *nnn*; see [umask\(2\)](#). If *nnn* is omitted, the current value of the mask is printed.

unset [ *name* **...** ]

For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2** and **IFS** cannot be unset.

wait [ *n* ]

Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

whatis [ *name* **...** ]

For each *name*, print the associated value as a parameter, function, builtin or executable file as appropriate. In each case, the value is printed in a form that would yield the same value if typed as input to the shell itself: parameters are printed as assignments, functions as their definitions, builtins as calls to **builtin**, and executable files as completed pathnames.

### Invocation

Normally the shell reads commands from the file named in its first argument (standard input default). The remaining arguments are interpreted as position parameters; see ‘Parameter substitution’ above. If the shell is invoked through [exec\(2\)](#) and the first character of argument zero is **-**, commands are read first from **\$HOME/.profile**, if it exists. Certain options modify this behavior:

- c** *string* Read commands from *string*; ignore remaining arguments.
- s** Write shell output (except for builtin commands) on file descriptor 2.
- i** Interactive. Ignore signal **SIGTERM** (interactive shell is immune to **kill 0**). Catch and ignore **SIGINT** (**wait** is interruptible). The shell always ignores **SIGQUIT**.

Other options are described under the **set** command above.

### FILES

**\$HOME/.profile**  
**/tmp/sh\***  
**/dev/null**

**SEE ALSO**

=(1), [echo\(1\)](#), [newgrp\(1\)](#), [test\(1\)](#), [dup\(2\)](#), [exec\(2\)](#), [fork\(2\)](#), [pipe\(2\)](#), [signal\(2\)](#), [umask\(2\)](#), [exit\(2\)](#), [environ\(5\)](#)

B. W. Kernighan and R. Pike, *The Unix Programming Environment*, Prentice-Hall, 1984

**DIAGNOSTICS**

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed; see also the **exit** command.

**BUGS**

Errors arising from builtins terminate shell scripts.



**NAME**

shstats – show histogram of share scheduler user priorities

**SYNOPSIS**

**shstats** [-#] [-cn] [-sn] [-un]

**DESCRIPTION**

*Shstats* prints a table of system scheduling parameters, the share variables for a selected user (default: the invoker), and a histogram of users vs. normalised usage. In the histogram, the current user selected is represented by the character ‘ ’, and other users by a unique character in the *ascii* range ‘!’ to ‘}’.

Invoked without arguments, *shstats* will print the table and exit. The flags affect operation as follows:-

<i>flag</i>	<i>meaning</i>
<b>-#</b>	All users except the invoker will be represented by the character #.
<b>-cn</b>	Continuous operation, where <b>n</b> , if present, limits the number of cycles.
<b>-sn</b>	Delay time between updates becomes <b>n</b> seconds [default 4].
<b>-un</b>	Change the selected user to the one whose <i>uid</i> matches <i>n</i> .

In continuous non-*hash* mode, *shstats* will read the *standard input* while waiting for the next display update for the invoker to type the identifier for a user to select. This can be either a single character to select the user represented by that character in the histogram, or a login name, or a uid.

**EXAMPLES**

**shstats -#c | dis**

**SEE ALSO**

dis(1), rates(1), ustats(1), share(5).

**BUGS**

Reading from the *standard input* when output is piped through *dis* doesn't seem to work, perhaps because of some weirdness in the way *curses* drives the screen.

**NAME**

sign, verify, enroll, resign – document certification

**SYNOPSIS**

**sign** [ **-n** *name* ] [ *file* ]

**verify** [ **-s** ] [ *file* ]

**enroll**

**resign**

**DESCRIPTION**

These routines provide a document-certification service.

*Sign* reads a document from the *file* or from the standard input, demands a signing password for the current login id, and places on standard output a signed and dated copy of the document, with a cryptographic certificate attached. The resulting document can be embedded in a larger one. The option is

**-n** *name*

Set the signing name; its password will be demanded.

*Verify* scans the *file* or the standard input for a certified document. If the document and date are as they were when certified, except possibly indented, the verified document is placed on the standard output with a statement of verification attached. The option is

**-s** Do not print the document; place only a statement of verification on the standard output.

The signer of a document must be registered with the certification service; the recipient need not be. Two commands handle registration:

*Enroll* demands a signing password and registers it for the current login id. It is unwise to use your login password.

*Resign* demands the signing password and, if it is correct, terminates the registration for the current login id.

A signed document and its date are tamperproof and thus are good for ordinary business purposes. The mere appearance of a certificate, however, is not proof of authenticity. That can be determined only by *verify*. The output of *verify* lacks a certificate; its authenticity cannot be attested at a later date.

There is no notion of an ‘original’ signed document; all copies are equally good and may be reverified at will.

Signers must trust *sign* and recipients must trust *verify* not to have been tampered with on their respective machines. Both parties must trust the verification service, which is on a separate secure machine, and the communication channels to it.

**EXAMPLES**

**sign** <doc.raw >doc.cert

**verify** <doc.suspect >doc.checked

**sign** <letter | mail whomever

The recipient can verify the letter from within *mail*(1) by using *mail*'s pipe command:  
|verify.

**SEE ALSO**

*notary*(8)

**DIAGNOSTICS**

*Verify* yields exit status 0 only on successful verification.

‘Bogus’ – the document has been tampered with, or the original password is no longer registered.

**BUGS**

Only one user with a given login name may be registered; thus the certification service cannot be extended too far.

To minimize dependence on the certification service, no password check is made at signing. A mistyped password will not show up until verification.

**NAME**

size – size of an object file

**SYNOPSIS**

**size** [ *object ...* ]

**DESCRIPTION**

*Size* prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each *object* file argument. If no file is specified, **a.out** is used.

**SEE ALSO**

[a.out\(5\)](#), [nm\(1\)](#)

**NAME**

size80 size of an object file for the 8008/8080 or Z80

**SYNOPSIS**

**size80** [ object ... ]

**DESCRIPTION**

*Size80* prints the decimal and octal number of bytes required by the text, data, and bss portions of each object-file argument. If no file is specified, **80.out** is used.

**BUGS**

**NAME**

sleep – suspend execution for an interval

**SYNOPSIS**

**sleep** *time*

**DESCRIPTION**

*Sleep* suspends execution for *time* seconds.

**EXAMPLES**

```
(sleep  
    Execute a command 100 seconds hence.
```

```
while :  
do
```

```
    command  
    sleep 30
```

```
done
```

Repeat a command every 30 seconds.

**SEE ALSO**

[alarm\(2\)](#), [sleep\(3\)](#)

**NAME**

sml – Standard ML compiler

**SYNOPSIS**

**sml** [ *arg ...* ]

**DESCRIPTION**

*Sml* is the Standard ML of New Jersey compiler. It reads declarations and expressions incrementally from standard input, compiles and evaluates them, and places results on the standard output. Some useful system-related facilities are

**System.argv : unit -> string list**

Return the argument list with which *sml* was invoked.

**System.environ : unit -> string list**

Return the environment list with which *sml* was invoked.

**use : string -> unit**

Temporarily take *sml* source from the file named in the argument.

**exportML : string->bool**

Save the current memory image as the named file, which may later be executed as an argument-less UNIX command. Return **true** in the original and **false** upon resumption of the saved image.

Save a function an executable file and quit ML. The function

takes a UNIX argument list and environment as input; see [exec\(2\)](#).

**system : string -> unit**

Invoke a shell command.

**cd : string -> unit**

Change working directory.

**System.Control.primaryPrompt : string ref****System.Control.secondaryPrompt : string ref**

Primary and secondary prompts analogous to **PS1** and **PS2** of [sh\(1\)](#).

**System.Control.Print.printDepth : int ref**

Limit on depth of printing complex objects; default 5.

**System.Control.Print.stringDepth : int ref**

Limit on length to which strings will be printed; default 70.

**System.Control.Print.signatures : bool ref**

Print signatures only if true.

**EXAMPLES**

```
fun timeit (f: unit->'a) = (* use the system timer *)
  let open System.Timer
    val start = start_timer()
    val result = f()
  in print(makestring(check_timer(start)));
    print "\n";
    result
  end;
```

**SEE ALSO**

Robert Harper, ‘Introduction to Standard ML’, Edinburgh University report ECS-LFSC-86-14 (1986)

Robert Harper, Robin Milner, and Mads Tofte, *The Definition of Standard ML*, MIT Press (1990)

**NAME**

sno – Snobol language interpreter

**SYNOPSIS**

**sno** [*file ...* ]

**DESCRIPTION**

*Sno* is a SNOBOL3 (with slight differences) compiler and interpreter. *Sno* obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspnt**.

*Sno* differs from SNOBOL3 in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b    unanchored search
a *x* b = x c  unanchored assignment
```

There is no back referencing.

```
x
a *x* x    unanchored search for "abc"
```

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and \* must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **syspnt** is not available.

**SEE ALSO**

[spitbol\(1\)](#), [snocone\(1\)](#), [awk\(1\)](#)

*SNOBOL, a String Manipulation Language*, by D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* **11** (1964), pp. 21-30.

**NAME**

snocone – snobol with syntactic sugar

**SYNOPSIS**

**snocone** *file ...*

**DESCRIPTION**

*Snocone* is a programming language, syntactically similar to C, that compiles into SNOBOL4. The *Snocone* compiler translates the concatenation of all the input files into a SNOBOL4 program, which it writes in `When a.out` is executed, the SNOBOL4 interpreter will automatically be invoked. A synopsis of *Snocone* syntax follows.

**Lexical conventions**

Everything after the first unquoted # on an input line is ignored.

Statements normally end at the end of the line. If the last character on a line is an operator, open parenthesis or bracket, or comma, the statement is continued on the next line.

**Binary operators**, grouped by decreasing precedence

- [] Array and table indexing (denoted in SNOBOL4 by <>).
- \$. conditional and immediate pattern value assignment, as in SNOBOL4
- ^ power; right-associative as in SNOBOL4
- \* / % multiplication, division, remainder; unlike SNOBOL4, all have the same precedence.
- + - addition, subtraction
- < > <= >= == != <: >: <=: >=: :>=: :>=: :>=:
- comparison operators; the ones surrounded by colons compare strings, the others compare numbers. These operators behave as SNOBOL4 predicates: they return the null string if the condition is true, and fail if it is false.
- && concatenation; evaluates its right operand only after its left operand has been successfully evaluated. It therefore acts as logical *and* when applied to predicates. The null string may be concatenated to any value.
- || the value of the left operand if possible, otherwise the value of the right operand.
- | pattern value alternation.
- ? pattern match. Returns the part of the left operand matched by the right operand, which must be a pattern. May be used on the left of an assignment if the left operand is appropriate. Right-associative.
- = assignment

**Unary operators**

- + The numeric equivalent of its argument.
- The numeric equivalent of its argument, with the sign reversed.
- \* Unevaluated expression, as in SNOBOL4.
- \$ If *v* is a value of type **name**, then  $\$v$  is the variable of that name.
- @ Pattern matching cursor assignment.
- Logical negation: returns the null string if its argument fails, and fails otherwise.
- ? Returns the null string if its argument succeeds, and fails otherwise.
- . Returns a value of type **name** that refers to its (lvalue) argument.

**Statements**

Statements may be prefixed by one or more labels. A label is an identifier followed by a colon, as in C. All labels are global: it is a good idea to prefix labels in procedures with the name of the procedure.

*expression*

The given *expression* is evaluated for its side effects.

{ *statement ...* }

The *statements* are executed sequentially.

**if** (*expression*) *statement* [ **else** *statement* ]

If evaluation of the *expression* succeeds, the first *statement* is executed. Otherwise, the second *statement*, if any, is executed. An *else* belongs to the closest unmatched **if**.



**while** (*expression*) *statement*

The *statement* is executed repeatedly, as long as the *expression* can be successfully evaluated.

**do** *statement* **while** (*expression*)

Like the **while** statement, except that the *statement* is executed once before the first time the *expression* is evaluated.

**for** (*e1*, *e2*, *e3*) *statement*

As in C, except that commas are used instead of semicolons.

**return** [*expression*]

returns the value of the *expression* from the current function. If *expression* fails or is missing, the value returned is that of the variable with the same name as the function. If that variable was never set, the function returns the null string.

**nreturn** [*expression*]

The *expression* must be the name of a variable. That variable is returned from the current function as an lvalue. If the *expression* fails or is missing, the variable with the same name as the function must have been set to the name of a variable.

**freturn**

The current function returns failure.

**goto** *label*

Transfer control to the given *label*.

Procedures may not be textually nested, but may be recursive and may call each other in forward references. The general form of a procedure declaration is:

```
procedure name (args) locals { statement ... }
```

The *args* and *locals* are lists of variable names, separated by commas. Since Snocone is a dynamically typed language, further declarations are not necessary. Although procedures are not textually nested, names are dynamically scoped: a procedure can reference the local variables and parameters of its caller as if they were global variables.

Assigning a (string) value to the variable `output` causes that value to be written as a single line on the standard output. Accessing the variable `input` causes a line to be read from the standard input. The access fails at end of file. Accessing or assigning to the variable `terminal` causes a line to be read from or written to the standard error file. Other input-output is as implemented by the Macrospitbol interpreter; see [langs\(1\)](#).

**SEE ALSO**

A. R. Koenig, 'The Snocone Programming Language', this manual, Volume 2  
[langs\(1\)](#)

**BUGS**

Run-time diagnostics refer to SNOBOL4 source statement numbers, not to Snocone line numbers. Extremely long statements can overflow the SNOBOL4 compiler's limits on input line length.

**NAME**

*soelim* – eliminate .so's from nroff input

**SYNOPSIS**

**soelim** [*file ...* ]

**DESCRIPTION**

*Soelim* reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

```
.so somefile
```

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

Note that inclusion can be suppressed by using `` instead of `.', i.e.

```
`so /usr/lib/tmac.s
```

A sample usage of *soelim* would be

```
soelim exum?.n | tbl | nroff -ms | col | lpr
```

**SEE ALSO**

*colcrt(1)*, *more(1)*

**AUTHOR**

William Joy

**BUGS**

The format of the source commands must involve no strangeness – exactly one blank must precede and no blanks follow the file name.

**NAME**

sort – sort and/or merge files

**SYNOPSIS**

**sort** [ **-cmusMbdfinrtx** ] [ **-o** *output* ] [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Sort* sorts lines of all the *files* together and writes the result on the standard output. The name **-** means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

- b** Ignore leading white space (spaces and tabs) in field comparisons.
- d** ‘Phone directory’ order: only letters, digits and white space are significant in string comparisons.
- f** Fold lower case letters onto upper case.
- i** Ignore characters outside the ASCII range 040-0176 in string comparisons.
- n** An initial numeric string, consisting of optional white space, optional sign, and a nonempty string of digits with optional decimal point, is sorted by value.
- g** Numeric, like **-n**, with **e**-style exponents allowed.
- M** Compare as month names. The first three characters after optional white space are folded to lower case and compared. Invalid fields compare low to jan.
- r** Reverse the sense of comparisons.
- tx** ‘Tab character’ separating fields is *x*.
- k** *pos1, pos2*  
Restrict the sort key to a string beginning at *pos1* and ending at *pos2*. *pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **Mbdfginr**; *m* counts fields from the beginning of the line and *n* counts characters from the beginning of the field. If any flags are present they override all the global ordering options for this key. If *.n* is missing from *pos1*, it is taken to be 1; if missing from *pos2*, it is taken to be the end of the field. If *pos2* is missing, it is taken to be end of line.

Under option **-tx** fields are strings separated by *x*; otherwise fields are non-empty strings separated by white space. White space before a field is part of the field, except under option **-b**. A **b** flag may be attached independently to *pos1* and *pos2*.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Except under option **-s**, lines with all keys equal are ordered with all bytes significant.

Single-letter options may be combined into a single string, such as **-cnrt**. The option combination **-di** and the combination of **-n** with any of **-diM** are improper. Posix argument conventions are supported.

These option arguments are also understood:

- c** Check that the single input file is sorted according to the ordering rules; give no output unless the file is out of sort.
- m** Merge; the input files are already sorted.
- u** Unique. Keep only the first of two lines that compare equal on all keys. Implies **-s**.
- s** Stable sort. When all keys compare equal, preserve input order. Unaffected by **-r**.
- o** *output*  
Place output in a designated file instead of on the standard output. This file may be the same as one of the inputs. The option may appear among the *file* arguments, except after **--**.
- T** *tempdir*  
Put temporary files in *tempdir* rather than in (the default) **/usr/tmp**.

**-ymemory**

Suggests using the specified number of bytes of internal store to tune performance; an unspecified *memory* size is taken to be huge.

**+pos1 -pos2**

Classical alternative to **-k**, with counting from 0 instead of 1, and *pos2* designating next-after-last instead of last character of the key. A missing character count in *pos2* means 0, which in turn excludes any **-t** tab character from the end of the key. Thus **+1 -1.3** means the same as **-k 2,2.3** and **+1r -3** means the same as **-k 2r,3**.

**EXAMPLES**

- `sort` Print in alphabetical order all the unique spellings in a list of words where capitalized words differ from uncapitalized.
- `sort` Print the password file (*passwd(5)*) sorted by *userid* (the third colon-separated field).
- `sort` Print the first instance of each month in an already sorted file.

**FILES**

`/usr/tmp/stm???`

**SEE ALSO**

[comm\(1\)](#), [join\(1\)](#), [uniq\(1\)](#), [look\(1\)](#)

**DIAGNOSTICS**

*Sort* comments and exits with non-zero status for various trouble conditions and for disorder discovered under option **-c**.

**BUGS**

The never-documented default *pos1*=0 for cases such as **sort -1** has been abolished.

Trouble (e.g. crash or file-system overflow) encountered while overwriting an input with **-o** is irrecoverable.

**NAME**

spell – find spelling errors

**SYNOPSIS**

**spell** [ *option* ] (.,.)[ *file* ] (.,.)SH DESCRIPTION *Spell* looks up words from the named *files* (standard input default) in a public spelling list and in a private list. Possible misspellings (words that occur in neither and are not plausibly derivable from the former) are placed on the standard output.

*Spell* ignores constructs of *troff*(1) and its standard preprocessors, or constructs of *tex*(1). It understands these options:

- b** Check British spelling.
- v** Print all words not literally in the spelling list, with derivations.
- x** Print on standard error, marked with =, every stem as it is looked up in the spelling list, along with its affix classes. Typically used for maintenance.
- c**
- C** Input is one word per line. Output is a single byte per word, delivered immediately: – if the word is rejected, + if the word is accepted under **-c**, and a digit if the word is accepted under **-C**. Digit zero indicates a word known directly; larger numbers indicate words derived by increasingly elaborate paths. Typically used by other programs piping queries to spell.

The private list, by default is arranged one word per line.

Pertinent files may be specified by environment variables, listed below with their default settings. To help in gathering local vocabularies, copies of all output are accumulated in the history file, if it exists and is writable.

As a matter of policy, *spell* does not admit multiple spellings of the same word. Variants that follow general rules are preferred over those that don't, even when the unruly spelling is more common. Thus, in American usage, 'modeled', 'sizable', and 'judgement' are preferred to 'modelled', 'sizeable', and 'judgment'. Agglutinated variants are shunned: 'crew member' and 'back yard' (noun) or 'back-yard' (adjective) are preferred to 'crewmember' and 'backyard'.

**FILES**

/usr/lib/spell/amspell  
spelling list, compressed (**D\_SPELL**)

/usr/lib/spell/brspell  
British spelling list

/usr/lib/spell/spellhist  
history file (**H\_SPELL**)

\$HOME/lib/spelldict  
private list (**A\_SPELL**)

/usr/lib/spell/sprog  
the main routine (**P\_SPELL**)

**deroff** (or **delatex**)

(or for removing punctuation and *troff*(1) constructs (**DEROFF**))

**SEE ALSO**

*dict*(7), *deroff*(1), *wwb*(1)

**BUGS**

Words in a private list are recognized only by exact match, including capitalization and affixing.

The heuristics of *deroff*(1) and *delatex*, used to excise formatting information, are imperfect.

The spelling list's coverage is uneven; in particular biology, medicine, and chemistry, and performe proper names, are covered very lightly.

British spelling was done by an American.

**NAME**

spelltell – find the correct spelling of a word

**SYNOPSIS**

**spelltell** [ **-flags** ] [ **-ver** ] [ wordpart | *grep*(1)-regular-expression ... ]

**DESCRIPTION**

The purpose of *spelltell* is to find the correct spelling of a word. The input to the program can be either:

1. any contiguous letters in the word that are known to be correct; or
2. a *grep*(1) regular expression representing a correctly spelled part of the word.

*Spelltell* will print all words in its dictionary of commonly misspelled words that contain the input sequence.

When *spelltell* is typed on a line with the input word(s), it will print the correct spelling(s) and then exit. When *spelltell* is typed on a line by itself, the program questions whether the user wants instructions, and prompts with ">" for word parts. To quit, type "q" after the prompt.

Two options give information about the program:

- flags** print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer's Workbench version number of the command, then exit.

**EXAMPLES**

1. The command:

**spelltell rece**

will print:

```
precede
preceding
recede
receiving
etc.
```

then exit.

2. The command:

**spelltell  
^proce**

will print "proceed" and other words that begin with "proce," and will then prompt the user for another letter sequence or regular expression.

**SEE ALSO**

*grep*(1), *spellwwb*(1), *proofr*(1), *wwb*(1).

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

spellwwb – find spelling errors  
 spelladd – add words to user spelling dictionary

**SYNOPSIS**

**spellwwb** [ **-flags** ] [ **-ver** ] [ **-f** *pfile* ] [ **-b** ] [ **-v** ] [ **-x** ] [file ...]  
**spelladd** [ **-flags** ] [ **-ver** ] word1 word2 ...

**DESCRIPTION**

*Spellwwb* is a modified version of *spell(1)*. *Spellwwb* allows the user to have his/her own file of additional legitimate spellings, by default *\$HOME/lib/spelldict*. Before reporting words not found in *spell*'s dictionary, *spellwwb* compares them with words in the specified file (*\$HOME/lib/spelldict* by default).

When a text is run through *spellwwb* for the first time, words such as names, acronyms, and unusual words will be listed as errors. Words like these that are spelled correctly can then be added to *\$HOME/lib/spelldict*, and future *spellwwb* lists will contain only real spelling errors.

The following options are available:

- f** *pfile* Use *pfile* instead of *\$HOME/lib/spelldict* as the additional file of legitimate spellings.
- b** Check British spelling.
- v** Print all words not literally in the spelling list, and show plausible derivations from the words in the spelling list.
- x** For each word, print every plausible stem with =.

Two options, which apply to both *spellwwb* and *spelladd*, give information about the programs:

- flags** print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer's Workbench version number of the command, then exit.

*Spelladd* adds words, specified by the user, to *\$HOME/lib/spelldict*, and maintains it in sorted order. It can be used in two ways.

1. Type:  
**spelladd** word1 word2 word3 ...
2. NOTE: Use this method when there are many words from one file to be added to *spelldict*.

First, correct all real spelling errors found by *spellwwb*.

Then, type the following command to have all the remaining words listed by *spell* (correct words) added to *\$HOME/lib/spelldict*:

**spell** *corrected-file* >> *\$HOME/lib/spelldict*; **spelladd**

*Spellwwb* is one of the programs run under the *proofr(1)* and *wwb(1)* commands.

**FILES**

*\$HOME/lib/spelldict* produced by *spelladd*  
*/tmp/\$\$\** temporary files used by *spellwwb* and *spelladd*

**SEE ALSO**

*proofr(1)*, *spell(1)*, *wwb(1)*, *deroff(1)*, *spelltell(1)*.

**SUPPORT**

**COMPONENT NAME:** Writer's Workbench  
**APPROVAL AUTHORITY:** Div 452  
**STATUS:** Standard  
**SUPPLIER:** Dept 45271  
**USER INTERFACE:** Stacey Keenan, Dept 45271, PY x3733  
**SUPPORT LEVEL:** Class B - unqualified support other than Div 452

**NAME**

`spin` – protocol analysis software

**SYNOPSIS**

`spin [ -nN ] [ -pglprsm ] [ -at ] [ file ]`

**DESCRIPTION**

*Spin* is a tool for analyzing the logical consistency of concurrent systems, specifically communication protocols. The system is specified in a guarded command language called Promela. The language, described in the reference, allows for the dynamic creation of processes, nondeterministic case selection, loops, gotos, variables and assertions. The tool has fast and frugal algorithms for analyzing liveness and safeness conditions.

Given a model system specified in Promela, *spin* can either perform random simulations of the system's execution or it can generate a C program that performs a fast exhaustive validation of the system state space. The validator can check, for instance, if user specified system invariants may be violated during a protocol's execution, or if any non-progress execution cycles exist.

Without any options the program performs a random simulation. With option

**-nN** the seed for the simulation is set explicitly to the integer value **N**.

The second group of options **-pglrs** is used to set the desired level of information that the user wants about the simulation run. Every line of output normally contains a reference to the source line in the specification that caused it.

**p** Show at each time step which process changed state.

**l** In combination with option **p**, show the current value of local variables of the process.

**g** Show at each time step the current value of global variables.

**r** Show all message-receive events, giving the name and number of the receiving process and the corresponding the source line number. For each message parameter, show the message type and the message channel number and name.

**s** Show all message-send events.

**m** Changes the semantics of send events. Ordinarily, a send action will be delayed if the target message buffer is full. With this option a message sent to a full buffer is lost. The option can be combined with **-a** (see below).

**a** Generate a protocol-specific analyzer. The output is written into a set of C files, named **pan.[cbhmt]**, that can be compiled (**cc pan.c**) to produce an executable analyzer. Large systems, that require more memory than available on the target machine, can still be analyzed by compiling the analyzer with a bit state space:

**cc -DBITSTATE pan.c**

This collapses the state space to 1 bit per system state, with minimal side-effects.

A compiled analyzer has its own set of options, which can be seen by typing **a.out -?**.

**t** If the analyzer finds a violation of an assertion, a deadlock, a non-progress loop, or an unspecified reception, it writes an error trail into a file named **pan.trail**. The trail can be inspected in detail by invoking *spin* with the **t** option. In combination with the options **pglrs** different views of the error sequence are then easily obtained.

**SEE ALSO**

*cospan* in [langs\(1\)](#)

G.J. Holzmann, 'Spin — A Protocol Analyzer', this manual, Volume 2.



**NAME**

spitbol – Snobol language compiler

**SYNOPSIS**

**spitbol** [ *options* ] *ifile* ...

**DESCRIPTION**

*Spitbol* is an upward compatible dialect of SNOBOL4.

All names used in a program are normally mapped to UPPER CASE during compilation and execution. For strict compatibility with SNOBOL4, use the `-f` option or `-CASE` control statement.

Each *ifile* is read in order before the standard input. Standard output comes only from assignments to OUTPUT and from error messages.

Compiler options:

- `-f`        don't fold lower case names to UPPER CASE
- `-e`        don't send error messages to the terminal
- `-l`        generate source listing
- `-c`        generate compilation statistics
- `-x`        generate execution statistics
- `-a`        like `-lcx`
- `-p`        long listing format; generates form feeds
- `-z`        use standard listing format
- `-h`        write *spitbol* header to standard output
- `-n`        suppress execution
- `-mdd`     max size (words) of created object (default 8192)
- `-sdd`     maximum size (words) of stack space (default 2048)
- `-idd`     size (words) of increment by which dynamic area is increased (default 4096)
- `-ddd`     size (words) of maximum allocated dynamic area (default 256K)
- `-u string`  
          executing program may retrieve string with HOST(0)
- `-o ofile`  
          write listing, statistics and dump to *ofile* and OUTPUT to standard output

Note: *dd* can be followed by a **k** to indicate units of 1024.

*Spitbol* has two input-output modes, *line mode*, where records are delimited by new-line characters, and *raw mode* where a predetermined number of bytes is transferred. Modes are specified in INPUT or OUTPUT function calls. The maximum length of an input record is set by the `-l` or `-r` argument. The form of the INPUT/OUTPUT function call is

INPUT/OUTPUT(.name,channel,file\_name args)

where *name* is the variable name to be input/output associated and *channel* is an integer or string that identifies the association to be used in subsequent calls for EJECT, ENDFILE, INPUT, OUTPUT, REWIND, and SET. If the *channel* is omitted or the null string, the association is made to the system's standard input or output stream. *file\_name args* specifies the source/destination of the input/output and any IO processing arguments. The *file\_name* can be either a path name to a file or a command string. Command strings are distinguished from file names by a leading "!". The character following the "!" is the delimiter used to separate the command string from any IO processing arguments. The ending delimiter may be omitted if there are no IO processing arguments. There must always be at least one space between the *file\_name* and *args*, even if the *file\_name* is null.

Input/output arguments are:

- a** Append output to existing file. If file doesn't exist then it is created. If **-a** is not specified then file is created.
- bdd** Set internal buffer size to *dd* characters. This value is the byte count used on all input/output transfers except for the last write to an output file (default 4096).
- c** Like **-r1**
- fdd** Use *dd* as file descriptor for IO. *spitbol* assumes that *dd* has been opened by the shell. File names and **-fdd** arguments are mutually exclusive. File descriptors 0, 1, and 2 may be accessed in this manner.
- ldd** Line mode: maximum input record length is *dd* characters (default 4096).
- rdd** Raw mode: input record length is *dd* characters.
- w** On output, each record is directly written to the file without any intermediate buffering (default for terminals). On input, each input operation uses exactly one *read(2)*, and fails if *read* returns 0.

More than one type of transfer may be associated with a channel. This is accomplished by calling INPUT/OUTPUT after the initial call with the name, channel, and file arguments. The file name or **-f** argument must not be specified on calls subsequent to the first.

Standard functions: SET(*channel,integer,integer*) The arguments are same as those to the *lseek(2)*, except that the first argument identifies a spitbol channel instead of a file descriptor.

EXIT(*command-string*)

causes the value of *command-string* to be handed to the Shell to be executed after *spitbol* terminates.

EXIT(*n*)

If *n* is greater than 0, a load module will be written in *a.out* before termination. Executing this load module will restore the state of the *spitbol* system to what it was when EXIT was called, except that any files other than the standard input, output, and error will have been closed. To the SNOBOL4 program, it will appear as if EXIT had returned a null string. If *n* is exactly 1, the generated load module will identify the version of *spitbol* that created it in a message when it begins execution. If *n* is greater than 1, it will resume quietly.

HOST()

returns the host string read from /usr/lib/spithost.

HOST(0)

returns the string specified with the **-u** option on the command line. If **-u** was not specified the null string is returned.

HOST(1,"command string")

executes the command string and continues.

HOST(2,*n*)

returns argument number *n* from the command line. It fails if *n* is out of range or not an integer.

HOST(3)

returns the index of the first command line argument that was not examined by *spitbol*.

HOST(4,"var")

returns the value of the environment variable *var*. If the value is too long for an internal buffer (presently 512 bytes) it is quietly truncated.

HOST(5,*n*)

sets (if *n* > 0) or resets (if *n* < 0) a trap for signal number *n* (see *signal(2)*). It returns 0 if no trap was previously set for that signal, 1 if a trap has been previously set but the signal has not occurred since the last call, or 2 if the signal has occurred.

## MISCELLANY

A file is not actually opened until the first attempt to read, write, SET, or REWIND it.

Folding of names to UPPER CASE can be controlled during compilation by the **-CASE** control statement

and during execution by the &CASE keyword. A value of 0 prevents folding to UPPER CASE and a value of 1 forces folding to UPPER CASE.

Integers are represented by 32-bit quantities. Real numbers are implemented in single precision.

Setting &STLIMIT = -1 inhibits statement limit checking and provides a way to execute arbitrarily many statements.

The name TERMINAL is available with default associations for input and output to the terminal.

If the first line of the first input file begins with #! then that line is ignored. This meshes with the way that *exec(2)* treats files beginning with #!.

Setting &PROFILE = 1 causes *spitbol* to accumulate profile information during program execution and print this information after the program terminates.

## FILES

/usr/lib/vaxspitv35.err – Error text.

/usr/lib/spithost – Host computer and operating system identifier.

## SEE ALSO

*Macro SPITBOL Program Reference Manual* by R. B. K. Dewar, A. P. McCann, R. E. Goldberg, and Steven G. Duff

*The SNOBOL Programming Language, Second Edition* by R. E. Griswold, J. F. Poage and I. P. Polonsky  
[sno\(1\)](#), [snocone\(1\)](#)

**NAME**

spline – fit a curve

**SYNOPSIS**

**spline** [ *option ...* ]

**DESCRIPTION**

*Spline* is a filter that interpolates extra points in an input list suitable for *graph(1)*. It is useful for making smooth-looking curves with *graph* or with *grap (1)*.

The following options are recognized, each as a separate argument.

- a** Similar to *graph(1)*. Supply abscissas automatically; no *x*-values appear in the input. Spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0, or the lower limit given by **-x**).
- x** Similar to *graph*. Next 1 (or 2) arguments are lower (and upper) *x* limits. Normally these quantities are determined automatically.
- k** The constant *k* used in the boundary value computation  

$$y''_0 = ky''_1, \quad y''_n = ky''_{n-1}, \quad (.)\text{.if t .ig}$$

$$(2\text{nd deriv. at end}) = k*(2\text{nd deriv. next to end})$$

$$(.)\text{.IP is set by the next argument. (Default } k = 0.)$$
- n** Space output points so that approximately *n* intervals occur between the lower and upper *x* limits. (Default *n* = 100.)
- p** Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.

**SEE ALSO**

*graph(1)*, *grap(1)*, *port(3)*

**DIAGNOSTICS**

When data are not strictly monotone in *x*, *spline* simply reproduces its input.

**BUGS**

*Spline* quietly discards points after the first 1000.

*Spline's* curves exhibit the classic ills of piecewise cubics.

**NAME**

split, fsplit – split a file into pieces

**SYNOPSIS**

**split** [ *option ...* ] [ *file* ]

**fsplit** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Split* reads *file* (standard input by default) and writes it in pieces of 1000 lines per output file. The names of the output files are **xaa**, **xab**, and so on to **xzz**. The options are

**-n** Split into *n*-line pieces.

**-e** *expression*

File divisions occur at each line that matches a *grep*-style regular *expression*; see [gre\(1\)](#). Multiple **-e** options may appear. If a subexpression of *expression* is contained in escaped parentheses `(...)`, the output file name is the portion of the line which matches the subexpression.

**-f** *stem*

Use *stem* instead of **x** in output file names.

**-s** *suffix*

Append *suffix* to names identified under **-e**.

**-x** Exclude the matched input line from the output file.

**-i** Ignore case in option **-e**; force output file names (excluding the suffix) to lower case.

*Fsplit* splits a collection of Fortran subprograms in one *file* into separate files. The options are

**-f**

**-e**

**-r** Set the file suffix: procedure `proc` will go into file `proc.f` (default), `proc.e`, or `proc.r` accordingly. Block data subprograms will go into files named `BLOCKDATA1.f`, etc.

**-i** Force output file names to lower case.

**-s** Strip off data beyond column 72 together with any resulting trailing blanks.

**SEE ALSO**

[sed\(1\)](#), [awk\(1\)](#) *grep* in [gre\(1\)](#)

**NAME**

strings – find printable strings in a file

**SYNOPSIS**

**strings** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Strings* looks for and prints ASCII strings in *files*. A string is a sequence of printing characters, tabs, or backspaces terminated by a newline or a null character. In object files, strings are normally looked for only in the text and data segments. The options are:

- t** Look for strings in the text segment of an object file.
- d** Look for strings in the data segment of an object file.
- s** Look for symbol strings in the symbol table of an object file.
- a** Look for strings throughout the file.
- o** Precede each string by its octal offset in the file.

**-number**

Ignore strings less than *number* characters long (excluding newlines). Default length is 4.

*Strings* is useful for identifying random object files and many other things.

**SEE ALSO**

[gre\(1\)](#), [xd\(1\)](#)

**BUGS**

Newlines are quietly inserted in very long strings.

**NAME**

strip – remove symbols and relocation bits

**SYNOPSIS**

**strip** [ **-s** ] [ **-g** ] [ **-v** ] *file* ...

**DESCRIPTION**

*Strip* removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This saves space and hampers debuggers. Option **-s** of *ld*(1) does the same thing. The options are

- s** Squeeze the symbol table by removing duplicate information.
- g** Delete line-number information, thus negating the effect of `cc -g`. Implies **-s**.
- v** Print size information.

**FILES**

shrink? temporary file

**SEE ALSO**

*cc*(1), *ld*(1) ENVIRONMENT VARIABLES.TH STRLIC 1 "30 June 1988"

**NAME**

strlic – Esterel parser

**SYNOPSIS**

**strlic** [ option ] ... [ file ]...

**DESCRIPTION**

*strlic* is the Esterel v3 parser, type-checker and intermediate code generator. It checks the Esterel input programs for syntax and type errors. It produces an *ic* format output. If no input file is specified, standard input is used. Output is produced only for valid input files i.e. files that did not generate any error. Warnings do not forbid code generation. Error messages and warnings are written to the standard error stream. Typical use is:

```
strlic < game1.strl > game.ic
or
strlic game1.strl game2.strl > game.ic
```

The following options are interpreted by *strlic*:

- version** Gives the version name and terminates ignoring all others arguments.
- memstat** Memory state after compiling.
- stat** Prints auxiliary informations onto the standard error stream: parsing, type-checking and coding times, size of the process.
- w** With this option, no warnings are generated.
- W** This options tells *strlic* to generated all possible warnings. Messages printed only if this option is specified are completely harmless

**FILES**

The caller of the command must have read/write permission for the directories containing the working files, and execute permission for the *strlic* file itself.

**DIAGNOSTICS**

The diagnostics produced by *strlic* compiler are intended to be self-explanatory. System errors are preceded by the message "\$\$Internal error".

**IDENTIFICATION**

Author: R. Bernhard  
CMA, Ecole des Mines de Paris,

Sophia-Antipolis, 06600 Valbonne, FRANCE

Revision Number: \$Revision: 1.14 \$ ; Release Date: \$Date: 88/07/11 17:21:41 \$ .

**SEE ALSO**

esterel(1), iclc(1), lcoc(1), ocxx(1).

Esterel v3 Programming Language Manual

Esterel v3 System Manuals.



**NAME**

struct – structure Fortran programs

**SYNOPSIS**

**struct** [ *option ...* ] *file*

**DESCRIPTION**

*Struct* translates the Fortran program specified by *file* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (e.g. `.GT.` into `>`). The output is appropriately indented.

The following options may occur in any order.

- s** Input uses standard Fortran comment and continuation conventions. Normally input is in the form accepted by [f77\(1\)](#)
- i** Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)
- a** Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch
{
    case pred1: code
    case pred2: code
    case pred3: code
    default: code
}
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

- b** Generate goto's instead of multilevel break statements.
- n** Generate goto's instead of multilevel next statements.
- tn** Make the nonzero integer *n* the lowest valued label in the output program (default 10).
- cn** Increment successive labels in the output program by the nonzero integer *n* (default 1).
- en** If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit a small code segment to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop. 'Small' is close to, but not equal to, the number of statements in the code segment. Values of *n* under 10 are suggested.

**FILES**

`/tmp/struct*`  
`/usr/lib/struct/*`

**SEE ALSO**

[f77\(1\)](#), [ratfor\(A\)](#)

B. S. Baker, 'An Algorithm for Structuring Flowgraphs', *JACM* **24** (1977) 376-391

**BUGS**

Struct knows Fortran 66 syntax, but not full Fortran 77.

If an input Fortran program contains identifiers which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.

The labels generated cannot go above 32767.

If you get a goto without a target, try **-e**.

**NAME**

stty – set terminal options

**SYNOPSIS**

stty [ *option ...* ]

**DESCRIPTION**

*Stty* sets certain I/O options for the terminal open on `/dev/tty` (file descriptor 3). With no argument, it reports the current settings of the options. The options are:

<b>even</b>	Allow even parity.
<b>-even</b>	Disallow even parity.
<b>odd</b>	Allow odd parity.
<b>-odd</b>	Disallow odd parity.
<b>raw</b>	Raw mode input: no erase, kill, interrupt, quit, EOT; parity bit passed to processes.
<b>-raw</b>	Turn off raw mode.
<b>8bit</b>	Eight-bit mode: don't strip parity in the device driver.
<b>-8bit</b>	Turn off eight-bit mode.
<b>cooked</b>	Same as <code>-raw</code> .
<b>cbreak</b>	Make each character available to <code>read(2)</code> as received; no erase and kill.
<b>-cbreak</b>	Make characters available to <code>read</code> only when newline is received.
<b>-nl</b>	Allow carriage return for new-line, and output CR-LF for carriage return or new-line.
<b>nl</b>	Accept only new-line to end lines.
<b>echo</b>	Echo back every character typed.
<b>-echo</b>	Turn off echo.
<b>lcase</b>	Map upper case to lower case.
<b>-lcase</b>	Preserve case.
<b>-tabs</b>	Replace tabs by spaces when printing.
<b>tabs</b>	Preserve tabs.
<b>ek</b>	Reset erase and kill characters to traditional backspace <code>@</code> .
<b>erase c</b>	Set erase character to <i>c</i> (initially backspace). In this and other character-setting options a <code>^</code> may precede <i>c</i> to signify control- <i>c</i> .
<b>kill c</b>	Set kill character to <i>c</i> (initially <code>@</code> ).
<b>intr c</b>	Set interrupt character to <i>c</i> (initially DEL).
<b>quit c</b>	Set quit character to <i>c</i> (initially control- <code>\</code> ).
<b>stop c</b>	Set stop character to <i>c</i> (initially control- <code>S</code> ).
<b>start c</b>	Set start character to <i>c</i> (initially control- <code>Q</code> ).
<b>eof c</b>	Set 'end of file' character to <i>c</i> (initially control- <code>D</code> ).
<b>brk c</b>	Set 'line-break' character to <i>c</i> (initially undefined).
<b>cr0 cr1 cr2 cr3</b>	Select style of delay for carriage return; see <code>ioctl(2)</code> .
<b>nl0 nl1 nl2 nl3</b>	Select style of delay for linefeed.
<b>tab0 tab1 tab2 tab3</b>	Select style of delay for tab.
<b>ff0 ff1</b>	Select style of delay for form feed.
<b>bs0 bs1</b>	Select style of delay for backspace.
<b>hup</b>	Hang up on last close.
<b>0</b>	Hang up immediately.
<b>50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb</b>	Set terminal baud rate to the number given, if possible.
<b>old</b>	Arrange to use normal teletype line discipline, <code>tyld(4)</code> .
<b>old!</b>	Force normal teletype line discipline, even if no teletype driver was present.
<b>notty</b>	Remove the top-level teletype driver.

**SEE ALSO**

`tyld(4)`, `ioctl(2)`, `tabs(1)`

**NAME**

`sum`, `treesum` – sum and count blocks in a file

**SYNOPSIS**

`sum` [ **-5ri** ] [ *file ...* ]

`treesum` [ *file ...* ]

**DESCRIPTION**

By default, `sum` calculates and prints a 32-bit checksum, a byte count and the name of each *file*. The checksum is also a function of the input length. If no files are given, the standard input is summed. Other summing algorithms are available. The options are

- i**      Read file names from standard input.
- r**      Sum with the algorithm of System V's **sum -r** and print the length (in 1K blocks) of the input.
- 5**      Sum with System V's default algorithm and print the length (in 512-byte blocks) of the input.

`Sum` is typically used to look for bad spots, to validate a file communicated over some transmission line or as a quick way to determine if two files might be the same.

`Treesum` is similar to **sum -r**, except that if *file* is a directory, then `treesum` recursively descends it, summing all non-directories encountered. If no files are given, `treesum` recursively sums the current directory.

**SEE ALSO**

[wc\(1\)](#)

**NAME**

*syl* – syllable counter

**SYNOPSIS**

**syl** [ **-flags** ] [ **-ver** ] [ **-num** ] [ file ... ]

**DESCRIPTION**

*Syl* counts the number of syllables in each word in the input text. The input text can be a file, or words typed in at the terminal. *Syl* prints each unique word in the file preceded by its syllable count, with the words ordered alphabetically within each syllable category. That is, all one-syllable words are printed first, in alphabetical order, followed by the two-syllable words, and so on.

One option is available:

**-num**      only print words that have at least *num* syllables, where *num* is an integer.

To use *syl* interactively, type **syl** (carriage return), then type in the word or words to be counted on the next line. *Syl* will print the syllable count for each word.

Two options give information about the program:

**-flags**      print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**      print the Writer's Workbench version number of the command, then exit.

**EXAMPLES**

The command:

**syl -5 filename**

will print all the words in *filename* that have five syllables or more.

The sequence:

**syl** (carriage return)

**Who needs a dictionary**

will print the syllable counts for each word in the line. When finished, type "control-d."

**BUGS**

Because there are minor rules and exceptions in English, not covered by the *syl* program, the program is about 98% accurate.

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

tabs – set terminal tabs

**SYNOPSIS**

**tabs** [ *+num* ] [ **-T***term* ]

**DESCRIPTION**

*Tabs* sets the tabs on a variety of terminals. *Term* is a name given in [term\(6\)](#). The name may also be supplied by the environment variable TERM.

The *+num* option offsets the left margin *num* positions.

**SEE ALSO**

[stty\(1\)](#), [term\(6\)](#), [tset\(A\)](#)

**NAME**

tail, readslow, head – print the last part of a file

**SYNOPSIS**

**tail** [ *option ... number*[lbc][rf] ] [ *file* ]

**DESCRIPTION**

*Tail* copies the named file to the standard output beginning at a designated place, normally 10 lines from the end. If no file is named, the standard input is used. The options are

*number*[lbc][rf]

Copying begins at position *+number* measured from the beginning, or *-number* from the end of the input. *Number* is counted in lines, 1K blocks or characters, according to the appended flag *l* (default), *b*, or *c*. Further flags *r* and *f* have the effect of options **-r** and **-l**.

**-r** Print lines from the end of the file in reverse order. Default line count is unbounded.

**-f** Follow. After printing to the end, keep watch and print further data as it appears.

**-c** *number*

**-n** *number*

*Number* may be signed, with sign *-* assumed by default. The effect is the same as **numberc** or **numberl** [sic] respectively.

**EXAMPLES**

**tail file**

Print the last 10 lines of a file.

**tail +0f file**

Print a file, and continue to watch data accumulate as it grows. A similar function is sometimes called *readslow*.

**sed 10q file**

Print the first 10 lines of a file. A similar function is sometimes called *head*.

**SEE ALSO**

[dd\(1\)](#)

**BUGS**

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length, even under option **-r**.

According to custom, option *+number* counts lines from 1, and counts blocks and characters from 0.

**NAME**

talk – talk to another user

**SYNOPSIS**

**talk** *user* [ *ttyname* ]

**DESCRIPTION**

*Talk* is identical to [write\(1\)](#) except that transmission takes place one character at a time, without waiting for the newline at the end of each line. *Talk* copies characters from your terminal to that of another user. When first called, it sends the message

yourname yourttyname talking...

The recipient of the message should talk back at this point. Communication continues until an interrupt (DEL) or EOT (CTRL-d) is sent. At that point *talk* writes ‘EOT’ on the other terminal and exits.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset talking is allowed. Certain commands, in particular *nroff* and [pr\(1\)](#) disallow messages in order to prevent messy output.

If the character ‘!’ is found at the beginning of a line, *talk* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *talk*: when you first talk to another user, wait for her to talk back before starting to send. Each party should end each message with a distinctive signal — **(o)** for ‘over’ is conventional — so that the other may reply. **(oo)** for ‘over and out’ is suggested when conversation is about to be terminated.

**FILES**

/etc/utmp	to find user
/bin/sh	to execute ‘!’

**SEE ALSO**

[write\(1\)](#), [mesg\(1\)](#), [who\(1\)](#), [mail\(1\)](#)

**NAME**

tape, mt – identify and manipulate magnetic tape

**SYNOPSIS**

**tape**

**mt** [ **-t** *tapename* ] *command* [ *count* ]

**DESCRIPTION**

*Tape* experiments with the magnetic tape drive and reports under which device name the tape mounted there can be read, how many files and records there are, and how big the records are. *Mt* applies a *command* to the named tape drive (default *count* times (default 1)). The *commands* are

**eof** write end-of-file mark

**fsf** forward space file

**fsr** forward space record

**bsf** backspace file

**bsr** backspace record

**rewind**

rewind

**offline**

rewind and take off line

**FILES**

/dev/nrmt1

**SEE ALSO**

[mt\(4\)](#), [dd\(1\)](#)



**NAME**

tar – tape archiver

**SYNOPSIS**

**tar** *key* [*file ...* ]

**DESCRIPTION**

*Tar* saves and restores files, normally on magnetic on tape. The *key* is a string that contains at most one function letter plus optional modifiers. Other arguments to the command are names of files or directories to be dumped or restored. A directory name implies all the contained files and subdirectories (recursively).

The function is one of the following letters:

- r**      The named files are written on the end of the tape.
- x**      Extract the named files from the tape. If a file is a directory, the directory is extracted recursively. Owners and modes are restored if possible. If no file argument is given, extract the entire tape. If the tape contains multiple entries for a file, the latest one wins.
- t**      List all occurrences of each *file* on tape, or of all files if there are no *file* arguments.
- u**      Add the named files if they are not on the tape or are newer than the tape version.
- c**      Create a new tape; writing begins at the beginning of the tape instead of after the last file.
- o**      Omit owner and modes of directories, for compatibility with old versions of *tar*.
- p**      Restore files to their original modes, ignoring the present *umask*(2). Setuid and sticky information will be restored when *tar* is executed by the super-user.

The modifiers are:

- 0,...,7**    Select a tape drive. The default is **1**. Incompatible with modifier **f**.
- v**      (verbose) Print the name of each file treated preceded by the function letter. With **t**, give more details about the tape entries.
- w**      Print the action to be taken followed by file name, then wait for user confirmation. If the answer begins with **y**, the action is performed. Any other input means don't do it.
- f**      Use the next argument as the name of the archive instead of the default `/dev/rmt1`. If the name of the file is `-`, *tar* writes to standard output or reads from standard input, whichever is appropriate. *Tar* can be used to move hierarchies thus:
 

```
(cd fromdir; tar cf - .) | (cd todir; tar xf -)
```
- b**      Write output in *n*×512-byte blocks, where *n* is the next argument, default 20, maximum 40. Useful for raw magnetic tape archives (see **f** above); destructive for disk archives.
- l**      Complain if links cannot be resolved. If **l** is not specified, no error messages are printed.
- L**      Write information needed to re-create symbolic links on the tape instead of following the links. Tapes thus written cannot be read on older versions of *tar*.

**FILES**

`/dev/rmt?`

`/tmp/tar*`

**SEE ALSO**

[cpio\(1\)](#), [bundle\(1\)](#), [mt\(4\)](#)

**BUGS**

There is no way to ask for any but the last occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow, and works only with archives on disk files.

File names are limited to 100 characters.

**NAME**

`tbl` – format tables for `nroff` or `troff`

**SYNOPSIS**

`tbl [ file ... ]`

**DESCRIPTION**

`Tbl` is a preprocessor for formatting tables for `nroff` or `troff(1)`. The input files are copied to the standard output, except for segments of the form

```
.TS
options ;
format .
data
.T&
format .
data
...
.TE
```

which describe tables and are replaced by `troff` requests to lay out the tables. If no arguments are given, `tbl` reads the standard input.

The (optional) `options` line is terminated by a semicolon and contains one or more of

```
center
    center the table; default is left-adjust

expand
    make table as wide as current line length

box
doublebox
    enclose the table in a box or double box

allbox
    enclose every item in a box

tab(x)
    use x to separate input items; default is tab

linesize(n)
    set rules in n-point type

delim(xy)
    recognize x and y as eqn(1) delimiters
```

Each line, except the last, of the obligatory `format` describes one row of the table. The last line describes all rows until the next `.T&`, where the format changes, or the end of the table at `.TE`. A format is specified by key letters, one per column, upper or lower case

```
L
    Left justify: the default for columns without format keys.
R
    Right justify.
C
    Center.
N
    Numeric: align at decimal point (inferred for integers) or at \&.
S
    Span: extend previous column across this one.
A
    Alabetic: left-aligned within column, widest item centered, indented relative to L rows.
```

- ^  
Vertical span: continue item from previous row into this row.
- Draw a horizontal rule in this column.
- Draw a double horizontal rule in this column.

Key letters may be followed by modifiers, also either case:

- | Draw vertical rule between columns.
- || Draw a double vertical rule between columns.
- n* Gap between column is *n* ens wide. Default is 3.
- F***font* Use specified *font*. **B** and **I** mean **FB** and **FI**.
- T** Begin vertically-spanned item at top row of range; default is vertical centering (with ^).
- P***n* Use point size *n*.
- V***n* Use *n*-point vertical spacing in text block; signed *n* means relative change.
- W**(*n*) Column width as a *troff* width specification. Parens are optional if *n* is a simple integer.
- E** Equalize the widths of all columns marked **E**.

Each line of *data* becomes one row of the table; tabs separate items. Lines beginning with . are *troff* requests. Certain special data items are recognized:

- \_ Draw a horizontal rule in this column.
- = Draw a double horizontal rule in this column. A data line consisting of a single \_ or = draws the rule across the whole table.
- \\_ Draw a rule only as wide as the contents of the column.
- \R*x* Repeat character *x* across the column.
- ^ Span the previous item in this column down into this row.
- T{ The item is a text block to be separately formatted by *troff* and placed in the table. The block continues to the next line beginning with T}. The remainder of the data line follows at that point.

When it is used in a pipeline with *eqn*, the *tbl* command should be first, to minimize the volume of data passed through pipes.

## EXAMPLES

Let <tab> represent a tab (which should be typed as a genuine tab).

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town<tab>Households
<tab>Number<tab>Size
Bedminster<tab>789<tab>3.26
Bernards Twp.<tab>3087<tab>3.74
Bernardsville<tab>2018<tab>3.30
.TE
```

```
Household Population
Town      Households
          Number  Size
Bedminster      789   3.26
Bernards Twp.   3087   3.74
Bernardsville   2018   3.30
address.fc
```

**SEE ALSO**

[troff\(1\)](#), [eqn\(1\)](#), [doctype\(1\)](#)

M. E. Lesk and L. L. Cherry, 'TBL—a Program to Format Tables', this manual, Volume 2

**NAME**

`tdc` – fill out TDC form

**SYNOPSIS**

`tdc`

`tdc [ -b ]`

`tdc bill-info [ card-info ]`

**DESCRIPTION**

This command will prepare a PostScript job that will produce a standard Telephone Discount Concession form (ATT363) when printed. With no arguments it simply gives usage information on the standard error output. Option **-b** causes PostScript for a blank form to be generated. To have the form filled in, *bill-info* and, optionally, *card-info* should be supplied. Each of these consists of five arguments, with the first set corresponding to the telephone bill and the second set corresponding to AT&T Universal Card calls. The five arguments are as follows, and must appear in the order shown:

*month* Billing month for the bill. This can be a month abbreviation or a month number. There is not room on the form for the full month name.

*day* Billing day for the bill.

*year* Billing year for the bill.

*amount*

Total Inter-LATA Toll Charges, including taxes (note that this is not quite the same as section 1 of the form—it's just the total amount as it appears on the bill).

*exclusions*

Total non-eligible charges as defined in section 2 of the form. The simplest way to compute this is to add the actual charges for all ineligible calls.

By default, the information for filling in the form is gleaned from the file **\$HOME/.tdc**. If this file does not exist, **tdc** will prompt you for the name of an alternate **tdc** profile residing in your **\$HOME** directory. This profile should consist of a number of lines, each giving one piece of information for the form. The format of the lines is an item name followed by the information. The names and the expected information are:

**firstname**

Your first name.

**homephone**

Your home phone number (the one for which you are vouchering the calls). The format is three numbers, giving area code, exchange and line number, with no extra characters like parenthesis or dashes.

**last8cardno**

The last 8 digits of your AT&T Universal Card number.

**lastname**

Your last name.

**middleinitial**

Your middle initial.

**officephone**

Your office phone number; see the description of **homephone** for the format of this entry.

**payment**

How you are paid. This can be **weekly**, **bi-weekly**, or **monthly** (the default).

**program**

The program you are in (see item 6 of the form). The possibilities are **management** (the default), **occupational1**, or **occupational2**.

**ssnumber**

Your Social Security number, with internal dashes, **xxx-xx-xxxx** .

**title** Your title. The default is **MTS**.

Some of these have default values, as indicated. The others will not appear on the form if you don't specify them. In addition to these values there are five others that are location-specific whose default values are built into **tdc** but which may be overridden in **\$HOME/.tdc**. These values are:

**location1****location2****location3**

Three lines of address information for your BL location.

**paycode**

Your Payroll Unit Code Number.

**tax** The tax rate (as a percentage) used on your phone bill. In New Jersey this is 9 (for 9%).

**EXAMPLES**

If you prepare TDC forms for more than one person, you should keep each person's data in separate **.tdc** files. For example, one file might be called **\$HOME/.tdcSmith**, and another might be called **\$HOME/.tdcJones**. Then, if **tdc** cannot find **\$HOME/.tdc**, it will prompt for the profile to be used:

```
Enter profile suffix: Smith
```

A typical **.tdc** file might look like this:

```
firstname Joe
middleinitial Q.
lastname Schlabotnik
ssnumber 123-45-6789
officephone 908 582 0000
homephone 908 999 0000
```

If you are paid weekly and are in the second occupational category of part 6 of the form, you would add these two lines:

```
payment weekly
program occupational2
```

A typical usage of **tdc** might be:

```
tdc Dec 26 1992 134.28 15.44 | lp
```

**FILES**

**\$HOME/.tdc\***                    **tdc data profiles**

**NAME**

tee – pipe fitting

**SYNOPSIS**

**tee** [ **-i** ] [ **-a** ] [ *file ...* ]

**DESCRIPTION**

*Tee* transcribes the standard input to the standard output and makes copies in the *files*. The options are

- i** Ignore interrupts.
- a** Append the output to the *files* rather than rewriting them.

**NAME**

telnet – user interface to the telnet protocol

**SYNOPSIS**

*telnet* [ *host* [ *port* ] ]

**DESCRIPTION**

*Telnet* communicates with another host using the TELNET protocol. If *telnet* is invoked without arguments, it prompts `telnet>`. In this mode it accepts the commands listed below. If it is invoked with arguments, it performs an `open` command (see below) with those arguments.

Once a connection has been opened, *telnet* sends typed text to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* escape character, initially `control-]`.

The following commands are available. Only a unique prefix of the command need be typed.

`open` *host* [ *port* ]

Open a connection to the named host. A missing port number defaults to a TELNET server. The *host* may be a host name or an Internet address specified in the ‘dot notation’.

**close** Close a TELNET session and return to command mode.

**quit** Close any open TELNET session and exit *telnet*.

`escape` [ *escape-char* ]

Set the *telnet* escape character. Control characters may be specified as `^` followed by a single letter.

**status** Show the current status of *telnet*.

**options**

Toggle viewing of TELNET options processing. When viewing is on, all TELNET option negotiations will be displayed. Options sent by *telnet* are displayed as SENT, options received as RCVD.

**crmod**

Toggle carriage return mode. When this mode is on, carriage return characters received from the remote host map into a carriage return and a line feed.

? [ *command* ]

Get help. With no arguments, print a help summary. With a command specified, print information about that command only.

**FILES**

`/usr/inet/lib/*`

**SEE ALSO**

[dcon\(1\)](#), [netstat\(8\)](#)



**NAME**

test, [, newer – condition commands

**SYNOPSIS**

**test** *expr*

[ *expr* ]

**newer** *file1 file2*

**DESCRIPTION**

*Test* evaluates the expression *expr*. If the value is true the exit status is 0; otherwise the exit status is nonzero. If there are no arguments the exit status is nonzero.

The following primitives are used to construct *expr*.

<b>-r</b> <i>file</i>	True if the file exists (is accessible) and is readable.
<b>-w</b> <i>file</i>	True if the file exists and is writable.
<b>-x</b> <i>file</i>	True if the file exists and has execute permission.
<b>-e</b> <i>file</i>	True if the file exists.
<b>-f</b> <i>file</i>	True if the file exists and is a plain file.
<b>-d</b> <i>file</i>	True if the file exists and is a directory.
<b>-c</b> <i>file</i>	True if the file exists and is a character special file.
<b>-b</b> <i>file</i>	True if the file exists and is a block special file.
<b>-L</b> <i>file</i>	True if the file is a symbolic link.
<b>-u</b> <i>file</i>	True if the file exists and has set userid permission.
<b>-g</b> <i>file</i>	True if the file exists and has set groupid permission.
<b>-s</b> <i>file</i>	True if the file exists and has a size greater than zero.
<b>-t</b> <i>fildev</i>	True if the open file whose file descriptor number is <i>fildev</i> (1 by default) is associated with a terminal device.
<b>-S</b>	True if the effective userid is zero.
<i>s1</i> = <i>s2</i>	True if the strings <i>s1</i> and <i>s2</i> are identical.
<i>s1</i> != <i>s2</i>	True if the strings <i>s1</i> and <i>s2</i> are not identical.
<i>s1</i>	True if <i>s1</i> is not the null string. (Deprecated.)
<b>-z</b> <i>s1</i>	True if the length of string <i>s1</i> is zero.
<i>n1</i> <b>-eq</b> <i>n2</i>	True if the integers <i>n1</i> and <i>n2</i> are arithmetically equal. Any of the comparisons <b>-ne</b> , <b>-gt</b> , <b>-ge</b> , <b>-lt</b> , or <b>-le</b> may be used in place of <b>-eq</b> . The (nonstandard) construct <b>-l</b> <i>string</i> , meaning the length of <i>string</i> , may be used in place of an integer.

These primaries may be combined with the following operators:

<b>!</b>	unary negation operator
<b>-o</b>	binary <i>or</i> operator
<b>-a</b>	binary <i>and</i> operator; higher precedence than <b>-o</b>
( <i>expr</i> )	parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

[ is a synonym for *test*, except that [ requires a closing ].

*Newer* returns a zero exit code if *file1* exists and *file2* does not, or if *file1* and *file2* both exist and *file1* was modified at least as recently as *file2*. It returns a non-zero return code otherwise.

**EXAMPLES**

*Test* is a dubious way to check for specific character strings: it uses a process to do what a shell case statement can do. The first example is not only inefficient but wrong, because *test* understands the purported string **"-c"** as an option. Furthermore **\$1** might be empty.

```
if test $1 = "-c" # wrong!
then echo OK
fi
```

A correct way is

```
case "$1" in
-c)  echo OK
esac
```

Test whether abc is in the current directory.

```
test -e abc -o -L abc
```

**SEE ALSO**

*sh(1)*, *find(1)*

**NAME**

`tex`, `dvips`, `dvit`, `dvicat` – text formatting and typesetting

**SYNOPSIS**

`tex` [*first-line* ]

`dvips` [*option ...* ] *dvifile*

`dvit` [*option ...* ] *dvifile*

`dvicat` *dvifile ...*

**DESCRIPTION**

*Tex* formats interspersed text and commands and outputs a `.dvi` (‘device independent’) file.

An argument given on the command line behaves as the first input line. That line should begin with a (possibly truncated) file name or a `\controlsequence`. Thus `tex` processes the file `paper.tex`. The basename of `paper` becomes the *jobname*, and is used in forming output file names. If no file is named, the jobname is `texput`. The default `.tex` extension can be overridden by specifying an extension explicitly.

The output is written on *jobname.dvi*, which can be printed using [lp\(1\)](#). A log of error messages goes into *jobname.log*.

As well as the standard TeX fonts, many *Postscript* fonts can be used (see the contents of The file `testfont.tex` (in the standard macro directory) will print a table of any font.

These environment variables adjust the behavior of *tex*:

**TEXINPUTS**

Search path for `\input` and `\openin` files. It should be colon-separated, and start with dot.  
Default: `./usr/lib/tex/macros`

**TEXFONTS**

Search path for font metric files. Default: `/usr/lib/tex/fonts/tfm`

**TEXFORMATS**

Search path for format files. Default: `/usr/lib/tex/macros`

**TEXPOOL**

Search path for strings. Default: `/usr/lib/tex`

**TEXEDIT**

Template for the switch-to-editor-on-error option, with `%s` for the filename and `%d` for the line number. Default: `/bin/ed`

*Dvips* and *dvit* convert `.dvi` files to Postscript and *troff* output format, respectively, writing the result on standard output. They are invoked by [lp\(1\)](#) and accept a subset of *lp* options that make sense for `.dvi` files. In the `-opagelist` option for only printing selected pages, the numbers refer to TeX page numbers. In addition, there is a `-Tdev` option, where *dev* is one of `laserwriter` (default for *dvips*), `jerq` (default for *dvit*), `gnot`, `fax`, or `lino` (the computer center’s high resolution Postscript service). The `-Tjerq` or `-Tgnot` options should be used for preparing output for [proof\(9\)](#) or [psi\(9\)](#)

When converting a number of short `.dvi` files to Postscript or using *lp* to print them, it is much more efficient first to combine them via *dvicat*. Simply concatenating `.dvi` files would not work, but *dvicat* achieves this effect and sends the result to standard output. Since the output is in binary, it must be directed to a file or piped into *lp*.

The following environment variables affect *dvips*:

**TEXPKS**

Search path for font bitmaps (PK files).

**TEXVFFONTS**

Search path for ‘virtual font’ descriptions.

*Dvips* and *dvit* understand some extended graphics commands that can be output using *tpic specials* in the TeX source. Many of them work by building up a path of *x,y* pairs, and then doing something with the path. The *tpic* coordinate system has its origin at the current *dvi* position when a drawing special is

emitted; all length arguments are in units of milli-inches, and the y-axis goes positive downward.

**\special{pa *x y*}**

Add *x,y* to the current path.

**\special{fp}**

Flush the current path: draw it as a polygonal line and reset the path to be empty.

**\special{da *dlen*}**

Like **fp** but draw dashed line, with dashes *dlen* milli-inches long.

**\special{dt *slen*}**

Like **fp** but draw a dotted line, with dots *slen* apart.

**\special{sp}**

Like **fp** but draw a quadratic spline. The spline goes through the midpoints of the segments of the path, and straight pieces extend it to the endpoints.

**\special{ar *x y xr yr s e*}**

Draw a circular or elliptical arc with center at *x,y* and radii *xr* and *yr*. The arc goes clockwise from angle *s* to angle *e* (angles measured clockwise from the positive x-axis).

**\special{pn *n*}**

Set line width (pen diameter) to *n* milli-inches.

**\special{bk}**

Set shading to black (will fill the next object drawn with black).

**\special{sh}**

Set shading to grey.

**\special{wh}**

Set shading to white.

**\special{psfile=*file options*}**

(Only *dvips*). Include *file*, which should be a Postscript illustration, making its origin be the current dvi position. The default Postscript transformation matrix will be in effect, but it can be modified by the *options*, a list of *key=value* assignments. Allowed keys are: *hoffset*, *voffset*, *hscale*, *vscale*, and *rotate*. If supplied, these values are supplied to Postscript *translate*, *scale*, and *rotate* commands, in that order. Also, keys *hsize* and *vsize* may be supplied, to cause clipping to those sizes.

**\special{include *horg yorg file*}**

(Only *dvi*). Include the *troff(1)* output *file* at the current place on TeX's page. The included file should have only one page. The *horg* and *yorg* distances give the origin of the included file; that point will be superimposed on the current position.

All of the specials leave TeX at the same position on the page that it started in.

## FILES

```
/usr/lib/tex/macros/*
    macros and preloaded format files

/usr/lib/tex/macros/doc/*
    more TeX-related documentation

/usr/lib/tex/fonts/tfm
    font metrics

/usr/lib/tex/fonts/psvf
    PostScript virtual font metrics

/usr/lib/tex/fonts/canonpk
    bitmaps for canon engines (300 dpi)

/usr/lib/tex/fonts/linopk
    bitmaps for Linotron (1270 dpi)
```

`/usr/lib/tex/*`

miscellaneous configuration files and Postscript headers

**SEE ALSO**

[latex\(6\)](#), [pic\(1\)](#), [lp\(1\)](#), [proof\(9\)](#) [psi\(9\)](#) [troff\(1\)](#), [monk\(1\)](#)

Donald E. Knuth, *The TEXbook*, Addison Wesley, 1986

**BUGS**

It should be possible to make TeX output go on standard output.

**NAME**

time – time a command

**SYNOPSIS**

**time** [ **-usrtdmfio** ] [ **-v** ] *command*

**DESCRIPTION**

The given command is executed; after it is complete, *time* reports statistics about the program on its standard error output. The statistics printed are controlled by the option string; the default is **-usr**; **-v** reports everything. The statistics are:

- u** User cpu time consumed by the command
- s** System cpu time attributed to the command
- r** Real (clock) time taken by the command
- t** Average resident text size, in Kb.
- d** Average resident data size, in Kb.
- m** Maximum total resident size, in Kb.
- f** Number of page faults resulting in disk I/O.
- i** Number of disk blocks read.
- o** Number of disk blocks written.

After the statistics, *time* prints a tab and the command, up to the fifth argument. Times are reported in seconds. The numbers are the sum of those for *command* and any children it spawns and waits for.

**SEE ALSO**

[lcomp\(1\)](#), [prof\(1\)](#)

**BUGS**

Elapsed time is accurate to the second, while the CPU times are measured to 1/60 second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

**NAME**

tk – paginator for the Tektronix 4014

**SYNOPSIS**

**tk** [ **-t** ] [ **-N** ] [ **-pL** ] [ *file* ]

**DESCRIPTION**

The output of *tk* is intended for a Tektronix 4014 terminal. *Tk* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page *tk* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command *!command* will send the *command* to the shell.

The command line options are:

- t**     Don't wait between pages; for directing output into a file.
- N**     Divide the screen into *N* columns and wait after the last column.
- pL**    Set page length to *L* lines.

**SEE ALSO**

[pr\(1\)](#)

**NAME**

tr – translate characters

**SYNOPSIS**

**tr** [ **-cds** ] [ *string1* [ *string2* ] ]

**DESCRIPTION**

*Tr* copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **-cds** may be used:

- c** Complement *string1*: replace it with a lexicographically ordered list of all other 8-bit unsigned characters.
- d** Delete from input all characters in *string1*.
- s** Squeeze repeated output characters that occur in *string2* to single characters.

In either string a noninitial sequence *-x*, where *x* is any character (possibly quoted), stands for a range of characters: a possibly empty sequence of codes running from the successor of the previous code up through the code for *x*. The character `\` followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A `\` followed by any other character stands for that character.

**EXAMPLES**

**tr A-Z a-z <mixed>lower**

Replace all upper-case letters by lower-case.

```
tr -cs A-Za-z '
' <file1 >file2
```

Create a list of all the words in `file1` one per line in `file2`, where a word is taken to be a maximal string of alphabetic. *String2* is given as a quoted newline.

**SEE ALSO**

[ed\(1\)](#), [ascii\(6\)](#)



**NAME**

tr2tex – convert a document from troff to latex

**SYNOPSIS**

tr2tex [ -m ] file ...

**DESCRIPTION**

Tr2tex attempts to convert into *latex(6)* form a document contained in *files* written in *troff(1)* with *ms(6)* macros, *eqn(1)* equations, and some simple *tbl(1)* specifications. The result is placed on standard output. There is one option.

**-m** Convert the document from *man(6)* form to *latex(6)*.

The output may need some hand polishing. *Troff* requests that cannot be converted are left as comments. Some of the converted document may be in plain TeX.

**FILES**

/usr/lib/tex/troffms.sty *tex(1)* macros into which some *troff* constructs convert.  
/usr/lib/tex/troffman.sty

**BUGS**

Option **-m** does not work with 10th edition *man* macros.

Commands that are not separated from their argument by a space are not properly parsed (e.g. **.sp3i**).

*Eqn* operators *rpile* and *lpile* are treated as *cpile*, *rcol* and *lcol* as *ccol*.

*Eqn* operators *size*, *gsize*, *fat*, *gfont*, *mark*, and *lineup* are ignored.

The closing member of a naturally bracketing pair of *troff* requests, such as **.nf-fi** or **.ft I-ft R**, is not supplied automatically after each paragraph.

When local motions are converted to `\raise` or `\lower`, an `\hbox` must be supplied manually.

The *eqn* expression `a sub i sub j` is misconverted; use `a sub { i sub j }` instead.

Line spacing cannot be changed within a paragraph.

Number registers cannot be accessed via **.nr**.

Redefinition of some *eqn* operators, such as *over*, *sub*, or *sup*, will cause trouble.

**AUTHOR**

Kamal Al-Yahya, Stanford University

**NAME**

*track* – selective remote file copy

**SYNOPSIS**

**track** [ **-vntfd** ] *file machine*

**track -r**

**DESCRIPTION**

*Track* uses Datakit to copy files from another machine to the local machine. If the version of the named file differs from that existing on the named machine, the remote file is copied. If the named file is a directory, the contents of the directory are considered recursively. Files are copied only if they exist on both machines. Options:

- v** Normally a report is given for each file copied. Giving the option causes more verbose reports, for example about files that exist locally but not remotely. Giving the option twice generates a report about each file considered.
- n** Do no copying; just report what would have been copied.
- t** Copy only if a remote file is newer than the local file.
- f** Interpret the following file as a list of files and directories to be handled.
- d prefix**  
Normally *track* copies from remote files with the same names as the local files. The **-d** option takes the next argument as a prefix for remote names; in constructing the remote name, the argument string that specifies the local file or directory is replaced by the prefix. directory.
- r** This option causes *track* to act as the remote partner; it is invoked in this way on the other machine, and is not intended for use by humans.

*Track* has no special privileges. Files must be readable remotely and writable locally by the invoker. It attempts to set the time of modification of a copied file to that of the remote original; the attempt can succeed only if the invoker of the local file owns it or is the super-user. This feature matters only when random libraries (archives) are being copied, because the loader uses this time to determine whether the symbol table is up-to-date.

**EXAMPLES**

*track*

Copy files from the remote directory `/bin` to the local directory. `/usr/local/bin`

**SEE ALSO**

[push\(1\)](#), [cp\(1\)](#), [newer\(1\)](#)

**NAME**

troff, nroff – text formatting and typesetting

**SYNOPSIS**

**troff** [ *option ...* ] [ *file ...* ]

**nroff** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Troff* formats text in the named *files* for printing on a phototypesetter; *nroff* for typewriter-like devices. Their capabilities are described in the references.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

- olist** Print pages in the comma-separated *list* of numbers and ranges. A range *N-M* means *N* through *M*; initial *-M* means up to *M*; final *N-* means from *N* to the end.
- nN** Number first generated page *N*.
- mname**  
Prepend the macro file `/usr/lib/tmac/tmac.name` to the input *files*.
- raN** Set register *a* (one character name) to *N*.
- i** Read standard input after the input files are exhausted.
- q** Invoke the simultaneous input-output mode of the **rd** request.
- z** produce no output: diagnostics and **.tm** messages only

**Troff only**

- a** Send a printable ASCII approximation of the results to the standard output.
- Tdest** Prepare output for typesetter *dest*:
  - Tpost** Apple LaserWriter and other PostScript printers (default)
  - T202** Mergenthaler Linotron 202
  - Taps** Autologic APS-5
- Fdir** Take font information from directory *dir*.

**Nroff only**

- sN** Halt prior to every *N* pages (default *N*=1) to allow paper loading or changing.
- Tname**  
Prepare output for specified terminal. Known *names* include **37** for the (default) Teletype model 37, **lp** ('line-printer') for any terminal without half-line capability, **450** for the DASI-450 (Diablo Hyterm), and **think** (HP ThinkJet, see [thinkblt\(9\)](#))
- e** Produce equally-spaced words in adjusted lines, using full terminal resolution.
- h** Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

**FILES**

- `/tmp/trtmp*`  
temporary file
- `/usr/lib/tmac/tmac.*`  
standard macro files
- `/usr/lib/term/*`  
terminal driving tables for *nroff*
- `/usr/lib/font/*`  
font width tables for *troff*

**SEE ALSO**

[lp\(1\)](#), [d202\(A\)](#), [proof\(9\)](#) [apsend\(1\)](#), [reader\(9\)](#)  
[eqn\(1\)](#), [tbl\(1\)](#), [prefer\(1\)](#), [pic\(1\)](#), [ideal\(1\)](#), [grap\(1\)](#), [dag\(1\)](#), [cip\(9\)](#) [ped\(9\)](#)

[doctype\(1\)](#), [ms\(6\)](#), [mpm\(6\)](#), [mbits\(6\)](#), [mpictures\(6\)](#), [mcs\(6\)](#), [font\(5\)](#), [monk\(1\)](#), [tex\(1\)](#)

J. F. Ossanna and B. W. Kernighan, 'Nroff/Troff User's Manual', this manual, Volume 2

B. W. Kernighan, 'A TROFF Tutorial', *ibid.*

**NAME**

true, false – provide truth values

**SYNOPSIS**

**true**

**false**

**DESCRIPTION**

*True* does nothing, successfully. *False* does nothing, unsuccessfully.

**SEE ALSO**

[sh\(1\)](#)

**DIAGNOSTICS**

*True* has exit status zero, *false* nonzero.

**BUGS**

For most purposes, *true* is a slow equivalent for the shell builtin command `:`.

**NAME**

tset – set terminal modes

**SYNOPSIS**

**tset** [ *options* ] [ **-m** test:type (..) ] [ *type* ]

**DESCRIPTION**

*Tset* conditionally sets erase and kill characters, tabs, delays, etc. for terminals. It is typically used in startup profiles; see *sh(1)*. In default of a specified terminal *type* (listed in the file the type is taken from the environment variable TERM. Option **-m** determines the type based on source and baud rate:

**-m** [*>baud*]:*type*

No sources are distinguished at present. The test *>* may be replaced by *<*, *=*, or *@* (same as *=*). The test may be preceded by *!* for negation. A *type* may be preceded by *?* to cause *tset* to query whether the guess is right. Tests are performed left-to-right until one is satisfied. A final default *type* prevails when all tests fail. Thus

```
tset -m '>1200:5620' '?hp'
```

assumes the terminal is a 5620 if the line speed exceeds 1200 baud. Otherwise it assumes an hp terminal but asks for confirmation, giving you a chance to name another type.

The **-s** option causes *tset* to place on the standard output shell commands for setting the environment variables TERM and TERMCAP. Use this feature thus:

```
eval `tset -s option ...`
```

On terminals that can backspace but not overstrike and when the erase character is the default erase character on standard systems), the erase character is changed to a Control-H (backspace).

Other options are:

- e** *c* set the erase character to *c*, or backspace if *c* is missing
- k** *c* set the kill character similarly; use control-X if *c* is missing
- I** suppress outputting terminal initialization strings
- Q** suppress printing 'Erase set to' and 'Kill set to' messages
- S** Outputs TERM and TERMCAP in the environment rather than in shell commands

**FILES**

/etc/ttytype  
terminal id to type map database

/etc/termcap  
terminal capability database

**SEE ALSO**

*sh(1)*, *stty(1)*, *environ(5)*, *termcap(5)*

**NAME**

tsort – topological sort

**SYNOPSIS**

**tsort** [*file* ]

**DESCRIPTION**

*Tsort* produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

**SEE ALSO**

[lorder\(1\)](#)

**DIAGNOSTICS**

Odd data: there is an odd number of fields in the input file.

**NAME**

tty, logtty – get terminal name

**SYNOPSIS**

tty [ -s ]

logtty

**DESCRIPTION**

*Tty* prints the pathname associated with the standard input file if it can be found in the `/dev` directory, `nameless pipe/` and a unique string if the file is a pipe, not a `tty` otherwise. Option `-s` suppresses output, returning exit status only.

*Logtty* prints the pathname associated with the terminal on which the current session was logged in. If the login terminal can't be found, `no login tty` is printed.

In a [mux\(9\)](#) window, *tty* reports the name of the window, while *logtty* reports the name of the terminal. But see **BUGS**.

**SEE ALSO**

[who\(1\)](#)

**DIAGNOSTICS**

Exit status is 0 if a real pathname was printed, 1 for not a `tty` or `no login tty`, 2 for a pipe.

**BUGS**

Mounting something atop the login terminal hides it from *logtty*. [Vismon\(9\)](#) does this.

Try `tty </`.



**NAME**

twig – tree-manipulation language

**SYNOPSIS**

**twig** [ **-wxx** ] *file* .**mt**

**DESCRIPTION**

*Twig* converts a tree-specification scheme consisting of pattern-action rules with associated costs into C functions that can be called to manipulate input trees. The C functions first find a minimum-cost covering of an input tree using a dynamic programming algorithm and then execute the actions associated with the patterns used in the covering. The tree-specification scheme may allow several coverings for an input tree, but the dynamic programming algorithm resolves any ambiguities by selecting a cheapest covering.

The input file containing the tree-specification scheme must have the suffix `.mt`. *Twig* produces two output files: which becomes the source file for the tree matcher, and which contains the definitions for the node and label symbols used in the source file.

To build *twig* uses an internal template file called where *xx* is the argument of the optional **-w** flag. If the flag is omitted, then *xx* defaults to `c1`.

**FILES**

`file.mt`  
input file

`walker.c`  
output tree matcher

`symbols.h`  
definitions of node and label symbols

**SEE ALSO**

[yacc\(1\)](#)

S. W. K. Tjiang, *The Twig Reference Manual*, Computing Science Technical Report No. 120, AT&T Bell Laboratories, Murray Hill, N.J.

A. V. Aho, M. Ganapathi, and S. W. K. Tjiang, *Code generation using tree matching and dynamic programming*.

**BUGS**

When tree matching fails, the debugging output is cryptic.

**NAME**

ul – print underlines on screen terminals

**SYNOPSIS**

**ul** [ **-i** ] [ **-t** *terminal* ] [ *file ...* ]

**DESCRIPTION**

*Ul* replaces backspaced, overstruck underscores by control sequences suitable for the terminal given by the environment variable `TERM` or by option **-t**. It reads from the standard input or the named files and writes on the standard output. Option **-i** represents underlining by a separate line of `-` characters.

**SEE ALSO**

[column\(1\)](#)

**NAME**

`uniq` – report repeated lines in a file

**SYNOPSIS**

**uniq** [ **-udc** [ **+num** ] ] [ *file* ]

**DESCRIPTION**

*Uniq* copies the input *file*, or the standard input, to the standard output comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed. Repeated lines must be adjacent in order to be found.

- u** Print unique lines.
- d** Print (one copy of) duplicated lines.
- c** Prefix a repetition count and a tab to each output line. Implies **-u** and **-d**.
- num** The first *num* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
- +num** The first *num* characters are ignored. Fields are skipped before characters.

**EXAMPLES**

**cut -d: -f3 /etc/passwd | sort | uniq -d**

Print duplicated userids from the password file, *passwd(5)*. *Cut* picks out the userid (the third colon-delimited field) and *sort* brings repetitions together.

**SEE ALSO**

[sort\(1\)](#), [comm\(1\)](#)

**BUGS**

Field-selection and comparison should be compatible with [sort\(1\)](#).

**NAME**

uptime – show how long system has been up

**SYNOPSIS**

**uptime**

**DESCRIPTION**

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w* (1) command.

**FILES**

/vmunix system name list

**SEE ALSO**

w(1) x T aps x res 723 1 1 x init x trailer V0 x stop

**NAME**

ustats – show bar graphs of share scheduler user usage rates

**SYNOPSIS**

**ustats** [-Kn] [-cn] [-g] [-sn] [-r] [-L] [-o[N|P|S|O]]

**DESCRIPTION**

*Ustats* prints bar graphs of user usage rates.

Invoked without arguments, *ustats* will print the graphs and exit. The flags affect operation as follows:-

*flag*    *meaning*

- Kn**    Set the *half-life* for decaying the displayed rate to **n** seconds [default 4].
- cn**    Continuous operation, where **n**, if present, limits the number of cycles.
- g**    Don't include scheduling groups.
- sn**    Delay time between updates becomes **n** seconds [default 4].
- r**    Don't include *root*'s usage.
- L**    The bar graphs are normalised between the minimum and maximum usages found. Quite often, one particular user (eg: *idle*) will dominate the usage, so this flag turns on the use of a logarithmic scale for normalisation.
- of**    The normal ordering of users is in kernel *Inode* table order. This allows a continuous display to show the least change as users log in and log out. There are alternate orders specifiable by the value of **f**:
  - N**    sort by user name
  - P**    sort by user priority
  - S**    sort by usage
  - O**    default (*Inode*) ordering.

**EXAMPLES**

**ustats -Lc | dis**

**SEE ALSO**

dis(1), rates(1), shstats(1), share(5).

**NAME**

uucp, uulog, uuname – unix-to-unix remote file copy

**SYNOPSIS**

**uucp** [ *options ...* ] *source ... destination*

**uulog** [ *option ...* ] [ *system* ]

**uuname** [ **-l** ]

**DESCRIPTION**

*Uucp* copies *source* files to the *destination* file or directory. A file name may be an ordinary path name, or may have the form:

*system-name!filename*

where *system-name* is a computer that *uucp* knows about.

Quoted shell metacharacters `?`, `*` and `[ ]` appearing in a remote *filename* will be expanded on the remote system.

Path names may be:

- (1) a full path name;
- (2) a path name preceded by *user!* where *user* is a login name on the specified system and is replaced by that user's login directory;
- (3) a path name preceded by */destination* where `destination` stands for The destination will be treated as a file name unless more than one file is being transferred by this request, the destination is already a directory, or the destination ends with `/`. For example, `/dan/` as the destination will make the directory `/usr/spool/uucppublic/dan` if it does not exist, and put the requested file(s) in that directory.
- (4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination* is a directory, the basename of the *filename* is used. *Uucp* preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod(2)*).

For obvious security reasons, the domain of remotely accessible files may be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. Similarly you will probably not be able to send files to arbitrary path names. By default, the only remotely accessible files are those whose names begin `/usr/spool/uucppublic/` (equivalent to `/`).

The options are

- C** Copy files in their present state to a spool directory for later transfer. **-c** Do not copy to the spool directory; send files in their state at the time of transmission (default).
- d** Make all necessary directories for the file copy (default).
- f** Do not make intermediate directories for the file copy.
- ggrade** Grade is a single letter/number; earlier ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation. The default is **N**.
- j** Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.
- m** Send mail to the requester when the copy is completed.
- nuser** Notify *user* on the remote system that a file was sent.
- r** Don't start the file transfer, just queue the job.
- xdebug-level** Produce debugging output. The *debug\_level* is a number between 0 and 9; higher numbers give more detailed information.

*Uulog* queries a log file of *uucp* or *uuxqt* transactions, optionally limited to a given *system*. Its options are

- f** Print recent transactions and follow further transactions as they occur.
- x** Look in the uuxqt log file for the given system.
- number**  
Print the last *number* transactions.

*Uuname* lists the uucp names of known systems. The **-l** option returns the local system name.

## FILES

`/usr/spool/uucp`  
spool directories

`/usr/spool/uucppublic`  
public directory for receiving and sending

`/usr/lib/uucp/*`  
other data and program files

`/usr/spool/uucp/.Log/uuxqt/system`  
log of uuxqt transactions with *system*

`/usr/spool/uucp/.Log/uucico/system`  
log of uucp transactions with *system*

## SEE ALSO

[uuto\(1\)](#), [mail\(1\)](#), [push\(1\)](#), [rcp\(1\)](#), [uux\(1\)](#), [uustat\(1\)](#), [uucico\(8\)](#)

P. Honeyman, 'UUCP—the Program that Wouldn't Go Away', this manual, Volume 2

## BUGS

For various reasons remote systems may decline to forward files transmitted through them.

All files received by *uucp* will be owned by user 'uucp'.

Option **-m** works only with a single file.

*Uucp* may run under a daemon userid, in which case files to be sent need general read permission.

**NAME**

uuencode, uudecode – encode/decode a binary file for transmission via mail

**SYNOPSIS**

**uuencode** [*file*] *remotedest*

**uudecode** [*file*]

**DESCRIPTION**

These routines are useful for sending binary files by *mail(1)*.

*Uuencode* places on the standard output an encoded version of the named *file* (standard input by default). The encoding, which uses only printing ASCII characters, includes the mode of the file and a name *remotedest* into which it will be decoded.

*Uudecode* reads encoded data from a *file* or from the standard input and recreates the original data with the mode and name given in the file. As the encoded file is ordinary text, the name or mode can be changed by editing.

An encoded file contains noise lines, a header line, data, trailer, and more noise in that order. The header contains `begin`, the octal mode, and the remote name separated by spaces. Each data line contains a count in the range 0-63, encoded as a single byte with value offset by 040 (space), followed by the encoding of that many bytes of source. 24-bit (3-byte) segments of source are coded in 4 6-bit pieces, again represented in offset-040 code. The trailer is a data line with count 0 and then the line end. **SEE ALSO** *uucp(1)*, *mail(1)*

**BUGS**

The interface is meretricious. The remote name should be decided by the recipient, not the sender. The command `uuencode` does not encode `myfile` but rather reads from standard input.



**NAME**

uurec – receive processed news articles via mail

**SYNOPSIS**

uurec

**DESCRIPTION**

*uurec* reads news articles on the standard input sent by *sendnews(1)*, decodes them, and gives them to *inews(1)* for insertion.

**SEE ALSO**

*inews(1)*, *readnews(1)*, *recnews(1)*, *sendnews(1)*, *newscheck(1)*

**NAME**

`uustat` – uucp status inquiry and job control

**SYNOPSIS**

`uustat` [ *option* ]

`uustat` [ *-ssystem* ] [ *-uuser* ]

**DESCRIPTION**

`Uustat` will display the status of, or cancel, previously specified `uucp` commands, or provide general status on `uucp` connections to other systems. The options are

- a** List the status of all pending `uucp` requests for all machines.
- k *jobid*** Kill the `uucp` request whose job identification is *jobid*. The killed `uucp` request must belong to the person issuing the `uustat` command unless one is the super-user.
- m** Report the status of accessibility of all machines.
- p** Report on the status of all processes that are in the lock files.
- q** List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. A parenthesized number next to the number of C or X files gives the age in days of the oldest file for that system. The retry field represents the number of hours until the next possible call. The count field is the number of failure attempts.
- r *jobid*** Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup demon from deleting the job until its modification time reaches the limit imposed by the demon.
- ssys** Report the status of all `uucp` requests for remote system *sys*.
- uuser** Report the status of all `uucp` requests issued by *user*.

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user.

Requests are listed in the form

```

    jobid  date    type    machine stuff
           date    type    machine stuff    ...

```

*Jobid* identifies the request; it is useful for **-k** and **-r**. The remainder of the line describes a transfer queued at *date* for *machine*. *Type* is **S** if a file is to be sent to *machine*, **R** if it is to be received. Ordinary files are followed by the requestor's userid, the length of the file in bytes, and the name of the spooled file; requests for remote execution are followed by the userid and the command. If the request involves more than one file, the remaining files are listed without a *jobid*.

The most common case is a mail request, which has two lines, one for the mail message itself and one for the request to execute `rmail` on the remote system.

**FILES**

`/usr/spool/uucp/*` spool directories

**SEE ALSO**

[uucp\(1\)](#), [uux\(1\)](#)

**NAME**

`uuto`, `uupick` – simplified unix-to-unix remote file copy

**SYNOPSIS**

`uuto` [ *option ...* ] *file ... recipient*

`uupick` [ `-s system` ]

**DESCRIPTION**

`Uuto` sends the named *files* to a *recipient* on another system by means of `uucp(1)`. The recipient is named thus:

`system!user`

The options are

- p** Make a copy of the *files*. This option allows you to delete or modify the files without worrying about whether they have yet been sent.
- m** Notify the sender by mail when the copy is complete. (The recipient is always notified.)

`Uupick` accepts or rejects files received from `uuto`. For each file currently available, it announces

**from system** *sys-name*: **file** *file-name*

followed by `?`. Give one of these answers on the standard input:

`<new-line>`

Go on to next entry.

**d** Delete the entry.

**m dir** Move the entry to directory *dir*. If *dir* is missing, use the current directory.

**a dir** Like **m**, but move all files from the given system.

**p** Print the file.

`<EOT>` (control-d)

**q** Stop.

`!command`

Escape to `sh(1)` command.

anything else

Print a usage summary.

Option `-s` picks files only from the named *system*.

**FILES**

`/usr/spool/uucppublic/receive/recipient/sendingssystem/*` where the files go

**SEE ALSO**

`mail(1)`, `push(1)`, `rcp(1)`, `uucp(1)`

**NAME**

`uux` – unix to unix command execution

**SYNOPSIS**

`uux` [ *options* ] *command-arg* ...

**DESCRIPTION**

*Uux* will execute a command on a specified system with files and standard output on specified files and systems. For security reasons, most machines allow only selected commands to be run, perhaps only receipt of incoming mail.

The *command-args* make up a *sh(1)* command with with command and file arguments possibly written *system!file*. A missing *system* is interpreted as the local system. *Files* may be prefixed by *xxx/* to represent the home directory for login name *xxx* on the specified system.

*Uux* copies all files to the execution system. Files to be returned as outputs must be enclosed in (escaped) parentheses. Files must have general read permission.

The options are

**-aname**

Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)

**-b** Return standard input to the command if the exit status is non-zero.

**-c** Don't copy local file to the spool directory for transfer to the remote machine (default).

**-C** Force the copy of local files to the spool directory for transfer.

**-ggrade**

*Grade* is a single letter/number; earlier ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation. The default is **N**.

**-j** Place the jobid, an ASCII string, on the standard output. The jobid can be used by *uustat(1)* to obtain the status or terminate a job.

**-n** Suppress mail notification about failures.

**-p**

- Take the standard input to *uux* as the standard input to the executed command.

**-r** Don't start the file transfer, just queue the job.

**-sfile** Report status of the transfer in *file*.

**-xdebug**

Produce debugging output on stdout. *Debug* is a number between 0 and 9; higher numbers give more detailed information.

**-z** Notify the user if the command succeeds.

**FILES**

`/usr/spool/uucp`  
spool directory

`/usr/lib/uucp/*`  
other data and programs

**SEE ALSO**

*uucp(1)*, *uucico(8)*, *uustat(1)*

**BUGS**

All the commands in a shell pipeline are executed on the machine of the first command.

Because files are gathered into a common directory, two files for one command cannot have the same basename. This won't work: `uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"`.

**NAME**

*ex*, *edit*, *vi* – text editor

**SYNOPSIS**

**ex** [*option* ... ] *file* ...

**edit** [*option* ... ] *file* ...

**vi** [*option* ... ] *file* ...

**DESCRIPTION**

*Ex* is elaborated from [ed\(1\)](#). *Vi* is the display editing aspect of *ex*; *edit* is a simplified subset of *ex*. The editors work on several *files* simultaneously. *Vi* gets the terminal type from environment variable **TERM**.

The options are:

**-r file** Recover the *file* after a crash in the editor. If no file is specified, list all saved files.

**-l** Lisp indent mode; give special meaning to editor commands `(){}|`.

**-wn** Default window size is *n* lines.

**-x** Edit encrypted files; a key will be prompted for.

**-R** Read only; no files will be changed.

**+cmd** Execute the specified *ex* command upon entering the editor.

The editors begin in ‘command mode’. An ESC character typed in command mode cancels the command. The commands **a A c C i I o O R s S** enter input mode, where arbitrary text may be entered. ESC or interrupt returns to command mode. The commands **: / ? !** take input from the last line of the screen, which is ended by a newline, or canceled by an interrupt.

A number preceding commands **z G |** specifies a line. A number before almost any other command causes it to be repeated.

In the following summary of *vi* commands **^** means the ‘control’ key. Commands so marked work in input mode; others work in command mode.

File manipulation **:w** write back changes **:w file** write *file* **:w! file** overwrite *file* **:q** quit **:q!** quit, discarding changes **:e file** edit *file* **:e!** reedit, discarding changes **:e +** file edit, starting at end **:e + startingat** line *n* **:sh** run shell, then return **!: cmd** run shell command **:n** edit next *file* argument **: ex-cmd** do editor command

Positioning in file **^F** forward screen **^B** backward screen **^D** down half screen **^U** up half screen **G** to specified line (end default) **/ pat** next line matching *pat* **? pat** previous line matching *n* repeat last **\** or **? /pat/±** *nnth* line after or before *pat* ( beginning of sentence ) end of sentence **{** beginning of paragraph **}** end of paragraph **%** find matching `(){}|`

Adjusting the screen **^L** clear and redraw **^R** retype, eliminate **@** lines **z** redraw, current line at window **z-** ... at bottom **z.** ... at center **/pat/z-** *pat* line at bottom **z n** use *n*-line window **^E** scroll down one line **^Y** scroll up one line

Marking and returning **“** move cursor to previous context **”** ... at first nonwhite in *m x* mark position with letter *x* **‘ x** move cursor to mark **’ x** ... at first nonwhite

Line positioning **H** top line of screen **L** bottom line **M** middle line **+** next line, at first nonwhite **-** previous line, at first nonwhite **<newline>** same as **+ j** next line, same column **k** previous line, same column

Character positioning **^** first nonwhite (not CTRL) **0** beginning of line **\$** end of line **l** forward **h** backward **^L** **^H** same as **l h** **<space>** same as **l f x** find *x* forward **F x** find backward **t x** upto *x* forward **T x** back upto *x*; repeat last **f F t**, opposite of **;** **|** to specified column **%** find matching `(){}|`

Words, sentences, paragraphs **w** word forward **b** word backward **e** end of word **)** next sentence **(** previous sentence **}** next paragraph **{** previous paragraph **W** blank-delimited word **B** backward to ... **E** to end of ...

Corrections **^H** erase last character **^W** erase last word **erase** your erase, same as **^H** **kill** your kill, erase this line **\** quotes your erase or kill **ESC** return to command mode **^?** interrupt, return to command mode **^D** backtab over autoindent **^V** quote nonprinting character

Insert and replace **a** append after cursor **A** append at end of line **i** insert before cursor **I** insert before first non-blank **o** open below line **O** open above **r x** replace one character **R text** replace characters

### Operators

Operators are followed by cursor motion and affect all text that would have been moved over: **dw** deletes a word. Double the operator, e.g. **dd**, to affect whole lines. **d** delete **c** change **y** yank lines to buffer **<** left shift **>** right shift **!** filter through command **=** Lisp indent

Miscellaneous operators **C** change rest of line (**c\$**) **D** delete rest of line (**d\$**) **s** substitute characters (**cd**) **S** substitute lines (**cc**) **J** join lines **x** delete characters (**dl**) **X** ... before cursor (**dh**) **Y** yank lines (**yy**)

Yank and put **p** put buffer after cursor **P** put buffer before cursor **" x p** put from buffer **x x y** yank to buffer **x x d** delete to buffer **x**

Undo, redo, retrieve **u** undo last change **U** restore current line **.** repeat last change **" n p** retrieve *n*-th last delete

### Special ex commands

*Ex* understands most *ed* commands, plus the commands indicated with **:** in the *vi* summary, plus the following, shown with their shortest possible spellings: **abbrev** **args** **copy** **map** **mark** **next** **number** **preserve** **put** **read** **recover** **rewind** **set** **shell** **source** **stop** **unabbrev** **undo** **unmap** **version** **visual** **xit** **yank** **z** **window** **< lshift >** **rshift** **^D** *scroll* The **set** command controls various options; **set** shows the list.

## EXAMPLES

←↓↑→

arrow keys move the cursor

**hjkl** same as arrow keys

**i***text*<ESC>

insert *text*

**cw***new*<ESC>

change word to *new*

**3dw** delete 3 words

**ZZ** exit, saving changes

**/***text*<newline>

search for *text*

**^U** **^D** scroll up or down

## FILES

**/usr/lib/ex?.?.recover**  
recover command

**/usr/lib/ex?.?.preserve**  
preserve command

**/etc/termcap**  
describes capabilities of terminals

**\$HOME/.exrc**  
editor startup file

**/tmp/Ex\***  
editor temporary

**/tmp/Rx\***  
named buffer temporary

**/usr/preserve**  
preservation directory

**SEE ALSO**

[ed\(1\)](#), [sam\(9\)](#) [sed\(1\)](#)

W. Joy and M. Horton, 'An Introduction to Display Editing with Vi', in *Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version*, 1980, Vol 2C (Berkeley)

**NAME**

view2d, regrid, vdata – movie of a function  $f(x, y, t)$

**SYNOPSIS**

**view2d** [ *option ...* ] *file*

**regrid** [ *option ...* ] *file*

**vdata** [ *option ...* ] *file*

**DESCRIPTION**

*View2d* displays a sequence of functional surfaces on a Teletype 5620 or frame buffer. On the 5620, the surface is ruled with a square mesh and projected isometrically; on a frame buffer, the surface is viewed from above, with height indicated by color. The options are

- T5** Output on the 5620;
- Tc** Output on frame buffer (default); **-Tc** is assumed when TERM is not 5620.
- Tp** Produce input for *plot(1)* to produce contour plots.
- cn** Use  $n$  colors ( $n=32$  by default) or contours (6 by default).
- ps** Run movie for  $s$  seconds ( $s=5$  by default).

There are two supported ways to generate input for *view2d*. *Vdata* takes ASCII input and allows scattered data; *view2d(3)* can be called from a program that already has data on a grid.

*Regrid* changes the grid sizes in *view2d* binary files, with options:

- b** Add a bar at the bottom of the image to indicate time.
- nNX,NY**  
Output grid is  $NX \times NY$ ;  $,NY$  may be omitted.
- fNF** Output will have  $NF$  frames.
- r** The plane of closest fit to the first frame is subtracted from all the frames.
- m fmin, fmax**  
Clip the data to the range  $fmin, fmax$ .

One option applies to both *view2d* and *regrid*:

- tTS,TE**  
Display frames from  $TS$  (default first frame time) to  $TE$  (may be omitted, default last frame time).

*Vdata* takes input in the following format and converts to input for *view2d*. The first line of each frame has the number of data points in the frame (an integer) and the time (a floating point value). The rest of the lines in the frame consist of blank-separated  $x y z$  triples. The order of points is irrelevant. The options are:

- nNX,NY**  
Output grid is  $NX \times NY$ ;  $,NY$  may be omitted.
- c** Use piecewise constant rather than linear interpolation.
- s** Scatterplot the data on a black background.
- i** The data is already on a square grid, in the order  $x$  and  $y$  ascending with  $x$  varying fastest.

**FILES**

/usr/lib/view2d/\*

**SEE ALSO**

*view2d(3)*, *view2d(5)*, *plot(1)*

/n/bowell/usr/lib/view2d/howto for more options and how to run equipment.



**NAME**

vis – show invisible characters

**SYNOPSIS**

**vis** [ **-t** ] [ **-s** ] [ *file ...* ]

**DESCRIPTION**

*Vis* reads each *file* in sequence and writes it on the standard output. Non-printing characters are translated on output as in the `l` command of [ed\(1\)](#). If no *file* is given *vis* reads from the standard input. The options are

**-t**      Do not translate tabs.

**-s**      Strip invisible characters; same effect as **tr -cd '<tab><newline><space>'**.

**SEE ALSO**

[cat\(1\)](#), [ed\(1\)](#), [xd\(1\)](#)

**NAME**

*visi* – mathematical spreadsheet

**SYNOPSIS**

**visi** [*file* ]

**DESCRIPTION**

*Visi* is a tabular mathematical worksheet for data analysis. If a *file* is specified, commands are read from that file when *visi* first starts.

*Visi* works only on cursor-controlled terminals such as the HP2621, and requires the environment variable TERM (see *environ(7)*) to be set appropriately.

*Visi* prompts for input at the top of the screen with '>>'. Input has one of the forms,

command parameters  
 variable = expression  
 variable = "string"

where a variable is a letter and number sequence, for example: 'A2, B10, BB23.' These variables represent locations on the worksheet; A2 is column A, row 2. If you type, in any order,

A1 = A2 + 5  
 A2 = 10

the values 15 and 10 will appear on the screen. If you later type

A2 = 20

the values will be updated to 25 and 20. *Visi* treats upper and lower case letters as identical.

Expressions are parsed, and standard mathematical precedence is retained. The operators +, -, \*, /, \*\* (or ^) can be used in expressions.

**Commands**

**copy** [*file* ]

Copy the screen image to the *file*. If a file is not specified, *visi* will prompt for one.

**debug** Toggle a flag to give *yacc(1)* debugging output, very unreadable.

**duplicate** *p1 thru p2 at p3*

Duplicate a block of definitions in another portion of the screen. *P1* and *p2* are the upper left corner and the lower right corner of the block to be duplicated. *P3* is the upper left corner of the destination.

**edit** Edit the commands list. If the environment variable 'ED' is set, it is used as the name of the editor. Otherwise, *ed(1)*. is called.

**help** Display a brief synopsis of the commands.

**list** List the current definitions on the terminal.

**quit** Quit the program.

**read** [*file* ]

Read input lines from the *file*. If a file is not specified, *visi* will prompt for one.

**redraw** Redraw the screen in the event the terminal output was corrupted.

**replicate** *p1 at p2 thru p3*

Replicate the single definition at *p1* throughout the block from *p2* in the upper left corner thru *p3* in the lower right.

**scale** [*column* ] *nnn*

Change the scale of the specified *column*, or of the entire tableau if a column is not specified. The scale *nnn* is the number of decimal places that are displayed to the right of the decimal point. Calculations are done in double precision regardless of *scale*.

- shift** *direction* [*nnn* ]  
Shift the current screen in any direction. The screen is only a window on the tableau. To see other portions of the tableau, the screen must be shifted. Valid directions are: **up**, **down**, **left**, **right**. *Nnn* is the number of positions to shift the screen (default 1).
- shell** Invoke */bin/sh* as an inferior process to *visi*.
- ver** Print the current version number of *visi*.
- width** *column nnn*  
Change the width of a column on the display, or of the entire tableau if no column is specified.
- write** [*file* ]  
Write commands to a file. If a file is not specified, *visi* will prompt for one.

**Built-in Functions**

- abs**(*e*) Absolute value of *e*.
- acos**(*e*) Arc cosine of *e*.
- asin**(*e*) Arc sine of *e*.
- atan**(*e*) Arc tangent of *e*.
- atan2**(*e1,e2*)  
Arc tangent of *e1/e2*.
- cos**(*e*) Cosine of *e*.
- exp**(*e*) Exponential function of *e*.
- gamma**(*e*) Log of the gamma function of *e*.
- hypot**(*e1,e2*)  
Square root of the sum of the squares of *e1* and *e2*.
- int**(*e*) The integer part of *e* (truncated toward zero.)
- log**(*e*) Natural log of *e*.
- pi** The constant 3.14159265358979....
- pow**(*e1,e2*)  
Same as  $e1^e2$ .
- sin**(*e*) Sine of *e*.
- sqrt**(*e*) Square root of *e*.

**Other Special Definitions**

- position**[*e1,e2*]  
The quantity at row *e1*, column *e2* of the tableau. Numbering for the columns is A = 1, B = 2, ..., AA = 27, and so on.
- ROW** The row number of this entry.
- COL** The column number of this entry.

**SEE ALSO**

[ed\(1\)](#), [exp\(3\)](#), [sin\(3\)](#)

**FILES**

[/usr/lib/visi.help](#)

**BUGS**

A circular list of variable declarations can cause *visi* to hang in a loop.  
*Scale* does truncation, not rounding.

**NAME**

vsw – video switcher

**SYNOPSIS**

**vsw** [ *option* ] [ *iopair ...* ]

**DESCRIPTION**

*Vsw* controls Dynair (System 10 and Dynasty) video switches. There is a composite video switch and a 3 channel (rgb) switch. Each switch has ten or more inputs and outputs. Inputs may be attached to one or more outputs (fanout). Outputs may not be attached to more than one input (no mixing).

The optional *iopair* arguments are two letters *io* which mean attach input letter *i* to output channel letter *o*. The options are

- c** Use the composite video switch. Exactly one of **-c** and **-r** must be specified.
- d** Describe the input and output channels.
- i** Initialize the switch to a default setting.
- p** Print the current switch configuration suitable for processing with *vsw*.
- r** Use the rgb video switch. Exactly one of **-c** and **-r** must be specified.
- v** Be more verbose.

By convention, video switch access to monitors is for 'INPUT B', which is normally selected by a button on the front. The Barco monitor is an exception; use 'COMP VIDEO 1'.

**EXAMPLES**

**x='vsw -cp'**  
Save the current configuration.

**vsw \$x**  
Restore the configuration.

**SEE ALSO**

[2500\(1\)](#)

**BUGS**

For timing reasons, *vsw* only works from fast or unloaded machines.

The input/output names are a little terse.

To power-cycle the rgb switch, look for the blue ribbon cable that daisy chains the three boxes. The box at one end of the chain has a coax cable connected to 'comm'; start power cycling at the other end of the daisy chain.

The switch may need to be reset after prolonged power outages (in this case, *vsw* will complain) by **cd /n/bowell/usr/src/cmd/vsw; mk init.vsw**

**NAME**

w – who is on and what they are doing

**SYNOPSIS**

w [ -h ] [ -s ] [ user ]

**DESCRIPTION**

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the total CPU time, the percentage of the CPU, the percentage of the total virtual memory, the percentage of the the total virtual memory loaded, and the name and arguments of the current process. The CPU and virtual memory items are based on all processes and their children associated with that terminal.

The -h flag suppresses the heading. The -s flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. -l gives the long output, which is the default.

If a *user* name is included, the output will be restricted to that user.

**FILES**

/etc/utmp  
/dev/kmem  
/dev/drum

**SEE ALSO**

who(1), ps(1)

**AUTHOR**

Mark Horton

**BUGS**

The notion of the “current process” is muddy. The current algorithm is “the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal”. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, w prints “-”.)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is “charged” with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

**NAME**

`wc` – word count

**SYNOPSIS**

`wc` [ **-lwc** ] [ *file ...* ]

**DESCRIPTION**

`Wc` counts lines, words and characters in the named *files*, or in the standard input if no file is named. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters **l**, **w**, or **c**.

**NAME**

who, whois, last – who is or was on the system

**SYNOPSIS**

**who** [ **-i** ] [ *who-file* ]

**who am i**

**whois** *username*

**last** [ **-f who-file** ] [ *userid ...* ] [ *terminal* ]

**DESCRIPTION**

*Who*, without an argument, lists the login name, terminal name, and login time for each current user. With the **-i** option, the report includes the number of minutes that the user's terminal has been idle.

With two arguments, as in `who am i`, *who* tells who you are logged in as.

Without an argument, *who* examines the `/etc/utmp` file to obtain its information. If a file is given, that file is examined. Typically the given file will be which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the `wtmp` file. Each login is listed with user name, terminal name (with `/dev/` suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with `x` in the place of the device name, and a fossil time indicative of when the system went down.

*Whois* consults administrative files to identify the *username*. (Actually, *whois* uses *grep* and can locate information by any useful key, such as real name or telephone number.)

*Last* reports logins and logouts in reverse chronological order. Optional arguments restrict attention to selected userids or terminals. Terminals **tty0**, **tty1**, ... may be abbreviated **0**, **1**, ...

By default, *last* examines the list of logins and logouts in option **-f** specifies a different file.

*Last* reports userid, terminal, time on, and time off for all users, or for selected *userids*. A pseudo-user, `reboot`, is logged in at reboots of the system.

Upon interrupt, *last* tells how far back it has looked; upon quit (control-`\`) it tells how far and keeps on looking.

**EXAMPLES**

**last reboot**

Report recent system outages.

**FILES**

`/etc/utmp`

`/usr/adm/wtmp`

`/usr/adm/usrlist`

`/etc/passwd`

**SEE ALSO**

[getuid\(1\)](#), [getuid\(2\)](#), [utmp\(5\)](#), [ac\(8\)](#), [tty\(1\)](#), *vwhois* in [vismon\(9\)](#)

**NAME**

`worduse` – print information about correct usage of words

**SYNOPSIS**

**worduse** [ **-flags** ] [ **-ver** ] [ word1 | "phrase 1" ... ]

**worduse** (carriage return)

>word1 or phrase1

(program response)

>word2 or phrase2

(program response)

>q

**DESCRIPTION**

*Worduse* searches a list of commonly confused and misused English words and phrases for the user's word or phrase. It describes the correct use of the word or phrase (if any), and compares it to any similar words or phrases.

When *worduse* is typed on a line with the input word(s) or phrase(s), it will print the explanation(s) and then exit. In this mode phrases of more than one word must be enclosed in double quotes.

When *worduse* is typed on a line by itself it acts as an interactive program, first printing brief instructions, then prompting with ">" for more words or phrases. To quit, type "q" after the prompt.

*Worduse* may be used as a supplement to *diction*(1), *dictplus*(1), or *proofr*(1), to learn why words were flagged by *diction*.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**EXAMPLES**

1. The command:

**worduse affect**

will print:

AFFECT: EFFECT: Affect is normally only a verb; effect is most often a noun...

2. The command:

**worduse "one of the"**

will print:

ONE OF THE: THE ONLY ONE OF THE: "One of the" is followed by a plural verb...

and then exit.

3. The command:

**worduse**

will first print instructions, then

>**alright**

will print:

ALL RIGHT: ALRIGHT: The only acceptable spelling is "all right."

and will then prompt the user for another word.

4. The command:

**worduse**

will first print instructions, then



**>one of the**

will print:

ONE OF THE: THE ONLY ONE OF THE: "One of the" is followed by a plural verb...

and will then prompt the user for another word.

## **FILES**

wwb/lib/wordlist            contains the list of words and descriptions

## **SEE ALSO**

diction(1), dictplus(1), proofr(1), wwb(1).

## **SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

write, mesg – write to other users, allow or forbid messages

**SYNOPSIS**

**write** *person* ...

**mesg** [ **n** ] [ **y** ]

**DESCRIPTION**

*Write* copies lines from your terminal to terminals of other *persons* designated either by login name or (to circumvent occasional ambiguities) by terminal name as given by *who(1)*. It announces to each *person* your login name, your terminal, and the other *persons*. To respond, each recipient should execute a corresponding *write* to the *persons* he wants to talk to.

When you are writing to more than one person, your messages are identified to the recipients. Writing ceases upon end of file or interrupt, and the message **EOF** is sent to the others.

*Write* recognizes certain commands during a conversation:

**!cmd** Execute a shell on the string *cmd* and then return to *write*.

**:a** *person*

Add *person* to the list of people to whom you are talking, and send an appropriate announcement to all parties. They must do **:a** for themselves if they want to include the new person.

**:d** *person*

Drop *person* from your list and make appropriate announcements.

**:l** Print a list of people to whom you are talking.

The following protocol is suggested for using *write*. When you invoke *write*, wait for the other user to write back before starting to send. Each party should end each message with a distinctive signal that the other may reply – the customary convention is (o) for ‘over’; (oo) for ‘over and out’ is suggested when conversation is about to be terminated.

*Mesg* with argument **n** forbids messages via *write(1)* by revoking non-user write permission on the user’s terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

Certain commands, in particular *nroff* and *pr(1)* disallow messages in order to prevent messy output.

**FILES**

/etc/utmp  
to find user

/bin/sh  
to execute !

**SEE ALSO**

*who(1)*, *mail(1)*, *vismon(9)*

**DIAGNOSTICS**

*Mesg* yields exit status 0 if messages are receivable, 1 if not, 2 on error.

**BUGS**

*Mux(9)* generally prevents the receipt of *write* messages, but *vismon(9)* permits them.

Messages ought to be identified when the recipient is receiving from more than one writer, rather than when a writer is sending to more than one recipient, but that requires cooperating processes and isn’t worth the effort.

**NAME**

`wwb` – Writer’s Workbench

**SYNOPSIS**

`wwb` [ **-flags** ] [ **-ver** ] [ **-mm** | **-ms** ] [ **-li** | **+li** ] [ **-tm** | **-t** | **-c** | **-x standards-file** ] [ **-s** ] [file ...]

**DESCRIPTION**

The Writer’s Workbench is a set of programs designed to aid writers and editors in editing documents. The command `wwb` runs modified versions of 2 major programs, which in turn run other programs.

The two major programs are:

- `proofr(1)` - automatic proofreading system that searches for spelling and punctuation errors, consecutive occurrences of words, wordy or misused phrases, and split infinitives.
- `prose(1)` - describes the writing style of a document, namely, readability and sentence characteristics, and suggests improvements.

`Prose` compares a document with standards for one of several document types, according to the following flags:

- tm** Compare input text to good Bell Laboratories Technical Memoranda. (this is the default.)
- t** Compare input text with good training documents.
- c** Evaluate text for craft suitability.
- x standards-file**  
Compare input text with standards contained in user-specified *standards-file*. See `mk-stand(1)` to set up the *standards-file*.

Other options are:

- s** Produce short versions of `proofr` and `prose`.

Two options give information about the program:

- flags**  
print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer’s Workbench version number of the command, then exit.

`Wwb` runs `deroff(1)` before looking at the text. Four options affect `deroff`:

- mm** eliminate `mm(1)` macros, and associated text that is not part of sentences (e.g. headings), from the analysis. This is the default.
- ms** eliminate `ms(1)` macros, and associated text that is not part of sentences from the analysis. The **-ms** flag overrides the default, **-mm**.
- li** eliminate list items, as defined by `mm` macros, from the analysis. This is the default.
- +li** Include list items in the analysis. This flag should be used if the text contains lists of sentences, but not if the text contains many lists of non-sentences.

**FILES**

`/tmp/$$*` temporary files

**SEE ALSO**

`prose(1)`, `proofr(1)`, `style(1)`, `spellwwb(1)`, `punct(1)`, `double(1)`, `diction(1)`, `splitinf(1)`, `deroff(1)`, `mk-stand(1)`, `wwbstand(1)`, `spelltell(1)`, `worduse(1)`, `wwbinfo(1)`, `wwbhelp(1)`, `wwbmail(1)`.

**BUGS**

Split infinitives in lists will not be found unless the `+li` option is used. See other manual pages for bugs in individual commands.

**SUPPORT**

**COMPONENT NAME:** Writer’s Workbench

**APPROVAL AUTHORITY:** Div 452

**STATUS:** Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

wwbhelp – print Writer's Workbench commands for a topic

**SYNOPSIS**

**wwbhelp** [ **-flags** ] [ **-ver** ] [ word or part of word ... ]

**DESCRIPTION**

*Wwbhelp* searches a file of descriptive topics covered by Writer's Workbench programs. It prints the topical descriptors followed by commands and options that can be used to get topically related information.

When *wwbhelp* is typed on a line with the input topic(s), it will print the commands and options and then exit. When *wwbhelp* is typed on a line by itself, the program prompts with ">" for a topic. To quit, type "q" after the prompt.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**EXAMPLE**

The command:

**wwbhelp syl**

returns:

syllable counts for words....syl file

syllables greater than N.....syl -N file

**FILES**

/wwb/lib/helpfile            contains the list of topics and programs

**SEE ALSO**

wwbinfo(1), wwb(1).

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

wwbinfo – print table of WWB commands and functions

**SYNOPSIS**

**wwbinfo** [ **-ver** ][ **-flags** ]

**DESCRIPTION**

*Wwbinfo* prints a table listing the Writer's Workbench (WWB) commands and their functions. Some commands run other commands. Subordinate commands are indented.

The table is divided into four parts: general commands, commands that give explanations, environmental tailoring commands, and user-specified dictionaries.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing the command flags and options, then exit.

**-ver**

print the Writer's Workbench version number of the command, then exit.

**SUPPORT**

*COMPONENT NAME:* Writer's Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

*SUPPORT LEVEL:* Class B - unqualified support other than Div 452

**NAME**

wwbmail – send mail to Writer’s Workbench development group

**SYNOPSIS**

**wwbmail** [ **-p** ] [ **-flags** ] [ **-ver** ]

**DESCRIPTION**

*Wwbmail* sends a user’s comments or requests for help directly to the Writer’s Workbench (WWB) development group.

The **-p** option allows the user to bypass the default (inter-system mail) and go directly to the procedure for obtaining a paper copy of the message.

The program prints instructions if the user needs them. The instructions are written at a level a casual user can understand.

If *wwbmail* knows of no electronic mail link between the user’s system and that of the WWB group, it gives instructions on how to print a pre-addressed paper copy of the message to send via regular mail.

Two options give information about the program:

**-flags**

print the command synopsis line (see above) showing command flags and options, then exit.

**-ver**

print the Writer’s Workbench version number of the command, then exit.

**FILES**

/tmp/\$\$\*                    temporary files

wwbmailtmp                output file containing address and message for paper copy

**SEE ALSO**

wwbhelp(1), wwbinfo(1), wwb(1).

**SUPPORT**

*COMPONENT NAME:* Writer’s Workbench

*APPROVAL AUTHORITY:* Div 452

*STATUS:* Standard

*SUPPLIER:* Dept 45271

*USER INTERFACE:* Stacey Keenan, Dept 45271, PY x3733

**NAME**

`wwbstand` – print standards used by `prose(1)`

**SYNOPSIS**

`wwbstand` [ **-flags** ] [ **-ver** ] [ **-tm** | **-t** | **-c** | **-x standards-file** ]

**DESCRIPTION**

*Wwbstand* reads the standards files that are used by *prose(1)*, and prints the average ranges of 12 *style(1)* scores for documents of certain kinds. *Prose* considers scores that fall within these ranges to be good.

Four options to *wwbstand* tell it what standards file to read:

- tm** print average ranges of *style* scores for Bell Labs Technical Memoranda judged good by department heads in the research area. This is the default.
- t** print average ranges of *style* scores for Bell Labs training documents from Department 45272.
- c** print suggested ranges of *style* scores for craft documents based on the research of E. Coke, department 11222.
- x standards-file**  
print average ranges of *style* scores contained in *standards-file*. The *standards-file* can be made using *mkstand(1)*.

Two options give information about the program:

- flags**  
print the command synopsis line (see above) showing command flags and options, then exit.
- ver** print the Writer's Workbench version number of the command, then exit.

**SEE ALSO**

`mkstand(1)`, `prose(1)`, `style(1)`, `wwb(1)`.

**SUPPORT**

**COMPONENT NAME:** Writer's Workbench

**APPROVAL AUTHORITY:** Div 452

**STATUS:** Standard

**SUPPLIER:** Dept 45271

**USER INTERFACE:** Stacey Keenan, Dept 45271, PY x3733

**SUPPORT LEVEL:** Class B - unqualified support other than Div 452



**NAME**

`wwv` – print or set the date from accurate clock

**SYNOPSIS**

`wwv` [ option ... ]

**DESCRIPTION**

`Wwv` connects to a clock synchronized with a time standard. With no argument it prints the time obtained. It accepts these options:

- b** Print both the WWV time and the system's current idea of the time.
- s** Set the system's time to agree with WWV. Only the super-user can do this. The system time cannot be changed by more than 20 minutes.
- f** With **-s**, force the time to be set by WWV even if the local time disagrees by more than 20 minutes.
- u** Report time in GMT rather than local time.

**FILES**

`/usr/adm/wtmp` to record time-setting; see [utmp\(5\)](#)

**SEE ALSO**

[date\(1\)](#), [time\(2\)](#), [stime\(2\)](#)

**NAME**

`xargs` – construct argument lists and execute command

**SYNOPSIS**

**xargs** [ *option* ] [ *command* [ *initial-arguments* ] ]

**DESCRIPTION**

*Xargs* combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. *Command* (by default) is located according to environment variable `PATH`.

Arguments read from standard input are delimited by white space (blanks, tabs, or new-lines). However, single or double quotes may be used to surround arguments that contain blanks or tabs, and backslash `\` may be used to quote single characters outside of quotes.

Normally the *initial-arguments* are followed by arguments read from standard input until an internal buffer is full, whereupon *command* is executed with the accumulated arguments. This process is repeated until there are no more arguments. Options modify this rule:

- ln** *Command* is executed upon reading each *n* (default 1) nonempty lines from standard input. Newlines preceded by blank or tab are not counted. Option **-x** is implied.
- is** Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *s*, `{ }` by default. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not exceed 255 characters. Option **-x** is implied.
- nn** Execute *command* using as many standard input arguments as possible, up to *n* arguments maximum.
- t** Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
- p** Prompt about whether to execute *command*. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by `?...`. The command will be executed if and only if the reply begins with `y`.
- x** Terminate if any argument list would be greater than *size* characters.
- size** The maximum total size of each argument list is *size* characters, 470 by default.
- eofstr**

*Eofstr* ( `_` by default) is the logical end-of-file string. Normally *xargs* reads standard input up to a logical or an actual end-of-file. Option **-e** with no *eofstr* turns off logical end-of-file testing.

*Xargs* will terminate if it receives a return code of **-1** from, or cannot execute, *command*.

**EXAMPLES**

`ls $1 | xargs -i -t mv $1/{ } $2/{ }` Move all files from directory `$1` to directory `$2`, and echo each move command just before doing it.

`(logname; date; echo $0 $*) | xargs >>log` Combine the output of the parenthesized commands onto one line, which is then echoed to the end of file `log`.

`ls | xargs -p -l ar r arch` Ask which files in the current directory are to be archived and archive them one at a time.

`ls | xargs -p -l | xargs ar r arch` Same, but archive many at a time.

`echo $** | xargs -n2 diff` Execute `diff(1)` with successive pairs of arguments originally typed as shell arguments.

**SEE ALSO**

[sh\(1\)](#), [apply\(1\)](#)

**NAME**

*xd*, *od* – hex, octal, decimal, or ASCII dump

**SYNOPSIS**

**xd** [ *option* ... ] [ *-format* ... ] [ *file* ... ]

**od** [ *-bcdox* ] [ *file* ] [ *+offset* ]

**DESCRIPTION**

*Xd* concatenates and dumps the *files* (standard input by default) in one or more formats. Groups of 16 bytes are printed in each of the named formats, one format per line. Each line of output is prefixed by its address (byte offset) in the input file. The first line of output for each group is zero-padded; subsequent are blank-padded.

Formats other than **-c** are specified by pairs of characters telling size and style, 4x by default. The sizes are

**1** or **b** 1-byte units.

**2** or **w** 2-byte units.

**4** or **l** 4-byte units.

The styles are

**o** Octal.  
**x** Hexadecimal.  
**d** Decimal.

Other options are

**-c** Format as **1x** but print ASCII representations or C escape sequences where possible.

**-astyle** Print file addresses in the given style (and size 4).

**-s** Reverse (swab) the order of bytes in each group of 4 before printing.

**-r** Print repeating groups of identical 16-byte sequences as the first group followed by an asterisk.

*Od* dumps a *file* or the standard input in one or more formats as selected by the first argument, octal by default.

The format characters mean

**b** Bytes in octal.  
**c** Bytes in ASCII with C escapes and 3-digit octal for other characters.  
**d** 16-bit words in decimal.  
**o** 16-bit words in octal.  
**x** 16-bit words in hex.

The *offset* argument tells where in the file to begin dumping. The offset, normally interpreted in octal, is interpreted in hexadecimal if it begins with **x** or **0x**, and in decimal if it ends with **.** or **.b**. If it ends in **b**, it is multiplied by 512. The preceding **+** may be omitted if *file* is present.

**SEE ALSO**

[adb\(1\)](#), [strings\(1\)](#), [vis\(1\)](#)

**BUGS**

The various output formats don't line up properly in the output of *xd*.

A spurious zero byte reported by *od* at the end of odd-length files is betrayed by the correctly printed final address.

The *offset* is ineffectual if [lseek\(2\)](#) won't work on the *file*.

On some raw devices offsets must be a multiple of the natural block size.

**NAME**

yacc, eyacc – yet another compiler-compiler

**SYNOPSIS**

**yacc** [ *option ...* ] *grammar*

**eyacc** [ **-v** ] [ *grammar* ]

**DESCRIPTION**

*Yacc* converts a context-free grammar and translation code into a set of tables for an LR(1) parser and translator. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, must be compiled by the C compiler to produce a program `yyparse`. This program must be loaded with a lexical analyzer function, `yylex()` (often generated by [lex\(1\)](#)), with a `main()` program, and with an error handling routine, `yyerror()`. Simple default versions of the last two are loaded by option **-ly** of [ld\(1\)](#).

The options are

**-o** *output*

Direct output to the specified file instead of

**-D** Create file containing diagnostic messages. To incorporate them in the parser, compile it with preprocessor symbol **YYDEBUG** defined. The amount of diagnostic output from the parser is regulated by values of an external variable `yydebug`:

Report errors.

1 Also report reductions.

2 Also report the name of each token returned by `yylex`.

**-v** Create file containing a description of the parsing tables and of conflicts arising from ambiguities in the grammar.

**-d** Create file containing **#define** statements that associate *yacc*-assigned ‘token codes’ with user-declared ‘token names’. Include it in source files other than `y.tab.c` to give access to the token codes.

**-s** *stem*

Change the prefix `y` of the file names and `y.output` to *stem*.

*Eyacc* is a special version of *yacc*, with systematic error recovery. It is used to compile *pascal(A)*

**FILES**

`y.output`

`y.tab.c`

`y.tab.h`

`y.debug`

`y.tmp.*`

temporary file

`y.acts.*`

temporary file

`/usr/lib/yaccpar`

parser prototype for C programs

`/usr/lib/liby.a`

library **-ly**

**SEE ALSO**

[lex\(1\)](#)

S. C. Johnson and R. Sethi, this manual, Volume 2

**BUGS**

The parser may not have full information when it writes to `y.debug` so that the names of the tokens returned by `yyllex` may be missing.

**NAME**

yes – be repetitively affirmative

**SYNOPSIS**

yes [ **expletive** ]

**DESCRIPTION**

**Yes** repeatedly outputs “y”, or if **expletive** is given, that is output repeatedly. Termination is by rubout.

**BUGS**

Boring.

**NAME**

zero – emit constant bytes

**SYNOPSIS**

**zero** [*byte*[*nbytes*]]

**DESCRIPTION**

*Zero* emits *nbytes* bytes, each with value *byte*. A missing *nbytes* is taken to be infinite. A missing *byte* is taken to be zero. Both arguments may be specified C-style.

**NAME**

uuclean – uucp spool directory clean-up

**SYNOPSIS**

**uuclean** [ option ] ...

**DESCRIPTION**

*Uuclean* will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

- p***pre* Scan for files with *pre* as the file prefix. Up to 10 **-p** arguments may be specified. A **-p** without any *pre* following will cause all files older than the specified time to be deleted.
- ntime** Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)
- m** Send mail to the owner of the file when it is deleted.

This program will typically be started by *cron*(8).

**FILES**

/usr/lib/uucp	directory with commands used by uuclean internally
/usr/lib/uucp/spool	spool directory

**SEE ALSO**

uucp(1C), uux(1C)



**NAME**

uudiff – directory comparison between machines

**SYNOPSIS**

**uudiff** [ -d ] local-name remote-name

**DESCRIPTION**

*Uudiff* compares the files in the directory *local-name* and the directory *remote-name*, (where *remote-name* may be of the form *system-name!directory-name* and *system-name* is a *uucp* Unix name). It reports (via mail) which files are added, deleted, or changed, and provides a *diff(1)* of altered printable files.

If a part of *remote-name* is omitted (either the system or the directory) the corresponding part of *local-name* is used. If *local-name* is a file, rather than a directory, *remote-name* is also assumed to be a file and the program degenerates into *diff(1)*.

The option **-d** does not diff altered files; only the summary by file names is prepared.

**FILES**

Lots. Files *zz[abcdeglmn]????* are used for scratch space; files brought back from the remote machine for *diffing* are stored (and left around) as *name.????* and the final report is left in *uudiff.????* (the exact name is reported by mail).

**SEE ALSO**

*diff(1)*, *uucp(1)*

**DIAGNOSTICS**

Almost none. Anything more serious than misspelling *local-name* causes unpredictable and obscure results.

**BUGS**

In addition to the standard *uucp* requirements a hook is needed at the remote system, and at present is only installed on the systems "research" and "inter".

This program is written in shell and should be translated to C so it could give diagnostics.

Even if "remote-system" is the local system, *uudiff* is subject to delays in *uucp* traffic.

It should probably write the standard output, instead of insisting on going into the background.

Since checksums are used there is a probability of 1 in  $2^{*32}$  of missing differences between files.

The *userid* syntax is not recognized.

**NAME**

uuencode,uudecode – encode/decode a binary file for transmission via mail

**SYNOPSIS**

**uuencode** [ source ] remotetest | **mail** sys1!sys2!...!decode

**uudecode** [ file ]

**DESCRIPTION**

*Uuencode* and *uudecode* are used to send a binary file via uucp (or other) mail. This combination can be used over indirect mail links even when *uusend(1)* is not available.

*Uuencode* takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotetest* for recreation on the remote system.

*Uudecode* reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user “decode” should be filtered through the uudecode program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using *delivermail* or by making *rmail* be a link to *Mail* instead of *mail*. In each case, an alias must be created in a master file to get the automatic invocation of uudecode.

If these facilities are not available, the file can be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

**SEE ALSO**

uuencode(5), uusend(1), uucp(1), uux(1), mail(1)

**AUTHOR**

Mark Horton

**BUGS**

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often *uucp*) must have write permission on the specified file.

**NAME**

uusend – send a file to a remote host

**SYNOPSIS**

**uusend** [ **-m** mode ] sourcefile sys1!sys2!...!remotefile

**DESCRIPTION**

*Uusend* sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of *uucp(1)* links needs to connect the two systems.

If the **-m** option is specified, the mode of the file on the remote end will be taken from the octal number given. Otherwise, the mode of the input file will be used.

The sourcefile can be “-”, meaning to use the standard input. Both of these options are primarily intended for internal use of uusend.

The remotefile can include the userid syntax.

**DIAGNOSTICS**

If anything goes wrong any further away than the first system down the line, you will never hear about it.

**SEE ALSO**

uux(1), uucp(1), uuencode(1)

**AUTHOR**

Mark Horton

**BUGS**

This command shouldn't exist, since *uucp* should handle it.

All systems along the line must have the *uusend* command available and allow remote execution of it.

Some uucp systems have a bug where binary files cannot be the input to a uux command. If this bug exists in any system along the line, the file will show up severely munged.

**NAME**

gplot – send a job to GCOS to plot draw files

**SYNOPSIS**

**gplot** [ *option* ] [ *file* ] ...

**DESCRIPTION**

The following *options* are used to print drawing files. The input to *gplot* may be generated using *draw -g* or *plot -g*. If the drawing contains a border and the **-b** flag is used the output will include the standard border used on 2S graph paper.

- The output is not sent to GCOS but printed on the standard output instead.
- c Produce Calcomp output, via the Honeywell 6000.
- f Produce microfilm output through the FR80 at Holmdel.
- s Produce output on Stare, via the Murray Hill Computer Center.
- v Produce a VU-Graph using the FR80 at Holmdel. There is no border with VU-Graph output.

The following variables from the environment(V) are used. **user** user name for \$IDENT card  
**acct** murray hill gcos account number **bin** btl bin number **sgrade** service grade for gcos job If any of these variables is not set its value is obtained using *getuid(I)*.

**NAME**

pins – look up pin names

**SYNOPSIS**

**pins** *pattern* ...

**DESCRIPTION**

*pins* prints the description of the chip type *pattern* from the file `/usr/lib/cda/lib/pins`. *pattern* is in a form suitable for `grep(I)`. If the name matched is a synonym for another part, both type names will be printed. The pin names and pin numbers are used by the circuit macro expander *cdm*.

**Naming Conventions**

A set of pin naming conventions is used, based on the traditional naming found in, for example, the Texas Instruments TTL Data Book. Function inputs and outputs are usually given as a single capital letter. Special inputs and outputs are given a short mnemonic name, such as **CLR** for clear. Lowercase letters are used where a subscript might normally be used. Where multiple gates exist within a single package, they are distinguished by appending a zero based numeric index. Pins which are active when low are indicated by adding a minus sign as the last character of the name.

General inputs are labeled with a single letter starting with the letter **A**. Functions with single inputs use the letter **D** with indices, as well as memories and flip-flops. When the outputs of logic elements are not synchronous with respect to another input they are named **Y**. Synchronous outputs, such as with flip-flops are called **Q**. Clock lines are called **CK**, clear lines are called **CLR**, preset lines are called **PR**. Select lines for multiplexors and data selectors are called **S**. The letter **G** is used for enable, chip enable, chip select, and output enable. The use of **OE** for output enable is used for tri-state devices where there is a separate enabling of the chip and its outputs (such as 74S373). Memories use the letter **A** for address lines, **WE** for write enable and **Y** for outputs.

**NAME**

prtx – graphics mode output filter for matrix printers

**SYNOPSIS**

prtx [ **-i** p ] [ **-o** device ] [ **-n** font ] [ **-w** width ] [ **-Ocx**f ] files

prtx [ **-i** t ] [ **-o** device ] [ **-n** font ] [ **-w** width ] [ **-Ocx**f ] [ **-s** charsize ] files

prtx [ **-i** g ] [ **-o** device ] [ **-n** font ] [ **-Ocxuf** ] [ **-r** region ] files

**DESCRIPTION**

*prtx* reads commands from the specified files and constructs pictures. printronix matrix printer. If no files are specified it reads from the standard input. The file name "-" also specifies the standard input.

The options are:

**-o device**

Output appropriate for the designated device. Currently supported devices are:

**prtx**

A printronix matrix printer. This is the default assumed if the **-i** option is omitted.

**hrprtx**

High resolution printronix. Some printronix printers have twice the horizontal resolution of the standard printer. This option produces output for such printers.

**tty**

Any ascii terminal. The output in this mode is crude and it is intended only for previewing command files whose output is ultimately intended for another device.

**trilog**. A trilog color printer.

**-i format**

This option specifies the format of the command file. The possible values are:

**prtx**

"Prtx" format. Ascii commands suitable for hand construction described in *prtx*(5). (This is the default format.)

**tplot**

"Tplot" format. Commands usually given to *tplot*(1) as described in *plot*(5). The **-s** option specifies the size of characters as an integer multiple of the normal height. (The default is 1.)

**troff**

"troff" format. Output from *newtroff*.

**gps**

*Gps*(5) format. Normally used for display on Tektronix terminals. Produced by the *stat*(1) commands. The **-r** and **-u** options are as for *ged*(1).

**-n font**

*Font* is a file that describes an alternate character set. If it starts with a '/' it is interpreted as a full path name. If it starts with './' it is relative to the current directory. Otherwise it is relative to the font library. If this option is omitted "*DEVICE*.font" is assumed. Where *DEVICE* is the **-o** option.

**-w width**

*Width* is the width (in inches) of the paper. It is a floating point number. It overrides the default width for the device.

**-O** Overlays all files into one picture. (Normally each file goes on a separate page.)

**-x** Exchange rows and columns. That is, turn picture on its side.

**-c** Let output continue vertically across page boundaries. This permits pictures of any height to be drawn.

**-f** Fit the output the to a page. I.e. rescale the locations of points so that the picture fits exactly on the page. The original picture may have been bigger or smaller than a page.

**EXAMPLES**

Typical use is:

```
prtx <sample | lpr -uk -o
```

The "-o" is necessary so that lpr doesn't complain about unprintable characters. Another example is

```
graph <numbers | prtx -t | lpr -uk -o
```

**SEE ALSO**

*prtx(5)*, *prtxfont(5)*, *graphics(1)*, *graph(1)*, *plot(5)*, *gps(5)*.

**FILES**

The font library is `/usr/marx/lib`.

**DIAGNOSTICS**

Many diagnostics are printed on the standard error output. In general an attempt is made to continue.

**BUGS**

The *boxit* prefix cannot always determine the boundary of the prefixed command.

This command is a descendant of one that only produced output for the printronix, hence its name.

The printronix, and trilog are slow devices in graphics mode.

Small narrow ellipses do not work.

Large input files can result in very poor performance because of a problem with *malloc*.

The "a" (arc) command for troff input is not yet implemented.

Many characters expected by troff users are not in the current font.

**AUTHOR**

Jerry Schwarz (harpo!jerry)

**NAME**

intro, errno – introduction to system calls and error numbers

**SYNOPSIS**

```
#include <errno.h>
```

**DESCRIPTION**

Section 2 describes the entries into the operating system.

**File I/O**

Files are opened for input or output by *open(2)* or *creat*. These calls return a integer called a *file descriptor* which identifies the file to subsequent I/O calls, notably *read(2)* and *write*. File descriptors range from 0 to 127 in the current system. The system gets to pick the numbers, but they may be reassigned by *dup(2)* and *dup2*.

By agreement among user programs, file descriptor 0 is the standard input, 1 is the standard output, 2 is for error messages, and 3 is the controlling terminal if any. The operating system is unaware of these conventions; it is permissible to close file 0, or even to replace it by a file open only for writing, but many programs will be confused by such chicanery.

Files are normally read or written in sequential order. *Lseek(2)* addresses arbitrary locations.

Files have associated status, consisting of ownerships, permission modes, access dates, and so on. The status is retrieved by *stat(2)*; the calls in *chmod(2)* alter parts of it.

New files are made with *creat* (in *open(2)*). An existing file may be given an additional name by *link(2)* or *symlink*; names are removed by *unlink(2)*. Directories are created and removed by *mkdir(2)* and *rmdir*.

Device files and communication channels (streams) admit a plethora of special operations, most specific to the device in question; see *ioctl(2)* and the device writeups in section 4, especially *tyld(4)* for terminals and *stream(4)* for communications. *Pipe(2)* creates nameless streams, useful for local communication. Several streams may be monitored in parallel by *select(2)*.

**Process execution and control**

A new process is created when an existing one calls *fork(2)*. The new (child) process starts out with copies of the address space and most other attributes of the old (parent) process. In particular, the child starts out running the same program as the parent; *exec(2)* will bring in a different one.

Each process has an integer process id, unique among all currently active processes; a process group id, used to distribute signals among processes in the same session or window; a userid and groupid, which determine access permissions; and a character-string login name for the current user (not the same as permissions). The calls in *getuid(2)* retrieve and change these values.

Various events cause software traps (signals): program errors like addressing violations, software events like the interrupt key on the terminal, the alarm clock set by *alarm(2)*, calls to *kill* (in *signal(2)*). Most signals terminate the process by default; *signal(2)* will arrange to trap or ignore them instead.

A process terminates on receiving a signal or by calling *exit(2)*. A parent process may call *wait* (in *exit(2)*) to wait for some child to terminate. A single byte of status information may be passed from *exit* to *wait*.

**Timekeeping**

*Time(2)* and *ftime* return the time of day and related information. *Times(2)* returns runtime accounting for this process and its children. *Profil* arranges to increment various locations in memory whenever the clock ticks; it is useful for execution profiling.

*Times*, *profil*, and a few other calls measure time in clock ticks. The clock frequency is given by the constant **HZ** in *<sys/param.h>*; 60 ticks per second in this system.

**SEE ALSO**

*intro(3)*, *perror(3)*

**DIAGNOSTICS**

A ‘Diagnostics’ paragraph appears in the page for each system call that has an error return. Unless otherwise stated, the error value is the integer  $-1$ , and the success value is 0. When an error occurs, an error number is assigned to the external variable *errno*. *Errno* is not cleared on successful calls, so it should be tested only after an error has occurred.



There is a table of messages that describe the errors and a routine for printing them; see [perror\(3\)](#). The list below gives the number, the name (as defined in `<errno.h>`), and the *perror* message for each error type. The reasons for error returns are explained in general terms; further explanations for less obvious error returns appear in the writeups of individual system calls.

- 0 [CB] Error 0  
No error has occurred.
- 1 [CB] EPERM Not owner  
An attempt was made to modify a file in some way forbidden except to its owner or the super-user, or an ordinary user attempted to do something allowed only to the super-user.
- 2 [CB] ENOENT No such file or directory  
A file name was specified and the file should exist but didn't, or one of the directories in a path name did not exist.
- 3 [CB] ESRCH No such process  
The process whose number was given to *kill* did not exist, or was already dead.
- 4 [CB] EINTR Interrupted system call  
A signal which the user has elected to catch occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.
- 5 [CB] EIO I/O error  
A physical I/O error or timeout occurred, usually in *read*, *write*, or *ioctl*. This error may in some cases be returned on a call following the one to which it actually applies.
- 6 [CB] ENXIO No such device or address  
I/O on a special file referred to a device which does not exist or is off line, or beyond the limits of the device.
- 7 [CB] E2BIG Arg list too long  
An argument list longer than 16384 bytes was presented to *exec*.
- 8 [CB] ENOEXEC Exec format error  
A request was made to execute a file which, although it had the appropriate permissions, did not start with a valid magic number, see [a.out\(5\)](#).
- 9 [CB] EBADF Bad file number  
A file descriptor referred to no open file, or a *read* (resp. *write*) a file which was open only for writing (resp. reading).
- 10 [CB] ECHILD No children  
In *wait*, the process had no living or unwaited-for children.
- 11 [CB] EAGAIN No more processes  
In *fork*, the system's process table was full or the user was not allowed to create any more processes.
- 12 [CB] ENOMEM Not enough memory  
During *exec* or *brk*, a program asked for more memory or swap space than the system was able to supply.
- 13 [CB] EACCES Permission denied  
An attempt was made to access a file in a way forbidden by the protection system.
- 14 [CB] EFAULT Bad address  
The system encountered a hardware fault in attempting to access the arguments of a system call.
- 15 [CB] EHASF Directory not empty  
An attempt was made to remove a nonempty directory.
- 16 [CB] EBUSY In use  
An attempt was made to mount a device that was already mounted (or crashed or was copied in mounted state), to dismount a device on which there was an active file (open file, current directory, mounted-on file, active text segment), or to remove the current directory of some process.

- 17 [CB]EEXIST File exists  
An existing file was mentioned in an inappropriate context, e.g. *link*.
- 18 [CB]EXDEV Cross-device link  
A link to a file on another device was attempted.
- 19 [CB]ENODEV No such device  
An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.
- 20 [CB]ENOTDIR Not a directory  
A non-directory was specified where a directory is required, for example in a path name or as an argument to *chdir*.
- 21 [CB]EISDIR Is a directory  
An attempt to write on a directory.
- 22 [CB]EINVAL Invalid argument  
Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in *signal*, reading or writing a file for which *lseek* has generated a negative pointer. Also set by math functions, see *intro(3)*.
- 23 [CB]ENFILE File table overflow  
The system's table of open files was full, and temporarily no more *opens* could be accepted.
- 24 [CB]EMFILE Too many open files  
The limit is 128 per process.
- 25 [CB]ENOTTY Illegal ioctl  
The function code mentioned in *ioctl* does not apply to the file or device.
- 26 [CB]ETXTBSY Text file busy  
An attempt to execute a pure-procedure program which was open for writing, or to open for writing a pure-procedure program that was being executed.
- 27 [CB]EFBIG File too large  
The size of a file exceeded the maximum (about  $10^9$  bytes).
- 28 [CB]ENOSPC No space left on device  
During a *write* to an ordinary file, there was no free space left on the device.
- 29 [CB]ESPIPE Illegal seek  
*Lseek* was issued to a stream device.
- 30 [CB]EROFS Read-only file system  
An attempt to modify a file or directory was made on a device mounted read-only.
- 31 [CB]EMLINK Too many links  
An attempt to make more than 32767 links to a file.
- 32 [CB]EPIPE Broken pipe  
*Write* to a stream that has been hung up, usually a pipe with no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.
- 33 [CB]EDOM Math argument  
The argument of a function in the math package (3M) was out of the domain of the function.
- 34 [CB]ERANGE Result too large  
The value of a function in the math package (3M) was unrepresentable within machine precision.
- 35 [CB]ELOOP Link loop  
An endless cycle of symbolic links was encountered.
- 36 [CB]ECONC Concurrency violation  
An *open* or *creat* was in violation of the concurrent access specified for the file.
- 37 [CB]EGREG It's all Greg's fault  
Something went wrong.

**BUGS**

Device and file system drivers may use error codes in unexpected or unconventional ways; it is infeasible to list them all. The writeups in section 4 list some such special cases.

**NAME**

access – determine accessibility of file

**SYNOPSIS**

```
int access(name, mode)
char *name;
```

**DESCRIPTION**

*Access* checks the given file *name* for accessibility. If *mode[CB]&4* is nonzero, read access is checked. If *mode[CB]&2* is nonzero, write access is checked. If *mode[CB]&1* is nonzero, execute access is checked. If *mode[CB]=0*, the file merely need exist. In any case all directories leading to the file must permit searches. 0 is returned if the access is permitted,, -1 if not.

Permission is checked against the real *userid* and *groupid* of the process; this call is most useful in *set-userid* and *set-groupid* programs.

Only access bits are checked. A directory may be announced as writable by *access*, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but *exec(2)* will fail unless it is in proper format.

If the *userid* of the process is the owner of the file access is determined by the three owner bits (0700). Otherwise, if the *groupid* of the process is the group of the file access is determined by the three group bits (0070). Otherwise access is determined by the three other bits (0007).

**SEE ALSO**

[stat\(2\)](#)

**DIAGNOSTICS**

**EACCES, EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EROFS, ETXTBSY**

**BUGS**

On symbolic links permissions are irrelevant and *access* returns nonsense.

**NAME**

*acct* – turn accounting on or off

**SYNOPSIS**

```
int acct(file)
char *file;
```

**DESCRIPTION**

*Acct*, with a null-terminated string naming an existing file as argument, turns on accounting; records for each terminating process are appended to *file*. An argument of (char \*)0 causes accounting to be turned off.

*Acct* may only be invoked by the super-user.

The accounting file format is given in [acct\(5\)](#).

**SEE ALSO**

[acct\(5\)](#), [sa\(8\)](#)

**DIAGNOSTICS**

**EACCES** (file not a regular file), **EBUSY** (accounting already turned on), **EFAULT**, **EIO**, **ELOOP**, **ENOENT**, **ENOTDIR**, **EPERM**, **EROFS**, **ETXTBSY**

**BUGS**

No accounting is produced for programs running when a crash occurs. In particular nonterminating programs are never accounted for.

**NAME**

alarm, nap, pause – schedule timing delays

**SYNOPSIS**

**unsigned alarm(seconds)**

**unsigned seconds;**

**void nap(ticks)**

**void pause()**

**DESCRIPTION**

*Alarm* causes signal **SIGALRM**, see [signal\(2\)](#), to be sent to the invoking process in the number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because the clock has a one second resolution, the signal may occur up to one second early; because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount.

The return value is the amount of time previously remaining in the alarm clock.

*Nap* suspends execution of the current process for the specified number of clock ticks. If *ticks* is negative, it is taken to be zero; if it is greater than two seconds, it is taken to be two seconds.

*Pause* only returns upon termination of a signal handler started during the *pause*. It is used to give up control while waiting for a signal, usually from *kill* (see [signal\(2\)](#)), *alarm(2)*, or the terminal driver [tyld\(4\)](#).

**SEE ALSO**

[kill\(1\)](#), [signal\(2\)](#), [setjmp\(3\)](#), [sleep\(3\)](#)

**BUGS**

If the argument to *alarm* is greater than 65535, it is treated as 65535.

If the alarm clock expires during a call to *alarm*, the return value will be 0, and the signal will be delivered immediately after the system call returns. If the routine calling *alarm* saves the return value and later restores it, it will disable any alarm set by the signal handler.

**NAME**

brk, sbrk – change core allocation

**SYNOPSIS**

**int** brk(addr)

**char \***addr;

**char \***sbrk(incr)

**DESCRIPTION**

*Brk* sets the system's idea of the lowest location not used by the program (called the break) to *addr* rounded up to the next multiple of 1024 bytes. Locations not less than *addr* and below the stack pointer may cause a memory violation if accessed.

In the alternate function *sbrk*, *incr* more bytes are added to the program's data space and a pointer to the start of the new area is returned. Rounding occurs as with *brk*, but a nominal break is remembered, so rounding does not accumulate.

When a program begins execution via *exec* the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use *brk*.

The error return from *sbrk* is (**char \***)-1.

**SEE ALSO**

*exec(2)*, *end(3)*, *malloc(3)*

**DIAGNOSTICS**

**ENOMEM**

**NAME**

chdir, chroot – change working or root directory

**SYNOPSIS**

**int chdir(dirname)**

**char \*dirname;**

**int chroot(dirname)**

**char \*dirname;**

**DESCRIPTION**

*Chdir* changes the working directory of the invoking process to *dirname*; *chroot* changes its root directory.

The root directory is the starting point when searching for pathnames beginning with /. The working directory is the starting point for pathnames that don't. The root directory normally points to the system root. *Login*(8) initially sets the working directory as specified in the password file.

After *chroot*, it is impossible to name a file outside the subtree rooted at the current root, provided that the current directory is located within the subtree and there are no links pointing outside the subtree (except for the entry `..` in the root directory).

*Chroot* may only be used by the super-user.

**SEE ALSO**

*sh*(1), *passwd*(5)

**DIAGNOSTICS**

**EACCES, EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EPERM** (*chroot* only)

**BUGS**

Using *chroot*, it is quite easy to fool set-userid programs about the contents of the password file (for example).



**NAME**

chmod, fchmod, chown, fchown, utime – change file mode, owner, group, or times

**SYNOPSIS**

```
int chmod(file, mode)
char *name;

int fchmod(fildes, mode)

int chown(file, uid, gid)
char *name;

int fchown(fildes, uid, gid)

#include <sys/types.h>

int utime(file, timep)
char *file;
time_t timep[2];
```

**DESCRIPTION**

These functions change inode information for the file named by a null-terminated string or associated with file descriptor *fildes*.

*Chmod* and *fchmod* change file permissions and other mode bits to *mode*. Mode values are defined in [stat\(2\)](#). Only the **07777** bits of *mode* are significant. Only the owner of a file (or super-user) may change the mode. Only a process in the file's group (or super-user) may set the set-group-id bit, **S\_ISGID**.

*Chown* and *fchown* change the owner, *uid*, and the group, *gid*, of a file. Only the super-user may change a file's owner. The owner of a file may change its group to match the current effective groupid. Other changes are restricted to the super-user.

*Utime* sets the **st\_atime** (access time) and **st\_mtime** (modify time) fields for *file* to **timep[0]** and **timep[1]** respectively. The **st\_ctime** (inode change time) field for *file* is set to the current time.

The caller must be the owner of the file or the super-user.

**SEE ALSO**

[stat\(2\)](#), [time\(2\)](#)

**DIAGNOSTICS**

all: **EIO**, **EPERM**

*chmod*, *chown*, *utime*: **ELOOP**, **ENOENT**, **ENOTDIR**, **EACCES**, **EFAULT**

*fchmod*, *fchown*: **EBADF**

**BUGS**

An attempt to change the inode data for a file on a read-only file system is quietly ignored.

**NAME**

reboot, vadvise, vlimit, vswapon, getgroups, setgroups – system calls to be avoided

**SYNOPSIS**

**int** reboot(**how**)

**int** vadvise(**how**)

**int** vlimit(**what**, **limit**)

**int** vswapon(**special**)

**char** \***special**;

**#include** <sys/param.h>

**int** getgroups(**ngroups**, **gidset**)

**short** \***gidset**;

**setgroups**(**ngroups**, **gidset**)

**short** \***gidset**;

**DESCRIPTION**

These calls are hangovers from prior versions of the system. Some exist only for system maintenance purposes; some depend on the virtual memory implementation. None should be used except as a last resort. Most are not included in **/lib/libc.a**.

*Reboot* finishes any pending I/O and reboots the system (if *how* is 0) or puts the system into a tight loop with interrupts disabled (if *how* is 8). It is restricted to the super-user.

*Vadvise* gives the virtual memory system hints about the paging behavior of the current process.

*Vlimit* sets various resource limits, such as the amount of memory allowed for text and data, and the maximum size of core images.

*Vswapon* adds the block device *special* to the pool of swap space. The device must be listed in a table compiled into the operating system; *vswapon* merely enables it.

*Getgroups* stores at most *ngroups* elements of the group access list of the current process in the array *gidset*.

*Setgroups* sets the group access list of the current user process from *gidset*. *Ngroups* gives the number of entries; it must not exceed **NGROUPS**, defined in **<param.h>**. Only the super-user may add groups to the list.

**SEE ALSO**

*Unix Programmer's Manual, Seventh Edition, Virtual VAX-11 Version*, Volume 1, 1980 (Berkeley)

**NAME**

*dirread* – read from directory, hiding format

**SYNOPSIS**

```
int dirread(fildes, buffer, nbytes)
char *buffer;
```

**DESCRIPTION**

*Dirread* reads at most *nbytes* bytes of directory information from the file pointer location in the directory associated with *fildes* into memory at *buffer*. The information is converted into a canonical form, independent of the format used by the file system in which the directory resides.

Each canonicalized entry consists of a decimal ASCII inode number, a tab character, a file name, and an ASCII NUL. *Buffer* is filled with as many entries as will fit; the number of bytes used is returned. 0 is returned when there are no more entries.

The file pointer is advanced by the number of bytes passed over in the directory, not the number of bytes placed in *buffer*. Ask *lseek* for the new pointer; don't try to compute it.

**SEE ALSO**

[open\(2\)](#), [read\(2\)](#), [directory\(3\)](#), [dir\(5\)](#)

**DIAGNOSTICS**

**EBADF**, **ENOSPC** (buffer too small), **ENOTDIR**, **EFAULT**, **EINVAL**

**BUGS**

Not all file system types support *dirread*. The routines in [directory\(3\)](#) know how to cope. Seeking in directories is dangerous, because the contents often change under foot.

**NAME**

`dup`, `dup2` – duplicate an open file descriptor

**SYNOPSIS**

```
int dup(fildes)
```

```
int fildes;
```

```
int dup2(fildes, fildes2)
```

```
int fildes, fildes2;
```

**DESCRIPTION**

Given a file descriptor *dup* allocates another file descriptor synonymous with the original. The new file descriptor is returned.

In *dup2*, *fildes* is a file descriptor referring to an open file, and *fildes2* is an integer in the range of legal file descriptors. *Dup2* causes *fildes2* to refer to the same file and returns *fildes2*. If *fildes2* already referred to another open file, it is closed first.

**SEE ALSO**

[open\(2\)](#), [pipe\(2\)](#), [fd\(4\)](#)

**DIAGNOSTICS**

**EBADF, EMFILE**

**BUGS**

*Dup* of a file descriptor greater than 63 turns into a *dup2* with a random second argument.

**NAME**

execl, execv, execl, execve, execlp, execvp, exect – execute a file

**SYNOPSIS**

```
int execl(name, arg0, arg1, ..., argn, (char *)0)
char *name, *arg0, *arg1, ..., *argn;
```

```
int execv(name, argv)
char *name, *argv[];
```

```
int execl(name, arg0, arg1, ..., argn, (char *)0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[];
```

```
int execve(name, argv, envp)
char *name, *argv[], *envp[];
```

```
int execlp(name, arg0, arg1, ..., argn, (char *)0)
char *name, *arg0, *arg1, ..., *argn;
```

```
int execvp(name, argv)
char *name, *argv[];
```

```
int exect(name, argv, envp)
char *name, *argv[], *envp[];
```

**DESCRIPTION**

*Exec* in all its forms overlays the calling process with the named file, then transfers to the entry point of the image of the file. There can be no return from a successful *exec*; the calling image is lost.

Files remain open across *exec* unless explicit arrangement has been made; see *ioctl(2)*. Signals that are caught (see *signal(2)*) are reset to their default values. Other signal settings are unchanged.

Each user has a *real* userid and groupid and an *effective* userid and groupid. The real userid (groupid) identifies the person using the system; the effective userid (groupid) determines access privileges. *Exec* changes the effective userid and groupid to the owner of the executed file if the file has the set-userid or set-groupid modes. The real userid is not affected.

*Name* points to the name of the file to be executed. It must be a regular file (type **S\_IFREG**, see *stat(2)*) and its permissions must allow execution. *Arg0*, *arg1*, ... or the pointers in *argv* address null-terminated argument strings to be made available when the new image starts. *Argv* must end with a 0 pointer. Conventionally argument 0 is the name of the program.

In *execl*, *execve*, and *exect*, the *envp* array contains pointers to a set of null-terminated strings composing the environment of the process. *Envp* must end with a 0 pointer. The other calls copy the present environment from the global cell *environ*; see *environ(5)*.

The *execl* and *execv* forms differ only in argument syntax; one is more convenient when the number of arguments is known in advance, the other when arguments are assembled on the fly.

If the first two bytes of the file are the characters #!, subsequent text up to a newline is examined. The first word, up to a blank or tab, names an interpreter program; anything left over is a single supplemental argument. The original *name*, preceded by the supplemental argument if any, is inserted in the argument list between *arg0* and *arg1* (or between the first pair of *argv* pointers). The interpreter is executed with the modified argument list.

If the file doesn't start with #!, a standard header for a binary image is expected; see *a.out(5)*. If the file doesn't begin with a valid header either, **ENOEXEC** is returned. The shell *sh(1)* takes this to mean that the file contains shell commands.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

*Argv* is the array of argument pointers passed to *exec*; *argc* is the number of arguments. *Argv* is directly usable in a subsequent *execv* because **argv[argc]==0**. *Envp* is the environment array; the same value has already been stored in *environ*.

*Execlp* and *execvp* take the same arguments as *execl* and *execv*, but search the directories listed in the **PATH** environment variable for an executable file called *name*, mimicking the shell's path search.

*Exec* is the same as *execve*, except that it arranges for the process to stop just before the first instruction of the new image; see [proc\(4\)](#).

## FILES

/bin/sh

shell, invoked if command file found by *execlp* or *execvp*

## EXAMPLES

This file, if created with execute permissions and run by *exec*, calls [awk\(1\)](#) to count the lines in all the files named in its arguments:

```
#!/usr/bin/awk -f
END { print NR }
```

## SEE ALSO

[sh\(1\)](#), [fork\(2\)](#), [ioctl\(2\)](#), [signal\(2\)](#), [proc\(4\)](#), [environ\(5\)](#)

## DIAGNOSTICS

**E2BIG**, **EACCES**, **EFAULT**, **EIO**, **ELOOP**, **ENOENT**, **ENOEXEC**, **ENOMEM**, **ENOTDIR**, **ENXIO**, **ETXTBSY**

## BUGS

If *execvp* is called to execute a file that turns out to be a shell command file, and the shell cannot be executed, some of the values in *argv* may be modified before return.

Neither the shell's path search nor that of *execlp* and *execvp* extends to the interpreter named after `#!`. The interpreter file may not itself begin with `#!`. The text after `#!` may be no more than 30 characters long, including the newline.

**NAME**

`_exit`, `wait` `wait3` – terminate process, wait for child to terminate

**SYNOPSIS**

```
void _exit(status)
int status;

int wait(status)
int *status;

int wait((int *)0)

#include <sys/vtimes.h>

wait3(status, options, ch_vt)
int *status;
struct vtimes *ch_vt;
```

**DESCRIPTION**

`_exit` closes all the process's files and notifies the parent process when the parent executes `wait`. The low-order 8 bits of `status` are available to the parent process. The call never returns.

The function `exit(3)`, which is the normal means of terminating a process, may cause cleanup actions before finally calling `_exit`. Therefore, `_exit` should be called to terminate a child process after a `fork(2)` to avoid flushing buffered output twice.

`Wait` delays until a signal is received or until a child process terminates or receives signal **SIGSTOP**. There is no delay if any child has died since the last `wait`, or if there are no extant children. The normal return yields the process id and status of one terminated child. The status of other children may be learned from further `wait` calls.

If `status` is nonzero, `wait` sets `*status = (s<<8)+t` where `s` is the low 8 bits of `status` from the child's `exit`, if any, and `t` is the termination status of the child. See `signal(2)` for a list of termination statuses (signals); status 0 indicates normal termination, 0177 a (restartable) process stopped on **SIGSTOP**. If the 0200 bit of the termination status is set, a core image of the process was produced by the system.

`Wait3` is similar to `wait`. An `option` value of 1 prevents waiting for extant, non-stopped children and causes 0 to be returned if children exist but none have reportable status. If `ch_vt` is nonzero, resource usage data for the child are reported as by `vtimes(2)`.

If the parent process terminates without waiting on its children, they are inherited by process 1 (the initialization process, `init(8)`).

**SEE ALSO**

`fork(2)`, `exit(3)`, `signal(2)`, `sh(1)`

**DIAGNOSTICS**

`wait`, `wait3`: **ECHILD**

**BUGS**

If the argument to `wait` is bogus, the user program gets a memory fault rather than an **EFAULT**. The 0 third argument to `wait3` is a required historical dreg.

**NAME**

fmount, funmount – mount or remove file system

**SYNOPSIS**

```
int fmount(type, fildes, name, flag)
char *name;
```

```
int funmount(name)
char *name;
```

**DESCRIPTION**

*Fmount* mounts a file system of the named *type* described by the file descriptor *fildes* on pathname *name*. Henceforth, references to *name* (the mount point) will refer to the root file on the newly mounted file system.

*Name* must already exist. Its old contents are inaccessible while the file system is mounted.

The meaning of *flag* varies with the file system type.

Allowed types are

- 0** Regular (block device) file system. *Fildes* must be a block special file. If *flag* is nonzero, the file system may not be written on; this must be used with physically write-protected media or errors will occur when access times are updated, even if no explicit write is attempted.
- 2** Process file system, *proc(4)*. *Fildes* is ignored.
- 3** Mounted stream. *Fildes* must refer to a stream; future calls to *open(2)* on *name* will reopen that stream. The mount is undone if the other end of the stream is closed or hung up.
- 4** Stream (network) file system. *Fildes* is a stream connected to a file system server, *netfs(8)*.

Types 5 and 6 are used internally to close off errors and for pipes; these types may not be mounted.

*Funmount* removes knowledge of the file system mounted at *name*. The mount point reverts to its previous interpretation.

The userid owning *name* may mount or unmount file systems of type 3 or 4. For other types, these calls are restricted to the super-user.

**SEE ALSO**

*mount(8)*, *netfs(8)*, *proc(4)*, *stream(4)*

**DIAGNOSTICS**

**EBADF, EBUSY, EINVAL, EIO, ENODEV**

**BUGS**

Although *fildes* for type 2 file systems is ignored, it must be a valid file descriptor.



**NAME**

fork – spawn new process

**SYNOPSIS**

**int** fork()

**DESCRIPTION**

*Fork* is the only way new processes are created. The new process's image is a copy of that of the caller of *fork*. The only distinction is that the value returned in the old (parent) process is the process id of the new (child) process, while the value returned in the child is 0. Process ids range from 1 to 30,000. The process id is used by *wait* (see [exit\(2\)](#)) and *kill* (see [signal\(2\)](#)).

Files open before the *fork* are shared, and have a common read-write pointer. This is the way that [sh\(1\)](#) passes standard input and output files and sets up pipes.

**SEE ALSO**

[exit\(2\)](#), [signal\(2\)](#), [sh\(1\)](#)

**DIAGNOSTICS**

EAGAIN, ENOMEM

**NAME**

getuid, getgid, geteuid, getegid, getlogname, getpid, getppid, getpgrp, setuid, setgid, setruid, setlogname, setpgrp – get or set user, group, or process identity

**SYNOPSIS**

**int** getuid()

**int** geteuid()

**int** getgid()

**int** getegid()

**int** getlogname(buf)

**char** \*buf;

**int** getpid()

**int** getppid()

**int** getpgrp(pid)

**int** pid;

**int** setuid(uid)

**int** setgid(gid)

**int** setruid(uid)

**int** setlogname(buf)

**char** buf[8];

**int** setpgrp(pid, pgrp)

**int** pid, pgrp;

**DESCRIPTION**

*Getuid* returns the real userid of the current process, *geteuid* the effective userid. The real userid identifies the person who is logged in, rather than the effective userid, which determines access permission at the moment. It is thus useful to set-userid programs to find out who invoked them.

*Getgid* returns the real groupid, *getegid* the effective groupid.

*Getlogname* copies the login name of the current process into the buffer pointed to by *buf*, which must be at least eight characters long.

*Getpid* returns the process id of the current process, *getppid* that of its parent process.

*Getpgrp* returns the process group id of process *pid*; means the current process.

*Setuid* (*setgid*) sets the effective and real userid (groupid) of the current process to *uid* (*gid*). Both the effective and the real userid (groupid) are set. These calls are permitted only if the process is super-user or if the argument is the real or effective userid (groupid).

*Setruid* sets the real userid only. It may only be used by the super-user.

*Setlogname* sets the login name returned by *getlogname*. It may only be used by the super-user.

*Setpgrp* sets the process group id of process *pid* to *pgrp*. *Pid* 0 is the current process. Only the super-user may set the process group of processes with other userids or set a process group to 0.

**SEE ALSO**

[getuid\(1\)](#), [getlogin\(3\)](#)

**DIAGNOSTICS**

*getlogname*: **EFAULT**

*setlogname*: **EFAULT, EPERM**

*setuid, setgid, setruid, setpgrp*: **EPERM**

**BUGS**

Non-super-user processes may set the process group of descendant processes; only certain unsupported shells use this, and the facility may vanish presently.

**NAME**

`ioctl` – miscellaneous control operations

**SYNOPSIS**

```
#include <sys/filio.h>
```

```
int ioctl(fildes, request, param)
```

```
void *param;
```

**DESCRIPTION**

*ioctl* performs a variety of *requests* on open files, most applying only to particular device files. The write-ups of various devices in section 4 discuss how *ioctl* applies to them; see [stream\(4\)](#) in particular for operations applying to communication devices.

*Param* points to a parameter buffer. Some *requests* read data from the buffer; others write data to the buffer; some do both. The buffer's size varies according to the request. A few provide meaningful data in the return value from *ioctl* as well; others return a conventional value of 0 for success, -1 for failure.

Requests for different drivers are defined in different header files. Two standard calls, defined in `<sys/filio.h>`, apply to any open file:

```
ioctl(fildes, FIOCLEX, (void *)0);  
ioctl(fildes, FIONCLEX, (void *)0);
```

The first causes the file to be closed automatically upon a successful [exec\(2\)](#); the second causes the file to be left open.

**SEE ALSO**

[exec\(2\)](#), [stream\(4\)](#)

**DIAGNOSTICS**

**EBADF**, **EFAULT**, **EIO**, **ENODEV**, **ENOTTY**

**BUGS**

*ioctl* requests vary among UNIX systems; undisciplined use is likely to compromise portability. The size of the parameter buffer should be an explicit argument.

**NAME**

limits – return or set limits structure

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/lnode.h>
#include <sys/retlim.h>
#include <sys/share.h>

limits(address, function)
struct lnode *address;
int function;
```

**DESCRIPTION**

This system call manipulates a kernel limits structure according to the value of *function*. Except where indicated below, *address* points to an *lnode* or an array of *lnodes*.

Function	Value	Meaning
L_MYLIM	0	Get user's own limits structure.
L_OTHLIM	1	Get limits associated with uid in lnode.
L_ALLLIM	2	All active limits structures are returned.
L_SETLIM	3*	Connect to a new limits structure.
L_DEADLIM	4	Wait for dead limits belonging to child.
L_CHNGLIM	5*	Changes limits fields in existing limits.
L_DEADGROUP	6*	Pick up a dead limits structure.
L_GETCOSTS	7	Get contents of system "shconsts" table.
L_SETCOSTS	8*	Set contents of system "shconsts" table.
L_MYKN	9	Get user's own "kern_lnode" structure.
L_OTHKN	10	Get structure associated with uid.
L_ALLKN	11	All active structures are returned.

The starred functions in the list are super-user only.

For L\_MYKN, L\_OTHKN, and L\_ALLKN *address* should point to a "struct kern\_lnode" defined in *<sys/lnode.h>*. For L\_SETCOSTS and L\_GETCOSTS *address* should point to a "struct shconsts" defined in *<sys/share.h>*. For L\_DEADLIM *address* should point to a "struct retlim" defined in *<sys/retlim.h>*.

L\_OTHLIM and L\_CHNGLIM require that the lnode pointed to by *address* contains the correct uid. L\_OTHKN requires that the kern\_lnode pointed to by *address* contains the correct uid. L\_MYLIM, L\_MYKN, L\_OTHLIM, and L\_OTHKN all return the number of processes currently attached to the node. L\_ALLLIM and L\_ALLKN both return the number of active nodes returned.

L\_SETLIM initialises a new limits structure with the passed lnode, and attaches the calling process to it. All children of that process will inherit the new structure.

L\_DEADGROUP looks for a dead limits structure, removes it from the list of active limits, and returns the lnode.

L\_DEADLIM performs a *wait(2)* system call, then returns a structure containing both the limits and process zombie structures. The value returned is the number of processes still attached to the node.

L\_SETCOST and L\_GETCOST deal with the constants structure for the scheduling algorithm.

Any other function is illegal, and will return an error of EINVAL. Unless otherwise specified the call returns the number of limits structures returned.

**DIAGNOSTICS**

**ESRCH** can be returned in *errno* by functions L\_DEADGROUP, L\_OTHKN, L\_OTHLIM and L\_CHNGLIM to indicate that the desired limits structure does not exist. **ESRCH** can also be returned by L\_SETLIM to indicate that this lnode's group has not been set-up.

**ETOOMANYU** is returned in *errno* for L\_SETLIM if there is no space left in the kernel limits table.

Any error causes a -1 to be returned.

**SEE ALSO**

setlimits(3), lnode(5), share(5).

**NAME**

link, symlink, readlink – link to a file

**SYNOPSIS**

```
int link(name1, name2)
char *name1, *name2;
```

```
int symlink(name1, name2)
char *name1, *name2;
```

```
int readlink(name, buf, size)
char *name, *buf;
```

**DESCRIPTION**

*Link* and *symlink* create a link to file *name1* with new name *name2*. Either name may be an arbitrary path name.

After *link*, *name2* is entirely equivalent to *name1*; it is a directory entry referring to the same file as *name1*. Only the super-user can make the link if *name1* is a directory.

After *symlink*, *name2* is a new symbolic link; when it is encountered in any path name, *name1* is substituted for *name2*, and path name parsing continues. If *name1* begins with the / character, it is interpreted with respect to the root directory; if not, it is interpreted with respect to the directory in which *name2* resides.

Symbolic links are slightly slower than normal links but may span file systems; normal links are confined to a single file system.

*Readlink* copies the pathname inside symbolic link *name* into memory at *buf*. No more than *size* bytes are copied; the actual number of bytes read is returned. The contents of *buf* will not be null-terminated. An error is returned if *name* is not a symbolic link.

**SEE ALSO**

[cp\(1\)](#), [unlink\(2\)](#), [stat\(2\)](#)

**DIAGNOSTICS**

all: **EIO**, **ELOOP**, **ENOENT**, **ENOTDIR**

*link*: **EEXIST**, **EROFS**, **EXDEV**

*symlink*: **EEXIST**, **EROFS**

*readlink*: **ENXIO**

**NAME**

`lseek`, `llseek` – seek, move read/write pointer

**SYNOPSIS**

**long** `lseek`(*fildev*, *offset*, *whence*)  
**long** *offset*;

**Long** `llseek`(*fildev*, *offset*, *whence*)  
**Long** *offset*;

**DESCRIPTION**

`Lseek` and `llseek` set the file pointer for the file associated with *fildev* as follows:

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset*.

If *whence* is 2, the pointer is set to the size of the file plus *offset*.

The new file pointer value is returned.

Type *Long* is a 64-bit quantity.

Seeking far beyond the end of a file, then writing, creates a gap or ‘hole,’ which occupies no physical space and reads as zeros.

**SEE ALSO**

[open\(2\)](#), [fseek\(3\)](#)

**DIAGNOSTICS**

**EBADF**, **ESPIPE**

**BUGS**

`Lseek` doesn’t affect some special files.



**NAME**

mkdir, rmdir – make or remove a directory

**SYNOPSIS**

```
int mkdir(name, mode)
char *name;
```

```
int rmdir(name)
char *name;
```

**DESCRIPTION**

*Mkdir* creates a new directory whose name is the null-terminated string pointed to by *name*. The mode of the directory is set to *mode*, as modified by the process's mode mask; see [stat\(2\)](#) and [umask\(2\)](#). The directory initially contains two entries: *.* (a link to the directory itself) and *..* (a link to the parent directory).

*Rmdir* removes the directory *name*, which must have only *.* and *..* entries.

**SEE ALSO**

[mkdir\(1\)](#), [rm\(1\)](#), [mknod\(2\)](#), [stat\(2\)](#), [umask\(2\)](#)

**DIAGNOSTICS**

*mkdir*: EEXIST, EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EROFS

*rmdir*: EFAULT, EHASH, EINVAL, EIO, ELOOP, ENOENT, ENOTDIR, EROFS

**NAME**

mknod – make a directory or a special file

**SYNOPSIS**

```
int mknod(name, mode, addr)
char *name;
```

**DESCRIPTION**

*Mknod* creates a new file whose name is the null-terminated string pointed to by *name*. The mode of the new file (including directory and special file bits) is initialized from *mode*. (The protection part of the mode is modified by the process's mode mask; see [stat\(2\)](#) and [umask\(2\)](#)). The first block pointer of the inode is initialized from *addr*. For ordinary files and directories *addr* is normally zero. For a special file, *addr* is the device number; see [mknod\(8\)](#) and the writeups in section 4.

*Mknod* may be invoked only by the super-user.

**SEE ALSO**

[open\(2\)](#) for [creat](#), [mkdir\(2\)](#), [stat\(2\)](#), [umask\(2\)](#), [filsys\(5\)](#), [mknod\(8\)](#)

**DIAGNOSTICS**

EEXIST, EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EPERM, EROFS

**NAME**

`mpx` – create and manipulate multiplexed files

**SYNOPSIS**

```
mpx(name, access)
char *name;
join(fd, xd)
chan(xd)
extract(i, xd)
attach(i, xd)
detach(i, xd)
connect(fd, cd, end)
npgrp(i, xd, pgrp)
ckill(i, xd, signal)
#include <sys/mx.h>
mpxcall(cmd, vec)
int *vec;
```

**DESCRIPTION**

`mpxcall(cmd, vec)` is the system call shared by the library routines described below. *Cmd* selects a command using values defined in `<sys/mx.h>`. *Vec* is the address of a structure containing the arguments for the command.

**mpx(name, access)**

*Mpx* creates and opens the file *name* with access permission *access* (see `creat(2)`) and returns a file descriptor available for reading and writing. A `-1` is returned if the file cannot be created, if *name* already exists, or if the file table or other operating system data structures are full. The file descriptor is required for use with other routines.

If *name* is 0, a file descriptor is returned as described but no entry is created in the file system.

Once created an `mpx` file may be opened (see `open(2)`) by any process. This provides a form of inter-process communication whereby a process B can ‘call’ process A by opening an `mpx` file created by A. To B, the file is ordinary with one exception: the `connect` primitive could be applied to it. Otherwise the functions described below are used only in process A and descendants that inherit the open `mpx` file.

When a process opens an `mpx` file, the owner of the file receives a control message when the file is next read. The method for ‘answering’ this kind of call involves using `attach` and `detach` as described in more detail below.

Once B has opened A’s `mpx` file it is said to have a *channel* to A. A channel is a pair of data streams: in this case, one from B to A and the other from A to B. Several processes may open the same `mpx` file yielding multiple channels within the one `mpx` file. By accessing the appropriate channel, A can communicate with B and any others. When A reads (see `read(2)`) from the `mpx` file data written to A by the other processes appears in A’s buffer using a record format described in `mpxio(5)`. When A writes (see `write(2)`) on its `mpx` file the data must be formatted in a similar way.

The following commands are used to manipulate `mpx` files and channels.

*join*- adds a new channel on an `mpx` file to an open file F. I/O on the new channel is I/O on F.

*chan*- creates a new channel.

*extract*- file descriptor maintenance.

*connect*- similar to `join` except that the open file F is connected to an existing channel.

*attach* and *detach*- used with call protocol.

*npgrp*- manipulates process group numbers so that a channel can act as a control terminal (see `tty(4)`).

*ckill*- send signal (see `signal(2)`) to process group through channel.

A maximum of 15 channels may be connected to an `mpx` file. They are numbered 0 through 14. *Join*

may be used to make one mpx file appear as a channel on another mpx file. A hierarchy or tree of mpx files may be set up in this way. In this case one of the mpx files must be the root of a tree where the other mpx files are interior nodes. The maximum depth of such a tree is 4.

An *index* is a 16-bit value that denotes a location in an mpx tree other than the root: the path through mpx ‘nodes’ from the root to the location is expressed as a sequence of 4-bit nibbles. The branch taken at the root is represented by the low-order 4-bits of an index. Each succeeding branch is specified by the next higher-order nibble. If the length of a path to be expressed is less than 4, then the illegal channel number, 15, must be used to terminate the sequence. This is not strictly necessary for the simple case of a tree consisting of only a root node: its channels can be expressed by the numbers 0 through 14. An index *i* and file descriptor *xd* for the root of an mpx tree are required as arguments to most of the commands described below. Indices also serve as channel identifiers in the record formats given in *mpxio(5)*. Since  $-1$  is not a valid index, it can be returned as a error indication by subroutines that normally return indices.

The operating system informs the process managing an mpx file of changes in the status of channels attached to the file by generating messages that are read along with data from the channels. The form and content of these messages is described in *mpxio(5)*.

**join(*fd*, *xd*)** establishes a connection (channel) between an mpx file and another object. *Fd* is an open file descriptor for a character device or an mpx file and *xd* is the file descriptor of an mpx file. *Join* returns the index for the new channel if the operation succeeds and  $-1$  if it does not.

Following *join*, *fd* may still be used in any system call that would have been meaningful before the join operation. Thus a process can read and write directly to *fd* as well as access it via *xd*. If the number of channels required for a tree of mpx files exceeds the number of open files permitted a process by the operating system, some of the file descriptors can be released using the standard *close(2)* call. Following a close on an active file descriptor for a channel or internal mpx node, that object may still be accessed through the root of the tree.

**chan(*xd*)** allocates a channel and connects one end of it to the mpx file represented by file descriptor *xd*. *Chan* returns the index of the new channel or a  $-1$  indicating failure. The *extract* primitive can be used to get a non-multiplexed file descriptor for the free end of a channel created by *chan*.

Both *chan* and *join* operate on the mpx file specified by *xd*. File descriptors for interior nodes of an mpx tree must be preserved or reconstructed with *extract* for use with *join* or *chan*. For the remaining commands described here, *xd* denotes the file descriptor for the root of an mpx tree.

**extract(*i*, *xd*)** returns a file descriptor for the object with index *i* on the mpx tree with root file descriptor *xd*. A  $-1$  is returned by *extract* if a file descriptor is not available or if the arguments do not refer to an existing channel and mpx file.

**attach(*i*, *xd*)**

**detach(*i*, *xd*)**. If a process A has created an mpx file represented by file descriptor *xd*, then a process B can open (see *open(2)*) the mpx file. The purpose is to establish a channel between A and B through the mpx file. *Attach* and *Detach* are used by A to respond to such opens.

An open request by B fails immediately if a new channel cannot be allocated on the mpx file, if the mpx file does not exist, or if it does exist but there is no process (A) with a multiplexed file descriptor for the mpx file (i.e. *xd* as returned by *mpx(2)*). Otherwise a channel with index number *i* is allocated. The next time A reads on file descriptor *xd*, the WATCH control message (see *mpxio(5)*) will be delivered on channel *i*. A responds to this message with *attach* or *detach*. The former causes the open to complete and return a file descriptor to B. The latter deallocates channel *i* and causes the open to fail.

One mpx file may be placed in ‘listener’ mode. This is done by writing *ioctl(xd, MXLSTN, 0)* where *xd* is an mpx file descriptor and MXLSTN is defined in */usr/include/sgtty.h*. The semantics of listener mode are that all file names discovered by *open(2)* to have the syntax *system!pathname* (see *uucp(1)*) are treated as opens on the mpx file. The operating system sends the listener process an OPEN message (see *mpxio(5)*) which includes the file name being opened. *Attach* and *detach* then apply as described above.

*Detach* has two other uses: it closes and releases the resources of any active channel it is applied to, and should be used to respond to a CLOSE message (see *mpxio(5)*) on a channel so the channel may be reused.

**connect(*fd*, *cd*, *end*)**. *Fd* is a character file descriptor and *cd* is a file descriptor for a channel, such as

might be obtained via *extract( chan(xd), xd)* or by *open(2)* followed by *attach*. *Connect* splices the two streams together. If *end* is negative, only the output of *fd* is spliced to the input of *cd*. If *end* is positive, the output of *cd* is spliced to the input of *fd*. If *end* is zero, then both splices are made.

**npgrp(i, xd, pgrp)**. If *xd* is negative *npgrp* applies to the process executing it, otherwise *i* and *xd* are interpreted as a channel index and mpx file descriptor and *npgrp* is applied to the process on the non-multiplexed end of the channel. If *pgrp* is zero, the process group number of the indicated process is set to the process number of that process, otherwise the value of *pgrp* is used as the process group number.

*Npgrp* normally returns the new process group number. If *i* and *xd* specify a nonexistent channel, *npgrp* returns  $-1$ .

**ckill(i, xd, signal)** sends the specified signal (see *signal(2)*) through the channel specified by *i* and *xd*. If the channel is connected to anything other than a process, *ckill* is a null operation. If there is a process at the other end of the channel, the process group will be interrupted (see *signal(2)*, *kill(2)*). *Ckill* normally returns *signal*. If *ch* and *xd* specify a nonexistent channel, *ckill* returns  $-1$ .

## FILES

/usr/include/sys/mx.h

/usr/include/sgtty.h

## SEE ALSO

*mpxio(5)*

## BUGS

Mpx files are an experimental part of the operating system more subject to change and prone to bugs than other parts.

Maintenance programs, e.g. *icheck(1)*, diagnose mpx files as an illegal mode.

Channels may only be connected to objects in the operating system that are accessible through the line discipline mechanism.

Higher performance line disciplines are needed.

The maximum tree depth restriction is not really checked.

A non-destructive *disconnect* primitive (inverse of *connect*) is not provided.

A non-blocking flow control strategy based on messages defined in *mpxio(5)* should not be attempted by novices; the enabling *ioctl* command should be protected.

The *join* operation could be subsumed by *connect*. A mechanism is needed for moving a channel from one location in an mpx tree to another.

**NAME**

nice – set program priority

**SYNOPSIS**

**void nice(incr)**

**DESCRIPTION**

The scheduling priority of the process is augmented by *incr*. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without flak from the administration. Priority 19 is recommended for programs that should only execute in “idle” time.

Only the super-user can effect negative increments. Priorities less than -20 (most urgent) are treated as -20. Priorities greater than 19 are treated as 19.

The priority of a process is passed to a child process by *fork(2)*. For a privileged process to return to normal priority from an unknown state, *nice* should be called successively with arguments -40 (goes to priority -20 because of truncation), then 20 (to get to 0).

**SEE ALSO**

*nice(1)*, *fork(2)*

**NAME**

open, creat, close – open a file for reading or writing, create file

**SYNOPSIS**

**int open(file, rwmode)**

**char \*file;**

**int creat(file, mode)**

**char \*file;**

**int close(fildes)**

**DESCRIPTION**

*Open* opens the *file*, for reading if *rwmode* is 0, for writing if *rwmode* is 1, or for both if *rwmode* is 2, and returns an associated file descriptor. *File* is a null-terminated string representing a path name. The file pointer is set to 0.

*Creat* creates a new *file* or prepares to rewrite an existing *file*, opens it for writing, and returns an associated file descriptor. If the file is new, the owner is set to the effective userid of the creating process; the group to that of the containing directory; the mode to *mode* as modified by the mode mask of the creating process; see [umask\(2\)](#). *Mode* need not allow writing. If the file already exists, it is truncated to 0 length; the mode, owner, and group remain unchanged, and must permit writing.

A program may reserve a filename for exclusive use by calling *creat* with a mode that forbids writing. If the file does not exist, *creat* will succeed; further attempts to *creat* the same file will be denied. More sophisticated (but less portable) concurrent access control may be obtained by setting the **S\_ICCTYP** field in the mode; see [stat\(2\)](#).

*Close* closes the file associated with a file descriptor. Files are closed upon termination of a process, but since there is a limit on the number of open files per process, *close* is necessary for programs which deal with many files. It is possible to arrange for files to be closed by [exec\(2\)](#); see **FIOCLEX** in [ioctl\(2\)](#).

**SEE ALSO**

[dup\(2\)](#), [pipe\(2\)](#), [read\(2\)](#), [exec\(2\)](#), [ioctl\(2\)](#), [stat\(2\)](#), [unlink\(2\)](#)

**DIAGNOSTICS**

*open, creat*: **EACCES**, **EBUSY**, **ECONC**, **EFAULT**, **EINTR**, **EIO**, **EISDIR**, **ELOOP**, **EMFILE**, **ENFILE**, **ENOENT**, **ENOTDIR**, **EROFS**, **ETXTBSY**

*creat*: **ENOSPC**

*close*: **EBADF**

**BUGS**

It should be possible to call *open* without waiting for carrier on communication lines.

The group of a newly-created file should (once again) be the effective groupid of the creating process.

The trick of creating a file with an unwritable mode fails for the super-user.

**NAME**

pipe – create an interprocess channel

**SYNOPSIS**

```
int pipe(fildes)
int fildes[2];
```

**DESCRIPTION**

*Pipe* creates a buffered channel for interprocess I/O communication. Two file descriptors returned in *fildes* are the ends of pair of cross-connected streams; see [stream\(4\)](#). Data written via **fildes[1]** is available for reading via **fildes[0]** and vice versa.

After the pipe has been set up, cooperating processes created by subsequent [fork\(2\)](#) calls may pass data through the pipe with *read* and *write* calls. The bytes placed on a pipe by one *write* are contiguous even if many process are writing. *Writes* induce a record structure: a *read* will not return bytes from more than one *write*; see [read\(2\)](#).

Write calls on a one-ended pipe raise signal **SIGPIPE**. Read calls on a one-ended pipe with no data in it return an end-of-file for the first several attempts, then raise **SIGPIPE**, and eventually **SIGKILL**.

**SEE ALSO**

[sh\(1\)](#), [fork\(2\)](#), [read\(2\)](#), [select\(2\)](#), [stream\(4\)](#)

**DIAGNOSTICS**

**EIO**, **EMFILE**, **ENFILE**, **ENXIO**

**BUGS**

Buffering in pipes connecting multiple processes may cause deadlocks.

Some line discipline modules discard the record delimiters inserted by *write*.

On many other versions of the system, only **fildes[0]** may be read and only **fildes[1]** may be written.



**NAME**

profil – execution time profile

**SYNOPSIS**

```
void profil(buff, bufsiz, offset, scale)
unsigned short *buff;
int bufsiz, offset;
unsigned scale;
```

**DESCRIPTION**

*Buff* points to an area of core whose length in bytes is given by *bufsiz*. After this call, the user's program counter is examined each clock tick; *offset* is subtracted from it, and the result multiplied by *scale*, divided by 65536 and then rounded up to a multiple of two. If the resulting number (*n*) is less than *bufsiz*, then **buff[n/2]** is incremented.

Profiling is turned off by giving a *scale* of 0. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an *exec* is executed, but remains on in child and parent both after a *fork*. Profiling is turned off if an update in *buff* would cause a memory fault.

**SEE ALSO**

[prof\(1\)](#), [monitor\(3\)](#)

**BUGS**

Because of the rounding up, single byte instructions cannot be exactly profiled.

**NAME**

read, write – read or write file

**SYNOPSIS**

```
int read(fildes, buffer, nbytes)
char *buffer;
```

```
int write(fildes, buffer, nbytes)
char *buffer;
```

**DESCRIPTION**

*Read* reads *nbytes* bytes of data from the file pointer location in the file associated with *fildes* into memory at *buffer*. The file pointer is advanced by the number of bytes read. It is not guaranteed that all *nbytes* bytes will be read; for example if the file refers to a terminal at most one line will be returned. In any event the number of characters read is returned. A return value of 0 is conventionally interpreted as end of file.

*Write* writes *nbytes* bytes of data starting at *buffer* to the file associated with *fildes* at the file pointer location. The file pointer is advanced by the number of bytes written. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Reads and writes which are aligned with file system blocks are more efficient than others; see [filsys\(5\)](#).

**SEE ALSO**

[open\(2\)](#), [dup\(2\)](#), [pipe\(2\)](#), [select\(2\)](#), [dirread\(2\)](#)

**DIAGNOSTICS**

*read*: **EBADF**, **EFAULT**, **EINTR**, **EINVAL**, **ENXIO**

*write*: **EBADF**, **EFAULT**, **EINTR**, **EINVAL**, **EIO**, **ENXIO**, **EPIPE**, **EROFS**

**BUGS**

A *read* or a *write* call may fail because of a prior call to [lseek\(2\)](#).

**NAME**

select – synchronous input/output multiplexing

**SYNOPSIS**

```
#include <sys/types.h>

int select(nfds, readfds, writefds, milli);
fd_set *readfds, *writefds;
```

**DESCRIPTION**

*Select* examines a set of file descriptors to see if they will block if read or written. *Readfds* points to an object of type **fd\_set**, which contains a set of descriptors to be checked for reading; *writefds* similarly for writing. Only descriptors 0 through *nfds*-1 are considered. The number of ready descriptors is returned, and the **fd\_set** pointed to by *readfds* (*writefds*) is overwritten with a set of descriptors ready to be read (written). The call waits until at least one descriptor is ready, or until *milli* milliseconds have passed.

Either *readfds* or *writefds* may be 0 if no descriptors are interesting.

The **fd\_set** type looks like

```
typedef struct {
    unsigned int fds_bits[FDWORDS];
} fd_set;
```

**FDWORDS** is sufficient to contain as many file descriptors as the system will allow (currently 128). If there are *B* bits in an **unsigned int**, file descriptor *n* is represented by  $1 \ll ((n \% B) - 1)$  in word **fds\_bits**[*n/B*].

These macros should be used to manipulate the contents of an **fd\_set**:

**FD\_ZERO(s)**

clear all bits in set *s*

**FD\_SET(n, s)**

set bit for file descriptor *n* in set *s*

**FD\_CLR(n, s)**

clear bit for file descriptor *n* in *s*

**FD\_ISSET(n, s)**

return a value of 1 if bit for file descriptor *n* is set in *s*, 0 otherwise

**EXAMPLES**

```
int p[2];
fd_set wfs;
pipe(p);
do {
    FD_SET(p[1], wfs);
    write(p[1], ".", 1);
    i++;
} while(select(p[1]+1, (fd_set*)0, wfs, 0) == 1);
printf("Pipe capacity = %d\n", i);
```

**SEE ALSO**

[read\(2\)](#)

**DIAGNOSTICS**

**EBADE, EFAULT, EINTR**

**BUGS**

*Milli* is rounded up to the nearest second.

*Select* is intended for use with streams; file descriptors referring to ordinary files or to non-stream special files always appear ready. This is a lie for some special files.

**NAME**

signal, kill – receive and send signals

**SYNOPSIS**

```
#include <signal.h>
```

```
SIG_TYP signal(sig, func)
```

```
SIG_TYP func;
```

```
int kill(pid, sig)
```

**DESCRIPTION**

A signal is generated by some abnormal event initiated by a user at a terminal (quit, interrupt), by a program error (bus error, etc.), or by *kill* in another process. Normally, most signals cause termination of the receiving process, but *signal* allows them either to be ignored or to be caught by interrupting to a specified function. The following signal names are defined in

[CB]SIGHUP	1hangup
[CB]SIGINT	2 interrupt
[CB]SIGQUIT	3*quit
[CB]SIGILL	4* illegal instruction (not reset when caught)
[CB]SIGTRAP	5*trace trap (not reset when caught)
[CB]SIGIOT	6* IOT instruction
[CB]SIGEMT	7*EMT instruction
[CB]SIGFPE	8* floating point exception
[CB]SIGKILL	9kill (cannot be caught or ignored)
[CB]SIGBUS	10*bus error
[CB]SIGSEGV	11*segmentation violation
[CB]SIGSYS	12*bad argument to system call
[CB]SIGPIPE	13write on a pipe with no one to read it
[CB]SIGALRM	14alarm clock
[CB]SIGTERM	15software termination signal
	16 unassigned
[CB]SIGSTOP	17+stop (cannot be caught or ignored)
[CB]SIGCONT	19#continue a stopped process
[CB]SIGCHLD	20#child has stopped or exited

- \* places core image in file **core** if not caught or ignored
- + suspends process until **SIGCONT** or **PIOCRUN**; see [proc\(4\)](#)
- # ignored if not caught

Signals 1 through **NSIG-1**, defined in the include file, exist. Those not listed above have no conventional meaning in this system. (Berkeley systems use 1-15 and 17-25.)

*Signal* specifies how signal *sig* will be handled. If *func* is **SIG\_DFL**, the default action listed above is reinstated. If *func* is **SIG\_IGN**, the signal will be ignored. Otherwise, when the signal occurs, it will be caught and a function, pointed to by *func*, will be called. The type of pointer *func* is **SIG\_TYP**:

```
typedef int (*SIG_TYP)();
```

It must point to a function such as,

```
int catcher(sig) { ... }
```

which will be called with a signal number as argument. A return from the catcher function will continue the process at the point it was interrupted.

Except as indicated, a signal is reset to **SIG\_DFL** after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during [read\(2\)](#) or *write* on a slow device (like a typewriter, but not a disk), and during *pause* and *wait*; see [alarm\(2\)](#) and [exit\(2\)](#). The interrupted system call will return error **EINTR**. The user's program may then, if it wishes, re-execute the call.

*Signal* returns the previous (or initial) value of *func* for the particular signal.

After a *fork(2)* the child inherits all signal settings. *Exec(2)* resets all caught signals to default action.

*Kill* sends signal *sig* to the process specified by process id *pid*. Signal 0 has no effect on the target process and may be used to test the existence of a process. The success of sending a signal is independent of how the receiving process treats the signal.

The effective userid of the sending process must be either 0 or the effective userid of the receiving process.

If *pid* is 0, the signal is sent to all other processes in the sender's process group; see *stream(4)*.

If *pid* is -1, and the user is the super-user, the signal is broadcast universally except to processes 0 (scheduler), 1 (initialization) and 2 (pageout); see *init(8)*. If *pid* is less than -1, it is negated and taken as a process group whose members should receive the signal.

Processes may send signals to themselves.

## FILES

*core*

## SEE ALSO

*kill(1)*, *setjmp(3)*, *stream(4)*

## DIAGNOSTICS

*signal*: **EINVAL**

*kill*: **EINVAL**, **EPERM**, **ESRCH**

## BUGS

The reason for a trap should be distinguishable by extra arguments to the signal handler.

If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

For historical reasons, the return value of a catcher function is **int**; it is **void** in ANSI standard C.

**NAME**

stat, lstat, fstat – get file status

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int stat(name, buf)
```

```
char *name;
```

```
struct stat *buf;
```

```
int lstat(name, buf)
```

```
char *name;
```

```
struct stat *buf;
```

```
int fstat(fildes, buf)
```

```
struct stat *buf;
```

**DESCRIPTION**

*Stat* puts detailed information about the file *name* in a structure whose address is *buf*. *Lstat* does the same except that when *name* is a symbolic link (see [link\(2\)](#)), it supplies information about the link itself. *Fstat* does what *stat* does for the file open on descriptor *fildes*.

It is unnecessary to have any permissions at all with respect to *name*, but all directories leading to the file must be searchable.

[CB]struct stat

```
{
    [CB]dev_t st_dev;           device number for this file system
    [CB]ino_t st_ino;          inode number
    [CB]unsigned short st_mode; file mode encoded as below
    [CB]short st_nlink;        number of links (not symbolic links)
    [CB]short st_uid;          uid of owner
    [CB]short st_gid;          gid of owner
    [CB]dev_t st_rdev;         if device file, the device number
    [CB]off_t st_size;         size in bytes
    [CB]time_t st_atime;       time file was last read or created
    [CB]time_t st_mtime;       time file was last written or created
    [CB]time_t st_ctime;       time file or inode was last written or created
};
```

The bits in **st\_mode** are defined by

```
[CB]S_IFMT          0170000file type
[CB]S_IFDIR         0040000directory
[CB]S_IFCHR         0020000character device
[CB]S_IFBLK        0060000block device
[CB]S_IFREG         0100000regular file
[CB]S_IFLNK        0120000symbolic link
[CB]S_ISUID         0004000set userid on execution
[CB]S_ISGID         0002000set groupid on execution
[CB]S_ICCTYP        0007000type of concurrency control
[CB]S_ISYNC         00010001 writer and n readers (synchronized access)
[CB]S_IEXCL         00030001 writer or n readers (exclusive access)
[CB] 0000400 read permission by owner
[CB] 0000200 write permission by owner
[CB] 0000100 execute permission (search on directory) by owner
[CB] 0000070 read, write, execute (search) by group
[CB] 0000007 read, write, execute (search) by others
```

**S\_IFMT** and **S\_ICCTYP** are field masks; the other constants encode modes. Codes contained in the **S\_IFMT** field are mutually exclusive. If bit 01000 is set, the concurrency modes contained in **S\_ICCTYP** are in effect; otherwise the set-id flags **S\_ISUID** and **S\_ISGID** apply.

The concurrency modes affect *open* and *creat* calls. Synchronized access, **S\_ISYNC**, guards against inconsistent updates by forbidding concurrent opens for writing. Exclusive access, **S\_IEXCL**, guards against inconsistent views by forbidding concurrent opens if one is for writing.

**SEE ALSO**

*chmod(1)*, *ls(1)*, *chmod(2)*, *filesystem(5)*

**DIAGNOSTICS**

*stat*, *lstat*: **EACCES**, **EFAULT**, **EIO**, **ELOOP**, **ENOENT**, **ENOTDIR**

*fstat*: **EBADF**, **EFAULT**, **EIO**

**BUGS**

For efficiency, **st\_atime** is not set when a directory is searched, although this might be more logical.

**NAME**

stime, biasclock – set time

**SYNOPSIS**

**int stime(tp)**

**long \*tp;**

**biasclock(milli)**

**long milli;**

**DESCRIPTION**

*Stime* sets the system's idea of the time and date. Time, pointed to by *tp*, is measured in seconds from 00:00:00 GMT Jan 1, 1970.

*Biasclock* informs the system that its idea of the time should be incremented by *milli* milliseconds. The system will make the adjustment gradually and without causing time to run backwards.

Only the super-user may use these calls.

**SEE ALSO**

[date\(1\)](#), [www\(1\)](#), [time\(2\)](#), [ctime\(3\)](#)

**DIAGNOSTICS**

**EPERM**



**NAME**

sync – force writing of system buffers

**SYNOPSIS**

**void sync()**

**DESCRIPTION**

*Sync* schedules all information in memory that should be on disk or other block media to be written out. This includes modified super-blocks, modified inodes, and delayed block I/O.

It should be used by programs which examine a file system, for example [icheck\(8\)](#). It is highly recommended before a shutdown or reboot.

**SEE ALSO**

[sync\(8\)](#)

**BUGS**

The writing, although scheduled, is not necessarily complete upon return from *sync*.

**NAME**

syscall – indirect system call

**SYNOPSIS**

**int** syscall(**number**, **arg**, ... )

**DESCRIPTION**

*Syscall* performs the system call with the specified *number* and arguments and returns its result. The numbers may be found in the system source.

**BUGS**

The simulation fails for system calls such as *pipe(2)*, which return multiple values.

**NAME**

time, ftime – get date and time

**SYNOPSIS**

```
long time((long *)0)
```

```
long time(tloc)
```

```
long *tloc;
```

```
#include <sys/types.h>
```

```
#include <sys/timeb.h>
```

```
ftime(tp)
```

```
struct timeb *tp;
```

**DESCRIPTION**

*Time* returns the time since the epoch 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If *tloc* is nonnull, the return value is also stored in the place to which *tloc* points.

*Ftime* stores a more accurate time and other horological data in the structure pointed to by *tb*:

```
[CB]struct timeb
```

```
{
```

[CB]time_t time;	time since the epoch in seconds
[CB]unsigned short millitm;	up to 1000 milliseconds of more-precise interval
[CB]short timezone;	local time zone measured in minutes of time westward from Greenwich
[CB]short dstflag;	if nonzero, daylight saving time applies locally during the appropriate part of the year

```
[CB]};
```

**SEE ALSO**

[date\(1\)](#), [stime\(2\)](#), [ctime\(3\)](#)

**DIAGNOSTICS**

*ftime*: **EFAULT**

**BUGS**

If the argument to *time* is bogus, the user program gets a memory fault rather than an **EFAULT**.

**NAME**

times – get process times

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <sys/times.h>
```

```
int times(buffer)
```

```
struct tms *buffer;
```

**DESCRIPTION**

*Times* delivers time-accounting information for the current process and for the terminated child processes of the current process. All times are in clock ticks.

```
[CB]struct tms
```

```
{
```

```
    [CB]time_t tms_utime;    user time for this process
```

```
    [CB]time_t tms_stime;    system time for this process
```

```
    [CB]time_t tms_cutime;    user time for all child processes
```

```
    [CB]time_t tms_cstime;    system time for all child processes
```

```
[CB]};
```

The children times are the sum of the children's process times and their children's times.

**SEE ALSO**

*time(1)*, *time(2)*

**DIAGNOSTICS**

**EFAULT**

**NAME**

umask – set file creation mode mask

**SYNOPSIS**

**int umask(complmode)**

**DESCRIPTION**

*Umask* sets the process mode mask. The mask modifies the *mode* argument of *creat* (see [open\(2\)](#)), [mkdir\(2\)](#), and [mknod\(2\)](#) thus:

```
mode &= (07777 & (complmode & 0777))
```

In other words, the mask specifies permission bits to be turned off when files are created.

The previous value of the mask is returned by the call. The initial value is set by [login\(8\)](#), and may be modified by the *umask* command of [sh\(1\)](#). The mask is inherited by child processes.

**SEE ALSO**

[open\(2\)](#), [mkdir\(2\)](#), [mknod\(2\)](#), [stat\(2\)](#)

**NAME**

unlink – remove directory entry

**SYNOPSIS**

```
int unlink(name)
char *name;
```

**DESCRIPTION**

*Unlink* removes the entry for the file pointed to by *name* from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

Only the super-user can *unlink* a directory, but see *rmdir* in [mkdir\(2\)](#).

**SEE ALSO**

[rm\(1\)](#), [link\(2\)](#), [mkdir\(2\)](#)

**DIAGNOSTICS**

EFAULT, EIO, ELOOP, ENOENT, ENOTDIR, EROFS

**NAME**

`vtimes` – get usage of time, space, and paging resources

**SYNOPSIS**

```
#include <sys/vtimes.h>

vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;
```

**DESCRIPTION**

`Vtimes` places accounting information for the current process in the area pointed to by `par_vm` and for its terminated children in the area pointed to by `ch_vm`. If either pointer is 0, the corresponding information is omitted.

After the call, each area contains information in the form

```
struct vtimes {
    int      vm_untime; /* user time */
    int      vm_stime; /* system time */
                                /* rss = resident storage size in 512-byte pages */
    unsigned vm_idrss; /* rss time integral, data+stack */
    unsigned vm_ixrss; /* rss time integral, text */
    int      vm_maxrss; /* maximum rss */
    int      vm_majflt; /* major page faults */
    int      vm_minflt; /* minor page faults */
    int      vm_nswap; /* number of swaps */
    int      vm_inblk; /* block reads */
    int      vm_oublk; /* block writes */
};
```

Times are expressed in clock ticks of 1/60 (or 1/50) second. The time integrals are computed by cumulating the number of 512-byte pages in use at each clock tick.

A major page fault involves a disk transfer; a minor fault gathers page-reference information. Block reads and writes are file system disk transfers; blocks found in the buffer pool are not counted.

**SEE ALSO**

[time\(2\)](#), [exit\(2\)](#)

**NAME**

killpg – send signal to a process or a process group

**SYNOPSIS**

**killpg(pgrp, sig)**

**cc ... -ljobs**

**DESCRIPTION**

*Killpg* sends the signal *sig* to the specified process group. See *sigsys(2)* for a list of signals; see *jobs(3)* for an explanation of process groups.

The sending process and members of the process group must have the same effective user ID, otherwise this call is restricted to the super-user. As a single special case the continue signal SIGCONT may be sent to any process which is a descendant of the current process. This allows a command interpreter such as *csh(1)* to restart set-user-id processes stopped from the keyboard with a stop signal.

The calls

**killpg(0, sig)**

and

**kill(0, sig)**

have identical effects, sending the signal to all members of the invoker's process group (including the process itself). It is preferable to use the call involving *kill* in this case, as it is portable to other UNIX systems.

**SEE ALSO**

*jobs(3)*, *kill(2)*, *sigsys(2)*, *signal(2)*, *csh(1)*, *kill(1)*

**DIAGNOSTICS**

Zero is returned if the processes are sent the signals; -1 is returned if any process in the process group cannot be sent the signal, or if there are no members in the process group.

**BUGS**

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of this system call and even the call itself are thus subject to change.



**NAME**

setpgrp, getpgrp – set/get process group

**SYNOPSIS**

**int** getpgrp(**pid**)

**setpgrp**(**pid**, **pgrp**)

**cc ... -ljobs**

**DESCRIPTION**

The process group of the specified process is returned by *getpgrp*. *Setpgrp* sets the process group of the specified process *pid* to the specified *pgrp*. If *pid* is zero, then the call applies to the current process.

If the invoker is not the super-user, then the affected process must have the same effective user-id as the invoker or be a descendant of the invoking process.

This call is used by *cs(1)* to create process groups in implementing job control. The TIOCGPGRP and TIOCSPGRP calls described in *tty(4)* are used to get/set the process group of the control terminal.

See *jobs(3)* for a general discussion of job control.

**SEE ALSO**

*jobs(3)*, *getuid(2)*, *tty(4)*

**BUGS**

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of these system calls and even the calls themselves are thus subject to change.

A system call *setpgrp* has been implemented in other versions of UNIX which are not widely used outside of Bell Laboratories; these implementations have, in general, slightly different semantics.

**NAME**

sigsys – catch or ignore signals

**SYNOPSIS**

```
#include <signal.h>
```

```
(*sigsys(sig, func))()
```

```
void (*func)();
```

```
cc ... -ljobs
```

**DESCRIPTION**

*N.B.:* The system currently supports two signal implementations. The one described in [signal\(2\)](#) is standard in version 7 UNIX systems, and retained for backward compatibility as it is different in a number of ways. The one described here (with the interface in [sigset\(3\)](#)) provides for the needs of the job control mechanisms (see [jobs\(3\)](#)) used by [csh\(1\)](#), and corrects the bugs in the standard implementation of signals, allowing programs which process interrupts to be written reliably.

The routine *sigsys* is not normally called directly; rather the routines of [sigset\(3\)](#) should be used. These routines are kept in the “jobs” library, accessible by giving the loader option **-ljobs**. The features described here are less portable than those of [signal\(2\)](#) and should not be used in programs which are to be moved to other versions of UNIX.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see [tty\(4\)](#)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals which cannot be blocked, the *sigsys* call allows signals either to be ignored, held until a later time (protecting critical sections in the process), or to cause an interrupt to a specified location. Here is the list of all signals with names as in the include file.

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught, held or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
	16	unassigned
SIGSTOP	17†	stop (cannot be caught, held or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19•	continue after stop
SIGCHLD	20•	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGTINT	23•	input record is available at control terminal
SIGXCPU	24	cpu time limit exceeded (see <a href="#">vlimit(2)</a> )
SIGXFSZ	25	file size limit exceeded (see <a href="#">vlimit(2)</a> )

The starred signals in the list above cause a core image if not caught, held or ignored.

If *func* is SIG\_DFL, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or †. Signals marked with • are discarded if the action is SIG\_DFL; signals marked with † cause the process to stop. If *func* is SIG\_HOLD the signal is remembered if it occurs, but not presented to the process; it may be presented later if the process changes the action for the signal. If *func* is SIG\_IGN the signal is subsequently ignored, and pending instances of the signal are discarded (i.e. if the action was previously SIG\_HOLD.) Otherwise when the signal occurs *func* will be called.

A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal, set with *sigsys*, is reset to SIG\_DFL after being caught. However by specifying DEFERSIG(*func*) as the last argument to *sigsys*, one causes the action to be set to SIG\_HOLD before the interrupt is taken, so that recursive instances of the signal cannot occur during handling of the signal.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a *read* or *write(2)* on a slow device (like a terminal; but not a file) and during a *pause* or *wait(2)*. When a signal occurs during one of these calls, the saved user status is arranged in such a way that, when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call. *Read* and *write* calls which have done no I/O, *ioctl*s blocked with SIGTTOU, and *wait3* calls are restarted.

The value of *sigsys* is the previous (or initial) value of *func* for the particular signal.

The system provides two other functions by oring bits into the signal number: SIGDOPAUSE causes the process to *pause* after changing the signal action. It can be used to atomically re-enable a held signal which was being processed and wait for another instance of the signal. SIGDORTI causes the system to simulate an *rei* instruction clearing the mark the system placed on the stack at the point of interrupt before checking for further signals to be presented due to the specified change in signal actions. This allows a signal package such as *sigset(3)* to dismiss from interrupts cleanly removing the old state from the stack before another instance of the interrupt is presented.

After a *fork(2)* or *vfork(2)* the child inherits all signals. *Exec(2)* resets all caught signals to default action; held signals remain held and ignored signals remain ignored.

## SEE ALSO

kill(1), ptrace(2), kill(2), jobs(3), sigset(3), setjmp(3), tty(4)

## DIAGNOSTICS

The value BADSIG is returned if the given signal is out of range.

## BUGS

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of this facility and the system calls supporting it are thus subject to change.

Since only one signal action can be changed at a time, it is not possible to get the effect of SIGDOPAUSE for more than one signal at a time.

The traps (listed below) should be distinguishable by extra arguments to the signal handler, and all hardware supplied parameters should be made available to the signal routine.

## ASSEMBLER (PDP-11)

(signal = 48.)

**sys signal; sig; label**

(old label in r0)

If *label* is 0, default action is reinstated. If *label* is 1, the signal is ignored. If *label* is 3, the signal is held. Any other even *label* specifies an address in the process where an interrupt is simulated. If *label* is otherwise odd, the signal is sent to the function whose address is the label with the low bit cleared with the action set to SIG\_HOLD. (Thus DEFERSIG is indicated by the low bit of a signal catch address. An RTI or RTT instruction will return from the interrupt.)

## NOTES (VAX-11)

The following defines the mapping of hardware traps to signals:

Arithmetic traps:	
Integer overflow	SIGFPE
Integer division by zero	SIGFPE
Floating overflow	SIGFPE
Floating underflow	SIGFPE
Floating/decimal division by zero	SIGFPE
Decimal overflow	SIGFPE
Subscript-range	SIGFPE
Length access control	SIGSEGV
Protection violation	SIGBUS
Reserved instruction	SIGILL
Customer-reserved instr.	SIGEMT
Reserved operand	SIGILL
Reserved addressing	SIGILL
Trace pending	SIGTRAP
Bpt instruction	SIGTRAP
Compatibility-mode	SIGEMT
Chme	SIGILL
Chms	SIGILL
Chmu	SIGILL

**NAME**

wait3 – wait for process to terminate

**SYNOPSIS**

```
#include <wait.h>
#include <sys/vtimes.h>

wait3(status, options, vtimep)
union wait status;
int options;
struct vtimes *vtimep;

cc ... -ljobs
```

**DESCRIPTION**

The *status* and *option* words are described by definitions and macros in the file <wait.h>; the union and its bitfield definitions and associated macros given there provide convenient and mnemonic access to the word of status returned by a *wait3* call. See this file for more information.

There are two *options*, which may be combined by *oring* them together. The first is WNOHANG which causes the *wait3* to not hang if there are no processes which wish to report status, rather returning a pid of 0 in this case as the result of the *wait3*. The second option is WUNTRACED which causes *wait3* to return information when children of the current process which are stopped but not traced (with *ptrace(2)*) because they received a SIGTTIN, SIGTTOU, SIGTSTP or SIGSTOP signal. See *sigsys(2)* for a description of these signals.

The *vtimep* pointer is an optional structure where a *vtimes* structure is returned describing the resources used by the terminated process and all its children. This may be given as “0” if the information is not desired. Currently this information is not available for stopped processes.

**SEE ALSO**

wait(2), exit(2), fork(2), sigsys(2)

**DIAGNOSTICS**

Returns -1 if there are no children not previously waited for, or 0 if the WNOHANG option is given and there are no stopped or exited children.

**BUGS**

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are subject to change. It may be replaced by other facilities in future versions of the system.

**NAME**

reboot – reboot system or halt processor

**SYNOPSIS**

```
#include <sys/reboot.h>
```

```
reboot(howto)
```

```
int howto;
```

**DESCRIPTION**

*Reboot* is used to cause a system reboot, and is invoked automatically in the event of unrecoverable system failures. *Howto* is a mask of options passed to the bootstrap program; some of the information in *howto* is interpreted by this program, and the information is further passed to the initialization process *init(8)* in the new system. When none of these options (e.g. RB\_AUTOBOOT) is given, the system is rebooted from file “vmunix” in the root file system of unit 0 of the drive with the same disk controller as the current root file system. An automatic consistency check of the disks is then normally performed.

The bits of *howto* are:

**RB\_HALT**

the processor is simply halted; no reboot takes place. This should be used with caution.

**RB\_ASKNAME**

Interpreted by the bootstrap program itself, causing it to inquire as to what file should be booted. Normally, the system is booted from the file “xx(0,0)vmunix” without asking, where *xx* is determined by a code in register *r10* (which is known as *devtype*) at entry to the bootstrap program. The code corresponds to the major device number of the root file system, i.e. “major(rootdev)”. Currently, the following values of *devtype* are understood:

0	hp	rm03/rm05/rp06 massbus disk
1	--	unused
2	up	emulex sc21 unibus controller; ampex 9300 disks
3	rk	rk07 unibus disks

Thus if *r10* contained a 2, the system

```
up(0,0)vmunix.
```

would be booted.

**RB\_SINGLE**

Normally, the reboot procedure involves an automatic disk consistency check and then multi-user operations. This prevents the consistency check, rather simply booting the system with a single-user shell on the console, from the file system specified by *r10*.

**SEE ALSO**

crash(8), halt(8), init(8), reboot(8)

**BUGS**

**NAME**

vadvise – give advice to paging system

**SYNOPSIS**

**vadvise(param)**

**DESCRIPTION**

*Vadvise* is used to inform the system that process paging behavior merits special consideration. Parameters to *vadvise* are defined in the file `<vadvise.h>`. Currently, two calls to *vadvise* are implemented:

The call

**vadvise(VA\_ANOM);**

advises that the paging behavior is not likely to be well handled by the system's default algorithm, since reference information collected over macroscopic intervals (e.g. 10-20 seconds) will not serve to indicate future page references. The system in this case will choose to replace pages with little emphasis placed on recent usage, and more emphasis on referenceless circular behavior. It is *essential* that processes which have very random paging behavior (such as LISP during garbage collection of very large address spaces) call *vadvise*, as otherwise the system has great difficulty dealing with their page-consumptive demands.

The call

**vadvise(VA\_NORM);**

restores default paging replacement behavior after a call to

**vadvise(VA\_ANOM);**

**BUGS**

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are expected to change. It is expected to be extended with additional facilities in future versions of the system. In particular it is expected that this call will be particular to a segment, and that other behaviors such as sequential behavior will be specifiable.

**NAME**

*vfork* – spawn new process in a virtual memory efficient way

**SYNOPSIS**

**vfork()**

**DESCRIPTION**

*Vfork* can be used to create new processes without fully copying the address space of the old process, which is horrendously inefficient in a paged environment. It is useful when the purpose of *fork(2)* would have been to create a new system context for an *exec*. *Vfork* differs from *fork* in that the child borrows the parents memory and thread of control until a call to *exec(2)* or an exit (either by a call to *exit(2)* or abnormally.) The parent process is suspended while the child is using its resources.

*Vfork* returns 0 in the child's context and (later) the pid of the child in the parents context.

*Vfork* can normally be used just like *fork*. It does not work, however, to return while running in the child's context from the procedure which called *vfork* since the eventual return from *vfork* would then return to a no longer existent stack frame. Be careful, also, to call *\_exit* rather than *exit* if you can't *exec*, since *exit* will flush and close standard I/O channels, and thereby mess up the parent processes standard I/O data structures. (Even with *fork* it is wrong to call *exit* since buffered data would then be flushed twice.)

Similarly when using the new signal mechanism of *sigset(3)* mechanism be sure to call *sigsys* rather than *signal(2)*.

**SEE ALSO**

*fork(2)*, *exec(2)*, *sigsys(2)*, *wait(2)*,

**DIAGNOSTICS**

Same as for *fork*.

**BUGS**

This system call may be unnecessary if the system sharing mechanisms allow *fork* to be implemented more efficiently; users should not depend on the memory sharing semantics of *vfork* as it could, in that case, be made synonymous to *fork*.

To avoid a possible deadlock situation, processes which are children in the middle of a *vfork* are never sent SIGTTOU or SIGTTIN signals; rather, output or *ioctl*s are allowed and input attempts result in an end-of-file indication.

This call is peculiar to this version of UNIX.



**NAME**

vhangup – virtually “hangup” the current control terminal

**SYNOPSIS**

**vhangup()**

**DESCRIPTION**

*Vhangup* is used by the initialization process *init(8)* to arrange that users are given “clean” terminals at login, by revoking access of the previous users’ processes to the terminal. To effect this, *vhangup* searches the system tables for references to the control terminal of the invoking process, revoking access permissions on each instance of the terminal which it finds. Further attempts to access the terminal by the affected processes will yield i/o errors (EBADF). Finally, a hangup signal (SIGHUP) is sent to the process group of the control terminal.

**SEE ALSO**

init (8)

**BUGS**

Access to the control terminal via **/dev/tty** is still possible.

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are subject to change.

**NAME**

vlimit – control maximum system resource consumption

**SYNOPSIS**

```
#include <sys/vlimit.h>
```

```
vlimit(resource, value)
```

**DESCRIPTION**

Limits the consumption by the current process and each process it creates to not individually exceed *value* on the specified *resource*. If *value* is specified as *-1*, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

**LIM\_NORAISE**

A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the *noraise* restriction.

**LIM\_CPU** the maximum number of cpu-seconds to be used by each process

**LIM\_FSIZE** the largest single file which can be created

**LIM\_DATA** the maximum growth of the data+stack region via *sbrk(2)* beyond the end of the program text

**LIM\_STACK** the maximum size of the automatically-extended stack region

**LIM\_CORE** the size of the largest core dump that will be created.

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to *csh(1)*.

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file i/o operation which would create a file which is too large will cause a signal SIGXFSZ to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal SIGXCPU is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the cpu time limit.

**SEE ALSO**

*csh(1)*

**BUGS**

If LIM\_NORAISE is set, then no grace should be given when the cpu time limit is exceeded.

There should be *limit* and *unlimit* commands in *sh(1)* as well as in *csh*.

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are subject to change. It may be extended or replaced by other facilities in future versions of the system.

**NAME**

vread – read virtually

**SYNOPSIS**

```
vread(fildes, buffer, nbytes)
char *buffer;
```

**DESCRIPTION**

**N.B.:** This call is likely to be replaced by more general virtual memory facilities in the near future.

A file descriptor is a word returned from a successful *open*, *creat*, *dup* or *pipe* call. *Buffer* is the location of *nbytes* contiguous bytes into which the input will be placed. It is not guaranteed that all *nbytes* will be read (see *read(2)*). In particular, if the returned value is 0, then end-of-file has been reached.

Unlike *read(2)*, *vread* does not necessarily or immediately fetch the data requested from *fildes*, but merely insures that the data will be fetched from the file descriptor *sometime before* the first reference to the data, at the system's discretion. Thus *vread* allows the system, among other possibilities, to choose to read data on demand, with whatever granularity is allowed by the memory management hardware, or to just read it in immediately as with *read*. A companion *vwrite(2)* call may be used with *vread* to provide an efficient mechanism for updating large files. The behavior of *vread* if other processes are writing to *fildes* is not defined.

Both the address of *buffer* and the current offset in *fildes* (as told by *tell(2)*) must be aligned to a multiple of VALSIZ (defined in `<valign.h>`). The library routine *valloc(3)* allocates properly aligned blocks from the free list.

Note for non-virtual systems: the *vread* system call can be simulated (exactly, if less efficiently) by *read*. If the unit on which a *vread* is done is not capable of supporting efficient demand initialization (e.g. a terminal or a pipe), then the system may choose to treat a call to *vread* as if it were a call to *read* at its discretion.

**SEE ALSO**

*read(2)*, *write(2)*, *vwrite(2)*, *valloc(3)*

**DIAGNOSTICS**

A 0 is returned at end-of-file. If the read was otherwise unsuccessful, a -1 is returned. Physical I/O errors, non-aligned or bad buffer addresses, preposterous *nbytes*, file descriptor not that of an input file, and file offset not properly aligned can all generate errors.

**BUGS**

You can't *close* a file descriptor which you have *vread* from while there are still pages in the file which haven't been fetched by the system into your address space. In no case can a file descriptor which had such pages at the point of a *vfork* be closed during the *vfork*.

The system refuses to truncate a file to which any process has a pending *vread*.

There is no primitive inverting *vread* to release the binding *vread* sets up so that the file may be closed. This can be only be done, clumsily, by reading another (plain) file onto the buffer area, or pulling the break back with *break(2)* to completely release the pages.

This call is peculiar to this version of UNIX. It will be superseded by more general virtual memory facilities in future versions of the system.

**NAME**

vswapon – add a swap device for interleaved paging/swapping

**SYNOPSIS**

**vswapon(name)**  
**char \*name;**

**DESCRIPTION**

*Vswapon* makes the argument block device available to the system for allocation for paging and swapping. The number of blocks to be made available, as well as the names of all potentially available devices are known to the system, and are present in the system configuration file (e.g. /usr/src/sys/conf/confhp.c).

**SEE ALSO**

swapon(8)

**BUGS**

There is no way to stop swapping on a disk so that the pack may be dismounted.

This call is peculiar to this version of UNIX.

**NAME**

`vtimes` – get usage of time, space, and paging resources

**SYNOPSIS**

```
#include <sys/vtimes.h>

vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;
```

**DESCRIPTION**

*Vtimes* places accounting information for the current process in the area pointed to by *par\_vm* and for its terminated children in the area pointed to by *ch\_vm*. If either pointer is 0, the corresponding information is omitted.

After the call, each area contains information in the form

```
    intvm_utime; /* user time */
    intvm_stime; /* system time */
    /* rss = resident storage size in 512-byte pages */
    unsignedvm_idrss; /* time integral - data+stack rss */
    unsignedvm_ixrss; /* time integral - text rss */
    intvm_maxrss; /* maximum rss */
    intvm_majflt; /* major page faults */
    intvm_minflt; /* minor page faults */
    intvm_nswap; /* number of swaps */
    intvm_inblk; /* block reads */
    intvm_oublk; /* block writes */
};
```

Times are expressed in clock ticks of 1/60 (or 1/50) second. The time integrals are computed by cumulating the number of 512-byte pages in use at each clock tick.

A major page fault results in disk activity; a minor fault gathers page-reference information. Block reads and writes are file system disk transfers; blocks found in the buffer pool are not counted.

**SEE ALSO**

[time\(2\)](#), [wait\(2\)](#)

**NAME**

`vwrite` – write (virtually) to file

**SYNOPSIS**

```
vwrite(filedes, buffer, nbytes)  
char *buffer;
```

**DESCRIPTION**

**N.B.:** This call is likely to be replaced by more general virtual memory facilities in the near future.

The `vwrite` system call is used in conjunction with `vread` to perform efficient updating of large files. After a call to `vread` and updating of the data in the buffer which was given to `vread`, a `vwrite` of the same buffer to the same `filedes` at the same offset in the file will cause data which has been modified since it was `vread` from (or `vwritten` to) the file to be returned to the file.

**SEE ALSO**

`vread(2)`

**DIAGNOSTICS**

Returns `-1` on error: bad descriptor, buffer address, count or alignment as well as on physical I/O errors.

**BUGS**

The result of `vwrite` is defined only when no other `vread`'s have occurred on `buffer` since the one matching the `vwrite`.

This call is peculiar to this version of UNIX. It will be superseded by more general virtual memory facilities in future versions of the system.

**NAME**

intro – introduction to library functions

**SYNOPSIS**

**#include** <libc.h>

**#include** <stdio.h>

**#include** <math.h>

**DESCRIPTION**

This section describes functions that may be found in various libraries, other than the system calls described in section 2. Functions are divided into various libraries distinguished by the section number at the top of the page:

- (3) These functions, together with those of section 2 and those marked (3S) and (3M), constitute library *libc*, which is automatically loaded by the C compiler *cc*(1) and the Fortran compiler *f77*(1). The same functions appear also in *libC*, which is automatically loaded by the C++ compiler; see *c++*(1). The link editor *ld*(1) searches this library under option **-lc** (**-lC** for *libC*). Declarations for some of these functions may be obtained from include files indicated on the appropriate pages. Other declarations can be found in
- (3F) These functions are in the Fortran library, *libF77*, automatically loaded by the Fortran compiler, and searched under option **-lF77** of the link editor.
- (3M) These functions constitute the math library, part of *libc*. (On some other systems they must be loaded by **-lm**). Declarations for these functions may be obtained from the include file
- (3S) These functions constitute the ‘standard IO package’ (see *stdio*(3)) part of *libc* already mentioned. Declarations for these functions may be obtained from the include file
- (3X) Various specialized libraries have not been given distinctive captions. Files in which such libraries are found are named on appropriate pages.
- (3+) C++ functions in *libC* that are not in *libc*.

**FILES**

/lib/libc.a

**SEE ALSO**

*stdio*(3), *nm*(1), *ld*(1), *cc*(1), *c++*(1), *f77*(1), *intro*(2)

**DIAGNOSTICS**

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases the external variable *errno* (see *intro*(2)) is set to the value **EDOM** or **ERANGE**, defined in the include file

**NAME**

abort – generate a fault

**SYNOPSIS**

**int abort()**

**DESCRIPTION**

*Abort* sends the current process a SIGIOT signal, which normally terminates the process with a core dump.

**SEE ALSO**

*adb(1)*, *signal(2)*, *exit(2)*

**DIAGNOSTICS**

Usually 'abort – core dumped' from the shell.



**NAME**

`abs`, `sgn`, `gcd`, `lcm`, `min`, `max`, `labs` – integer arithmetic functions: absolute value, sign, greatest common divisor, least common multiple, minimum, maximum

**SYNOPSIS**

**int** `abs(a)`

**int** `sgn(a)`

**int** `gcd(a, b)`

**long** `lcm(a, b)`

**int** `min(a, b)`

**int** `max(a, b)`

**long** `labs(a)`

**long** `a`;

**DESCRIPTION**

*Abs* returns the absolute value of *a*.

*Sgn* returns  $-1$ ,  $0$ ,  $1$ , if  $a < 0$ ,  $a = 0$ ,  $a > 0$ , respectively.

*Gcd* returns the greatest common divisor of *a* and *b*. More precisely, *gcd* returns the largest machine-representable generator of the ideal generated by *a* and *b*. This means that **gcd(0,0) = 0**, and **gcd(N,0) = gcd(0,N) = N**, where **N** is the most negative integer.

*Lcm* returns the least common multiple of *a* and *b*. When the result is representable, it satisfies **abs(a\*b) == lcm(a,b)\*gcd(a,b)**.

*Min* (*max*) returns the minimum (maximum) of *a* and *b*.

**SEE ALSO**

[\*floor\(3\)\*](#) for *fabs*

**DIAGNOSTICS**

*Abs* returns the most negative integer when the true result is unrepresentable.

There are no guarantees about the value of *lcm* when the true value is unrepresentable.

**BUGS**

The result of *lcm* is undefined when it doesn't fit in a long.

*Labs*, provided for ANSI compatibility, is lonely; there is no *lsign*, *lmax*, etc.

**NAME**

assert – assertion checking

**SYNOPSIS**

```
#include <assert.h>
```

```
void assert(expression);
```

**DESCRIPTION**

*Assert* is a macro that indicates *expression* is expected to be nonzero at this point in the program. It causes an *abort(3)* with a diagnostic comment on the standard output when *expression* is zero. Compiling with the *cc(1)* option **-DNDEBUG** effectively makes the expression always nonzero.

**DIAGNOSTICS**

'Assertion failed: file *f* line *n*', where *f* is the source file and *n* the source line number of the *assert* statement.

**NAME**

*atof*, *atoi*, *atol*, *strtod*, *strtol*, *strtoul* – convert ASCII to numbers

**SYNOPSIS**

**double** *atof*(*nptr*)

**char** \**nptr*;

**int** *atoi*(*nptr*)

**char** \**nptr*;

**long** *atol*(*nptr*)

**char** \**nptr*;

**double** *strtod*(*nptr*, *rptr*)

**char** \**nptr*, \*\**rptr*;

**long** *strtol*(*nptr*, *rptr*, *base*)

**char** \**nptr*, \*\**rptr*;

**unsigned long** *strtoul*(*nptr*, *rptr*, *base*)

**char** \**nptr*, \*\**rptr*;

**DESCRIPTION**

*Atof*, *atoi*, and *atol* convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

*Atof* recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optionally signed integer.

*Atoi* and *atol* recognize an optional string of tabs and spaces, then an optional sign, then a string of decimal digits.

*Strtod*, *strtol*, and *strtoul*, behave similarly to *atof*, and *atol* and, if *rptr* is not zero, set \**rptr* to point to the input character immediately after the string converted.

*Strtol* and *strtoul* interpret the digit string in the specified *base*, from 2 to 36, each digit being less than the base. Digits with value over 9 are represented by letters, a-z or A-Z. If *base* is 0, the input is interpreted as an integral constant in the style of C (with no suffixed type indicators): numbers are octal if they begin with 0, hexadecimal if they begin with 0x or 0X, otherwise decimal. *Strtoul* does not recognize signs.

**SEE ALSO**

[scanf\(3\)](#)

**DIAGNOSTICS**

Zero is returned if the beginning of the input string is not interpretable as a number.

If overflow is detected by *atof*, *strtod*, *strtol*, or *strtoul*, a maximum value of the correct sign is returned and *errno* is set to **ERANGE**.

**BUGS**

*Atoi* and *atol* have no provisions for overflow.

**NAME**

*besj0*, *besj1*, *besjn*, *besy0*, *besy1*, *besyn* – *bessel* functions

**SYNOPSIS**

```
#include <math.h>
```

```
double besj0(x)
```

```
double x;
```

```
double besj1(x)
```

```
double x;
```

```
double besjn(n, x)
```

```
double x;
```

```
double besy0(x)
```

```
double x;
```

```
double besy1(x)
```

```
double x;
```

```
double besyn(n, x)
```

```
double x;
```

**DESCRIPTION**

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

**DIAGNOSTICS**

Negative arguments cause *besy0*, *besy1*, and *besyn* to return a huge negative value and set *errno* to **EDOM**.

**NAME**

bits – variable length bit strings

**SYNOPSIS**

```
#include <Bits.h>
```

```
typedef unsigned long Bits_chunk;
```

```
struct Bits {
```

```
    Bits() { }
```

```
    Bits(Bits_chunk, unsigned = 1);
```

```
    Bits(const Bits&)
```

```
    Bits();
```

```
    Bits& operator= (const Bits&); // also = &= |= ^=
```

```
    Bits& operator<<= (int); // also >>=
```

```
    int operator[] (unsigned);
```

```
    operator Bits_chunk();
```

```
    unsigned size();
```

```
    unsigned size(unsigned);
```

```
    Bits& compl();
```

```
    Bits& concat(const Bits&);
```

```
    Bits& set(unsigned, unsigned long = 1);
```

```
    Bits& reset(unsigned);
```

```
    Bits& compl(unsigned);
```

```
    unsigned count();
```

```
    unsigned signif();
```

```
    unsigned trim();
```

```
};
```

```
Bits operator (const Bits&);
```

```
Bits operator& (const Bits&, const Bits&); // also | ^
```

```
Bits operator<< (const Bits&, int); // also >>
```

```
int operator< (const Bits&, const Bits&); // also > <= >= == !=
```

**DESCRIPTION**

A **Bits** object contains a variable-length bit string. The bits of a **Bits** object *b* are numbered from 0 through *b.size()*–1, with the rightmost bit numbered 0.

**Bits\_chunk** is the largest unsigned integral type acceptable for conversion to and from **Bits**, **unsigned long** in this implementation.

**Constructors**

**Bits()** An empty bit string.

**Bits(*n*)** The bits are copied from the binary representation of *n* with the ones-digit of *n* becoming bit 0. Leading zero-bits are removed, except that **Bits(0)** is a one-bit string.

**Bits(*n*,*m*)** The same, but padded with leading zeros to size *m* if necessary.

**Operators**

Bit strings have value semantics; assigning a **Bits** object or passing it to or returning it from a function creates a copy of its value. The meanings of operators are mostly predictable from C.

Under binary and comparison operators, the shorter operand is considered to be padded on the left with zeros to the length of the longer. If, after padding, two strings of different length compare equal, the shorter is deemed the smaller.

Negative shift amounts reverse the sense of shift operators.

Under conversion (or assignment) to a **Bits\_chunk**, a **Bits** is interpreted as an unsigned integer. If it has a value small enough to fit, that value is assigned. Otherwise, a non-zero value is assigned. Thus a **Bits** is considered ‘true’ in an **if** test if it contains any one-bit, ‘false’ otherwise.

*a[s]* Bit number *s* of *a*; 0 if *s* is out of bounds.

#### Other functions

*a.size(s)* Set the size of *a* to *s* by truncating or padding with zeros on the left as necessary. Return the new size.

*a.compl()* Complement the bits of *a*. Return *a*.

*a.set(s)*

*a.reset(s)*

*a.compl(s)*

Set, reset, or complement bit *s* of *a*. No effect if *a.size()* ≤ *s*. Return *a*.

*a.set(s,n)*

If *n* is 0, reset bit *s* of *a*, otherwise set bit *s*. Equivalent to **n? a.set(s): a.reset(s)**.

*a.count()*

Return the number of one-bits in *a*.

*a.signif()*

Return the number of significant bits in **a**: one more than the number of the leftmost one-bit, or zero if there is no one-bit.

*a.trim()*

Discard high-order zero-bits. Equivalent to **a.size(a.signif())**.

*a.concat(b)*

Concatenate the value of *b* to the right (low-order) end of *a*. Return *a*.

**concat(a,b)**

Return a newly created concatenated object.

#### DIAGNOSTICS

An operation that runs out of memory sets the length of the affected **Bits** to zero.

#### BUGS

Too bad C++ can't support **a[s] = n**.

Things would be more consistent if **Bits(0).size()** were zero.

**NAME**

block – adjustable arrays

**SYNOPSIS**

```
#include <Block.h>

Blockdeclare(T)
Blockimplement(T)

struct Block(T) {
    Block(T)(unsigned = 0);
    Block(T)(const Block(T&);
    Block(T);
    Block(T)& operator= (const Block(T)&);
    T& operator[] (int);
    operator T* ();
    unsigned size();
    unsigned size(unsigned);
    T* end();
    void swap(const Block(T)&);
}
```

**DESCRIPTION**

A **Block**(*T*) is an array of zero or more *elements* of type *T*, where *T* is a type name. *T* must have assignment (**operator=**) and initialization (*T*(**T&**)) operations.

The macro call **Blockdeclare**(*T*) declares the class **Block**(*T*). It must appear once in every source file that uses type **Block**(*T*). The macro call **Blockimplement**(*T*) defines the functions that implement the block class. It must appear exactly once in the entire program.

New elements are initialized to the value of an otherwise uninitialized static object of type *T*.

**Constructors**

<b>Block</b> ( <i>T</i> )	An empty block.
<b>Block</b> ( <i>T</i> )( <i>n</i> )	A block of <i>n</i> elements.
<b>Block</b> ( <i>T</i> )( <i>b</i> )	A new block whose elements are copies of the elements of <b>b</b> .

**Operations**

Assignment copies elements and size.

<i>b</i> [ <i>k</i> ]	A reference to element <i>k</i> of block <i>b</i> ; undefined if <i>k</i> is out of bounds.
( <i>T</i> *) <i>b</i>	A pointer to the first element of block <i>b</i> .

**Other functions**

<i>b</i> .size()	Return the number of elements in <i>b</i> .
<i>b</i> .size( <i>n</i> )	Set the size of <i>b</i> to <i>n</i> . If the new size is greater than the old. Otherwise, <i>n</i> old elements are kept. Return the new size.
<i>b</i> .reserve( <i>n</i> )	The size of <i>b</i> is increased, if necessary, to some value greater than <i>n</i> . If <i>b</i> already has room, <i>b</i> is not changed. Return zero if memory could not be allocated and non-zero otherwise.
<i>b</i> .end()	Returns a pointer to just past the last element in <b>b</b> . Equivalent to ( <b>T</b> *) <b>b</b> + <b>b.size</b> () .
<i>a</i> .swap( <i>b</i> )	The memory associated with blocks <i>a</i> and <i>b</i> is exchanged.

**Performance**

Most operations are implemented by the obvious uses of the `new` and `delete` operators. *Reserve* checks the size inline. If it isn't big enough, the size is increased by multiplying by 3/2 (and adding one) enough times to increase it beyond *n*.

**EXAMPLES**

```
Blockdeclare(long)
unsigned n = 0;
Block(long) b;
long x;
while (cin >> x) {
    b.reserve(n);
    b[n++] = x;
}
```

**SEE ALSO**

[malloc\(3\)](#), [map\(3\)](#)

**DIAGNOSTICS**

The only error detected is running out of memory; this is indicated in all cases by setting the size of the block for which allocation failed to zero.

**BUGS**

Elements are copied during reallocation by using `T::operator=` instead of `T(T&)`.  
Because the 'type parameter' *T* is implemented by the C preprocessor, white space is forbidden inside the parentheses of `Block(T)`.



**NAME**

bopen, bclose, bseek, bfirst, bkey, breclen, bread, bdelete, bwrite – compressed B-tree subroutines

**SYNOPSIS**

```
#include <cbt.h>

bfile *bopen(name, typ) char *name;

void bclose(b) bfile *b;

bseek(b, key) bfile *b; mbuf key;

bfirst(b) bfile *b;

mbuf bkey(b) bfile *b;

breclen(b) bfile *b;

bread(b, key, val) bfile *b; mbuf *key, *val;

bdelete(b, key) bfile *b; mbuf key;

bwrite(b, key, val) bfile *b; mbuf key, val;
```

**DESCRIPTION**

These functions manipulate files of key/value records. Such files are created by **cbt creat**; see [cbt\(1\)](#). To load the functions use the [ld\(1\)](#) option **-lcbt**.

The records occur sorted (increasing lexicographical order) by their keys, which must be distinct. Both keys and values are arrays of characters accessed through the structure

```
[CB]typedef struct {
    [CB]char *mdata;address of data bytes
    [CB]short mlen;number of data bytes
[CB]} mbuf;
```

*Bopen* attempts to open a named B-tree, and if successful establishes a read pointer pointing to the beginning of the file and returns a pointer to be used in calling the other routines. *Typ* is 0 for read-only or 2 for read-write. *Bopen* returns a descriptor that identifies the file to the other functions.

*Bclose* closes a B-tree.

*Bseek* positions the read pointer of the file to the record whose key is the first not less than *key*. The routine returns 1 if *key* is in the file, **EOF** if *key* is greater than any key in the file, and 0 otherwise.

*Bfirst* sets the read pointer to the beginning of the file. It has the same error return as **bseek**.

*Bkey* returns the current key. The element *mdata* of the returned structure is 0 on errors, otherwise it points to a static buffer.

*Breclen* returns the length of the value part of the current record.

*Bread* reads the value at the read pointer into the space pointed to by **val->mdata**, places its length in **val->mlen**, and advances the read pointer to the record with the next greater key. If *key* is not 0 the key of the record is read into the space pointed to by **key->mdata** and its length is placed in **key->mlen**. *Bread* returns 0 if successful.

*Bdelete* removes the record with the given key, returning 1 if the record was found, -1 if there was an error, and 0 otherwise. The read pointer is left undefined.

*Bwrite* writes the given value with the given key. An existing record with that key will be replaced. The read pointer is left undefined.

**FILES**

*name*.T  
*name*.F

**SEE ALSO**

[cbt\(1\)](#), [dbm\(3\)](#)

## **DIAGNOSTICS**

Routines which return pointers return 0 on errors, routines which return integers return -1.

## **BUGS**

The length of any key is limited to 255.

The **mbuf** arguments are passed inconsistently to the routines, sometimes by value and sometimes by reference.

*Cbt* files are not directly portable between big-endian and little-endian machines.

**NAME**

`chrtab` – simple character bitmaps

**SYNOPSIS**

```
extern char chrtab[95][16];
```

**DESCRIPTION**

*Chrtab* contains 8-by-16 bitmaps for ASCII printing characters. The 16 bytes pointed to by `chrtab[c-']` are the 16 rows of character *c* from top to bottom. The most significant bit is the leftmost bit. The bottom row is always empty (has all bits 0).

**SEE ALSO**

[font\(9\)](#)

**BUGS**

The bitmaps of *chrtab* are intended for use with line printers, not bitmap devices.

**NAME**

closeshares – close shares file

**SYNOPSIS**

**int** closeshares()

**DESCRIPTION**

*Closeshares* closes the shares file (if open) which otherwise remains open across shares file routine calls.

**FILES**

/etc/shares      Shares data-base.

**SEE ALSO**

getshares(3), getshput(3), openshares(3), putshares(3), sharesfile(3).

**NAME**

*crypt*, *setkey*, *encrypt* – DES encryption

**SYNOPSIS**

**char \**crypt*(key, salt)**

**char \**key*, \**salt*;**

**setkey(*key*)**

**char \**key*;**

**encrypt(*block*, *edflag*)**

**char \**block*;**

**DESCRIPTION**

*Crypt* is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other functions provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to *encrypt* is also a character array of length 64 containing 0's and 1's. The 64 argument 'bits' are encrypted in place by the DES algorithm using the key previously set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

**SEE ALSO**

[crypt\(1\)](#), [passwd\(1\)](#), [passwd\(5\)](#), [getpass\(3\)](#)

**BUGS**

The return value points to static data whose content is overwritten by each call.

*Encrypt* is not available outside the United States and Canada.

**NAME**

ctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII

**SYNOPSIS**

```
#include <time.h>

char *ctime(clock)
long *clock;

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

**DESCRIPTION**

*Ctime* converts a time pointed to by *clock* such as returned by [time\(2\)](#) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

*Localtime* and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

```
[CB]struct tm {
    [CB]int tm_sec;    seconds (range 0..59)
    [CB]int tm_min;    minutes (0..59)
    [CB]int tm_hour;   hours (0..23)
    [CB]int tm_mday;   day of the month (1..31)
    [CB]int tm_mon;    month of the year (0..11)
    [CB]int tm_year;   year A.D. – 1900
    [CB]int tm_wday;   day of week (0..6, Sunday = 0)
    [CB]int tm_yday;   day of year (0..365)
    [CB]int tm_isdst;  zero means normal time, nonzero means daylight saving time
[CB]};
```

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The peculiarities of this conversion are read from the file which contains lines of the form

```
y0 y1 bmon bday boff emon eday eoff
```

meaning that for years between *y0* and *y1* inclusive, daylight saving time begins (ends) *boff* (*eoff*) days after the first Sunday after the day *bmon/bday* (*emon/eday*).

*Timezone* returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced. Thus, as Afghanistan is 4:30 ahead of GMT, `timezone(-(60*4+30), 0)` returns[CB] "GMT+4:30".

**SEE ALSO**

[time\(2\)](#), [timec\(3\)](#)

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – character classification

**SYNOPSIS**

```
#include <ctype.h>
```

**isalpha(c)**

**isupper(c)**

**islower(c)**

**isdigit(c)**

**isxdigit(c)**

**isalnum(c)**

**isspace(c)**

**ispunct(c)**

**isprint(c)**

**isgraph(c)**

**iscntrl(c)**

**isascii(c)**

**DESCRIPTION**

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value **EOF**; [stdio\(3\)](#).

*isalpha*

*c* is a letter, a-z or A-Z

*isupper*

*c* is an upper case letter, A-Z

*islower*

*c* is a lower case letter, a-z

*isdigit* *c* is a digit, 0-9

*isxdigit*

*c* is a hexadecimal digit, 0-9 or a-f or A-F

*isalnum*

*c* is an alphanumeric character, a-z or A-Z or 0-9

*isspace*

*c* is a space, horizontal tab, vertical tab, carriage return, newline, or formfeed (040, 011, 012, 013, 014, 015)

*ispunct*

*c* is a punctuation character (one of !"#\$%&'()\*+,-./:;<=>?@[\\]^\_`{|} )

*isprint*

*c* is a printing character, 040 (space) through 0176 (tilde)

*isgraph*

*c* is a visible printing character, 041 (exclamation) through 0176 (tilde)

*iscntrl*

*c* is a delete character, 0177, or ordinary control character, 000 through 037

*isascii*

*c* is an ASCII character, 000 through 0177

**SEE ALSO**

[tolower\(3\)](#), [ascii\(6\)](#)

**NAME**

curses – screen functions with ‘optimal’ cursor motion

**DESCRIPTION**

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then *refresh()* tells the routines to make the current screen look like the new one. The initialization routine *initscr()* must be called before any other routines that deal with windows and screens. The routine *endwin()* should be called before exiting.

To load the functions use the *ld(1)* options **-lcurses -ltermcap**.

**SEE ALSO**

*ioctl(2)*, *termcap(5)*

Ken Arnold, ‘Screen Updating and Cursor Movement Optimization: A Library Package’, *UNIX Programmer’s Manual*, Seventh Edition, Virtual VAX-11 Version, 1980 (Berkeley)

**FUNCTIONS**

[CB]addch(ch)	<b>add a character to stdscr</b>
[CB]addstr(str)	<i>add a string to stdscr</i>
[CB]box(win,vert,hor)	<i>draw a box around a window</i>
[CB]cbreakmode()	set cbreak mode
[CB]clear()	<i>clear stdscr</i>
[CB]clearok(scr,boolf)	<i>set clear flag for scr</i>
[CB]clrtoeol()	<i>clear to bottom on stdscr</i>
[CB]clrtoeol()	<i>clear to end of line on stdscr</i>
[CB]delwin(win)	<i>delete win</i>
[CB]echo()	<i>set echo mode</i>
[CB]endwin()	end window modes
[CB]erase()	<i>erase stdscr</i>
[CB]getch()	<i>get a char through stdscr</i>
[CB]getstr(str)	<i>get a string through stdscr</i>
[CB]gettmode()	<i>get tty modes</i>
[CB]getyx(win,y,x)	<i>get (y,x) co-ordinates</i>
[CB]inch()	<i>get char at current (y,x) co-ordinates</i>
[CB]initscr()	initialize screens
[CB]leaveok(win,boolf)	<i>set leave flag for win</i>
[CB]longname(termbuf,name)	<i>get long name from termbuf</i>
[CB]move(y,x)	<i>move to (y,x) on stdscr</i>
[CB]mvcur(lasty,lastx,newy,newx)	<i>actually move cursor</i>
[CB]newwin(lines,cols,begin_y,begin_x)	create a new window
[CB]nl()	<i>set newline mapping</i>
[CB]nocrmode()	unset cbreak mode
[CB]noecho()	<i>unset echo mode</i>
[CB]nonl()	unset newline mapping
[CB]noraw()	<i>unset raw mode</i>
[CB]overlay(win1,win2)	overlay win1 on win2
[CB]overwrite(win1,win2)	<i>overwrite win1 on top of win2</i>
[CB]printw(fmt,arg1,arg2,...)	<i>printf on stdscr</i>
[CB]raw()	<i>set raw mode</i>
[CB]refresh()	make current screen look like <i>stdscr</i>
[CB]resetty()	<i>reset tty flags to stored value</i>
[CB]savetty()	stored current tty flags
[CB]scanw(fmt,arg1,arg2,...)	<i>scanf through stdscr</i>
[CB]scroll(win)	<i>scroll win one line</i>
[CB]scrollok(win,boolf)	<i>set scroll flag</i>
[CB]setterm(name)	set term variables for name
[CB]standend()	<i>end standout mode</i>
[CB]standout()	start standout mode



[CB]subwin(win,lines,cols,begin\_y,begin\_x)  
[CB]touchwin(win)  
[CB]unctrl(ch)  
[CB]waddch(win,ch)  
[CB]waddstr(win,str)  
[CB>wclear(win)  
[CB>wclrto bot(win)  
[CB>wclrtoeol(win)  
[CB>werase(win)  
[CB>wgetch(win)  
[CB>wgetstr(win,str)  
[CB>winch(win)  
[CB>wmove(win,y,x)  
[CB>wprintw(win,fmt,arg1,arg2,...)  
[CB>wrefresh(win)  
[CB>wscanw(win,fmt,arg1,arg2,...)  
[CB>wstandend(win)  
[CB>wstandout(win)

*create a subwindow*  
*'change' all of win*  
*printable version of ch*  
*add char to win*  
*add string to win*  
*clear win*  
*clear to bottom of win*  
*clear to end of line on win*  
*erase win*  
*get a char through win*  
*get a string through win*  
*get char at current (y,x) in win*  
*set current (y,x) co-ordinates on win*  
*printf on win*  
*make screen look like win*  
*scanf through win*  
*end standout mode on win*  
*start standout mode on win*

**NAME**

dbminit, fetch, store, delete, firstkey, nextkey – database subroutines

**SYNOPSIS**

**dbminit(file)**

**char \*file;**

**datum fetch(key)**

**datum key;**

**store(key, value)**

**datum key, value;**

**delete(key)**

**datum key;**

**datum firstkey()**

**datum nextkey(key)**

**datum key;**

**DESCRIPTION**

These functions maintain key/value pairs (each pair is a *datum*) in a data base. The functions will handle very large databases in one or two file system accesses per key. The functions are loaded with *ld(1)* option **-ldbm**. A datum is defined as

```
typedef struct {
    char    *dptr;
    int     dsize;
} datum;
```

A **datum** object specifies a string of **dsize** bytes pointed to by **dptr**. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has *.dir* as its suffix. The second file contains all data and has *.pag* as its suffix.

Before a database can be accessed, it must be opened by *dbminit*. At the time of this call, the files *file.dir* and *file.pag* must exist. (An empty database has empty *.dir* and *.pag* files.)

The value associated with a key is retrieved by *fetch* and assigned by *store*. A key and its associated value are deleted by *delete*. A linear pass through all keys in a database may be made, in random order, by use of *firstkey* and *nextkey*. *Firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

```
for(key = firstkey(); key.dptr != NULL; key = nextkey(key))
```

**SEE ALSO**

*cbt(3)*

**DIAGNOSTICS**

All functions that return integers indicate errors with negative values. A zero return indicates success. Routines that return a **datum** indicate errors with zero **dptr**.

**BUGS**

The *.pag* file contains holes; its apparent size is about four times its actual content. These files cannot be copied by normal means (*cat(1)*, *tar(1)*, *cpio(1)*, *ar(1)*) without filling in the holes.

Pointers returned by these subroutines refer to static data that is changed by subsequent calls.

The sum of the sizes of a key/value pair must not exceed a fixed internal block size. Moreover all key/value pairs that hash together must fit on a single block. *Store* will return an error in the event that a disk block fills with inseparable data.

*Delete* does not physically reclaim file space, although it does make it available for reuse.

**NAME**

dialout – place call on ACU

**SYNOPSIS**

```
int dialout(telno, class)
char *telno, *class;
```

**DESCRIPTION**

*Dialout* places a data call via an automatic calling unit directly attached to the calling computer. To use an ACU on Datakit see the example in [ipc\(3\)](#).

*Dialout* searches for an ACU of the appropriate service class and places a data call on the associated line, using the given telephone number. If successful, it returns an open file descriptor for the line. The file is in raw mode, and has exclusive-use and hangup-on-close modes set. It returns `-1` if all ACUs of the given class are busy, `-3` if carrier could not be set, and `-9` if the service class is unidentifiable.

The routine consults a data file `/etc/aculist` that consists of lines containing six blank- or tab-separated fields.

service class

Each line with a service class matching the one specified is tried in turn until an unoccupied one is found. Service classes specify a switching office and a baud rate.

Defined service classes at the ‘research’ site are **300** and **1200**, for 300- and 1200-baud calls on 665- phone lines, with synonyms **D300** and **D1200**. Internal calls on these lines require 5 digits. Service classes **C300** and **C1200** use 582- phone lines. Internal calls on these lines require 4 digits and reach only other 582- lines.

file           The file name of the associated special file for the telephone line.

acu            The file name of the associated ACU. If specified as **none**, no ACU is used and the telephone number is ignored. This is for hardwired connections.

speed          The bit rate of the interface, chosen from the numbers given in [ttyld\(4\)](#).

prefix         A string to be prefixed to the number. This is handy for shared ACUs in which the first digit specifies a line. The prefix `-` is taken to be an empty prefix.

postfix        A string to be postfixed to the number to be dialed. Some ACUs require an ‘end of number’ code; it should be specified here.

**FILES**

`/etc/aculist`

**SEE ALSO**

[ttyld\(4\)](#), [cu\(1\)](#), [ipc\(3\)](#)

**NAME**

opendir, readdir, telldir, seekdir, closedir – directory operations

**SYNOPSIS**

```
#include <sys/types.h>
#include <ndir.h>

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

closedir(dirp)
DIR *dirp;
```

**DESCRIPTION**

*Opendir* opens the directory named by *filename* and associates a ‘directory stream’ with it. *Opendir* returns a pointer to be used to identify the directory stream in subsequent operations. The pointer value 0 is returned if *filename* cannot be accessed or is not a directory.

*Readdir* returns a pointer to the next directory entry. It returns 0 upon reaching the end of the directory or detecting an invalid *seekdir* operation.

*Telldir* returns the current location associated with the named directory stream.

*Seekdir* sets the position of the next *readdir* operation on the directory stream. The new position reverts to the one associated with the directory stream when the *telldir* operation was performed. Values returned by *telldir* are good only for the lifetime of the **DIR** pointer from which they are derived.

*Closedir* causes the named directory stream to be closed, and the structure associated with the **DIR** pointer to be freed.

```
struct direct {
    [CB]u_long          d_ino; inode for the entry
    [CB]short d_reclen; don't use
    [CB]short          d_namlen; equivalent to [CB]strlen(d_name)
    [CB]char           d_name [MAXNAMLEN+1]; null-terminated entry name
[CB]};
```

The preferred way to search the current directory is:

```
DIR *dirp;
dirp = opendir(".");
for(dp = readdir(dirp); dp != 0; dp = readdir(dirp))
    if(strcmp(dp->d_name, name) == 0)
        break;
closedir(dirp);
/* found name if dp != 0 */
```

**SEE ALSO**

[dir\(5\)](#), [open\(2\)](#), [dirread\(2\)](#), [read\(2\)](#), [lseek\(2\)](#), [ftw\(3\)](#)

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

*ecvt*, *fcvt*, *gcvt* – convert numbers to ascii

**SYNOPSIS**

**char \**ecvt*(value, ndigit, decpt, sign)**

**double value;**

**int ndigit, \*decpt, \*sign;**

**char \**fcvt*(value, ndigit, decpt, sign)**

**double value;**

**int ndigit, \*decpt, \*sign;**

**char \**gcvt*(value, ndigit, buf)**

**double value;**

**char \*buf;**

**DESCRIPTION**

*Ecvt* converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

*Fcvt* is similar to *ecvt* and produces output for the Fortran format **F\*.ndigit**. If *decpt* ≤ *ndigit*, then the returned string is null. Otherwise, *decpt*+*ndigit*+1 characters (including terminating null) are returned.

*Gcvt* converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf*. It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

**SEE ALSO**

[printf\(3\)](#)

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

*end*, *etext*, *edata* – last locations in program

**SYNOPSIS**

**extern *end*;**  
**extern *etext*;**  
**extern *edata*;**

**DESCRIPTION**

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break coincides with *end*, but it is reset by the routines [brk\(2\)](#), [malloc\(3\)](#), standard input/output ([stdio\(3\)](#)), the profile (**-p**) option of [cc\(1\)](#), etc. The current value of the program break is reliably returned by **sbrk(0)**; see [brk\(2\)](#).

**SEE ALSO**

[brk\(2\)](#), [malloc\(3\)](#), [stdio\(3\)](#), [cc\(1\)](#)

**NAME**

erf, erfc – error function

**SYNOPSIS**

```
#include <math.h>
```

```
double erf(x)
```

```
double x;
```

```
double erfc(x)
```

```
double x;
```

**DESCRIPTION**

These functions calculate the error function  $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$  (.) if  $t$  .ig = (2/sqrt(pi)) integral from 0 to x  $\exp(-t^2) dt$  (.) and the complementary error function  $\text{erfc}(x) = 1 - \text{erf}(x)$ . The error criterion for both *erf* and *erfc* is relative.

**DIAGNOSTICS**

There are no error returns.

**NAME**

exit, atexit, onexit – terminate process

**SYNOPSIS**

```
void exit(status)
```

```
int status;
```

```
int atexit(fn)
```

```
int (*fn)();
```

**DESCRIPTION**

*Exit* is the conventional way to terminate a process. Before calling `_exit` (see [exit\(2\)](#)) with *status* as an argument, it calls in reverse order all the functions recorded by *atexit*.

*Exit* can never return.

*Atexit* records *fn* as a function to be called by *exit*. It returns zero if it failed, nonzero otherwise. Typical uses include cleanup routines for [stdio\(3\)](#) and profiling; see [monitor\(3\)](#).

Calling *atexit* twice (or more) with the same function argument causes *exit* to invoke the function twice (or more).

The function *fn* should be declared as

```
int fn()
```

The function *onexit* is an obsolescent synonym for *atexit*. The constant **NONEXIT** defined in `<libc.h>` determines how many functions can be recorded.

**SEE ALSO**

[exit\(2\)](#)



**NAME**

*exp*, *log*, *log10*, *pow*, *sqrt* – exponential, logarithm, power, square root

**SYNOPSIS**

```
#include <math.h>
```

```
double exp(x)
```

```
double x;
```

```
double log(x)
```

```
double x;
```

```
double log10(x)
```

```
double x;
```

```
double pow(x, y)
```

```
double x, y;
```

```
double sqrt(x)
```

```
double x;
```

**DESCRIPTION**

*Exp* returns the exponential function of  $x$ .

*Log* returns the natural logarithm of  $x$ ; *log10* returns the base 10 logarithm.

*Pow* returns  $x^y$

*Sqrt* returns the square root of  $x$ .

**SEE ALSO**

[hypot\(3\)](#), [sinh\(3\)](#), [intro\(2\)](#)

**DIAGNOSTICS**

*Exp* and *pow* return a huge value when the correct value would overflow; *errno* is set to **ERANGE**. *Pow* returns 0 and sets *errno* to **EDOM** when the second argument is negative and non-integral and when both arguments are 0.

*Log* returns a huge negative value when  $x$  is zero or negative; *errno* is set to **EDOM**.

*Sqrt* returns 0 when  $x$  is negative; *errno* is set to **EDOM**.

**NAME**

feof, ferror, clearerr, fileno – stream status inquiries

**SYNOPSIS**

```
#include <stdio.h>
```

```
int feof(stream)
```

```
FILE *stream;
```

```
int ferror(stream)
```

```
FILE *stream
```

```
int clearerr(stream)
```

```
FILE *stream
```

```
int fileno(stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Feof* returns non-zero if end of file has been encountered on the named input *stream*, otherwise zero. After returning **EOF**, [stdio\(3\)](#) functions will not necessarily return **EOF** again; *feof* provides a lasting indication.

*Ferror* returns non-zero when an error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

*Clearerr* resets the error indication on the named *stream*.

*Fileno* returns the integer file descriptor associated with the *stream*, see [open\(2\)](#).

These functions are implemented as macros; they cannot be redeclared.

**SEE ALSO**

[stdio\(3\)](#)

**NAME**

fgets, puts, fputs, gets – string input/out on streams

**SYNOPSIS**

```
#include <stdio.h>
```

```
char *fgets(s, n, stream)
```

```
char *s;
```

```
FILE *stream;
```

```
int puts(s)
```

```
char *s;
```

```
int fputs(s, stream)
```

```
char *s;
```

```
FILE *stream;
```

**DESCRIPTION**

*Fgets* reads *n*-1 characters, or up to a newline character, whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

*Puts* copies the null-terminated string *s* to the standard output stream *stdout* and appends a newline character.

*Fputs* copies the null-terminated string *s* to the named output *stream*.

Neither routine copies the terminal null character. Both return the result of calling *putc* with the last character written; see [getc\(3\)](#).

**SEE ALSO**

[getc\(3\)](#) [stdio\(3\)](#)

**DIAGNOSTICS**

*Fgets* returns a null pointer upon end of file or error. *Puts* and *fputs* return the constant on write error.

**BUGS**

For safety reasons the ANSI standard function **char \*gets(s)**, which reads from standard input up to a newline and discards the newline, is not supported.

*Puts* appends a newline, *fputs* does not, all in the name of backward compatibility.

**NAME**

filebuf – buffer for file input/output

**SYNOPSIS**

```
#include <iostream.h>
typedef long streamoff, streampos;
class ios {
public:
    enum        seek_dir { beg, cur, end };
    enum open_mode { in, out, ate, app };
    // and lots of other stuff ...
};
#include <fstream.h>
class filebuf : streambuf {
public:
    filebuf() ;
    filebuf(int d);
    filebuf(int d, char* p, int len) ;
    filebuf*   attach(int d) ;
    int       close();
    int       fd();
    int       is_open();
    filebuf*   open(char *name, open_mode om, int prot=0664) ;
    streampos seekoff(streamoff o, seek_dir d, open_mode m) ;
    streampos seekpos(streampos p, open_mode m) ;
    streambuf* setbuf(char* p, int len) ;
    int       sync() ;
};
```

**DESCRIPTION**

filebufs specialize streambufs to use a file as source or sink of characters. Characters are consumed by doing writes to the file, and are produced by doing reads. When the file is seekable, a filebuf allows seeks. At least 4 characters of putback are guaranteed. When the file permits reading and writing the buffer permits both storing and fetching. No special action is required between gets and puts. A filebuf that is connected to a file descriptor is said to be *open*.

Assume:

- **f** is a filebuf.
- **pfb** is a filebuf\*.
- **psb** is a streambuf\*.
- **i**, **d**, **len**, and **prot** are int
- **name** and **ptr** are char\*.
- **m** is open\_mode.
- **off** is streamoff.
- **p** and **pos** are streampos.
- **dir** is seek\_dir.

Constructors:

**filebuf()**

Constructs an initially closed filebuf.

**filebuf(d)**

Constructs a filebuf connected to **d**.

**filebuf(d,p,len)**

Constructs a filebuf connected to **d** and initialized to use the reserve area starting at **p** and containing **len** bytes. If **p** is null or **len** is zero or less, the filebuf will be unbuffered.

Members:

**pfb=f.attach(d)**

Connects **f** to an open file descriptor, **d**. **pfb** is normally **&f** but is 0 if **f** is already open.

**i=f.close()**

Flushes any waiting output, closes the file descriptor, and disconnects **f**. Unless an error occurs, **f**'s error state will be cleared. **i** is 0 unless errors occur in which case it is EOF. Even if errors occur **attach** leaves the file descriptor and **f** closed.

**i=f.fd()**

Returns **i**, the file descriptor **f** is connected to. If **f** is closed **i** is EOF.

**i=f.is\_open()**

Returns non-zero when **f** is connected to a file descriptor, and zero otherwise.

**pfb=f.open(name,m,prot)**

Opens a file with the specified pathname, mode, and protection, and connects **f** to it. `open_modes` may be or'ed together to form **m**. `in` and `out` translate to corresponding UNIX modes. `ate` and `app` both cause the file to be positioned at its end during the open. `app` implies output. In addition, `app` causes all subsequent writes to occur at the end of a file. (In some systems this is supported directly by the kernel, in other instances the desired effect is approximated by seeking to the end of the file before each write.) `out` may be specified even if **prot** does not permit output. Normally **pfb** is **&f** but if an error occurs it is 0.

**p=f.seekoff(off,dir,m)**

Moves the get/put pointer as designated by **off** and **dir**. It fails if the file that **f** is attached to does not support seeking, or if the attempted motion is otherwise invalid (such as attempting to seek to a position before the beginning of file). **off** is interpreted as a count relative to the place in the file specified by **dir** as described in *sbuf.pub(3C++)*. **m** is ignored. **p** is the position after movement, or EOF if a failure occurs. The position of the file after a failure is undefined.

**p=f.seekpos(pos,m)**

Moves the file to a position **pos** as described in *sbuf.pub(3C++)*. **m** is ignored. Normally **p** is **pos**, but on failure it is EOF.

**psb=f.setbuf(ptr,len)**

Sets up the reserve area as **len** bytes beginning at **p**. If **ptr** is null or **len** is less than or equal to 0, the **f** will be unbuffered. Normally **psb** is **&f**, but it is 0 if **f** is open, and in the latter case no changes are made to the reserve area or buffering status.

**i=f.sync()**

Attempts to force the state of get/put pointers of **f** to agree (be synchronized) with the state of the file **f.fd()**. This means it may write characters to the file if some have been buffered for output or attempt to reposition (seek) the file if characters have been read and buffered for input. Normally **i** is 0, but it is EOF if synchronization is not possible. Sometimes it is necessary to guarantee that certain characters are written together. To do this, the program should use **setbuf** (or a constructor) to guarantee that the reserve area is at least as large as the number of characters that must be written together. It can then do a **sync**, followed by storing the characters, followed by another **sync**.

**CAVEATS**

**attach** and the constructors should test if the file descriptor they are given is open, but I can't figure out a portable way to do that.

There is no way to force atomic reads.

Unix does usually report failures of seek (e.g. on a tty) and so a filebuf does not either.

**SEE ALSO**

*sbuf.pub(3C++)* *sbuf.prot(3C++)* *fstream(3C++)*

**NAME**

Finit, Frdline, Fgetc, Fread, Fseek, Fundo, Fputc, Fprint, Fwrite, Fflush, Ftie, Fclose, Fexit – fast buffered input/output

**SYNOPSIS**

```
#include <fio.h>

void Finit(fd, buf)
char *buf;

int Fclose(fd);

int Fprint(fildes, format [, arg ...])
int fildes;
char *format;

char *Frdline(fd)

int FIOLINELEN(fd)

long FIOSEEK(fd)

int Fgetc(fd)

void Fundo(fd)

long Fseek(fd, offset, ptr)
long offset;

int Fputc(fd, c)

long Fread(fd, addr, nbytes)
char *addr;
long nbytes;

long Fwrite(fd, addr, nbytes)
char *addr;
long nbytes;

int Fflush(fd)

void Ftie(ifd, ofd)

Fexit(type)
```

**DESCRIPTION**

These routines provide buffered I/O, faster than, and incompatible with *stdio(3)*. The routines can be called in any order. I/O on different file descriptors is independent.

*Finit* initializes a buffer (whose type is *Fbuffer*) associated with the file descriptor *fd*. Any buffered input associated with *fd* will be lost. The buffer can be supplied by the user (it should be at least **sizeof(Fbuffer)** bytes) or if *buf* is (**char \***)**0**, *Finit* will use *malloc(3)*. *Finit* must be called after a stretch of *non-fio* activity, such as *close* or *lseek(2)*, between *fio* calls on the same file descriptor number; it is unnecessary, but harmless, before the first *fio* activity on a given file descriptor number.

*Fclose* flushes the buffer for *fd*, *frees* the buffer if it was allocated by *Finit*, and then closes *fd*.

*Frdline* reads a line from the file associated with the file descriptor *fd*. The newline at the end of the line is replaced by a 0 byte. *Frdline* returns a pointer to the start of the line or (**char**) on end of file or read error. The macro *FIOLINELEN* returns the length (not including the 0 byte) of the most recent line returned by *Frdline*. The value is undefined after a call to any other *fio* routine.

*Fgetc* returns the next character from the file descriptor *fd*, or a negative value at end of file.

*Fread* reads *nbytes* of data from the file descriptor *fd* into memory starting at *addr*. The number of bytes read is returned on success and a negative value is returned if a read error occurred.

*Fseek* applies *lseek(2)* to *fd* taking buffering into account. It returns the new file offset. The macro *FIOSEEK* returns the file offset of the next character to be processed.

*Fundo* makes the characters returned by the last call to *Frdline* or *Fgetc* available for reading again.

There is only one level of undo.

*Fputc* outputs the low order 8 bits of *c* on the file associated with file descriptor *fd*. If this causes a *write* (see *read(2)*) to occur and there is an error, a negative value is returned. Otherwise, zero is returned.

*Fprint* is a buffered interface to *print(3)*. If this causes a *write* to occur and there is an error, a negative value is returned. Otherwise, the number of chars output is returned.

*Fwrite* outputs *nbytes* bytes of data starting at *addr* to the file associated with file descriptor *fd*. If this causes a *write* to occur and there is an error, a negative value is returned. Otherwise, the number of bytes written is returned.

*Fflush* causes any buffered output associated with *fd* to be written; it must precede a call of *close* on *fd*. The return is as for *Fputc*.

*Ftie* links together two file descriptors such that any *fio*-initiated *read(2)* on *ifd* causes a *Fflush* of *ofd* (if it has been initialized). It is appropriate for most programs used as filters to do **Ftie(0,1)**. The tie may be broken by **Ftie(ifd, -1)**.

*Fexit* is used to clean up all *fio* buffers. If *type* is zero, the buffers are *Fflushed*, otherwise they are *Fclosed*. **Fexit(0)** is automatically called at *exit(3)*.

## SEE ALSO

*open(2)*, *print(3)*, *stdio(3)*

## DIAGNOSTICS

*Fio* routines that return integers yield **-1** if *fd* is not the descriptor of an open file or if the operation is inapplicable to *fd*.

## BUGS

*Frdline* deletes characters from lines longer than 4095 characters, ignores characters after the last newline in a file, and will read past and end-of-file indication on a stream.

The data returned by *Frdline* may be overwritten by calls to any other *fio* routine.

*Fgetc* is much slower than access through a pointer returned by *Frdline*.

There is no *scanf(3)* analogue.

**NAME**

*fabs*, *fmod*, *floor*, *ceil* – absolute value, remainder, floor, ceiling functions

**SYNOPSIS**

```
#include <math.h>
```

```
double floor(x)
```

```
double x;
```

```
double ceil(x)
```

```
double x;
```

```
double fabs(x)
```

```
double x;
```

```
double fmod(x,y)
```

```
double x, y;
```

**DESCRIPTION**

*Fabs* returns the absolute value  $|x|$ .

*Floor* returns the largest integer not greater than  $x$ .

*Ceil* returns the smallest integer not less than  $x$ .

*Fmod* returns  $x$  if  $y$  is zero, otherwise the number  $f$  with the same sign as  $x$ , such that  $x = iy + f$  for some integer  $i$ , and  $|f| < |y|$ .

**SEE ALSO**

[\*arith\(3\)\*](#), [\*frexp\(3\)\*](#)



**NAME**

fopen, freopen, fdopen, fclose, fflush – open, close, or flush a stream

**SYNOPSIS**

```
#include <stdio.h>
```

```
FILE *fopen(filename, type)
```

```
char *filename, *type;
```

```
FILE *freopen(filename, type, stream)
```

```
char *filename, *type;
```

```
FILE *stream;
```

```
FILE *fdopen(fildes, type)
```

```
char *type;
```

```
int fclose(stream)
```

```
FILE *stream;
```

```
int fflush(stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Fopen* opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

*Type* is a character string having one of the following values:

[CB]"r" *open for reading*

[CB]"w" *create for writing*

[CB]"r+w"

[CB]"w+r" *open for reading and writing*

[CB]"a" *append: open for writing at end of file, or create for writing*

*Freopen* substitutes the named file in place of the open *stream*. It returns the original value of *stream*. The original stream is closed. *Freopen* is typically used to attach the preopened constant names `stdin`, `stdout` and `stderr` to specified files.

*Fdopen* associates a stream with a file descriptor. The *type* of the stream must agree with the mode of the open file.

*Fclose* causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

*Fclose* is performed automatically upon calling *exit(3)*.

*Fflush* causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

**SEE ALSO**

[open\(2\)](#), [popen\(3\)](#), [stdio\(3\)](#), [ferror\(3\)](#)

**DIAGNOSTICS**

*Fopen* and *freopen* return **NULL** if *filename* cannot be accessed.

*Fclose* and *fflush* return **EOF** if *stream* is not associated with a file, or if buffered data cannot be transferred to that file.

**NAME**

*fread*, *fwrite* – buffered binary input/output

**SYNOPSIS**

```
#include <stdio.h>
```

```
int fread(ptr, sizeof(*ptr), nitems, stream)
```

```
FILE *stream;
```

```
int fwrite(ptr, sizeof(*ptr), nitems, stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Fread* reads from the named input *stream* at most *nitems* of data of the type of *\*ptr* into a block beginning at *ptr*. It returns the number of items actually read.

*Fwrite* appends to the named output *stream* at most *nitems* of data of the type of *\*ptr* from a block beginning at *ptr*. It returns the number of items actually written.

**SEE ALSO**

[read\(2\)](#), [stdio\(3\)](#)

**DIAGNOSTICS**

*Fread* and *fwrite* return a short count upon end of file or error.

**BUGS**

Write errors from *fwrite* may be delayed until a subsequent [stdio\(3\)](#) writing, seeking, or file-closing call. These routines are much slower than you might imagine.

**NAME**

frexp, ldexp, modf – split into mantissa and exponent

**SYNOPSIS**

```
#include <math.h>
```

```
double frexp(value, eptr)
```

```
double value;
```

```
int *eptr;
```

```
double ldexp(value, exp)
```

```
double value;
```

```
double modf(value, iptr)
```

```
double value, *iptr;
```

**DESCRIPTION**

*Frexp* returns the mantissa of *value* and stores the exponent indirectly through *eptr*. If  $1/2 \leq |X| < 1$ , then  $x = \text{frexp}(x \times 2^{*eptr}, eptr)$ .

*Ldexp* returns the quantity  $value \times 2^{exp}$ .

*Modf* returns the positive fractional part of *value* and stores the integer part indirectly through *iptr*.

**DIAGNOSTICS**

On underflow *ldexp* returns 0; on overflow it returns a properly signed largest value. In both cases it sets *errno* to **ERANGE**.

**NAME**

*fseek*, *ftell*, *rewind* – reposition a stream

**SYNOPSIS**

```
#include <stdio.h>
```

```
int fseek(stream, offset, ptrname)
```

```
FILE *stream;
```

```
long offset;
```

```
long ftell(stream)
```

```
FILE *stream;
```

```
int rewind(stream)
```

**DESCRIPTION**

*Fseek* sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, as *ptrname* has the value 0, 1 or 2 respectively.

*Ftell* returns the current value of the file pointer for the file associated with the named *stream*.

*Rewind(stream)* is equivalent to **fseek(stream, 0L, 0)**.

**SEE ALSO**

[lseek\(2\)](#), [stdio\(3\)](#)

**DIAGNOSTICS**

*Fseek* returns -1 for improper seeks. *Ftell* returns -1 if seeking is impossible.

**BUGS**

The interaction of *fseek* and [ungetc\(3\)](#) is undefined.

**NAME**

`fstream` – `iostream` and `streambuf` specialized to files

**SYNOPSIS**

```
#include <fstream.h>
typedef long streamoff, streampos;
class ios {
public:
    enum        seek_dir { beg, cur, end };
    enum open_mode { in, out, ate, app };
    // and lots of other stuff ...
};
class ostream : ostream {
    ostream();
    ostream(char* name, open_mode mode, int prot=0664);
    ostream(int fd);
    ostream(int fd, char* p, int l);
    void        attach(int fd);
    void        close();
    void        open(char* name, open_mode, int=0664);
    filebuf*    rdbuf();
};
class ifstream : istream { same as ostream };
class fstream : iostream { same as ostream };
```

**DESCRIPTION**

`ifstream`, `ofstream` and `fstream` specialize `istream`, `ostream` and `iostream` (respectively) to files. That is, the associated `streambuf` will be a `filebuf`.

Assume

- **f** any of `ifstream`, `ofstream` or `fstream`.
- **pfb** is a `filebuf*`.
- **psb** is a `streambuf*`.
- **name** and **ptr** are `char*`.
- **i**, **fd**, **len** and **prot** are `int`.
- **mode** is an `open_mode`.

The constructors for `xstream`, where *x* is either `if`, `of` or `f`, are:

**xstream()**

Constructs an unopened `xstream`.

**xstream()(name,mode,prot)**

Constructs an `xstream` and tries to opens it using **name**, **mode**, and **prot**. The status of the constructed `xstream` will indicate failure in case the **open** fails.

**xstream()**

Constructs an `xstream` connected to file descriptor **d**, which must be previously opened.

**xstream(d,ptr,len)**

Constructs an `xstream` connected to file descriptor **fd** and in addition initializes the associated `filebuf` to use the **len** bytes at **ptr** as the reserve area. If **ptr** is null or **len** is 0, the `filebuf` will be unbuffered.

Member functions:

**f.attach(d)**

Connects **f** to the file descriptor **d**. A failure occurs when **f** is already connected to a file. A failure sets `failbit` in **f**'s error state.

**f.close()**

Closes any associated `filebuf` and thereby breaks the connection of the **f** to a file. `\fBf`'s error state is cleared except on failure. A failure occurs when the call to `f.rdbuf()->close` fails.

**f.open(name,mode,prot)**

Opens file **name** and connects **f** to it. If the file is created, it is created with protection mode **prot**. **open** normally returns 0, but it returns EOF on failure. Failure occurs if **f** is already open, or the call to **f.rdbuf()->open** fails. **failbit** is set in **f**'s error status on failure. The members of **open\_mode** are bits that may be or'ed together. The meaning of these bits in **mode** is

- app    A seek to the end of file is performed. Subsequent data written to the file is always added (appended) at the end of file. On some systems this is implemented in the kernel. In others it is implemented by seeking to the end of the file before each write. **app** implies output.
- ate    A seek to the end of the file is performed during the **open**. **ate** does not imply output.
- in     Implied by construction and opens of **istream**s. For **fstreams** it indicates that input operations should be allowed possible. It is legal to include **in** in the modes of an **ostream** in which case it implies that the original file (if it exists) should not be truncated.
- out    Implied by construction and opens of **ostream**s. For **fstream** it says that output operations are to be allowed.

**pfb=f.rdbuf()**

Returns a pointer to the associated **filebuf**. **fstream::rdbuf** has the same meaning as **istream::rdbuf** but is typed differently.

**psb=f.setbuf(p,len)**

Has the usual effect of a **setbuf**, offering space for a reserve area or requesting unbuffered I/O. Normally the returned **psb** is **f.rdbuf()**, but it is 0 on failure. A failure occurs if **f** is open or the call to **f.rdbuf()->setbuf** fails.

**SEE ALSO**

**filebuf(3C++) istream(3C++) ios(3C++) ostream(3C++) sbuf.pub(3C++)**

**NAME**

*ftw* – file tree walk

**SYNOPSIS**

```
#include <ftw.h>
```

```
int ftw(path, fn, depth)
char *path;
int (*fn)();
int depth;
```

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
fn(name, statb, code, S)
char *name;
struct stat *statb;
struct FTW *S;
```

**DESCRIPTION**

*Ftw* recursively descends the directory hierarchy rooted in *path*. For each entry in the hierarchy, *ftw* calls *fn*, passing it information about the entry: a pointer to a null-terminated pathname string, a pointer to a **stat** structure (see [stat\(2\)](#)), and a pointer to the following structure.

```
struct FTW {
    int quit;  see below
    int base;  &name[base] points to basename
    int level; recursion level (initially 0)
};
```

Possible values of *code*, defined in are

**FTW\_D**

Entry is a directory (before visiting descendants).

**FTW\_DP**

Entry is a directory (after visiting descendants).

**FTW\_SL**

Entry is a symbolic link.

**FTW\_F**

Entry is some other kind of file.

**FTW\_DNR**

Entry is a directory that cannot be read; no descendants will be visited.

**FTW\_NS**

*Lstat* (see [stat\(2\)](#)) failed on *name*; contents of *statb* are undefined

**FTW\_NSL**

*Lstat* succeeded, but *stat* failed; contents of *statb* are undefined.

The tree traversal continues until the tree is exhausted or *fn* returns a nonzero value. When the tree is exhausted, *ftw* returns zero. When *fn* returns a nonzero value, *ftw* stops and returns that value.

Normally symbolic links are not followed. But if on a symbolic link (**FTW\_SL**) *fn* sets **S->quit** to **FTW\_FOLLOW**, *ftw* will next attempt to follow the link.

*Ftw* normally visits a readable directory twice, before and after visiting its descendants. But if on a pre-visit (**FTW\_D**) *fn* sets **S->quit** to **FTW\_SKD**, *ftw* will skip the descendants and the postvisit (**FTW\_DP**).

*Ftw* uses one file descriptor for each level in the tree up to a maximum of *depth* (or 1, if *depth*<1) descriptors. *Depth* must not exceed the number of available file descriptors; small values of *depth* may cause *ftw* to run slowly, but will not change its effect.

**SEE ALSO**

[stat\(2\)](#), [directory\(3\)](#)

**DIAGNOSTICS**

*Ftw* returns `-1` with *errno* set to **ENOMEM** when [malloc\(3\)](#) fails.

*Errno* is set appropriately when *ftw* calls *fn* with code **FTW\_DNR**, **FTW\_NS**, or **FTW\_NSL**.



**NAME**

*galloc*, *gfree*, *garbage* – storage allocation with garbage collection

**SYNOPSIS**

**char \*galloc(n)**

**unsigned n;**

**void gfree(p)**

**char \*p;**

**void garbage()**

**DESCRIPTION**

These functions perform heap storage allocation with garbage collection.

*Galloc* allocates a block of at least *n* bytes and returns a pointer to it. *Gfree* frees a block previously allocated by *galloc*.

When space gets tight, garbage blocks are freed automatically. A block allocated by *galloc* is deemed to be garbage unless it is reachable. A reachable block is one whose first byte is pointed to by a declared C variable or by a pointer in a reachable block.

The frequency of garbage collection is controlled by external variables declared

**long gcmx = 5000, gcmin = 50;**

No more than *gcmx* allocations may intervene between automatic collections; this feature will help contain the growth of virtual address space. At least *gcmin* allocations must intervene, otherwise garbage collection will be abandoned as fruitless. *Garbage* may be called to do garbage collection at an arbitrary time.

*Malloc(3)* and *galloc* allocate from the same arena, but garbage collection affects only *galloc* blocks. *Free* (see *malloc(3)*) must not be used on blocks allocated with *galloc*.

**SEE ALSO**

[malloc\(3\)](#)

**DIAGNOSTICS**

*Galloc* returns 0 when space cannot be found.

**BUGS**

Garbage collection is conservative; blocks that appear to be pointed to from within declared storage will not be freed, regardless of whether the apparent ‘pointers’ were declared as such.

**NAME**

gamma – log gamma function

**SYNOPSIS**

```
#include <math.h>
```

```
double gamma(x)
```

```
double x;
```

```
extern int signgam;
```

**DESCRIPTION**

*Gamma* returns  $\ln |\Gamma(x)|$ . The sign of  $\Gamma(x)$  is returned in the external integer *signgam*.

**EXAMPLES**

Computation of the gamma function:

```
errno = 0;
y = gamma(x);
if(errno || (y > 88.0))
    error();
y = signgam*exp(y);
```

**DIAGNOSTICS**

A large value is returned for negative integer arguments and *errno* is set to **EDOM**.

**BUGS**

There should be a positive indication of error.

The name should indicate the answer is a logarithm, perhaps *lgamma*.

**NAME**

getarg, iargc – command arguments to Fortran

**SYNOPSIS**

**subroutine** *getarg*(**argno**, **string**)

**integer** *argno*

**character** *\*(\*) string*

**iargc**()

**DESCRIPTION**

These procedures permit Fortran programs to access the command arguments. The integer function *iargc* returns the number of command arguments. The subroutine *getarg* stores the *argno*th command argument in its second argument. The string is truncated or padded with blanks, in accord with the rules of Fortran character assignment.

**EXAMPLES**

a.out arg1 arg2 In a program invoked this way *iargc* will return 2.

character\*4 s

call *getarg*(2, s) Place arg2 in s.

**SEE ALSO**

[exec\(2\)](#)

**NAME**

`getc`, `getchar`, `fgetc`, `getw`, `putc`, `putchar`, `fputc`, `putw` – character- or word-at-a-time stream input/output

**SYNOPSIS**

```
#include <stdio.h>
```

```
int getc(stream)
```

```
FILE *stream;
```

```
int getchar()
```

```
int getw(stream)
```

```
FILE *stream;
```

```
int fgetc(stream)
```

```
FILE *stream;
```

```
int putc(c, stream)
```

```
char c;
```

```
FILE *stream;
```

```
putchar(c)
```

```
putw(w, stream)
```

```
FILE *stream;
```

```
fputc(c, stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Getc* returns the next character from the named input *stream*.

*Getchar*() is identical to *getc(stdin)*

*Getw* returns the next word (32-bit integer on a VAX) from the named input *stream*. *Getw* assumes no special alignment in the file.

*Putc* appends the character *c* to the named output *stream*. It returns the character written.

*Putchar*(*c*) is identical to *putc(c, stdout)*

*Putw* appends word (i.e. **int**) *w* to the output *stream*. It returns the word written. *Putw* neither assumes nor causes special alignment in the file.

*Fgetc* and *fputc* behave like *getc* and *putc*, but are genuine functions, not macros; they may be used to save object text.

The standard output stream **stdout** is normally buffered, but is flushed whenever *getc* causes a buffer to be refilled from **stdin**. The standard error stream **stderr** is normally unbuffered. These defaults may be changed by *setbuf*(3). When an output stream is unbuffered, information appears on the destination file or terminal as soon as written. When an output stream is buffered, many characters are saved up and written as a block. *Flush* may be used to force the block out early; see *fopen*(3).

**SEE ALSO**

[fopen\(3\)](#), [ungetc\(3\)](#), [stdio\(3\)](#)

**DIAGNOSTICS**

These functions return the integer constant **EOF** at end of file or error. For *getw* or *putw* this indication is ambiguous; *ferror*(3) may be used to distinguish.

**BUGS**

Because they are implemented as macros, *getc* and *putc* treat *stream* arguments with side effects incorrectly. For example, **getc(\*f++)** is wrong.

The routines in *printf*(3) provide temporary buffering even when buffering has been turned off.

Write errors may be delayed until a subsequent *stdio* (3) writing, seeking, or file-closing call.

**NAME**

getdate – convert time and date from ASCII

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <sys/timeb.h>
```

```
time_t getdate(buf, now)
```

```
char *buf;
```

```
struct timeb *now;
```

**DESCRIPTION**

*Getdate* converts common time specifications to standard UNIX format. *Buf* is a character string containing the time and date. *Now* is the assumed current time (used for relative specifications); if the pointer is null (0) *ftime* (see [time\(2\)](#)) is used to obtain the current time and timezone.

The character string consists of 0 or more specifications of the following form:

**tod**     A *tod* is a time of day, in the form *hh:mm[:ss]* (or *hhmm*) [*meridian*] [*zone*]. If no meridian – **am** or **pm** – is specified, a 24-hour clock is used. A *tod* may be specified as just *hh* followed by a *meridian*.

**date**     A *date* is a specific month and day, and possibly a year. Acceptable formats are *mm/dd[/yy]* and *monthname dd[, yy]*. If omitted, the year defaults to the current year; if a year is specified as a number less than 100, 1900 is added. If a number not followed by a day or relative time unit occurs, it will be interpreted as a year if a *tod*, *monthname*, and *dd* have already been specified; otherwise, it will be treated as a *tod*. This rule allows the output from [date\(1\)](#) or [ctime\(3\)](#) to be passed as input to *getdate*.

**day**     A *day* of the week may be specified; the current day will be used if appropriate. A *day* may be preceded by a *number*, indicating which instance of that day is desired; the default is **1**. Negative *numbers* indicate times past. Some symbolic *numbers* are accepted: **last**, **next**, and the ordinals **first** through **twelfth** (**second** is ambiguous, and is not accepted as an ordinal number). The symbolic number **next** is equivalent to **2**; thus, *next monday* refers not to the immediately coming Monday, but to the one a week later.

**relative time**

Specifications relative to the current time are also accepted. The format is [*number*] *unit*; acceptable units are **year**, **month**, **fortnight**, **week**, **day**, **hour**, **minute**, and **second**.

The actual date is formed as follows: first, any absolute date and/or time is processed and converted. Using that time as the base, day-of-week specifications are added; last, relative specifications are used. If a date or day is specified, and no absolute or relative time is given, midnight is used. Finally, a correction is applied so that the correct hour of the day is produced after allowing for daylight savings time differences.

*Getdate* accepts most common abbreviations for days, months, etc.; in particular, it will recognize them with upper or lower case first letter, and will recognize three-letter abbreviations for any of them, with or without a trailing period. Units, such as **weeks**, may be specified in the singular or plural. Timezone and meridian values may be in upper or lower case, and with or without periods.

**FILES**

/usr/lib/libu.a

**SEE ALSO**

[ctime\(3\)](#), [time\(2\)](#)

**AUTHOR**

Steven M. Bellovin (unc!smb)

Dept. of Computer Science

University of North Carolina at Chapel Hill

**BUGS**

Because [yacc\(1\)](#) is used to parse the date, *getdate* cannot be used a subroutine to any program that also needs *yacc*.

The grammar and scanner are rather primitive; certain desirable and unambiguous constructions are not

accepted. Worse yet, the meaning of some legal phrases is not what is expected; *next week* is identical to *2 weeks*.

The daylight savings time correction is not perfect, and can get confused if handed times between midnight and 2:00 am on the days that the reckoning changes.

Because *localtime(2)* accepts an old-style time format without zone information, attempting to pass *getdate* a current time containing a different zone will probably fail.

**NAME**

getenv – value for environment name

**SYNOPSIS**

**char \*getenv(name)**

**char \*name;**

**DESCRIPTION**

*Getenv* searches the environment list (see [environ\(5\)](#)) for a string starting with *name[CB]=*. If no such a string is found, 0 is returned. Otherwise, the address of the character following the = is returned.

**SEE ALSO**

[printenv\(1\)](#), [environ\(5\)](#), [exec\(2\)](#)

**BUGS**

*Getenv* ignores shell functions; see [sh\(1\)](#).

**NAME**

getfields, getmfields, setfields – break a string into fields

**SYNOPSIS**

```
int getfields(str, ptrs, nptrs)
```

```
char *str, **ptrs;
```

```
int getmfields(str, ptrs, nptrs)
```

```
char *str, **ptrs;
```

```
char *setfields(fielddelim)
```

```
char *fielddelim;
```

**DESCRIPTION**

*Getfields* breaks the null-terminated string *str* into at most *nptrs* null-terminated fields and places pointers to the start of these fields in the array *ptrs*. It returns the number of fields and terminates the list of pointers with a zero pointer. It overwrites some of the bytes in *str*. If there are *nptr* or more fields, the list will not end with zero and the last ‘field’ will extend to the end of the input string and may contain delimiters.

A field is defined as a maximal sequence of characters not in a set of field delimiters. Adjacent fields are separated by exactly one delimiter. No field follows a delimiter at the end of string. Thus a string of just two delimiter characters contains two empty fields, and a nonempty string with no delimiters contains one field.

*Getmfields* is the same as *getfields* except that fields are separated by maximal strings of field delimiters rather than just one.

*Setfields* makes the field delimiters (space and tab by default) be the characters of the string *fielddelim* and returns a pointer to a string of the previous delimiters.

**EXAMPLES**

Print the words in a string, where words are non-whitespace strings. There is no bound on the number of words.

```
printwords(string)
char *string;
{
    char *ptrs[2];
    int n;
    setfields(" \t\n");
    for(n=2; n>=2; string=ptrs[1]) {
        n = getmfields(string, ptrs, 2);
        if(n == 0)
            break;
        if(ptrs[0][0] != 0) /* skip initial blanks */
            printf("%s\n", ptrs[0]);
    }
}
```

**SEE ALSO**

[string\(3\)](#)



**NAME**

getflags – process flag arguments in argv

**SYNOPSIS**

```
#include </usr/include/getflags.h>

int getflags(argc, argv, flags)
char **argv, *flags;

usage(tail)
char *tail;

extern char **flag[], cmdline[], *cmdname, *flagset[];
```

**DESCRIPTION**

*Getflags* digests an argument vector *argv*, finding flag arguments listed in *flags*. *Flags* is a string of flag letters. A letter followed by a colon and a number is expected to have the given number of parameters. A flag argument starts with '-' and is followed by any number of flag letters. A flag with one or more parameters must be the last flag in an argument. If any characters follow it, they are the flag's first parameter. Otherwise the following argument is the first parameter. Subsequent parameters are taken from subsequent arguments.

The global array *flag* is set to point to an array of parameters for each flag found. Thus, if flag **-x** was seen, **flag['x']** is non-zero, and **flag['x'][i]** is the flag's *i*th parameter. If flag **-x** has no parameters **flag['x']==flagset**. Flags not found are marked with a zero. Flags and their parameters are deleted from *argv*. *Getflags* returns the adjusted argument count.

*Getflags* stops scanning for flags upon encountering a non-flag argument, or the argument --, which is deleted.

*Getflags* places a pointer to *argv[0]* in the external variable *cmdname*. It also concatenates the original members of *argv*, separated by spaces, and places the result in the external array *cmdline*.

*Usage* constructs a usage message, prints it on the standard error file, and exits with status 1. The command name printed is *argv[0]*. Appropriate flag usage syntax is generated from *flags*. As an aid, explanatory information about flag parameters may be included in *flags* in square brackets as in the example. *Tail* is printed at the end of the message. If *getflags* encountered an error, *usage* tries to indicate the cause.

**EXAMPLES**

```
main(argc, argv)
char *argv[];
{
    if((argc=getflags(argc, argv, "vinclbhse:1[expr]", 1))==-1)
        usage("[file ...]");
}
might print:
Illegal flag -u
Usage: grep [-vinclbhs] [-e expr] [file ...]
```

**SEE ALSO**

[getopt\(3\)](#)

**DIAGNOSTICS**

*Getflags* returns -1 on error: a syntax error in *flags*, setting a flag more than once, setting a flag not mentioned in *flags*, or running out of argv while collecting a flag's parameters.

**NAME**

getflds – read a line from a file and break it into fields

**SYNOPSIS**

```
#include <stdio.h>
```

```
char **getflds(stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Getflds* reads a line from the *stream* given as argument, breaks it into fields, and returns a pointer to a null-terminated array of character pointers, each of which points to a null-terminated string representing one field. These strings, and the array that points to them, persist until the next call to *getflds*. On end of file, a null pointer is returned.

If the first character of a line is a #, that line is considered to be a comment and ignored.

Fields are separated by white space (spaces or tabs). Leading and trailing white space on a line is ignored. White space may appear inside a field in one of three ways: each space or tab may be preceded by a backslash, the white space may be enclosed in single or double quotes, or characters may be described in octal, as detailed below.

A field, or any parts of a field, may be enclosed in single or double quotes. Within quotes, white space and quotes of the other sort are treated as ordinary characters, but a closing quote is silently inserted before a newline.

Inside or outside quotes, a backslash and the character(s) after it are changed as follows:

```
[CB]\b    backspace
[CB]\f    form feed
[CB]\n    newline
[CB]\r    return
[CB]\t    horizontal tab
[CB]\v    vertical tab
[CB]\\    \
[CB]\`    `
[CB]"    "
[CB]\#    #
[CB]\space space
[CB]\tab  tab
```

[CB]*newline* completely ignored; this allows a logical line to span any number of physical lines.

[CB]*digits* the one, two, or three octal digits are the value of the character to be used.

If any other character follows the \, both characters lose their special interpretation.

**DIAGNOSTICS**

If *getflds* detects an error, it returns 0, just as it does at end of file. However, the error will have changed *errno* (see [intro\(2\)](#)). To distinguish between error and end of file set *errno* to zero before calling *getflds* and test it afterwards.

**BUGS**

*Getflds* provides no way to distinguish a null character within a field from the end of the field.

**NAME**

getfsent, getfsspec, getfsfile, setfsent, endfsent – get file system description file entry

**SYNOPSIS**

```
#include <fstab.h>

struct fstab *getfsent()

struct fstab *getfsspec(name)
char *name;

struct fstab *getfsfile(name)
char *name;

int setfsent()

int endfsent()
```

**DESCRIPTION**

*Getfsent*, *getfsspec* and *getfsfile* each return a pointer to a structure containing the broken-out fields of a line in *fstab(5)*, which describes mountable file systems.

```
struct fstab {
    char fs_spec[FSNMLG]; block device name
    char fs_file[FSNMLG]; file system mount point
    int fs_fstype;        file system type
    int fs_flags;        file system flags
    int fs_passno;       pass number for parallel fsck(8)
};
```

Type numbers and flags are listed in *fmount(2)*. Entries that aren't file systems (should not be mounted) have negative values for *fs\_fstype*:

**FSSWAP** (-2) *fs\_spec* is a device available for swapping.

*Getfsent* reads the next line of the file, opening the file if necessary.

*Setfsent* opens and rewinds the file.

*Endfsent* closes the file.

*Getfsspec* and *getfsfile* sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered.

**FILES**

/etc/fstab

**SEE ALSO**

*fmount(2)*, *fstab(5)*

**DIAGNOSTICS**

Zero is returned on EOF or error.

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry

**SYNOPSIS**

```
#include <grp.h>

struct group *getgrent();
struct group *getgrgid(gid)
struct group *getgrnam(name)
char *name;
int setgrent();
int endgrent();
```

**DESCRIPTION**

*Getgrent*, *getgrgid* and *getgrnam* each return pointers to a structure containing the broken-out fields of a line in

```
struct group {
    [CB]char    *gr_name;the group name
    [CB]char    *gr_passwd;the encrypted group passwd
    [CB]int     gr_gid;the numeric groupid
    [CB]char    **gr_mem;null-terminated vector of pointers to the individual
    [CB]};
```

*Getgrent* simply reads the next line while *getgrgid* and *getgrnam* search until a matching *gid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

**FILES**

/etc/group

**SEE ALSO**

[getlogin\(3\)](#), [getpwent\(3\)](#), [passwd\(5\)](#)

**DIAGNOSTICS**

Zero is returned on EOF or error.

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

getlogin – get login name

**SYNOPSIS**

**char \*getlogin()**

**DESCRIPTION**

*Getlogin* returns a pointer to the login name as set by *setlogname*; see [getuid\(2\)](#).

It is preferable, but less portable, to call *getlogname*; see [getuid\(2\)](#).

**FILES**

/etc/utmp

**SEE ALSO**

[getpwent\(3\)](#), [getgrent\(3\)](#), [utmp\(5\)](#), [getuid\(2\)](#)

**DIAGNOSTICS**

Zero is returned if the name could not be found.

**BUGS**

The return values point to static data whose content is overwritten by each call.

**NAME**

getopt – get option letter from argv

**SYNOPSIS**

```
int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind;
extern int opterr;
```

**DESCRIPTION**

*Getopt* returns the next option letter in *argv* that matches a letter in *optstring*. *Optstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument, which may or may not be separated from it by white space. *Optarg* is set to point to the start of the option argument, if any.

*Optind*, initially 1, holds the index in *argv* of the next argument to be processed. When *opterr* is nonzero (the default state), errors cause diagnostic messages.

Option letters appear in nonempty clusters preceded by -. The special option -- may be used to mark the end of the options.

**EXAMPLES**

This fragment processes arguments for a command that can take option **a** and option **f**, which requires an argument.

```
main (argc, argv) char **argv;
{
    int c, errflg = 0;
    extern int optind;
    extern char *optarg, *ifile;
    while((c = getopt(argc, argv, "af:")) != -1)
        switch (c){
            case 'a':  aflag=1; break;
            case 'f':  ifile = optarg; break;
            case '?':  errflg=1; break;
        }
    if(errflg){
        fprintf(stderr, "usage: . . . ");
        exit(2);
    }
    for( ; optind < argc; optind++){
        if(access(argv[optind], 4)){
            ...
        }
    }
    ...
}
```

**SEE ALSO**

[getflags\(3\)](#)

**DIAGNOSTICS**

When all options have been processed, -1 is returned; *optind* refers to the first non-option argument.

When *getopt* encounters an option letter not included in *optstring* or finds an option argument missing, it prints a diagnostic on **stderr** under control of *opterr* and returns a question mark ?.

**NAME**

getpass – read a password

**SYNOPSIS**

```
char *getpass(prompt)
char *prompt;
```

**DESCRIPTION**

*Getpass* reads a password from the file or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

**FILES**

/dev/tty

**SEE ALSO**

[crypt\(3\)](#)

**BUGS**

The return value points to static data whose content is overwritten by each call.

**NAME**

getpwent, getpwuid, getpwnam, setpwent, endpwent, pwdecode – get password file entry

**SYNOPSIS**

```
#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()

struct passwd *pwdecode(p)
char *p;
```

**DESCRIPTION**

*Getpwent*, *getpwuid* and *getpwnam* each return a pointer to a structure containing the broken-out fields of a line in

```
struct passwd {
    [CB]char    *pw_name;login name
    [CB]char    *pw_passwd;encrypted password
    [CB]int     pw_uid;numeric userid
    [CB]int     pw_gid;numeric groupid
    [CB]int     pw_quota;unused
    [CB]char    *pw_comment;unused
    [CB]char    *pw_gecos;field for local use
    [CB]char    *pw_dir;login directory
    [CB]char    *pw_shell;program to use as Shell
    [CB]};
```

*Getpwent* reads the next line (opening the file if necessary); *setpwent* rewinds the file; *endpwent* closes it.

*Getpwuid* and *getpwnam* search from the beginning until a matching *uid* or *name* is found (or until end-of-file is encountered).

*Pwdecode* breaks out a null-terminated character string *p* containing a password file entry. The input string is modified by the call and the output structure contains pointers into it.

**FILES**

/etc/passwd

**SEE ALSO**

[getlogin\(3\)](#), [getgrent\(3\)](#), [passwd\(5\)](#)

**DIAGNOSTICS**

These routines return 0 for end of file or error.

**BUGS**

The return values point to static data whose content is overwritten by each call.



**NAME**

getshares – get shares file entry given uid

**SYNOPSIS**

```
#include <shares.h>
```

```
unsigned long getshares(lp, uid, lock)
```

```
struct lnode * lp;
```

```
int uid;
```

```
int lock;
```

**DESCRIPTION**

*Getshares* finds the shares entry with the same uid as *uid* and reads it into an area pointed to by *lp*. *lock* should be non-zero if the shares data-base should be opened for writing. Returns zero if no entry exists, (in which case all non-*uid* fields of *\*lp* will have been cleared), or the “last used” time of the entry. *lp->L\_uid* will always have been set to *uid* on return.

**SEE ALSO**

closeshares(3), getshput(3), openshares(3), putshares(3), sharesfile(3).

**DIAGNOSTICS**

*Getshares* returns 0 if entry can't be found, or on error.

**NAME**

getshput – read, modify, and write shares file entry

**SYNOPSIS**

```
#include <shares.h>
```

```
unsigned long getshput(lp, func)
```

```
struct lnode * lp;
```

```
unsigned long (*func)();
```

**DESCRIPTION**

*Getshput* finds the shares entry with the same uid as *lp*→*l\_uid* and reads it into an internal *lnode* structure. The function pointed to by *func* is then called with two arguments, the first pointing to the original *lnode* structure, and the second pointing to the internal *lnode* structure. If *func* returns with a value other than 0 the (presumably modified) second structure is written back to the shares file, with the “extime” field in the shares record filled with the value returned by *func*. Otherwise *getshput* simply returns. *Getshput* returns the value returned by *func*.

**SEE ALSO**

closeshares(3), getshares(3), openshares(3), putshares(3), sharesfile(3).

**DIAGNOSTICS**

*Getshput* returns 0 if the entry can't be modified.

**NAME**

*getwd*, *getcwd* – get current directory

**SYNOPSIS**

**char \**getwd*(*buf*)**

**char \**buf*;**

**char \**getcwd*(*buf*, *size*)**

**char \**buf*;**

**DESCRIPTION**

*Getwd* and *getcwd* fill *buf* with a null-terminated string representing the current directory and return *buf*.

*Getwd* is in the style of BSD systems and *getcwd* in that of System V. If *buf* is 0, *getcwd* will call [malloc\(3\)](#) to allocate *size* bytes for the buffer.

**SEE ALSO**

[pwd\(1\)](#)

**DIAGNOSTICS**

On error, zero is returned and *buf* is filled with a diagnostic message.

**NAME**

`huff` – huffman codebook/tree generator

**SYNOPSIS**

```
#include <huff.h>
```

```
NODE *huff(inrout)
```

```
int (*inrout)();
```

**DESCRIPTION**

*Huff* generates a binary Huffman codebook. It obtains a list of messages one at a time from an input routine, *inrout*, declared as

```
int inrout(str, p)
char ** str;
float *p;
```

*Inrout* makes *\*str* point to a null-terminated string identifying a message, and places in *\*p* the (arbitrarily normalized) frequency of the message. *Inrout* returns non-zero when data is returned and zero when there is no more data.

*Huff* returns a pointer to a root of type **NODE**:

```
typedef struct node {
    char *datump;
    struct node *to;
    struct node *from;
    struct node *ldad;
    struct node *rdad;
    struct node *kid;
    float prob;
} NODE;
```

The root heads a linked list and the Huffman tree. The doubly linked list, connected via **from** and **to**, is ordered as the codebook was generated. The tree is connected via **kid**, **ldad**, and **rdad**, with null pointers at the various ends. The **kid** field points towards the root, **ldad** and **rdad** point away: **node->ldad->kid==node** and **node->rdad->kid==node**. the **datump** field is null or points to a message identifier.

The codeword for a message may be read off from the path from the root to the node containing the message identifier, counting *ldad* branches as 0 and *rdad* branches as 1.

**BUGS**

A code with only one message dumps core.

**NAME**

hypot, cabs – euclidean distance

**SYNOPSIS**

```
#include <math.h>
```

```
double hypot(x, y)
```

```
double x, y;
```

```
double cabs(z)
```

```
struct { double x, y; } z;
```

**DESCRIPTION**

*Hypot* and *cabs* return  $\sqrt{x*x + y*y}$ , taking precautions against unwarranted overflows.

**SEE ALSO**

[exp\(3\)](#)

**NAME**

`in_host`, `in_ntoa`, `in_address`, `in_service` – internet networking functions

**SYNOPSIS**

```
#include <sys/inet/in.h>

char *in_host(hostaddr)
in_addr hostaddr;

char *in_ntoa(hostaddr)
in_addr hostaddr;

in_addr in_address(hostname)
char *hostname;

struct in_service *in_service(name, proto, port)
char *name, *proto;
unsigned long port;
```

**DESCRIPTION**

These routines are loaded by the **-lin** option of *ld(1)*.

Internet addresses, type *in\_addr*, are 32-bit quantities global to the network. The ASCII representation of an *in\_addr* can be either a host name or of the form *b1.b2.b3.b4*, where each '*bx*' is the value of the *x*'th byte of the address in decimal. Since host names are considered local 'aliases' for internet addresses, the host-to-address mapping is subjective.

*In\_address* maps an internet host name to an address and returns 0 if the name is not found in the host table.

*In\_host* maps an internet address into a host name. If the host is not found in the host table, the ASCII representation of the address is returned.

*In\_ntoa* maps an internet address to its ASCII numeric format.

*In\_service* returns the closest match to *name* in the services file. If either *name* or *port* are 0, they will match any name or port. If *proto* is (**char \***)0, the **tcp** protocol is assumed.

**FILES**

[CB]/usr/inet/lib/hosts	<b>mapping between host names and addresses</b>
[CB]/usr/inet/lib/networks	mapping between network names and addresses
[CB]/usr/inet/lib/services	<b>database of services</b>
[CB]/usr/inet/lib/hosts.equiv	machines with common administration

**SEE ALSO**

*ipc(3)*, *tcp(3)*, *udp(3)*

**BUGS**

The mappings between internet addresses and names is arbitrary at best. The hosts file may contain many addresses for each name and/or many names for each address. *In\_address* and *in\_host* each start at the beginning of the file and search sequentially for a match. Therefore, **in\_addr(in\_host(addr)) == addr** is not necessarily true.

**NAME**

ios – input/output formatting

**SYNOPSIS****#include <iostream.h>****struct fmtinfo {**

<b>ostream*</b>	<b>tie;</b>
<b>short</b>	<b>convbase;</b>
<b>short</b>	<b>width;</b>
<b>short</b>	<b>precision;</b>
<b>char</b>	<b>fill;</b>
<b>char</b>	<b>lajust;</b>
<b>char</b>	<b>showbase;</b>
<b>char</b>	<b>skip;</b>
<b>char</b>	<b>floatfmt;</b>

**};****class ios {****public:**

<b>enum</b>	<b>io_state { goodbit=0, eofbit, failbit, badbit };</b>
<b>enum</b>	<b>open_mode { in, out, ate, app };</b>
<b>enum</b>	<b>seek_dir { beg, cur, end };</b>

**public:**

	<b>ios(streambuf* b);</b>
<b>int</b>	<b>bad();</b>
<b>void</b>	<b>clear(state_value i = _good);</b>
<b>int</b>	<b>convbase();</b>
<b>void</b>	<b>convbase(int i);</b>
<b>int</b>	<b>eof();</b>
<b>int</b>	<b>fail();</b>
<b>char</b>	<b>fill();</b>
<b>void</b>	<b>fill(char c);</b>
<b>char</b>	<b>floatfmt();</b>
<b>void</b>	<b>floatfmt(char c);</b>
<b>fmtinfo</b>	<b>fmt();</b>
<b>void</b>	<b>fmt(fmtinfo&amp; info);</b>
<b>int</b>	<b>lajust();</b>
<b>void</b>	<b>lajust(int a);</b>
<b>int</b>	<b>good();</b>
<b>int</b>	<b>operator!();</b>
	<b>operator int();</b>
<b>int</b>	<b>popfmt();</b>
<b>int</b>	<b>precision();</b>
<b>void</b>	<b>precision(int i);</b>
<b>void</b>	<b>pushfmt();</b>
<b>streambuf*</b>	<b>rdbuf();</b>
<b>int</b>	<b>rdstate();</b>
<b>int</b>	<b>showbase();</b>
<b>void</b>	<b>showbase(int i);</b>
<b>int</b>	<b>skip();</b>
<b>int</b>	<b>skip(int i);</b>
<b>void</b>	<b>sync_with_stdio();</b>
<b>streampos</b>	<b>tellg();</b>
<b>ostream*</b>	<b>tie();</b>
<b>ostream*</b>	<b>tie(ostream* iosp);</b>
<b>int</b>	<b>width();</b>
<b>void</b>	<b>width(int i);</b>
<b>ios&amp;</b>	<b>operator&lt;&lt;(ios&amp; (*)(ios&amp;));</b>
<b>ios&amp;</b>	<b>operator&gt;&gt;(ios&amp; (*)(ios&amp;));</b>

```

private:
    iosstream&      iosstream(iostream&) ;
                   operator=(iosstream&);
};
class ios_withassign : ios {
    ios_withassign();
    ios&           operator=(ios&);
    ios&           operator=(streambuf*);
};
ios&              dec(ios& i) ;
ios&              hex(ios& i) ;
ios&              oct(ios& i) ;
ios&              popfmt(ios& i) ;
ios&              pushfmt(ios& i) ;

```

## DESCRIPTION

The stream classes derived from `ioss` provide a high level interface that supports transferring formatted and unformatted information into and out of `streambufs`.

In the following assume:

- **s** is an `ios`.
- **swa** is an `ios_withassign`.
- **sp** is a `ios*`.
- **i** and **n** are `int`.
- **c** is a `char`.
- **osp** is an `ostream*`.
- **sb** is a `streambuf*`.
- **pos** is a `streampos`.
- **off** is a `streamoff`.
- **info** is a `formatinfo`.
- **dir** is a `seek_dir`.
- **mode** is a `seek_dir`.
- **fcn** is a function with type `ios& (*)(ios&)`.

Constructors and assignment:

### **ios(sb)**

**sb** becomes the `streambuf` associated with the constructed **ios**. This association is fixed for the life of the `ios`. If **sb** is null the effect is undefined.

### **ios\_withassign()**

Uninitialized variable.

### **ios(ios&)**

#### **ios=ios**

Copying of `ios`'s is not in general well defined and the constructor and assignment operators are made private so that the compiler will complain about attempts to do so. Usually what is desired is copying of pointers to `iostreams`.

#### **swa=sb**

Associates **sb** with **swa** and initializes the entire state of **swa**.

**swa=s** Associates `s->rdbuf()` with **swa** and initializes the entire state of **swa**.

An **ios** has an internal error state (which is a collection of the bits declared as `io_states`). Members related to the error state:

#### **i=s.rdstate()**

Returns the current error state.

#### **s.clear(i)**

Stores **i** as the error state. If **i** is zero this clears all bits. To set a bit without clearing previously set bits requires something like `s.clear(badbit|s.rdstate())`.



**i=s** The `int` conversion operator is non-zero if `badbit` or `failbit` is set in the error state.

**i=s.good()**  
Non-zero if the error state has no bits set.

**i=s.eof()**  
Non-zero if `eofbit` is set in the error state. Normally this bit is set when an end of file has been encountered doing extraction.

**i=s.fail()**  
Non-zero if either `badbit` or `failbit` are set in the error state. Normally this indicates that some operation has failed.

**i=s.bad()**  
Non-zero if **badbit** is set in the error state. Normally this indicates that some operation on `s.rdbuf()` has failed.

An `ios` has a collection of format state variables that are used by input and output operations to control the details of formatting operations. Otherwise their values have no particular effect and they may be set and examined arbitrarily by user code.

**s.convbase(i)**  
Sets the "conversion base" format state variable to **i**.

**i=s.convbase()**  
Returns the "conversion base" format state variable.

**s.fill(c)**  
Sets the "fill character" format state variable to **c**.

**c=s.fill()**  
Returns the "fill character" format state variable.

**s.fill(c)**  
Sets the "fill character" format state variable to **c**.

**c=s.fill()**  
Returns the "fill character" format state variable.

**s.floatfmt(c)**  
Sets the "floating format" format state variable to **c**.

**c=s.floatfmt()**  
Returns the "floating format" format state variable.

**s.ladjust(i)**  
Sets the "left adjustment" format state variable to **i**.

**i=s.ladjust()**  
Returns the "left adjustment" format state variable.

**s.precision(i)**  
Sets the "precision" format state variable to **i**.

**i=s.precision()**  
Returns the "precision" format state variable.

**s.showbase(i)**  
Sets the "show explicit base" format state variable to **i**.

**i=s.showbase()**  
Returns the "show explicit base" format state variable.

**sp=s.skip(sp)**  
Sets the "skip whitespace" format state variable to **sp**.

**sp=s.skip()**  
Returns the "skip whitespace" format state variable.

**osp=s.tie(osp)**  
Sets the "tie" format state variable **sp**.

**osp=s.tie()**  
Returns the "tie" format state variable.

**s.width(i)**  
Sets the "field width" format state variable to **i**.

**i=s.width()**  
Returns the "field width" format state variable.

**s.fmt(info)**  
Sets the collection of all format state variables to **info** in a single operation.

**info=s.fmt()**  
Returns the collection of all format state variables in a single operation.

**s.pushfmt()**  
Pushes a `formatinfo` structure onto a (dynamically allocated) stack associated with **s**. This copies the current values of format state variables without changing them.

**i=s.popfmt()**  
Pops the topmost `formatinfo` from the stack associated with **s** and sets the format state variables accordingly. Normally **i** is non-zero, but it will be zero if the stack is empty (i.e., more **popfmt** than **pushfmt** operations have been performed).

Other members:

**sb=s.rdbuf()**  
Returns a pointer to the `streambuf` associated with **s** when **s** was constructed.

**((ios\*)0)->sync\_with\_stdio()**  
Solves problems that arise when mixing `stdio` and `iostreams`. The first time it is called it will reset the standard `iostreams` (`istream`, `ostream`, `errstream`, `logstream`) to be streams using `stdiobufs`. After that input and output using these streams may be mixed with input and output using the corresponding `FILEs` and will be properly synchronized.

Convenient manipulators (functions that take an `ios&` and return their argument).

**ios<<dec**  
**ios<<dec**  
**ios>>dec**  
Sets the convbase to 10.

**ios<<hex**  
**ios>>hex**  
Sets the convbase to 16.

**ios<<oct**  
**ios>>oct**  
Sets the convbase to 8.

**ios<<popfmt**  
**ios>>popfmt**  
Does **s.popfmt()**

**ios<<pushfmt**  
**ios>>pushfmt**  
Does **s.pushfmt()**

The `streambuf` associated with an `ios` may be manipulated by other methods than through `ios`. For example, characters may be stored in a queue-like `streambuf` through an `ostream` while they are being fetched through an `istream`. Or for efficiency some part of a program may choose to do `streambuf` gets directly rather than through the `ios`. In most cases the program does not have to worry about

this possibility, because an `ios` never saves information about the internal state of a `streambuf`. For example if the `streambuf` is repositioned between extraction operations the extraction (input) will proceed normally.

### CAVEATS

The effect of `ios.sync_with_stdio()` does not depend on `ios`. `sync_with_stdio` ought to be a global function but it is a member of `iostream` to avoid name space pollution. The need for `sync_with_stdio` is a wart. The old stream package did this as a default, but in the `iostream` package unbuffered `stdiobufs` are too inefficient to be the default.

The stream package had a constructor that took a **FILE\*** argument. This is now replaced by `stdiostream`. It is not declared even as an obsolete form to avoid having `iostream.h` depend on `stdio.h`.

The old stream package allowed copying of streams. This is disallowed by the `iostream` package. Old code using copying can usually be rewritten to use pointers and copy pointers.

For compatibility with the old stream package, the versions of `tie` and `skip` that set the state variables also return the old value.

### SEE ALSO

IOS.INTRO(3C++) `streambuf(3C++)` `istream(3C++)` `ostream(3C++)`

**NAME**

ipccreat, ipcopen, ipclisten, ipcaccept, ipcreject, ipcexec, ipcpath, ipclogin, ipclogin – set up connections between processes or machines

**SYNOPSIS**

```
#include <ipc.h>

char *ipcpath(name, network, service)
char *name;
char *network;
char *service;

int ipcopen(name, param)
char *name;
char *param;

int ipccreat(name, param)
char *name;
char *param;

ipcinfo *ipclisten(fd)
int fd;

int ipcaccept(ip)
ipcinfo *ip;

int ipcreject(ip, no, str)
ipcinfo *ip;
int no;
char *str;

int ipcexec(name, param, cmd)
char *name;
char *param;
char *cmd;

int ipclogin(fd)
int fd;

int ipclogin(fd, opt)
int fd;
char *opt;

extern char *errstr;
```

**DESCRIPTION**

These routines establish communication between unrelated processes, often for networking purposes. They are loaded by the **-lipc** option of *ld(1)*.

End points in the network are identified by names of the form: **element[!element]...**. The name is translated element by element relative to the name space selected by the previous element. The first element is always a name in the local file system. By convention, all network interfaces and services mount themselves in For example:

*/cs/exec*

refers to a local process which has mounted itself (via *ipccreat*) on

*/cs/dk!nj/astro/voice*

refers to a voice synthesizer attached to Datakit; process */cs/dk* is the Datakit interface.

*/cs/dk!dutoit!exec*

is the process that has mounted itself on */cs/exec* in machine 'dutoit'.

*Ipcpath*, forms a network name from its arguments and returns a pointer to it. It takes three arguments: the destination *name*, the default *network*, and the default *service*. It assumes that *name* is a three part name of the form: *network!host!service*. If either *network* or *service* is missing from *name*, *ipcpath* supplies them from the default arguments. It then tacks a */cs* on the front and returns a pointer to that.

Thus,

```
ipcpath("dutoit", "dk", "dcon")
```

returns a pointer to the string `/cs/dk!dutoit!dcon`.

*Ipcopen* places a call to a named network end point and returns a file descriptor for the resulting connection. *Param*, a whitespace-delimited string of values, specifies properties which the connection must have. At present four parameter values are defined:

**heavy**

**light** Heavy (usually computer-to-computer) or light (computer-to-terminal) traffic is expected.

**delim** The connection must support delimiters; see [stream\(4\)](#).

**hup** **SIGHUP** must be generated at end of file; see [signal\(2\)](#).

*Ipccreat* attaches a process to a name space. It returns a file descriptor representing the attachment. *Name* and *param* mean the same as for *ipcopen*.

*Ipclisten* waits for calls (from *ipcopen* in other processes) appearing on file descriptor *fd* (obtained from *ipccreat*). When a call arrives, it returns an **ipcin**fo structure, defined in

```
typedef struct {
    int reserved[5];
    char *name;          that being dialed
    char *param;         parameters used to set up call
    char *machine;       machine id of caller
    char *user;          user name of caller
    int uid, gid;        uid, gid of caller
} ipcin
```

The call may be accepted by *ipcaccept* or rejected by *ipcreject*. *Ipcaccept* returns a file descriptor for the connection. *Ipreject* takes an integer error number and an error message string, which will be passed back to the caller as *errno* and *errstr*.

A higher-level routine, *ipcexec*, executes the command, *cmd*, on a named machine. The file descriptor returned by *ipcexec* is the standard input, standard output, and standard error of the command. As in *ipcopen*, *param* lists properties required of the channel.

Once a connection is established using *ipcopen* it is often necessary to authenticate yourself to the destination. This is done using *ipclogin* and *ipcrogin*. *Ipclogin* runs the client side of the authentication protocol described in [svcmgr\(8\)](#) for the *v9auth* action. The supplied *fd* is the descriptor returned by *ipcopen*. Until the authentication is accepted, *ipclogin* will prompt the user (using for a login id and password to be sent over *fd*).

*Ipcrogin* runs the client side of the authentication protocol used by BSD's *rlogin* and *rsh* services. Unlike *ipclogin*, it will not prompt the user if the authentication fails. *Ipcrogin* takes a second argument that is written to *fd* after the authentication is accepted.

## EXAMPLES

To connect to the voice synthesizer attached to the Datakit:

```
#include <ipc.h>
main() {
    int fd;
    fd = ipcopen(ipcpath("voice", "dk", 0), "light");
    if (fd < 0) {
        printf("can't connect: %s\n", errstr);
        exit(1);
    }
    ...
    close(fd);
}
```

To place a Dataphone call via Datakit; the service name is derived in an obvious way from the ACU service class; see [dialout\(3\)](#).

```
fd = ipcopen(ipcpath("9-1-201-582-0000", "dk", "dial1200"), "light");
```

To announce as a local service and wait for incoming calls:

```
#include <ipc.h>
main() {
    int fd;
    ipcinfo *ip;
    fd = ipccreat("/tmp/service", 0);
    if(fd<0){
        printf("can't announce: %s\n", errstr);
        exit(1);
    }
    while(ip = ipclisten(fd)){
        int nfd;
        if(i_hate_this_user(ip->machine, ip->user)) {
            ipcreject(ip, EACCES, "i hate you");
            continue;
        }
        nfd = ipcaccept(ip);
        ...
        close(nfd);
    }
    printf("lost the announced connection somehow\n");
    exit(1);
}
```

## FILES

/cs/dk  
the announce point for the Datakit dialer

/cs/tcp  
the announce point for the internet dialer

## SEE ALSO

[dialout\(3\)](#), [connlnd\(4\)](#), [dkmgr\(8\)](#), [svcmgr\(8\)](#), [tcpmgr\(8\)](#)

D. L. Presotto, 'Interprocess Communication in the Eighth Edition UNIX System', this manual, Volume 2

## DIAGNOSTICS

Integer return values of  $-1$  and pointer return values of  $0$  denote error. *Errno* contains an error code (see [intro\(2\)](#)) and *errstr* points to an explanatory string.

## BUGS

Files created by *ipccreat* in the local name space are not removed when the file descriptor returned by *ipc-creat* is closed.

Information in **ipcinfo** is no more trustworthy than its origin. Information, such as user name, sent by foreign machines may be suspect. On Ethernet or dialup connections (but not on Datakit) machine names can be forged. Let's not even think about wire-swappers and wiretappers.

**NAME**

*iread* – insistent read

**SYNOPSIS**

**read(fildes, buf, n) char \*buf;**

**DESCRIPTION**

*Iread*, like *read(1)*, reads *n* bytes from the file associated with the given file descriptor into the block of store beginning at *buf*. *Iread*, however, always places exactly *n* bytes in *buf*, unless *read* would return 0.

*Iread* returns the number of bytes placed in *buf*, which is an integer in the range 0-*n*.

**DIAGNOSTICS**

*Iread* returns -1 for an error; see *read(1)*

**SEE ALSO**

*read(2)*

**NAME**

istream – formatted and unformatted input

**SYNOPSIS**

```

#include <iostream.h>
typedef long streamoff, streampos;
class ios {
public:
    enum        seek_dir { beg, cur, end };
    enum        open_mode { in, out, ate, app };
    // and lots of other stuff ...
};
class istream : ios {
public:
    int         gcount();
    istream&    get(char* ptr, int len, char delim='\n');
    istream&    get(unsigned char* ptr,int len, char delim='\n');
    istream&    get(unsigned char& c);
    istream&    get(char& c);
    istream&    get(streambuf&, char delim ='\n');
    int         get();
    istream&    getline(char* ptr, int len, char delim='\n');
    istream&    getline(unsigned char* ptr, int len, char delim='\n');
    istream&    getline(streambuf& sb, int len, char delim='\n');
    istream&    ignore(int len=1,int delim=EOF);
    int         ipfx(int need=0);
    int         peek();
    istream&    putback(char c);
    istream&    read(char* s,int n);
    istream&    read(unsigned char* s,int n);
    istream&    seekg(streampos pos);
    istream&    seekg(streamoff off, seek_dir d);
    streampos   tellg();
    istream&    operator>>(char*);
    istream&    operator>>(unsigned char*);
    istream&    operator>>(unsigned char&);
    istream&    operator>>(char&);
    istream&    operator>>(short&);
    istream&    operator>>(int&);
    istream&    operator>>(long&);
    istream&    operator>>(unsigned short&);
    istream&    operator>>(unsigned int&);
    istream&    operator>>(unsigned long&);
    istream&    operator>>(float&);
    istream&    operator>>(double&);
    istream&    operator>>(streambuf&);
    istream&    operator>>(istream& (*)(istream&));
};
class istream_withassign {
    istream&    istream_withassign();
    istream&    operator=(istream&);
    istream&    operator=(streambuf*);
};
extern istream_withassign cin;
istream&       ws(istream& i) ;

```



## DESCRIPTION

`istream`s support interpretation of characters fetched from an associated `streambuf`. These are commonly referred to as input operations.

Assume that

- **ins** has type `istream`.
- **inwa** has type `istream_withassign`.
- **insp** has type `istream*`.
- **c** has type `char&`
- **delim** has type `char`.
- **ptr** has type `char*` or `unsigned char*`.
- **sb** has type `streambuf&`.
- **i**, **len**, **d**, and **need** have type `int`.
- **pos** is a `streampos`.
- **off** is a `streamoff`.
- **dir** is a `seek_dir`.
- **manip** is a function with type `istream& (*)(istream&)`.

Constructors and assignment:

### **istream(sb)**

Initializes **ios** state variables and associates stream with buffer **sb**. **istream\_withassign()** Does no initialization. This allows a file static variable of this type (**cin** for example) to be used before it is constructed provided it is assigned to first.

### **inswa=sb**

Associates **sb** with **swa** and initializes the entire state of **inswa**.

### **inswa=ins**

Associates **ins->rdbuf()** with **swa** and initializes the entire state of **inswa**.

Input prefix function:

### **ins.ipfx(need)**

If **ins**'s error state is non-zero return immediately. If necessary (and it is non-null) **f.tie** is flushed. Flushing is necessary if either **need==0** or there are less than **need** characters immediately available. If **f.skip()** is non-zero and **need** is zero then leading whitespace characters are extracted from **ins**. Return zero or an error occurs while skipping whitespace. Otherwise it returns non-zero.

Formatted input functions:

### **ins>>x**

Calls **ipfx(0)** and if that returns non-zero extracts characters from **ins** and converts them according to the type of **x**. It stores the converted value in **x**. Errors are indicated by setting the error state of **ins**. `failbit` means that characters in **ins** were not a representation of the required type. `badbit` indicates that attempts to extract characters failed. **ins** is always returned.

The details of conversion depend on the values of **ins**'s format state variable (see `ios(3C++)`) and the type of **x**. Except as noted, the extraction operators do not change the value of the format state variables.

`char*`, `unsigned char*`

Characters are stored in the array pointed at by **x** until a whitespace character is found in **ins**. The terminating whitespace is left in **ins**. If **ins.width()** is non-zero it is taken to be the size of the array, and no more than **ins.width()-1** characters are extracted. A terminating null character (0) is always stored (even when nothing else is done because of **ins**'s status). **ins.width()** is reset to 0.

`char&`, `unsigned char&`

A character is extracted and stored in **x**.

`short&`, `unsigned short&`, `int&`, `unsigned int&`, `long&`, `unsigned long&`

Characters are extracted and converted to an integral value according to the conversion specified by **ins.convbase()**. The first character may be a sign (+ or -). After that, if

**ins.convbase()** is 8, 10, or 16 the conversion is octal, decimal, or hexadecimal respectively. Conversion is terminated by the first "non-digit," which is left in **ins**. Octal digits are the characters '0' to '7'. Decimal digits are the octal digits plus '8' and '9'. Hexadecimal digits are the decimal digits plus the letters 'a' through 'f' (in either upper or lower case). If **ins.convbase()** is 0 then the number is interpreted according to C lexical conventions. That is, if the first characters (after the optional sign) are 0x or 0X a hexadecimal conversion is performed on following hexadecimal digits. Otherwise if the first character is a 0 an octal conversion is performed and in all other cases a decimal conversion is performed. **failbit** is set if there are no digits (not counting the 0 in 0x or 0X) during hex conversion) available.

**float&, double&**

Converts the characters according to C++ syntax for a float or double, and stores the result in **x**. **failbit** is set if there are no digits available in **ins** or if it does not begin with a well formed floating point number.

The type and name(**operator>>**) of the extraction operations are chosen to give a convenient syntax for sequences of input operations. The operator overloading of C++ permits extraction functions to be declared for user defined classes. These operations can then be used with the same syntax as the member functions described here.

Unformatted input functions, which call **ipfx(1)** and proceed only if it returns non-zero:

**insp=&ins.get(ptr,len,delim)**

Extracts characters and stores them in the byte array beginning at **ptr** and extending for **len** bytes. Extraction stops when **delim** is encountered (**delim** is left in **ins** and not stored), when **ins** has no more characters, or when the array has only one byte left. **get** always stores a terminating null, even if it doesn't extract any characters from **ins** because of its error status. **failbit** is set only if **get** encounters an end of file before it stores any characters.

**insp=&ins.get(c)**

Extracts a single character and stores it in **c**.

**insp=&ins.get(sb,delim)**

Extracts characters from **ins.rdbuf()** and stores them into **sb**. It stops if it encounters end of file or if a store into **sb** fails or if it encounters **delim** (which it leaves in **ins**). **failbit** is set if it stops because the store into **sb** fails

**i=ins.get().**

Extracts a character and returns it. **i** is EOF if extraction encounters end of file. **failbit** is never set.

**insp=&ins.getline(ptr,len,delim)**

Does the same thing as **ins.get(ptr,len,delim)** with the exception that it extracts a terminating **delim** character from **ins**. In case **delim** occurs when exactly **len** characters have been extracted, termination is treated as being due to the array being filled, and this **delim** is left in **ins**.

**insp=&ins.ignore(n,d)**

Extracts and throws away up to **n** characters. Extraction stops prematurely if **d** is extracted or end of file is reached. If **d** is EOF it can never cause termination.

**insp=&ins.read(ptr,n)**

Extracts **n** characters and stores them in the array beginning at **ptr** If end of file is reached before **n** characters have been extracted, **read** stores whatever it can extract and sets **failbit**. The number of characters extracted can be determined via **ins.gcount()**.

Other members are:

**i=ins.gcount()**

Returns the number of characters extracted by the last unformatted input function. Formatted input functions may call unformatted input functions and thereby reset this number.

**i=ins.peek()**

Begins by calling **ins.ipfx(1)**. If that call returns zero or if **ins** is at end of file, it returns EOF. Otherwise it returns (without extracting it) the next character.

**insp=&ins.putback(c)**

Attempts to back up **ins.rdbuf()**. **c** must be the character before **ins.rdbuf()**'s get pointer. (Unless other activity is modifying **ins.rdbuf()** this is the last character extracted from **ins**.) If it is not, the effect is undefined. **putback** may fail (and set the error state). Although it is a member of **istream**, **putback** never extracts characters, so it does not call **ipfx**. It will, however, return without doing anything if the error state is non-zero.

**ins>>manip**

Equivalent to **manip(ins)**. Syntactically this looks like an extractor operation, but semantically it does an arbitrary operations rather than converting a sequence of characters and storing the result in **manip**.

Member functions related to positioning.

**insp=&ins.seekg(off,dir)**

Repositions **ins.rdbuf()**'s get pointer. See *sbuf.pub(3C++)* for a discussion of positioning.

**insp=&ins.seekg(pos)**

Repositions **ins.rdbuf()**'s get pointer. See *sbuf.pub(3C++)* for a discussion of positioning.

**pos=ins.tellg()**

The current position of **ios.rdbuf()**'s get pointer. See *sbuf.pub(3C++)* for a discussion of positioning.

Manipulator:

**ins>>ws**

Extracts whitespace characters.

**CAVEATS**

There is no overflow detection on conversion of integers. There should be, and overflow should cause the error state to be set.

There should be a way to input an arbitrary length string without knowing a maximum size beforehand.

**SEE ALSO**

*ios(3C++)* *sbuf.pub(3C++)* *manip(3C++)*

**NAME**

*l3tol*, *l3tol3* – convert between 3-byte integers and long integers

**SYNOPSIS**

***l3tol*(*lp*, *cp*, *n*)**

**long \**lp*;**

**char \**cp*;**

***l3tol3*(*cp*, *lp*, *n*)**

**char \**cp*;**

**long \**lp*;**

**DESCRIPTION**

*L3tol* converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

*L3tol3* performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

**SEE ALSO**

[\*flsys\*\(5\)](#)

**NAME**

malloc, free, realloc, calloc, cfree – memory allocator

**SYNOPSIS**

**char \*malloc(size)**  
**unsigned size;**

**free(ptr)**  
**char \*ptr;**

**char \*realloc(ptr, size)**  
**char \*ptr;**  
**unsigned size;**

**char \*calloc(nelem, elsize)**  
**unsigned nelem, elsize;**

**cfree(ptr)**  
**char \*ptr;**

**DESCRIPTION**

*Malloc* and *free* provide a simple memory allocation package. *Malloc* returns a pointer to a new block of at least *size* bytes. The block is suitably aligned for storage of any type of object. No two active pointers from *malloc* will have the same value.

The argument to *free* is a pointer to a block previously allocated by *malloc*; this space is made available for further allocation.

*Realloc* changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. The call **realloc((char\*)0, size)** means the same as `malloc(size)`.

*Calloc* allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros. *Cfree* frees such a block.

**SEE ALSO**

[galloc\(3\)](#), [brk\(2\)](#), [pool\(3\)](#), [block\(3\)](#)

**DIAGNOSTICS**

*Malloc*, *realloc* and *calloc* return 0 if there is no available memory or if the arena has been detectably corrupted.

**BUGS**

When *realloc* returns 0, the block pointed to by *ptr* may have been destroyed.

User errors can corrupt the storage arena. The most common gaffes are (1) freeing an already freed block, (2) storing beyond the bounds of an allocated block, and (3) freeing data that was not obtained from the allocator. To help find such errors, a diagnosing allocator may be loaded; use flag **-ldmalloc** of [cc\(1\)](#). An even more stringently checking version may be created by recompilation; see the source.

**NAME**

manipulators – iostream out of band manipulations

**SYNOPSIS**

```

#include <iostream.h>
#include <iomanip.h>
IOMANIPdeclare(T) ;
class SMANIP(T) {
    SMANIP(T)( ios& (*)(ios&,T), T);
};
class SAPP(T) {
    SAPP(T)( ios& (*)(ios&,T));
    SMANIP(T) operator()(T);
};
ios& SMANIP(T)::operator<<(ios&);
ios& SMANIP(T)::operator>>(ios&);
istream& SMANIP(T)::operator>>(istream&);
ostream& SMANIP(T)::operator<<(ostream&);
iostream& SMANIP(T)::operator<<(iostream&);
iostream& SMANIP(T)::operator>>(iostream&);
class IMANIP(T) {
    IMANIP(T)( istream& (*)(istream&,T), T);
};
class IAPP(T) {
    IAPP(T)( istream& (*)(istream&,T));
    IMANIP(T) operator()(T);
};
istream& operator>>(IMANIP(T), istream&);
class OMANIP(T) {
    OMANIP(T)( ostream& (*)(ostream&,T), T);
};
class OAPP(T) {
    OAPP(T)( ostream& (*)(ostream&,T));
    OMANIP(T) operator()(T);
};
ostream& operator<<(OMANIP(T), ostream&);
class IOMANIP(T) {
    IOMANIP(T)( ios& (*)(ios&,T), T);
};
class IOAPP(T) {
    IOAPP(T)( ios& (*)(ios&,T));
    IOMANIP(T) operator()(T);
};
istream& operator<<(SMANIP(T), istream&);
istream& operator>>(SMANIP(T), istream&);
IOMANIPdeclare(int) ;
IOMANIP(int) setbase(int b) ;
IOMANIP(int) setw(int w) ;

```

**DESCRIPTION**

Manipulators are values that may be "inserted into" or "extracted from" streams in order to achieve some effect (other than to insert a representation of their value). Ideally the types relating to manipulators would be parameterized as "templates". The macros defined in `iosmanip.h` are used to simulate a templates. `OP(T)` declares the various classes and operators. (All code is declared as inlines so that no separate definitions are required.) Each of the other `T` is used to construct the real names and therefore must be a single identifier. Each of the other macros also requires an identifier and expands to a name.

— `t` is a `T`.

— `s` is an `ios`.

- **i** is an istream.
- **o** is an ostream.
- **io** is an iostream.
- **f** is an ios& (\*)(ios&).
- **if** is an istream& (\*)(istream&).
- **of** is an ostream& (\*)(ostream&).
- **iof** is an iostream& (\*)(iostream&).

```

s<<SMANIP(T)(f,t)
s<<SMANIP(T)(f,t)
s>>SMANIP(T)(f,t)
s<<SAPP(T)(f)(t)
s>>SAPP(T)(f)(t)
i>>SMANIP(T)(if,t)
i>>SAPP(T)(if)(t)
o<<SMANIP(T)(of,t)
o<<SAPP(T)(of)(t)
io<<SMANIP(T)(iof,t)
io>>SMANIP(T)(iof,t)
io<<SAPP(T)(iof)(t)
io>>SAPP(T)(iof)(t)

```

Returns **fct(s,t)**, where **s** is the left operand of the insertion or extractor operator and **fct** is **f**, **ifr** or **iof**.

```

i>>IMANIP(T)(if,t)
i>>IAPP(T)(if)(t)

```

Return **if(i,t)**.

```

o<<OMANIP(T)(of,t)
o<<OAPP(T)(of)(t)

```

Return **of(i,t)**.

```

io<<IOMANIP(T)(iof,t)
io>>IOMANIP(T)(iof,t)
io<<IOAPP(T)(iof)(t)
io>>IOAPP(T)(iof)(t)

```

Return **iof(s,t)**.

`iomanip.h` contains a declaration, `IOMANIPdeclare(int)` and some functions:

```

o<<setbase(n)
i>>setbase(n)
io<<setbase(n)
io>>setbase(n)

```

Sets the conversion base of the stream (left hand operand) to **n**.

```

o<<setw(n)
i>>setw(n)
io<<setw(n)
io>>setw(n)

```

Sets the width format state variable the stream (left hand operand) to **n**.

## CAVEATS

Syntax errors will be reported if **IOMANIP(T)** occurs more than once in a file with the same **T**.

## SEE ALSO

`ios(3C++)` `istream(3C++)` `ostream(3C++)`

**NAME**

map – associative array classes

**SYNOPSIS**

```
#include <Map.h>

Mapdeclare(S,T)
Mapimplement(S,T)

struct Map(S,T) {
    Map(S,T)();
    Map(S,T)(const T&);
    Map(S,T)(const Map(S,T)&);
    Map(S,T);
    Map(S,T)& operator= (const Map(S,T)&);
    T& operator[] (int);
    int size();
    Mapiter(S,T) element(const S&);
    Mapiter(S,T) first();
    Mapiter(S,T) last();
};

struct Mapiter(S,T) {
    Mapiter(S,T) (const Map(S,T)&);
    Mapiter(S,T);
    operator int();
    S key();
    T value();
    Mapiter(S,T)& operator++ (Mapiter(S,T)&);
    Mapiter(S,T)& operator-- (Mapiter(S,T)&);
};
```

**DESCRIPTION**

A *map* is a collection of *elements*, each of which contains a *key* part of type *S* and a *value* part of type *T*, where *S* and *T* are type names. Both *S* and *T* must have value semantics: assignment or initialization have the effect of copying. (It is unlikely for *S* and *T* to be pointers.)

Map elements are ordered by key: type *S* must have a transitive boolean **operator<**.

The macro call `Mapdeclare(S,T)` declares the classes **Map(S,T)** and **Mapiter(S,T)**. It must appear once in every source file that uses either. The macro call `Mapimplement(S,T)` defines the functions that implement the map classes. It must appear exactly once in the entire program.

**Map constructors**

- Map(S,T)()** An empty map. The value part of future elements is the value of an otherwise uninitialized static object of type *T*.
- Map(S,T)(x)** An empty map whose future elements have default value *x*.
- Map(S,T)(m)** A copy of map *m* obtained by copying the elements and default value of *m*.

**Map operators**

- n = m* All the elements of map *n* are deleted and copies of the elements of *m* are added. The default value of *n* does not change. Running time is  $O(\log(|m|) + \log(|n|))$ , where  $|m|$  means *m.size()*.
- m[k]* A reference to the value part of the element of map *m* with key **k**. If the element does not exist, it is created. Running time is  $O(\log(|m|))$ .

**Other Map functions**

- m.size()* The number of elements in *m*. Running time is  $O(1)$
- m.element(k)* A map iterator referring to the element of *m* with key *k* if such an element exists. Otherwise the result is vacuous. Running time is  $O(\log(|m|))$ .



***m.first()***  
***m.last()*** A map iterator referring to the element of *m* with the smallest (or largest) key. If *m* has no elements, the result is *vacuous*. Running time is  $O(\log(|m|))$ .

### Map iterators

For every class **Map**(*S,T*) there is a class **Mapiter**(*S,T*). A map iterator identifies a map object and possibly an element in that map. An iterator that does not identify an element is *vacuous*.

### Mapiter constructors

**Mapiter**(*S,T*)(*m*)  
 A *vacuous* iterator referring to map *m*. Running time is  $O(1)$

### Mapiter operators

*i = j* Make iterator *i* refer (for now) to the same map as does *j*.  
**(int)***i* Zero if iterator *i* is *vacuous*, otherwise nonzero.  
 ++*i*  
 --*i* If iterator *i* is *vacuous*, make it refer to the map element with the smallest (or largest) key. Otherwise, make it refer to the map element with the next key greater (or less) than the key of the current element. If no such element exists, *i* becomes *vacuous*. The running time of a single increment operation for map *m* is  $O(\log(|m|))$ . However an iterator takes only time  $O(|m|)$  to sequence through the whole map.

### Other mapiter functions

*i.key()*  
*i.value()* The key (or value) part of the element referred to by *i*. If *i* is *vacuous*, return the value of an otherwise uninitialized static object of appropriate type. Running time is  $O(1)$

## EXAMPLES

```
struct city { char name[100]; };
typedef int population;
int operator< (const city&, const city&);

Mapdeclare(name,population)
Map(name,population) gazetteer;

// Print big cities; set populations of others to zero.

for(Mapiter(name,population) i = gazetteer.first(); i; i++)
    if(i.value() > 1000000)
        printf("%s\n", i.key().name);
    else
        gazetteer[i.key()] = 0;
```

## BUGS

A 'type name' **Map**(*S,T*) or **Mapiter**(*S,T*) that contains spaces will be mangled by [cpp\(8\)](#).  
 There is no way to delete a single element.  
 Ambiguities can occur if the type name *S* contains an underscore.  
 No precautions are taken against running out of memory.

**NAME**

memcpy, memchr, memcmp, memcpy, memmove, memset – memory operations

**SYNOPSIS**

**char \*memcpy(s1, s2, c, n)**

**char \*s1, \*s2;**

**int c, n;**

**char \*memchr(s, c, n)**

**char \*s;**

**int c, n;**

**int memcmp(s1, s2, n)**

**char \*s1, \*s2;**

**int n;**

**char \*memcpy(s1, s2, n)**

**char \*s1, \*s2;**

**int n;**

**char \*memmove(s1, s2, n)**

**char \*s1, \*s2;**

**int n;**

**char \*memset(s, c, n)**

**char \*s;**

**int c, n;**

**DESCRIPTION**

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

*Memcpy* copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. It returns a pointer to the character after the copy of *c* in *s1*, or zero if *c* was not found in the first *n* characters of *s2*.

*Memchr* returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or zero if *c* does not occur.

*Memcmp* compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*.

*Memcpy* copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

*Memmove* is the same as *memcpy*, except it is guaranteed to handle overlapping strings as if the move had been made to a temporary and then to the destination.

*Memset* sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

**SEE ALSO**

[string\(3\)](#)

**BUGS**

*Memcmp* uses native character comparison, which is signed on some machines, unsigned on others; thus the sign of the value returned when a character has its high-order bit set is implementation-dependent. Thanks to ANSI X3J11 for the *memcpy/memmove* distinction.

**NAME**

mktemp, tmpnam – make a unique file name

**SYNOPSIS**

```
char *mktemp(template)
```

```
char *template;
```

```
#include <tmpnam.h>
```

```
char *tmpnam(s)
```

```
char s[L_tmpnam];
```

**DESCRIPTION**

*Mktemp* replaces *template* by a unique file name, and returns the address of the template. The template should look like a file name with six trailing Xs, which will be replaced with the current process id and a unique letter.

*Tmpnam* places in the string pointed to by *s* a unique file name referring to the standard `/tmp` directory for temporary files and returns *s*. If *s* is 0, *tmpnam* returns the address of a fixed internal buffer that contains the name. (Note: it is bad form to leave files in the temporary directory.)

**SEE ALSO**

*getpid* in [getuid\(2\)](#)

**BUGS**

After many calls to *tmpnam*, the resulting filenames may have strange characters.

**NAME**

monitor – prepare execution profile

**SYNOPSIS**

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];
```

**DESCRIPTION**

An executable program created by `cc -p` automatically includes calls for *monitor* with default parameters; *monitor* needn't be called explicitly except to gain fine control over profiling.

*Monitor* is an interface to *profil(2)*. *Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* bytes. *Monitor* arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. At most *nfunc* call counts can be kept; only calls of functions compiled with the profiling option `-p` of *cc(1)* are recorded. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. The default values for *bufsize* and *nfunc* are  $(highpc - lowpc) / 8$  and 300 respectively.

To profile the entire program, use

```
extern etext();
. . .
monitor((int (*)())2, etext, buf, bufsize, nfunc);
```

*Etext* lies just above all the program text, see *end(3)*. For the highest resolution profiling on the VAX, use `bufsize = ((int)highpc) - ((int)lowpc) + 12 + 8 * nfunc`.

To stop execution monitoring and write the results on the file *mon.out*, use

```
monitor((int (*)())0);
```

then *prof(1)* can be used to examine the results.

**FILES**

mon.out

**SEE ALSO**

*prof(1)*, *profil(2)*, *cc(1)*

**NAME**

itom, mfree, madd, msub, mult, mdiv, sdiv, msqrt, mgcd, min, mout, fmin, fmout, move, mcmp, rpow, mpow – multiple precision integer arithmetic

**SYNOPSIS**

```
#include <mp.h>
#include <stdio.h>
```

```
mint *itom(n)
short n;
```

```
mfree(a)
mint *a;
```

```
madd(a, b, c)
mint *a, *b, *c;
```

```
msub(a, b, c)
mint *a, *b, *c;
```

```
mult(a, b, c)
mint *a, *b, *c;
```

```
mgcd(a, b, c)
mint *a, *b, *c;
```

```
mdiv(a, b, q, r)
mint *a, *b, *q, *r;
```

```
sdiv(a, n, q, r)
mint *a, *q;
short n, *r;
```

```
msqrt(a, b, r)
mint *a, *b, *r;
```

```
rpow(a, n, c)
mint *a, *c;
```

```
mpow(a, b, m, c)
mint *a, *b, *m, *c;
```

```
move(a, b)
mint *a, *b;
```

```
mcmp(a, b)
mint *a, *b;
```

```
int min(a)
mint *a;
```

```
mout(a)
mint *a;
```

```
int fmin(a, f)
mint *a;
FILE *f;
```

```
fmout(a, f)
mint *a;
FILE *f;
```

**DESCRIPTION**

These routines perform arithmetic on arbitrary-length integers of defined type *mint*. The functions are obtained with the *ld(1)* option **-lmp**.

Pointers to *mint* must be initialized using the function *itom*, which sets the initial value to *n*. Thereafter space is managed automatically. The space may be freed by *mfree*, making the variable uninitialized.

*Madd*, *msub*, *mult*, and *mgcd* assign to their third arguments the sum, difference, product, and greatest common divisor, respectively, of their first two arguments.

*Mdiv* assigns the quotient and remainder, respectively, to its third and fourth arguments. The remainder is nonnegative and less than the divisor in magnitude. *Sdiv* is like *mdiv* except that the divisor is an ordinary integer.

*Msqrt* assigns the square root and remainder to its second and third arguments, respectively.

*Rpow* calculates  $a$  raised to the power  $n$ ; *mpow* calculates this reduced modulo  $m$ .

*Move* assigns (by copying) the value of its first argument to its second argument.

*Mcmp* returns a negative, zero, or positive integer if the value of its first argument is less than, equal to, or greater than, respectively, the value of its second argument.

*Min* and *mout* do decimal conversion from **stdin** and to **stdout**, *fnin* and *fmout* use file  $f$ ; see [stdio\(3\)](#).  
*Min* and *fnin* return **EOF** on end of file.

## DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

## BUGS

*Itom* and *sdiv* fail if  $n$  is the most negative short integer.

**NAME**

nlist – get entries from name list

**SYNOPSIS**

```
#include <nlist.h>
nlist(filename, nl)
char *filename;
struct nlist nl[];
```

**DESCRIPTION**

*Nlist* examines the name list in the given executable output file and selectively extracts a list of values. The list is terminated with a null name.

```
struct nlist {
    [CB]char *n_name;           symbol name
    [CB]unsigned char n_type;   type flag
    [CB]char n_other;          unused
    [CB]short n_desc;          unused
    [CB]unsigned long n_value; value (or offset) of this symbol
};
```

Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in `n_type` and `n_value` respectively. If the name is not found, both entries are set to 0.

This subroutine is useful for examining the system name list kept in In this way programs can obtain system addresses that are up to date.

**SEE ALSO**

[a.out\(5\)](#)

**DIAGNOSTICS**

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

**BUGS**

On some UNIX systems you must include `<a.out.h>` rather than This is unfortunate, but `<a.out.h>` can't be used on the VAX because it contains a union, which can't be initialized.

**NAME**

openshares – open shares file

**SYNOPSIS**

```
int openshares(lock)
int lock;
```

**DESCRIPTION**

*Openshares* attempts to open the shares file for writing, otherwise, if *lock* is 0, it attempts to open it for reading. If one of the opens succeeds, it returns 1, otherwise it returns 0.

If the shares file is already open, this routine does nothing.

**FILES**

/etc/shares      Shares data-base.

**SEE ALSO**

closeshares(3), getshares(3), getshput(3), putshares(3), sharesfile(3).



**NAME**

ostream – formatted and unformatted output

**SYNOPSIS**

```
#include <iostream.h>
typedef long streamoff, streampos;
class ios {
public:
    enum          seek_dir { beg, cur, end };
    enum          open_mode { in, out, ate, app };
    // and lots of other stuff ...
};
class ostream : ios {
public:
    ostream&      flush();
    int           opfx();
    ostream&      put(char c);
    ostream&      seekp(streampos pos);
    ostream&      seekp(streamoff off, seek_dir d);
    streampos     tellp();
    ostream&      write(const char* ptr,int n);
    ostream&      write(const unsigned char* ptr, int n);
    ostream&      operator<<(const char*);
    ostream&      operator<<(char);
    ostream&      operator<<(short);
    ostream&      operator<<(int a);
    ostream&      operator<<(long);
    ostream&      operator<<(double);
    ostream&      operator<<(unsigned char);
    ostream&      operator<<(unsigned short);
    ostream&      operator<<(unsigned int);
    ostream&      operator<<(unsigned long);
    ostream&      operator<<(void*);
    ostream&      operator<<(const streambuf&);
    ostream&      operator<<(ostream& (*)(ostream&));
};
class ostream_withassign {
    ostream_withassign();
    istream&      operator=(istream&);
    istream&      operator=(streambuf*);
};
extern ostream_withassign cout;
extern ostream_withassign cerr;
extern ostream_withassign clog;
ostream&         endl(ostream& i) ;
ostream&         ends(ostream& i) ;
ostream&         flush(ostream& i) ;
```

**DESCRIPTION**

istreams support insertion (storing) into an streambuf. These are commonly referred to as output operations.

Assume:

- **outs** is an ostream.
- **outswa** is an ostream\_withassign.
- **outsp** is an ostream\*.
- **c** is a char.
- **ptr** is a char\* or unsigned char\*.
- **sb** is a streambuf\*

- **i** and **n** are int.
- **pos** is a streampos.
- **off** is a streamoff.
- **dir** is a seek\_dir.
- **manip** is a function with type ostream& (\*)(ostream&).

Constructors and assignment:

#### **ostream(sb)**

Initializes **ios** state variables and associates stream with buffer **sb**. **ostream\_withassign()** Does no initialization. This allows a file static variable of this type (**cout** for example) to be used before it is constructed provided it is assigned to first.

#### **outswa=sb**

Associates **sb** with **swa** and initializes the entire state of **outswa**.

#### **inswa=ins**

Associates **ins->rdbuf()** with **swa** and initializes the entire state of **outswa**. The inserters:

#### **outs<<x**

First call **outs.opfx()**. If that returns 0 do nothing. Otherwise insert a sequence of characters representing **x** into **outs.rdbuf()**. Errors are indicated by setting the error state of **outs**. **outs** is always returned.

**x** is converted into a sequence of characters (its representation) according to rules that depend on **x**'s type and **outs**'s format state variables (see *ios(3C++)*):

char\*

The representation is the sequence of characters up to (but not including) the terminating null of the string **x** points at.

*any integral type*

If **x** is positive the representation contains a sequence of decimal, octal, or hexadecimal digits with no leading zeros according to whether **outs.convbase()** is 10, 8, or 16 respectively. If **x** is zero the representation is a single zero character(0). If **x** is negative, decimal conversion converts it to a minus sign (-) followed by decimal digits. The other conversions treat all values as unsigned. A value of 0 for **outs.convbase()** also causes decimal conversion. The effect of other values of **outs.convbase()** is undefined. If **outs.showbase()** is non-zero the hex representation contains 0x before the hex digits. If **outs.showbase()** is non-zero the octal representation contains a leading 0.

void\*

Pointers are cast to integral values and then converted to hexadecimal numbers as if **outs.showbase()** were set.

float, double

The arguments are converted according to the current values of **outs.precision()**, **outs.floatfmt()**, and **outs.width()** using the printf formatting conventions for "%floatfmt". The default values for **outs.floatfmt()** and **outs.precision()** are g and 6 respectively.

After the representation is determined, padding occurs. If **outs.width()** is greater than 0 and the representation contains less than **outs.width()** characters, then enough **outs.fill()** characters are added to bring the total number of characters to **ios.width()**. If **ios.ladjust()** is non-zero the sequence is left adjusted. That is, characters are added after the characters determined above. Otherwise the padding is added before the characters determined above. **ios.width()** is reset to 0, but all other format state variables are unchanged. The resulting sequence (padding plus representation) is inserted into **outs.rdbuf()**.

#### **outs<<sb**

If **outs.opfx()** returns non-zero the sequence of characters that can be fetched from **sb** are inserted into **outs.rdbuf**. Insertion stops when no more characters can be fetched from **sb**. No padding is performed. Always return **outs**.

Other members:

**i=outs.opfx()**

If **outs**'s error state is nonzero returns immediately. If **outs.tie()** is non-null flush it. Returns non-zero except when **outs**'es error state is nonzero.

**outsp=&outs.flush()**

storing characters into a streambuf does not always cause them to be consumed (e.g., written to the external file) immediately. **flush** causes any characters that may have been stored but not yet consumed to be consumed by calling **outs.rdbuf()->sync**.

**outs<<manip**

Equivalent to **manip(outs)**. Syntactically this looks like an insertion operation, but semantically it does an arbitrary operations rather than converting **manip** to a sequence of characters as do the insertion operators.

Unformatted output functions:

**outsp=&outs.put(c)**

Inserts **c** into **outs.rdbuf()**. Sets the error state if the insertion fails.

**outsp=&outs.write(s,n)**

Inserts the **n** characters starting at **s** into **outs.rdbuf()**. These characters may include zeros (i.e., **s** need not be a null terminated string).

Positioning functions:

**outsp=&ins.seekp(off,dir)**

Repositions **outs.rdbuf()**'s put pointer. See *sbuf.pub(3C++)* for a discussion of positioning.

**outsp=&outs.seekp(pos)**

Repositions **outs.rdbuf()**'s put pointer. See *sbuf.pub(3C++)* for a discussion of positioning.

**pos=outs.tellp()**

The current position of **outs.rdbuf()**'s put pointer. See *sbuf.pub(3C++)* for a discussion of positioning.

Manipulators:

**outs<<endl**

Ends a line by inserting a newline character and flushing.

**outs<<ends**

Ends a string by inserting a null(0) character.

**outs<<flush**

Flushes **outs**

**SEE ALSO**

ios(3C++) sbuf.pub(3C++) manip(3C++)

**NAME**

`perror`, `sys_errlist`, `sys_nerr` – system error messages

**SYNOPSIS**

```
perror(s)  
char *s;  
  
int sys_nerr;  
char *sys_errlist[];
```

**DESCRIPTION**

*Perror* produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. The error number is taken from the external variable *errno* (see [intro\(2\)](#)), which is changed only when errors occur.

*sys\_errlist* is a vector of message strings. *Errno* can be used as an index in this table to get the message string without the newline. *sys\_nerr* is the number of messages in the table.

**SEE ALSO**

[intro\(2\)](#)

**NAME**

picopen\_r, picopen\_w, picread, picwrite, picclose, picputprop, picgetprop, picunpack, picpack, picerror – picture file I/O

**SYNOPSIS**

```
#include <picfile.h>

PICFILE *picopen_r(name)
char *name;

PICFILE *picopen_w(name, type, x0, y0, w, h, chan, argv, cmap)
char *name, *type, *chan, *argv[], *cmap;

int picread(pf, buf)
PICFILE *pf;
char *buf;

int picwrite(pf, buf)
PICFILE *pf;
char *buf;

void picclose(pf)
PICFILE *pf;

PICFILE *picputprop(pf, name, value)
PICFILE *pf;
char *name, *value;

char *picgetprop(pf, name)
PICFILE *pf;
char *name;

void picunpack(pf, pix, fmt, arg ...)
PICFILE *pf;
char *pix, *fmt;

void picpack(pf, pix, fmt, arg ...)
PICFILE *pf;
char *pix, *fmt;

void picerror(string)
char *string;
```

**DESCRIPTION**

These functions read and write raster images in *picfile(5)* format. They are loaded by option **-lpicfile** of *ld(1)*. Open picture files are referred to by pointers of type **PICFILE\***.

*Picopen\_r* opens the named picfile for reading and returns a pointer to the open file. If *name* is "IN", standard input is used.

*Picopen\_w* similarly creates the named image file for writing. The name "OUT" refers to standard output. *Type* is a **TYPE** attribute, as described in *picfile(5)*; *x0* and *y0* are the upper left coordinates of the **WINDOW** attribute; *w* and *h* are the image width and height in pixels. *Chan* is a string specifying the order of channels for the **CHAN** attribute; the length of this string becomes the value of **NCHAN**. *Argv*, if nonzero, is conventionally the second argument of the main program; see *exec(2)*. It becomes a **COMMAND** attribute recording the provenance of the file.

The special call **picopen\_w(name, PIC\_SAMEARGS(pf))** creates a file with the same attributes as an already open picfile. **PIC\_SAMEARGS** mentions *argv* by name, hence the name must be visible at the point of call.

*Picread* and *picwrite* read or write a single row of pixels using the character array *buf*. The length of the row is determined from the file's **WINDOW** and **NCHAN** attributes. One-bit-per-pixel images (of type **bitmap** or **ccitt-g4**, for example) are decoded to one byte per pixel, 0 for black, 255 for white, and are encoded as 1 for pixel values less than 128 and 0 otherwise. Files of type **ccir601** are decoded into conventional **rgb** channels.

*Picclose* closes a picfile and frees associated storage.

*Picputprop* called after *picopen\_w* but before *picwrite* adds header attributes, returning a (probably changed) value of the **PICFILE** pointer.

*Picgetprop* returns a pointer to the value of the named attribute, or 0 if the picfile does not have the attribute. In both *Picputprop* and *picgetprop*, with multiple appearances (e.g. **COMMAND**) are expressed as a sequence of values separated by newlines.

The header file defines macros to extract commonly-used attributes:

```
PIC_NCHAN(pf), PIC_WIDTH(pf), PIC_HEIGHT(pf),
PIC_SAMEARGS(pf) (see picopen_w)
```

*Picunpack* extracts the channels of pixel array *pix* into separate array *args* of types described by the *fmt* character string. Format characters are **c**, **s**, **l**, **f**, **d**, for arrays of types unsigned char, short, long, float, and double. Format character **\_** designates a picfile channel to be skipped. *Picpack* reverses the process. These routines effect a standard machine-independent byte ordering.

*Picerror* prints messages for errors resulting from calls to *picfile* routines. (*Perror(3)* cannot describe some error conditions, like malformed header lines.)

## EXAMPLES

Unpack the green and z channels from a file with channels **rgbz...**

```
PICFILE *pf = picopen_r("file");
extern char pixels[], green[][1000];
extern float zdepth[][1000];
for(i=0; picread(pf, pixels); i)
    picunpack(pf, pixels, "_c_f", green[i], zdepth[i]);
```

Reflect a picture about its vertical midline.

```
PICFILE *in = picopen_r("picture");
PICFILE *out = picopen_w("OUT", PIC_SAMEARGS(in));
int w = PIC_WIDTH(in);
int n = PIC_NCHAN(in);
char *buffer = malloc(w*n), *temp = malloc(n);
while (picread(in, buffer)) {
    char *left = buffer;
    char *right = buffer + n*(w - 1);
    for( ; left<right; left+=n, right-=n) {
        strncpy(temp, left, n);
        strncpy(left, right, n);
        strncpy(right, temp, n);
    }
    picwrite(out, buffer);
}
```

## SEE ALSO

[picfile\(5\)](#), [pico\(1\)](#), [bcp\(1\)](#)

## DIAGNOSTICS

*Picread* returns 1 on success, 0 on end of file or error.

*Picopen\_r* and *picopen\_w* return 0 for unopenable files.

## BUGS

*Picpack* and *picunpack* store and retrieve floating point channels (types **f** and **d**) using native floating-point, rather than something machine independent like IEEE format.

**NAME**

pipebuf – streambuf specialized as circular queue

**SYNOPSIS**

```
#include <iostream.h>
#include <pipestream.h>
class pipebuf : streambuf {
    pipebuf();
    int empty();
    int full();
    streambuf* setbuf(char* ptr, int len);
};
```

**DESCRIPTION**

A `pipebuf` uses its reserve area to support a circular queue of characters. In terms of the abstract notion of buffer a `pipebuf` is a potentially infinite sequence in which the put pointer and get pointer move independently. The put pointer is always at the end of the sequence, and puts extend the sequence. As long as the get pointer remains behind the put pointer, but not too far behind, fetching and storing can continue indefinitely. Seeks are not supported.

Assume

- **pb** is a `pipebuf`.
- **ptr** is a `char*`.
- **i** and **len** are an `int`.
- **sb** is `streambuf*`.

Constructor:

**pipebuf()**

Constructs an empty buffer.

Members:

**i=pb.empty()**

Returns non-zero if the get pointer is at the end of the sequence, and attempts to get characters will therefore fail.

**i=pb.full()**

Returns non-zero if there is no more room for putting characters. In the current implementation, the capacity of the buffer is one less than the size of the reserve area.

**sb=pb.setbuf(ptr,len)**

Establishes the **len** bytes starting at **ptr** as the reserve area. Normally it will return **&pb**. But it will return a null pointer if it fails. Failure occurs if **pb.empty()** is zero, or if **len** is less than 2.

**CAVEATS**

There ought to be a version with an unbounded capacity.

**SEE ALSO**

`streambuf(3C++)` `pipestream(3C++)`

**NAME**

pipestream – iostream specialized as circular buffer

**SYNOPSIS**

```
#include <iostream.h>
#include <pipestream.h>
class pipestream : public iostream {
public:
    pipestream();
    pipestream(char* ptr, int len);
    pipebuf* rdbuf();
};
```

**DESCRIPTION**

pipestream specializes iostream to use a circular buffer (pipebuf).

Assume

- **ps** is a pipestream.
- **ptr** is a char\*.
- **len** is an int.
- **pb** is a pipebuf\*.

Constructors:

**pipestream()**

Constructs a pipestream with an empty pipebuf.

**pipestream(ptr,len)**

Constructs a pipestream with a pipebuf that uses the **len** bytes at **ptr** as a reserve area.

Members:

**pb=ps.rdbuf()**

Returns the associated **pipebuf**. **pipebuf::rdbuf** has the same semantics as **streambuf::rdbuf**, but the type of the result is more precise.

**SEE ALSO**

pipebuf(3C++) ios(3C++)



**NAME**

vec, move, etc. – plot graphics interface

**SYNOPSIS**

```
#include <plot.h>  
openpl(s)  
char *s;  
closepl()  
erase()  
move(x, y)  
double x, y;  
rmove(dx, dy)  
double dx, dy;  
point(x, y)  
double dx, dy;  
vec(x, y)  
double x, y;  
rvec(dx, dy)  
double dx, dy;  
line(x1, y1, x2, y2)  
double x1, y1, x2, y2;  
arc(x1, y1, x2, y2, x, y, r)  
double x1, y1, x2, y2, x, y, r;  
circle(xc, yc, r)  
double xc, yc, r;  
box(x1, y1, x2, y2)  
double x1, y1, x2, y2;  
sbox(x1, y1, x2, y2)  
double x1, y1, x2, y2;  
parabola(x1, y1, x2, y2, x3, y3)  
double x1, y1, x2, y2, x3, y3;  
fill(n, arr) int n[];  
double *arr[];  
poly(n, arr) int n[];  
double *arr[];  
spline(n, arr)  
int n[];  
double *arr[];  
cspline(n, arr)  
int n[];  
double *arr[];  
fspline(n, arr)  
int n[];  
double *arr[];  
lspline(n, arr)  
int n[];  
double *arr[];  
dspline(n, arr)  
int n[];
```

```

double *arr[];
text(s)
char *s;
color(s)
char *s;
cfill(s)
char *s;
pen(s)
char *s;
range(x1, y1, x2, y2)
double x1, y1, x2, y2;
frame(x1, y1, x2, y2)
double x1, y1, x2, y2;
grade(x)
double x;
save()
restore()
ppause()

```

## DESCRIPTION

These functions generate either a device-independent graphic stream (see [plot\(5\)](#)) or device-dependent graphics commands. The include file `<plot.h>` is used only for device-independent output. An alternative include file, `<iplot.h>`, supports device-independent output using identically named functions of integer, instead of double, arguments.

Libraries for different devices are loaded with the following [ld\(1\)](#) flags:

```

-lplot  general stream output
-l2621  HP2621 terminal
-l4014  Tektronix 4014 terminal
-ltr    Troff input, tuned for the Mergenthaler Linotron 202 phototypesetter
-lpen   HP7580 pen plotter
-l5620  5620 terminal running mux

```

String arguments are null-terminated and may not contain embedded newlines. For details on string arguments, see [plot\(5\)](#). *Poly*, *fill*, and the various spline functions take an integer array and an array of pointers to double floating point arrays. The integers specify the number of vertices (*x*-*y* pairs) in the floating point array. The last integer entry should be 0.

## SEE ALSO

[plot\(1\)](#), [plot\(5\)](#)

## BUGS

The `-ltr` library should be tuned for PostScript.

**NAME**

poly\_lk, poly\_read – polyhedron database routines

**SYNOPSIS**

```
#include <poly.h>
```

```
int poly_lk(name)
```

```
char *name;
```

```
int poly_read(p, dir, n)
```

```
Polyhedron *p;
```

```
char *dir;
```

**DESCRIPTION**

These routines access the [poly\(7\)](#) database of polyhedra.

*Poly\_lk* tries to interpret *name* as a polyhedron reference. If it is a number, it returns that number. Otherwise, it returns the number of the first polyhedron for which *name* is a prefix of the polyhedron's name.

*Poly\_read* forms an in-core description of the polyhedron number *n* in the directory *dir*. If *dir* is 0, the normal directory (`/usr/lib/polyhedra`) is used.

**SEE ALSO**

[poly\(5\)](#), [poly\(7\)](#)

**DIAGNOSTICS**

*Poly* returns `-1` on unknown names.

*Poly\_read* returns zero on success, nonzero on error.

**NAME**

pool – fast memory allocation

**SYNOPSIS**

```
#include <Pool.h>

struct Pool {
    Pool(unsigned);
    Pool();
    void* alloc();
    void free(void*)
};
```

**DESCRIPTION**

Every `Pool` is a collection of *elements*, each of which is an array of bytes. All elements of a pool are the same size. Pool functions are

<b>Pool(<i>n</i>)</b>	Construct a pool whose elements are of size <i>n</i> .
<b><i>p</i>.alloc()</b>	Allocate a new element in pool <i>p</i> . Return a pointer to the element.
<b><i>p</i>.free(<i>ep</i>)</b>	Free the element of <i>p</i> pointed to by <i>ep</i> . The element must have been allocated from <i>p</i> .

Destroying a pool frees all the memory occupied by its elements.

The memory in a pool element is aligned on the same boundary as memory returned by [malloc\(3\)](#) so that it may be used to contain an object of any type. In typical use, there would be one pool per class, with the pool known only to the **new** and **delete** operators of that class.

**Performance**

Pool memory is allocated in chunks that are typically about 1,000 bytes each. Once a chunk is allocated to a particular pool, that chunk is only released when the pool itself is destroyed.

Elements are allocated inline except when a new chunk must be added to the pool. Elements are always freed inline.

**EXAMPLES**

```
#include <Pool.h>

struct Mytype {
    static Pool mypool;
    // constructors and members
    void* operator new(unsigned) { return mypool.alloc(); }
    void operator delete(void* p) { mypool.free(p); }
};
```

```
Pool Mytype::mypool(sizeof(Mytype));
```

**SEE ALSO**

[malloc\(3\)](#)

**NAME**

*popen*, *ppopen*, *vepopen*, *pclose* – open a pipe to/from a process

**SYNOPSIS**

```
#include <stdio.h>
```

```
FILE *popen(command, type)
```

```
char *command, *type;
```

```
FILE *ppopen(command, type)
```

```
char *command, *type;
```

```
FILE *vepopen(command, type, args, env)
```

```
char *command, *type, **args, **env;
```

```
int pclose(stream)
```

```
FILE *stream;
```

**DESCRIPTION**

The first argument to *popen* is a pointer to a null-terminated string containing a command line for *sh(1)*. *Type* is as in *fopen(3)*. *Popen* creates a pipe between the calling process and the command and returns a stream pointer that can be used to write to the standard input of the command or to read from the standard output.

*Pppopen* uses the **-p** shell flag to restrict the environment of the shell. Both *popen* and *ppopen* set the effective userid to the real userid before calling the shell.

*Vepopen* has arguments akin to those of *execve* (see *exec(2)*): a file to be executed, a mode as above, a null-terminated vector of argument strings, and a null-terminated vector of environment strings. The shell is not called, and the effective userid is preserved.

A stream opened by these routines should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because the *command* inherits open files, in particular standard input and output, a type "r" call may be used to insert a filter in the input, and type "w" in the output.

**SEE ALSO**

*exec(2)*, *pipe(2)*, *fopen(3)*, *stdio(3)*, *system(3)*

**DIAGNOSTICS**

*Popen* returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

*Pclose* returns -1 if there is no process to wait for.

**BUGS**

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by calling *fflush*; see *fopen(3)*.

The resetting of the userid is probably gratuitous; it is there as a defense against incautious use of the routine by set-uid programs.

I/O type "r+w" exists but is not useful.

**NAME**

port – mathematical library for Fortran

**DESCRIPTION**

The Port library of hundreds of scientific subroutines covers approximation, ordinary differential equations, partial differential equations, linear algebra, optimization and mathematical programming, quadrature, differentiation, roots, special functions, and transforms. It is built upon a framework of service routines for error handling, stack management, and machine constant parameterization.

The routines are loaded by the [ld\(1\)](#) option `-lport`.

The manual describes the software and gives examples.

**SEE ALSO**

P. A. Fox, *The PORT Mathematical Subroutine Library*, AT&T Bell Laboratories, May 8, 1984.

P. A. Fox, *The PORT Mathematical Subroutine Library Installation Manual*, AT&T Bell Laboratories, September, 1984.

**NAME**

print, fprintf, sprintf, fmtinstall – print formatted output

**SYNOPSIS**

```
int print(format [ , arg ] ... )
char *format;

int fprintf(fildes, format [ , arg ] ... )
int fildes;
char *format;

int sprintf(s, format [ , arg ] ... )
char *s, *format;

fmtinstall(c, fn)
char c;
int (*fn)();

strconv(s, f1, f2)
char *s;

extern int printcol;
```

**DESCRIPTION**

*Print* places output on the standard output. *Fprint* places output on the named output file descriptor; a buffered form is described in [fio\(3\)](#). *Sprintf* places output followed by the null character (`\0`) in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the `\0` in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its trailing arguments under control of a *format* string. The *format* contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more arguments. The results are undefined if there are arguments of the wrong type or too few arguments for the format. If the format is exhausted while arguments remain, the excess are ignored.

Each conversion specification has the following format

```
% [flags] [[-] digits [. digits]] verb
```

The *flags* modify the meaning of the conversion verb. The first (possibly negative) number is called *f1*, the second number is *f2*. The flags and numbers are arguments to the *verb* described below.

The numeric verbs **d**, **o**, and **x** format their arguments in decimal, octal and hex respectively. Each interprets the flags **h**, **l**, **u**, to mean short, long, and unsigned. If neither short nor long is specified, then the arg is an **int**. If unsigned is specified, then the arg is interpreted as a positive number and no sign is output. *F1* is the minimum field width and, if negative, means left justified rather than right justified; in both cases, padding is done with blanks. The converted number is padded with `0` on the left to at least *f2* characters.

The floating point verbs **f**, **e**, and **g** take a double argument. No flags apply to floating point conversions. *F1* is the minimum field width and, if negative, means left justified. *F2* is the number of digits that are converted after the decimal place. The first unconverted digit has suffered decimal rounding. The **f** verb produces output of the form `[-]digits[.digits]`. **e** conversion appends an exponent `e[-]digits`. The **g** verb will output the arg in either **e** or **f** with the goal of producing the smallest output.

The **s** verb copies a string (pointer to character) to the output. The number of characters copied (*n*) is the minimum of the size of the string and *f2*. These *n* characters are justified within a field of *f1* characters as described above.

The **c** verb copies a single character (int) justified within a field of *f1* characters as described above.

*Fmtinstall* is used to install your own conversions and flags. *Fn* should be declared as

```
int fn(o, f1, f2, f3)
void *o;
int f1, f2, f3;
```

*Fn* is passed a pointer *o* to whatever argument appears in the list to *print*. *Fn* should return the size of the

argument in bytes so *print* can skip over it. *F1* and *f2* are the decoded numbers in the conversion. A missing *f1* is denoted by the value zero. A missing *f2* is denoted by a negative number. *F3* is the logical or of all the flags seen in the conversion. If *c* is a flag, *fn* should return a negative number that is negated and then logically *ored* with any other flags and ultimately passed to a conversion routine. All interpretation of *f1*, *f2*, and *f3* is left up to the conversion routine. The standard flags are h(2), l(1), and u(4).

*Sprintf* is designed to be recursive in order to help prepare output in custom conversion routines.

The output of any conversion routine must be passed through *strconv*. *S* is the character string, *f1* and *f2* have the same meaning as above.

*Printcol* indicates the position of the next output character. Tabs, backspaces and carriage returns are interpreted appropriately.

## EXAMPLES

This adds a verb to print complex numbers.

```
typedef struct {
    double r, i;
} complex;
complex x = { 1.5, -2.3 };
int Xconv();
main()
{
    fmtinstall('X', Xconv);
    print("x = %X\n", x);
}
Xconv(o, f1, f2, f3)
complex *o;
{
    char str[50];
    sprintf(str, "(%g,%g)", o->r, o->i);
    strconv(str, f1, f2);
    return(sizeof(complex));
}
```

## SEE ALSO

[fio\(3\)](#), [printf\(3\)](#)

## BUGS

There are internal buffers which may overflow silently.



**NAME**

printf, fprintf, sprintf, snprintf – print formatted output

**SYNOPSIS**

```
#include <stdio.h>
```

```
int printf(char *format, ... );
```

```
int fprintf(FILE *stream, char *format, ... );
```

```
int sprintf(char *s, char *format, ... );
```

```
int snprintf(char *s, int len, char *format, ... );
```

**DESCRIPTION**

*Printf* places output on the standard output stream *stdout*. *Fprintf* places output on the named output stream. *Sprintf* places output followed by the null character (`\0`), in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available. *Snprintf* corresponds to *sprintf* except that no more than *len* bytes are placed into *s*. Each function returns the number of characters transmitted (not including the `\0` in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its trailing arguments under control of a *format* string. The *format* contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more arguments. The results are undefined if there are arguments of the wrong type or too few arguments for the format. If the format is exhausted while arguments remain, the excess are ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it is padded to the field width. When the width specification begins with zero, padding is with leading zeros. Otherwise padding is with leading spaces (trailing spaces, with the left-adjustment flag `-`, described below) to the field width.

A *precision* that gives the minimum number of digits to appear for the **d**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e** and **f** conversions, the maximum number of significant digits for the **g** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (`.`) followed by a decimal digit string; a null digit string is treated as zero.

An optional **l** (ell) specifying that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*. An **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (`*`) or an exclamation point (`!`) instead of a digit string. In this case, an integer *arg* supplies the field width or precision.

The flag characters and their meanings are:

- `-` The result of the conversion is left-justified within the field.
- `+` The result of a signed conversion always begins with a sign (`+` or `-`).
- blank If the first character of a signed conversion is not a sign, a blank is prefixed to the result. This implies that if the blank and `+` flags both appear, the blank flag is ignored.
- `#` This flag specifies that the value is to be converted to an “alternate form.” For **c**, **d**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** or **X** conversion, a non-zero result has **0x** or **0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result always contains a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeros are *not* be removed from the result as they normally are.

The conversion characters and their meanings are:

[CB]d,[CB]o,[CB]u,[CB]x,[CB]X

The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading zeros. (For compatibility with other versions of *printf*, a field width with a leading zero results in padding with leading zeros. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

**f** The float or double *arg* is converted to decimal notation in the style `[-]d.ddd`, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly 0, no decimal point appears.

**e, E** The float or double *arg* is converted in the style `[-]d.ddde±dd`, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, six digits are produced; if the precision is zero, no decimal point appears. The **E** format code produces a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

**g, G** The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** is used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeros are removed from the result; a decimal point appears only if it is followed by a digit. Precision 0 yields a result with just enough significance to round to exactly the original value when converted back to binary as by [scanf\(3\)](#).

**c** The character argument is printed.

**s** The argument is taken to be a string (character pointer) and characters from the string are printed until a null character (**\0**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A zero value for the argument yields undefined results. (For compatibility with other versions of *printf*, a field width with a leading zero results in zero-padding the string instead of blank-padding it. This does not imply an octal value for the field width.)

**%** Print a **%**; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if *putc* had been called; see [getc\(3\)](#).

## EXAMPLES

```
printf("%s, %s %d, %d:%.2d", weekday, month, day, hour, min);
```

Print a date and time in the form 'Sunday, July 3, 10:02', where *weekday* and *month* are pointers to null-terminated strings.

```
printf("pi = %.5f", 4*atan(1.0));
```

Print  $\pi$  to 5 decimal places.

## SEE ALSO

[ecvt\(3\)](#), [scanf\(3\)](#), [stdio\(3\)](#), [print\(3\)](#)

## BUGS

The **!** indicator for field width is nonstandard.

**NAME**

orient, normalize – map projections

**SYNOPSIS**

```
orient(lat, lon, rot)
double lat, lon, rot;

normalize(p)
struct place *p;
```

**DESCRIPTION**

Users of *map(7)* may skip to the description of ‘Projection generators’ below.

The functions *orient* and *normalize* plus a collection of map projection generators are loaded by option **-lmap** of *ld(1)*. Most of them calculate maps for a spherical earth. Each map projection is available in one standard form, into which data must be normalized for transverse or nonpolar projections.

Each standard projection is displayed with the Prime Meridian (longitude 0) being a straight vertical line, along which North is up. The orientation of nonstandard projections is specified by *orient*. Imagine a transparent gridded sphere around the globe. First turn the overlay about the North Pole so that the Prime Meridian (longitude 0) of the overlay coincides with meridian *lon* on the globe. Then tilt the North Pole of the overlay along its Prime Meridian to latitude *lat* on the globe. Finally again turn the overlay about its ‘North Pole’ so that its Prime Meridian coincides with the previous position of (the overlay’s) meridian *rot*. Project the desired map in the standard form appropriate to the overlay, but presenting information from the underlying globe. It is not useful to use *orient* without using *normalize*.

*Normalize* converts latitude-longitude coordinates on the globe to coordinates on the overlaid grid. The coordinates and their sines and cosines are input to *normalize* in a **place** structure. Transformed coordinates and their sines and cosines are returned in the same structure.

```
struct place {
    double radianlat, sinlat, coslat;
    double radianlon, sinlon, coslon;
};
```

The projection generators return a pointer to a function that converts normalized coordinates to *x-y* coordinates for the desired map, or 0 if the required projection is not available. The returned function is exemplified by *proj* in this example:

```
struct place pt;
int (*proj)() = mercator();
double x, y;

orient(45.0, 30.0, 180.0);      /* set coordinate rotation */
. . .                          /* fill in the pt structure */
normalize(&pt);                 /* rotate coordinates */
if((*proj>(&pt, &x, &y) > 0)    /* project onto x,y plane */
    plot(x, y);
```

The projection function (**\*proj**()) returns 1 for a good point, 0 for a point on a wrong sheet (e.g. the back of the world in a perspective projection), and -1 for a point that is deemed unplotable (e.g. points near the poles on a Mercator projection).

Scaling may be determined from the *x-y* coordinates of selected points. Latitudes and longitudes are measured in degrees for ease of specification for *orient* and the projection generators but in radians for ease of calculation for *normalize* and *proj*. In either case latitude is measured positive north of the equator, and longitude positive west of Greenwich. Radian longitude should be limited to the range  $-\pi \leq lon < \pi$ .

**Projection generators**

Equatorial projections centered on the Prime Meridian (longitude 0). Parallels are straight horizontal lines.

```
mercator() equally spaced straight meridians, conformal, straight compass courses
sinusoidal() equally spaced parallels, equal-area, same as bonne(0)
cylequalarea(lat0) equally spaced straight meridians, equal-area, true scale on lat0
```

**cylindrical()** central projection on tangent cylinder  
**rectangular(lat0)** equally spaced parallels, equally spaced straight meridians, true scale on *lat0*  
**gall(lat0)** parallels spaced stereographically on prime meridian, equally spaced straight meridians, true scale on *lat0*  
**mollweide()** (homalographic) equal-area, hemisphere is a circle

Azimuthal projections centered on the North Pole. Parallels are concentric circles. Meridians are equally spaced radial lines.

**azequidistant()** equally spaced parallels, true distances from pole  
**azequalarea()** equal-area  
**gnomonic()** central projection on tangent plane, straight great circles  
**perspective(dist)** viewed along earth's axis *dist* earth radii from center of earth  
**orthographic()** viewed from infinity  
**stereographic()** conformal, projected from opposite pole  
**laue()**  $radius = \tan(2 \times colatitude)$ , used in xray crystallography  
**fisheye(n)** stereographic seen from just inside medium with refractive index *n*  
**newyorker(r)**  $radius = \log(colatitude/r)$ : extreme 'fish-eye' view from pedestal of radius *r* degrees

Polar conic projections symmetric about the Prime Meridian. Parallels are segments of concentric circles. Except in the Bonne projection, meridians are equally spaced radial lines orthogonal to the parallels.

**conic(lat0)** central projection on cone tangent at *lat0*  
**simpleconic(lat0,lat1)** equally spaced parallels, true scale on *lat0* and *lat1*  
**lambert(lat0,lat1)** conformal, true scale on *lat0* and *lat1*  
**albers(lat0,lat1)** equal-area, true scale on *lat0* and *lat1*  
**bonne(lat0)** equally spaced parallels, equal-area, parallel *lat0* developed from tangent cone

Projections with bilateral symmetry about the Prime Meridian and the equator.

**polyconic()** parallels developed from tangent cones, equally spaced along Prime Meridian  
**aitoff()** equal-area projection of globe onto 2-to-1 ellipse, based on *azequalarea*  
**lagrange()** conformal, maps whole sphere into a circle  
**bicentric(lon0)** points plotted at true azimuth from two centers on the equator at longitudes  $\pm lon0$ , great circles are straight lines (a stretched gnomonic projection)  
**elliptic(lon0)** points are plotted at true distance from two centers on the equator at longitudes  $\pm lon0$   
**globular()** hemisphere is circle, circular arc meridians equally spaced on equator, circular arc parallels equally spaced on 0- and 90-degree meridians  
**vandergrinten()** sphere is circle, meridians as in *globular*; circular arc parallels resemble *mercator*

Doubly periodic conformal projections.

**guyou()** W and E hemispheres are square  
**square()** world is square with Poles at diagonally opposite corners  
**tetra()** map on tetrahedron with edge tangent to Prime Meridian at S Pole, unfolded into equilateral triangle  
**hex()** world is hexagon centered on N Pole, N and S hemispheres are equilateral triangles

Miscellaneous projections.

**harrison(dist,angle)** oblique perspective from above the North Pole, *dist* earth radii from center of earth, looking along the Date Line *angle* degrees off vertical  
**trapezoidal(lat0,lat1)** equally spaced parallels, straight meridians equally spaced along parallels, true scale at *lat0* and *lat1* on Prime Meridian

Retroazimuthal projections. At every point the angle between vertical and a straight line to 'Mecca', latitude *lat0* on the prime meridian, is the true bearing of Mecca.

**mecca(lat0)** equally spaced vertical meridians  
**homing(lat0)** distances to 'Mecca' are true

Maps based on the spheroid. Of geodetic quality, these projections do not make sense for tilted orientations. For descriptions, see corresponding maps above.

`sp_mercator()`  
`sp_albers(lat0,lat1)`

**SEE ALSO**

*map(7), map(5), plot(3)*

**BUGS**

Only one projection and one orientation can be active at a time.  
The west-longitude-positive convention betrays Yankee chauvinism.

**NAME**

putshares – write shares file entry

**SYNOPSIS**

```
#include <shares.h>
```

```
int putshares(lp, extime)
struct lnode * lp;
unsigned long extime;
```

**DESCRIPTION**

*Putshares* writes the shares entry with the same uid as *lp->l\_uid*. *Putshares* returns -1 if any error occurs, 0 if *lp->l\_uid* is greater than MAXUID, or the size of the entry if successfully written.

**SEE ALSO**

closeshares(3), getshares(3), getshput(3), openshares(3), sharesfile(3).

**DIAGNOSTICS**

*Putshares* returns 0 if *lp->l\_uid* is greater than MAXUID, or -1 on error.

**NAME**

qsort – quicker sort

**SYNOPSIS**

**qsort**(base, nel, width, compar)

**char** \*base;

**int** (\*compar)();

**DESCRIPTION**

*Qsort* (quicker sort) sorts an array into nondecreasing order. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of a comparison routine to be called with pointers to elements being compared. It should be declared as

```
    compar(a, b)
```

```
    char *a, *b;
```

The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

**SEE ALSO**

[sort\(1\)](#)

**NAME**

rand, lrand, frand, nrand, srand – random number generator

**SYNOPSIS**

**int** rand()

**long** lrand()

**double** frand()

**int** nrand(val)

**int** val;

**srand**(seed)

**int** seed;

**DESCRIPTION**

*Rand* uses a linear feedback random number generator to return uniform pseudo-random numbers  $x$ ,  $0 \leq x < 2^{15-32}$ .

*Lrand* returns a uniform long  $x$ ,  $0 \leq x < 2^{31-48}$ .

*Frاند* returns a uniform double  $x$ ,  $0.0 \leq x < 1.0$ , always a multiple of  $2^{-31}$ .

*Nrand* returns a uniform integer  $x$ ,  $0 \leq x < val$ .

The generators are initialized by calling *srand* with whatever you like as argument. To get a different starting value each time,

```
srand((int)time((long *)0));
```

will work as long as it is not called more often than once per second.

**BUGS**

*Rand* and *lrand* are quite machine-dependent. Although *frاند* and *nrand* are more portable, they appear in few versions of Unix.



**NAME**

`re_bm`, `re_cw`, `re_re` – string and pattern matching

**SYNOPSIS**

```
#include <re.h>

re_bm *re_bmcomp(b, e, map)
char *b, *e;
unsigned char map[256];

int re_bmexec(pat, rdfn, matchfn)
re_bm *pat;
int (*rdfn)(), (*matchfn)();

void re_bmfree(pat);
re_bm *pat;

re_cw *re_cwinit(map)
unsigned char map[256];

void re_cwadd(pat, b, e)
re_cw *pat;
char *b, *e;

void re_cwcomp(pat)
re_cw *pat;

int re_cwexec(pat, rdfn, matchfn)
re_cw *pat;
int (*rdfn)(), (*matchfn)();

void re_cwfree(pat);
re_cw *pat;

re_re *re_recomp(b, e, map)
char *b, *e;
unsigned char map[256];

re_rexec(pat, b, e, match)
re_re *pat;
char *b, *e, *match[10][2];

void re_refree(pat);
re_re *pat;

void re_error(str);
char *str;
```

**DESCRIPTION**

These routines search for patterns in strings. The `re_re` routines search for general regular expressions (defined below) using a lazily evaluated deterministic finite automaton. The more specialized and faster `re_cw` routines search for multiple literal strings using the Commentz-Walter algorithm. The still more specialized and efficient `re_bm` routines search for a single string using the Boyer-Moore algorithm. The routines handle strings designated by pointers to the first character of the string and to the character following the string.

To use the `re_bm` routines, first build a recognizer by calling `re_bmcomp`, which takes the search string and a character map; all characters are compared after mapping. Typically, `map` is initialized by a loop similar to

```
for(i = 0; i < 256; i++) map[i] = i;
```

and its value is no longer required after the call to `re_bmcomp`.

The recognizer can be run (multiple times) by calling `re_bmexec`,

which stops and returns the first non-positive return from either `rdfn`

or  
*matchfn*.  
 The recognizer calls the supplied function  
*rdfn*  
 to obtain input and  
*matchfn*  
 to report text matching the search string.

*Rdfn*  
 should be declared as

```
int rdfn(pb, pe)
char **pb, **pe;
```

where **\*pb** and **\*pe** delimit an as yet unprocessed text fragment (none if **\*pb==\*pe**) to be saved across the call to *rdfn*. On return, **\*pb** and **\*pe** point to the new text, including the saved fragment. *Rdfn* returns 0 for EOF, negative for error, and positive otherwise. The first call to *rdfn* from each invocation of *re\_bmexec* has **\*pb==0**.

*Matchfn* should be declared as

```
int matchfn(pb, pe)
char **pb, **pe;
```

where **\*pb** and **\*pe** delimit the matched text. *Matchfn* sets **\*pb**, **\*pe**, and returns a value in the same way as *rdfn*.

To use the *re\_cw* routines, first build the recognizer by calling *re\_cwinit*, then *re\_cwadd* for each string, and finally *re\_cwcomp*. The recognizer is run by *re\_cwexec* analogously to *re\_bmexec*.

A full regular expression recognizer is compiled by *re\_recomp* and executed by *re\_reexec*, which returns 1 if there was a match and 0 if there wasn't. The strings that match subexpressions are returned in array *match* using the above convention. *match[0]* refers to the whole matched expression. If *match* is zero, then no match delimiters are set.

The routine *re\_error* prints its argument on standard error and exits. You may supply your own version for specialized error handling. If *re\_error* returns rather than exits, the compiling routines (e.g. *re\_bmcomp*) will return 0.

The recognizers that these routines construct occupy storage obtained from *malloc(3)*. The storage can be deallocated by *re\_refree*.

## Regular Expressions

The syntax for a regular expression **e0** is

```
e3: literal | charclass | '.' | '^' | '$' | '\n | '(' e0 ')'
e2: e3
   | e2 REP
REP: '*' | '+' | '?' | '\{' RANGE '\}'
RANGE: int | int ',' | int ',' int
e1: e2
   | e1 e2
e0: e1
   | e0 ALT e1
ALT: '|' | newline
```

A literal is any non-metacharacter or a metacharacter (one of `.*+?[]()|^$`) preceded by `\`.

A charclass is a nonempty string *s* bracketed [*s*] (or [*^s*]); it matches any character in (or not in) *s*. In *s*, the metacharacters other than ] have no special meaning, and ] may only appear as the first letter. A substring *a-b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters between *a* and *b*.

A `\` followed by a digit *n* matches a copy of the string that the parenthesized subexpression beginning with the *n*th (, counting from 1, matched.

A `.` matches any character.

A `^` matches the beginning of the input string; `$` matches the end.

The **REP** operators match zero or more (`*`), one or more (`+`), zero or one (`?`), exactly  $m$  (`\{m\}`),  $m$  or more (`\{m,\}`), and any number between  $m$  and  $n$  inclusive (`\{m,n\}`), instances respectively of the preceding regular expression **e2**.

A concatenated regular expression, **e1 e2**, matches a match to **e1** followed by a match to **e2**.

An alternative regular expression, **e0 ALT e1**, matches either a match to **e0** or a match to **e1**.

A match to any part of a regular expression extends as far as possible without preventing a match to the remainder of the regular expression.

### SEE ALSO

[regexp\(3\)](#), [gre\(1\)](#)

### DIAGNOSTICS

Routines that return pointers return 0 on error.

### BUGS

Between [re\(3\)](#) and [regexp\(3\)](#) there are too many routines.

**NAME**

`re_comp`, `re_exec` – regular expression handler

**SYNOPSIS**

**char \*re\_comp(s)**

**char \*s;**

**re\_exec(s)**

**char \*s;**

**DESCRIPTION**

*Re\_comp* compiles a regular expression into an internal form suitable for pattern matching. *Re\_exec* checks the argument string against the last string passed to *re\_comp*.

*Re\_comp* returns 0 if the string *s* was compiled successfully; otherwise a string containing an error message is returned. If *re\_comp* is passed 0 or a null string, it returns without changing the currently compiled regular expression.

*Re\_exec* returns 1 if the string *s* matches the last compiled regular expression, 0 if the string *s* failed to match the last compiled regular expression, and -1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both *re\_comp* and *re\_exec* may have trailing or embedded newline characters; they are terminated by nulls. The regular expressions are otherwise as described for *ed(1)*.

**SEE ALSO**

*ed(1)*, *expr(1)*, *regex(3)*

**DIAGNOSTICS**

*Re\_exec* returns -1 for an internal error.

**BUGS**

These routines have been superseded by the more general routines of *regex(3)*. They have been retained only for compatibility.

**NAME**

regcomp, regexc, regsub, regerror – regular expression

**SYNOPSIS**

```
#include <regexp.h>

regexp *regcomp(exp)
char *exp;

int regexc(prog, string, match, msize)
regexp *prog;
char *string;
regsubexp *match;
int msize;

void regsub(source, dest, match, msize)
char *source, *dest;
regsubexp *match;
int msize;

void regerror(msg)
char *msg;
```

**DESCRIPTION**

*Regcomp* compiles a regular expression and returns a pointer to a compiled regular expression. The space is allocated by *malloc(3)* and may be released by *free*. Regular expressions are as in *re(3)* except that newlines are not operators and back-references (with *\n*) are not supported.

*Regexc* matches a null-terminated *string* against the compiled regular expression in *prog*. If it matches, *regexc* returns a non-zero value and fills in the array *match* with character pointers to the substrings of *string* that correspond to the parenthesized subexpressions of *exp*: **match[i].sp** points to the beginning and **match[i].ep** points just beyond the end of the *i*th substring. (Subexpression *i* begins at the *i*th left parenthesis, counting from 1.) Pointers in **match[0]** pick out the substring that corresponds to the whole regular expression. Unused elements of *match* are filled with zeros. Matches involving \*, +, and ? are extended as far as possible. The number of array elements in *match* is given by *msize*. The structure of elements of *match* is:

```
typedef struct {
    char *sp;
    char *ep;
} regsubexp;
```

*Regsub* places in *dest* a substitution instance of *source* in the context of the last *regexc* performed using *match*. Each instance of *\n*, where *n* is a digit, is replaced by the string delimited by **match[n].sp** and **match[n].ep**. Each instance of & is replaced by the string delimited by **match[0].sp** and **match[0].ep**.

*Regerror*, called whenever an error is detected in *regcomp*, *regexc*, or *regsub*, writes the string *msg* on the standard error file and exits. *Regerror* can be replaced to perform special error processing.

**SEE ALSO**

*gre(1)*, *re(3)*, *expr(1)*

**DIAGNOSTICS**

*Regcomp* returns (**regexp \***)**0** for an illegal expression or other failure. *Regexc* returns 0 if *string* is not accepted.

**NAME**

streambuf – interface for derived classes

**SYNOPSIS**

```
#include <iostream.h>
typedef long streamoff, streampos;
class ios {
public:
    enum        seek_dir { beg, cur, end };
    enum        open_mode { in, out, ate, app };
    // and lots of other stuff ...
};
class streambuf {
public:
    streambuf() ;
    streambuf(char* p, int len, int i=0);
protected:
    void        dbp() ;
    int         allocate();
    char*       base();
    int         blen();
    char*       eback();
    char*       ebuf();
    char*       egptr();
    char*       epptr();
    void        gbump(int n);
    char*       gptr();
    char*       pbase();
    void        pbump(int n);
    char*       pptr();
    void        setg(char* eb, char* g, char* eg);
    void        setp(char* p, char* ep);
    void        setb(char* b, char* eb, int a = 0 );
    int         unbuffered();
    void        unbuffered(int);
    virtual int doallocate();
    virtual int pbackfail(int c);
    virtual     streambuf() ;
public:
    virtual int overflow(int c=EOF);
    virtual int underflow();
    virtual streambuf*
        setbuf(char* p, int len);
    virtual streampos
        seekpos(streampos, open_mode=input|output);
    virtual streampos
        seekoff(streamoff, seek_dir, open_mode=input|output);
    virtual int sync();
};
```

**DESCRIPTION**

streambufs implement the buffer abstraction described in *sbuf.pub*(3C++). But the `streambuf` class itself contains only basic members for manipulating the characters and normally a class derived from `streambuf` will be used. This man page describes the interface needed by programmers who are coding a derived class. Broadly speaking there are two kinds of members described here. The non-virtual functions are provided for manipulating a **streambuf** in ways that are appropriate in a derived class. Their descriptions reveal details of the implementation that would be inappropriate in the public interface. The virtual functions permit the derived class to specialize the **streambuf** class in ways appropriate to the

specific sources and sinks that it is implementing. The descriptions of virtuals explain the obligations of the virtuals of the derived class. If the virtuals behave as specified, the **streambuf** will behave as specified in the public interface. However, if the virtuals do not behave as specified, then the `streambuf` may not behave properly, and an `iostream` (or any other code) that relies on proper behavior of the `streambuf` may not behave properly either.

Assume

- **sb** is a `streambuf*`.
- **i** and **n** are `int`.
- **ptr**, **b**, **eb**, **p**, **ep**, **eb**, **g**, and **eg** are `char*`.
- **c** is an `int` character (positive or EOF).
- **pos** is a `streampos`. (See `iostream(3C++)`.)
- **off** is a `streamoff`.
- **dir** is a `seekdir`.
- **mode** is a `open_mode`.

Constructors:

#### **streambuf()**

Constructs an empty buffer corresponding to an empty sequence.

#### **streambuf(b,len,i)**

Constructs an empty buffer and then sets up the reserve area to be the **len** bytes starting at **b**. (**i** is present for backward compatibility with the stream package. The effect if it is not 0 is undefined.)

The protected members of `streambuf` present an interface to derived classes organized around three areas (arrays of bytes) managed cooperatively by the base and derived classes. They are the get area, the put area, and the reserve area. The get and the put area are normally disjoint, but they may both overlap the reserve area, whose primary purpose is to be a resource in which space for the put and get areas can be allocated. The get and the put areas are changed as characters are put into and gotten from the buffer, but the reserve area normally remains fixed. The areas are defined by a collection of `char*` values. The buffer abstraction is described in terms of pointers that point between characters, but the `char*` values must point at `chars`. To establish a correspondence the `char*` values should be thought of as pointing just before the byte they really point at.

Functions to examine the pointers are:

#### **ptr=sb->base()**

Returns a pointer to the first byte of the reserve area.  
Space between **sb->base()** and **sb->eback()** is the reserve area.

#### **ptr=sb->eback()**

Returns a pointer to a lower bound on **sb->gptr()**. Space between **sb->eback()** and **sb->gptr()** is available for putback.

#### **ptr=sb->ebuf()**

Returns a pointer to the byte after the last byte of the reserve area.

#### **ptr=sb->egptr()**

Returns a pointer to the byte after the last byte of the get area.

#### **ptr=sb->epptr()**

Returns a pointer to the byte after the last byte of the put area.

#### **ptr=sb->gptr()**

Returns a pointer to the first byte of the get area. The available characters are those between **sb->gptr()** and **sb->egptr()**. The next character fetched will be **\*sb->gptr()** unless **sb->egptr()** is less than or equal to **sb->gptr()**.

#### **ptr=sb->pbase()**

Returns a pointer to the put area base. Characters between **sb->pbase()** and **sb->pptr()** have been stored into the buffer and not yet consumed.

**ptr=sb->pptr()**

Returns a pointer to the first byte of the put area. The space between **sb->pptr()** and **sb->epptr()** is the put area and characters will be stored here.

The member functions for setting the pointers:

**sb->setb(b,eb,i)**

Sets **base** and **ebase** to **b** and **eb** respectively. **i** controls whether the area will be subject to automatic deletion. If **i** is non zero, then `delete b` will be done when **base** is changed by another call of **setb**, or when the destructor is called for **\*sb**. If **b** and **eb** are both null then we say that there is no reserve area. If **b** is non-null, there is a reserve area even if **eb** is less than **b** and so the reserve area has zero length.

**sb->setp(p,ep)**

Sets **pptr** to **p**, **pbase** to **p**, and **epptr**.

**sb->setg(eg,g,eb)**

Sets **eback** to **eb**, **gptr** to **g**, and **egptr** to **eg**.

Other non-virtual members:

**i=sb->allocate()**

Tries to set up a reserve area. If a reserve area already exists or if **sb->unbuffered()** is nonzero returns 0 without doing anything. If the attempt to allocate space fails **allocate** returns EOF. Otherwise (allocation succeeds) **allocate** returns 1. **allocate** is not called by any member of `streambuf` except virtuals.

**i=sb->blen()**

Returns the current size (in chars) of the current reserve area.

**dbp()** Writes directly on file descriptor 1 information in ASCII about the state of the buffer. It is intended for debugging and nothing is specified about the form of the output. It is considered part of the protected interface because the information it prints can only be understood in relation to that interface, but it is a public function so that it can be called anywhere during debugging.

**sb->gbump(n)**

Increments **gptr** by **n** which may be positive or negative. No checks are made on whether the new value of **gptr** is in bounds.

**sb->pbump(n)**

Increments **pptr** by **n** which may be positive or negative. No checks are made on whether the new value of **pptr** is in bounds.

**sb->unbuffered(i)****i=sb->unbuffered()**

There is a private variable known as **sb**'s buffering state. **sb->unbuffered(i)** sets the value of this variable to **i** and **sb->unbuffered()** returns the current value. This state is independent of the actual allocation of a reserve area. Its primary purpose is to control whether a reserve area is allocated automatically by **allocate**.

Virtual functions must be redefined in derived classes to specialize the behavior of **streambufs**:

**i=sb->doallocate()**

Is called when **allocate** determines that space is needed. **doallocate** is required to call **setb** to provide a reserve area or to return EOF if it cannot. It is only called if **sb->unbuffered()** is non-zero and **sb->base()** is non-zero.

**i=overflow(c)**

Is called to consume characters. If **c** is not EOF it also must either save **c** or consume it. Usually it is called when the put area is full and an attempt is being made to store a new character, but it can be called at other times. The normal action is to consume the characters between **pbase** and **pptr**, call **setp** to establish a new put area, and if **c!=EOF** store it (using **sputc**). If **sb->unbuffered()** is non-zero, **overflow** is not allowed to call **setp** and so must consume **n** **sb->overflow** should return EOF to indicate an error; otherwise it should return something else.



**i=sb->pbackfail(c)**

Is called when **eback** equals **gp** and an attempt has been made to putback **c**. If this situation can be dealt with (e.g., by repositioning an external file), **pbackfail** should return **c**; otherwise it should return EOF.

**pos=sb->seekoff(off,dir,mode)**

Repositions the get and/or put pointers (i.e., the abstract get and put pointers, not **p** and **g**). The meanings of **off** and **dir** are discussed in *sbuf.pub(3C++)*. **mode** specifies whether the put pointer (**output** bit set) or the get pointer (**input** bit set) is to be modified. Both bits may be set in which case both pointers should be affected. A class derived from **streambuf** is not required to support repositioning. **seekoff** should return EOF if the class does not support repositioning. If the class does support repositioning, **seekoff** should return the new position or EOF on error.

**pos=sb->seekpos(pos,mode)**

Repositions the streambuf get and/or put pointer to **pos**. **mode** specifies which pointers are affected as for **seekoff**. Returns **pos** (the argument) or EOF if the class does not support repositioning or an error occurs.

**sb=sb->setbuf(ptr,len)**

Offers the array at **ptr** with **len** bytes should be used as a reserve area. The normal interpretation is that if **ptr** or **len** are zero then this is a request to make the **sb** unbuffered. The derived class may use this area or not as it chooses. It may accept or ignore the request for unbuffered state as it chooses. **setbuf** should return **sb** if it honors the request. Otherwise it should return 0.

**i=sb->sync()**

Is called to give the derived class a chance to look at the state of the areas, and synchronize them with any external representation. Normally **sync** should consume any characters that have been stored into the put area, and if possible give back to the source any characters in the get area that have not been fetched. When **sync** returns there should not be any unconsumed characters, and the get area should be empty. **sync** should return EOF if some kind of failure occurs.

**i=sb->underflow()**

Is called to supply characters for fetching, i.e., to create a condition in which the get area is not empty. If it is called when there are characters in the get area it should return the first character. If the get area is empty it should create a nonempty get area and return the next character (which it should also leave in the get area). If there are no more characters available **underflow** should return EOF and leave an empty put area.

The default definitions of the virtual functions:

**i=sb->streambuf::doallocate()**

Attempts to allocate a reserve area using operator `new`.

**i=sb->streambuf::overflow(n)**

Is compatible with the old stream package, but that behavior is not considered part of the specification of the iostream package. So **streambuf::overflow** should be treated as if it had undefined behavior. That is, derived classes should always define it.

**i=sb->streambuf::pbackfail(n)**

Returns EOF.

**pos=sb->streambuf::seekpos(pos,mode)**

Returns **sb->seekoff(streamoff(pos),seek\_beg,mode)**. Thus to define seeking in a derived class, it is frequently only necessary to define **seekoff** and use the inherited **streambuf::seekpos**.

**pos=sb->streambuf::seekoff(off,dir,mode)**

Returns EOF.

**sb=sb->streambuf::setbuf(ptr,len)**

Will honor the request when ever there is no reserve area.

**i=sb->streambuf::sync()**

Returns 0 if the get area is empty and there are no unconsumed characters. Otherwise it returns EOF.

**i=sb->streambuf::underflow()**

Is compatible with the old stream package, but that behavior is not considered part of the specification of the iostream package. So **streambuf::underflow** should be treated as if it had undefined behavior. That is, it should always be defined in derived classes.

**CAVEATS**

The constructors are public for compatibility with the old stream package. They ought to be protected.

The interface for unbuffered actions is awkward. It's hard to write **underflow** and **overflow** virtuals that behave properly for unbuffered `streambufs` without special casing. Also there is no way for the virtuals to react sensibly to multi character gets or puts.

Although the public interface to `streambufs` deals in characters and bytes, the interface to derived classes deals in `chars`. Since a decision had to be made on the types of the real data pointers, it seemed easier to reflect that choice in the types of the protected members than to duplicate all the members with both plain and unsigned char versions. But perhaps all these uses of `char*` ought to have been with a typedef.

The implementation contains a variant of **setbuf** that accepts a third argument. It is present only for compatibility with the old stream package.

**SEE ALSO**

`sbuf.pub(3C++)` `streambuf(3C++)` `iostream(3C++)`

**NAME**

streambuf – public interface of character buffering class

**SYNOPSIS**

```
#include <iostream.h>
typedef long streamoff, streampos;
class ios {
public:
    enum        seek_dir { beg, cur, end };
    enum        open_mode { in, out, ate, app };
    // and lots of other stuff ...
};
class streambuf {
public :
    int         in_avail();
    int         out_waiting();
    int         sbumpc();
    streambuf*  setbuf(char* ptr, int len, int count=0);
    streampos   seekpos(streampos,open_mode);
    streampos   seekoff(streamoff,seek_dir,open_mode);
    int         sgetc();
    int         sgetn(char* ptr,int n);
    int         snextc();
    int         sputbackc(char c);
    int         sputc(int c);
    int         sputn(const char* s,int n);
    void        stoss();
    int         sync();
}
```

**DESCRIPTION**

The `streambuf` class supports buffers into which characters can be inserted (put) or from which characters can be fetched (gotten). Abstractly such a buffer is a sequence of characters together with one or two pointers (a get and/or a put pointer) that define the location at which characters are to be inserted or fetched. The pointers should be thought of as pointing between characters rather than at them. This makes it easier to understand the boundary conditions (a pointer before the first character or after the last). Some of the effects of getting and putting are defined by this class but most of the details are left to specialized classes derived from `streambuf`.

Classes derived from `streambuf` vary in their treatments of the get and put pointers. The simplest are unidirectional buffers which permit only gets or only puts. Such classes serve as pure sources (producers) or sinks (consumers) of characters. Queue-like buffers have a put and a get pointer which move independently of each other. In such buffers characters that are stored are held (i.e., queued) until they are later fetched. File-like buffers permit both gets and puts but have only a single pointer. (An alternative description is that the get and put pointers are tied together so that when one moves so does the other.)

Most `streambuf` member functions are organized into two phases. As far as possible, operations are performed inline by storing into or fetching from arrays (the get area and the put area). From time to time, virtual functions are called to deal with collections of characters. Generally the user of a `streambuf` does not have to know anything about these details, but some the public members pass back information about the state of the areas.

Assume:

- **i**, **n**, and **len** are `int`.
- **c** is an `int`. It always holds a "character" value or EOF. A "character" value is always positive even when `char` is normally sign extended.
- **sb** and **sb1** are `streambuf*`.
- **ptr** is a `char*`.
- **off** is a `streamoff`.
- **pos** is a `streampos`.

- **dir** is a `seekdir`.
- **mode** is a `open_mode`.

Public member functions:

**i=sb->in\_avail()**

Returns the number of characters that are immediately available in the get area for fetching. That many characters may be fetched with a guarantee that no errors will be reported.

**i=sb->out\_waiting()**

Returns the number of characters in the put area that have not been consumed by virtuals.

**c=sb->sbumpc()**

Moves the get pointer forward one character and returns the character moved over. Returns EOF if the get pointer is currently at the end of the sequence.

**pos=sb->seekoff(off,dir,mode)**

Repositions the get and/or put pointers. **mode** specifies whether the put pointer (**output** bit set) or the get pointer (**input** bit set) is to be modified. Both bits may be set in which case both pointers should be affected. **off** is interpreted as a byte offset. (Notice that it is a signed quantity.) The meaning of possible values of **dir** are

- beg** The beginning of the stream.
- cur** The current position.
- end** The end of the stream. (End of file.)

Not all classes derived from **streambuf** support repositioning. **seek** will return EOF if the class does not support repositioning. If the class does support repositioning, **seek** will return the new position or EOF on error.

**pos=sb->seekpos(pos,mode)**

Repositions the streambuf get and/or put pointer to **pos**. **mode** specifies which pointers are affected as for **seekoff**. Returns **pos** (the argument) or EOF if the class does not support repositioning or an error occurs. In general a `streampos` should be treated as a "magic cookie" and no arithmetic should be performed on it. But two particular values have special meaning:

**streampos(0)**

The beginning of the file.

**streampos(EOF)**

Used as an error indication.

**c=sb->sgetc()**

Returns the character after the get pointer. Contrary to what most people expect from the name *IT DOES NOT MOVE THE GET POINTER*. Returns EOF if there is no character available.

**sb1=sb->setbuf(ptr,len,i)**

Offers the **len** bytes starting at **ptr** as the reserve area. If **ptr** is null or **len** is zero or less, then an unbuffered state is requested. Whether the offered area is used, or a request for unbuffered state is honored depends on details of the derived class. **setbuf** normally returns **sb**, but if it does not accept the offer or honor the request, it returns 0.

**i=sb->sgetn(ptr,n)**

Fetches the **n** characters following the get pointer and copies them to the area starting at **ptr**. When there are less than **n** characters left before the end of the sequence **sgetn** fetches whatever characters remain. **sgetn** repositions the get pointer following the fetched characters and returns the number of characters fetched.

**c=sb->snextc()**

Moves the get pointer forward one character and returns the character following the new position. It returns EOF if the pointer is currently at the end of the sequence or is at the end of the sequence after moving forward.

**i=sb->sputbackc(c)**

Moves the get pointer back one character. **c** must be the current content of the sequence just before the get pointer. The underlying mechanism may simply back up the get pointer or may

rearrange its internal data structures so the **c** is saved. Thus the effect of **sputbackc** is undefined if **c** is not the character before the get pointer. **putbackc** returns EOF when it fails. The conditions under which it can fail depend on the details of the derived class.

**i=sb->sputc(c)**

Stores **c** after the put pointer, and moves the put pointer over the stored character. Usually this extends the sequence. It returns EOF when an error occurs. The conditions that can cause errors depend on the derived class.

**i=sb->sputn(ptr,n)**

Stores the **n** characters starting at **ptr** after the put pointer. Moves the put pointer over them. Returns the number of characters stored successfully. Normally this is **n**, but it may be less when errors occur.

**sb->stoss()**

Moves the get pointer forward one character. If the pointer started at the end of the sequence this function has no effect.

**i=sb->sync()**

Establishes consistency between the internal data structures and the external source or sink. The details of this function depend on the derived class. Usually this "flushes" any characters that have been stored but not yet consumed, and "gives back" any characters that may have been produced but not yet fetched.

**CAVEATS**

**setbuf** does not really belong in the public interface. It is there for compatibility with the stream package.

Requiring the program to provide the previously fetched character to `sputback` is probably a botch.

**SEE ALSO**

`iostream(3C++)`, `sbuf.prot(3C++)`

**NAME**

scanf, fscanf, sscanf – formatted input

**SYNOPSIS**

```
#include <stdio.h>
```

```
scanf(format [ , pointer ] ... )
```

```
char *format;
```

```
fscanf(stream, format [ , pointer ] ... )
```

```
FILE *stream;
```

```
char *format;
```

```
sscanf(s, format [ , pointer ] ... )
```

```
char *s, *format;
```

**DESCRIPTION**

*Scanf* reads from the standard input stream *stdin*. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string *format*, described below, and a set of arguments, normally pointers, indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character \*, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by \*. Conversions other than c and [ skip white space and consume non-white-space characters up to the next inappropriate character or until the field width, if specified, is exhausted. The field width is either an integer constant or !. In the latter case, the width is taken from an integer argument that precedes the next pointer argument.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

- % A single % is expected in the input at this point; no assignment is done.
- d A decimal integer is expected; the corresponding argument should be an integer pointer.
- o an octal integer is expected; the corresponding argument should be an integer pointer.
- x A hexadecimal integer is expected; the corresponding argument should be an integer pointer.
- s A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which will be added. The input field is terminated by a space character or a newline.
- c A character is expected; the corresponding argument should be a character pointer. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.
- e
- f A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an **E** or **e** followed by an optionally signed integer.
- [
- [^ A character string is expected. The left bracket (or bracket and circumflex) is followed by a set of characters and a right bracket. When the set is introduced by [ (or [^), the string consists only of characters in (or not in) the set. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be preceded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be preceded by **l** to indicate a pointer to **double** rather than to **float**. The conversion characters **d**, **o** and **x** may be preceded by **h** to indicate a pointer to **short**.

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant **EOF** is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line

```
25 54.32E-1 thompson
```

will assign to *i* the value 25, *x* the value 5.432, and *name* will contain thompson\0. Or,

```
int i; float x; char name[50];
scanf("%2d%f*%d%[1234567890]", &i, &x, name);
```

with input

```
[CB]56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip 0123, and place the string 56\0 in *name*. The next call to *getchar* will return a.

## SEE ALSO

[atof\(3\)](#), [stdio\(3\)](#), [ungetc\(3\)](#)

## DIAGNOSTICS

The *scanf* functions return **EOF** on end of input, and a short count for missing or illegal data items.

## BUGS

The success of literal matches and suppressed assignments is not directly determinable.

The input scan stops short of the end of excessively long numbers.

There is no `%#`.

When no maximum field width is given in a `%s` or `%[ ]` conversion specification, improper input can overrun the output string and corrupt the program in arbitrarily malicious ways. The best alternative, `%!s`, is nonstandard.

A deprecated usage allows upper-case conversion characters as equivalents for lower-case characters preceded by `l`.

**NAME**

setbuf – assign buffering to a stream

**SYNOPSIS**

```
#include <stdio.h>
```

```
setbuf(stream, buf)
```

```
FILE *stream;
```

```
char buf[BUFSIZ];
```

**DESCRIPTION**

*Setbuf* is used after a stream has been opened but before it is read or written. It causes the character array *buf* to be used instead of an automatically allocated buffer. If *buf* is the constant pointer `NULL`, input/output will be completely unbuffered.

A buffer is normally obtained from [malloc\(3\)](#) upon the first [getc\(3\)](#) or [putc](#) on the file. Initially, the standard stream *stderr* is unbuffered, and the standard stream *stdout* is flushed automatically whenever new data is read by [getc](#). The latter magic may be dissolved by a call to *setbuf*.

**SEE ALSO**

[stdio\(3\)](#), [malloc\(3\)](#)



**NAME**

setjmp, longjmp – non-local goto

**SYNOPSIS**

```
#include <setjmp.h>
```

```
setjmp(env)
```

```
jmp_buf env;
```

```
longjmp(env, val)
```

```
jmp_buf env;
```

**DESCRIPTION**

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

*Setjmp* saves its stack environment in *env* for later use by *longjmp*. It returns value 0.

*Longjmp* restores the environment saved by the last call of *setjmp*. It then causes execution to continue as if the call of *setjmp* had just returned with value *val*. The invoker of *setjmp* must not itself have returned in the interim. All accessible data have values as of the time *longjmp* was called.

**SEE ALSO**

[signal\(2\)](#)

**NAME**

setlimits – set limits structure

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <sys/lnode.h>
```

```
setlimits(limits)
```

```
struct lnode *limits;
```

**DESCRIPTION**

This library routine sets an in-core limits structure. If necessary, it also sets any group limits structures. *Limits* points to an *lnode*.

The *lnode* pointed to by the argument *limits* is first examined to see if its scheduling group is *root*. If not, the *lnode* for the group is obtained (via [getshares\(3\)](#)) and passed to a recursive call to *setlimits*. Finally the original *lnode* is set with the L\_SETLIM call to the [limits\(2\)](#) system call.

If the details for any group encountered cannot be found in the *limits* data-base, then the group is set to *root*.

Note that the */etc/shares* file may be left open by this routine.

**DIAGNOSTICS**

As for the [limits\(2\)](#) system call.

Any error cause a -1 to be returned.

**SEE ALSO**

[limits\(2\)](#), [closeshares\(3\)](#), [getshares\(3\)](#), [setupshares\(3\)](#).

**NAME**

setupgroups – set access group vector for invoker

**SYNOPSIS**

**setupgroups(name, gid)**

**char \*name;**

**int gid;**

**DESCRIPTION**

This library routine sets up the invoker's access group vector by searching the */etc/group* file for groups matching *name*. The invoker's real group ID should be passed in *gid*, as there is no need to load this into the vector.

*Setgroups* returns 0 for success, or *-1* on error with *errno* set appropriately.

This routine is safe to use on systems where the multiple access groups system calls have not been installed.

**SEE ALSO**

setgroups(2), getgrent(3), login(8).

**NAME**

setupshares – set kernel shares for a user

**SYNOPSIS**

**setupshares**(uid, efp)

**int** uid;

**void** (\*efp)();

**DESCRIPTION**

This library routine sets up a kernel shares structure for the user represented by *uid*. It extracts the share details for the user from the shares data-base, decays the usage figure up to the current time, and uses [setlimits\(3\)](#) to install the shares in the kernel.

If the system is out of *lnode* structures, then the structure for the default user “other” is used. If this also fails, then the structure for the super-user is used.

If there are any errors, and the second argument is non-NULL, the function will be called with a [printf\(3\)](#) format string and at most one extra argument. A non-zero result is returned for un-recoverable errors. Otherwise, *setupshares* returns **0**.

This routine is safe to use on systems where the share scheduler has not been installed, or is inactive.

**DIAGNOSTICS**

*Setupshares* returns a non-zero result if [setlimits\(3\)](#) returns an error other than ETOOMANYU. The optional error routine is called if [setlimits\(3\)](#) returns any error, or if no shares have been allocated to the user.

**SEE ALSO**

getshares(3), closeshares(3), setlimits(3), share(5).

**NAME**

sharesfile – change name of shares file

**SYNOPSIS**

```
int sharesfile(s)  
char * s;
```

**DESCRIPTION**

This routine closes the old shares file (if open) and resets its name to the string passed.

**SEE ALSO**

closeshares(3), getshares(3), getshput(3), openshares(3), putshares(3).

**NAME**

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

**SYNOPSIS**

```
#include <math.h>
```

```
double sin(x)
```

```
double x;
```

```
double cos(x)
```

```
double x;
```

```
double tan(x)
```

```
double x;
```

```
double asin(x)
```

```
double x;
```

```
double acos(x)
```

```
double x;
```

```
double atan(x)
```

```
double x;
```

```
double atan2(y, x)
```

```
double x, y;
```

**DESCRIPTION**

*Sin*, *cos* and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

*Asin* returns the arc sin in the range  $-\pi/2$  to  $\pi/2$ .

*Acos* returns the arc cosine in the range 0 to  $\pi$ .

*Atan* returns the arc tangent of  $x$  in the range  $-\pi/2$  to  $\pi/2$ .

*Atan2* returns the arc tangent of  $y/x$  in the range  $-\pi$  to  $\pi$ .

**DIAGNOSTICS**

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0; *errno* is set to EDOM. The value of *tan* at its singular points is a huge number, and *errno* is set to ERANGE.

**BUGS**

The value of *tan* for arguments greater than about  $2^{31}$  is garbage.

**NAME**

sinh, cosh, tanh – hyperbolic functions

**SYNOPSIS**

```
#include <math.h>
```

```
double sinh(x)
```

```
double x;
```

```
double cosh(x)
```

```
double x;
```

```
double tanh(x)
```

```
double x;
```

**DESCRIPTION**

These functions compute the designated hyperbolic functions for real arguments.

**DIAGNOSTICS**

*Sinh* and *cosh* return a huge value of appropriate sign when the correct value would overflow and set *errno* to EDOM.

**NAME**

sleep – suspend execution for an interval

**SYNOPSIS**

**sleep(seconds)**  
**unsigned seconds;**

**DESCRIPTION**

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to one second less than that requested, because scheduled wakeups occur at fixed second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent one second later.

**SEE ALSO**

[alarm\(2\)](#)



**NAME**

strstreambuf – streambuf specialized to arrays

**SYNOPSIS**

```
#include <iostream.h>
#include <strstream.h>
class strstreambuf : streambuf {
public:
                                strstreambuf() ;
                                strstreambuf(char*,int,char*);
                                strstreambuf(int);
                                strstreambuf(unsigned char*, int, unsigned char*);
                                strstreambuf(void* (*a)(long), void(*f)(void*));
                                void      freeze(int n=1) ;
                                char*     str();
                                streambuf* setbuf(char*,int)
};
```

**DESCRIPTION**

A `strstreambuf` is a `streambuf` that uses an array of bytes (a string) to hold the sequence of characters. Given the convention that a `char*` should be interpreted as pointing just before the `char` it really points at, the mapping between the abstract get/put pointers and `char*` pointers is direct. Moving the pointers corresponds exactly to incrementing and decrementing the `char*` values.

To accommodate the need for arbitrary length strings `strstreambufs` supports an automatic mode. When a **strstreambuf** is in automatic mode, space for the character sequence is allocated as needed. When the sequence is extended too far, it will be copied to a new array.

Assume

- **ssb** is a `strstreambuf*`.
- **n** is an `int`.
- **ptr** and **pstart** are `char*` or `unsigned char*`.
- **a** is `void* (*)(long)`.
- **f** is `void* (*)(void*)`.

The constructors:

**strstreambuf()**

Constructs an empty buffer in dynamic mode. This means that space will be automatically allocated to accommodate the characters that are put into the buffer (using operators `new` and `delete`). Because this may require copying the original characters, it is recommended that when large strings will be used that **setbuf** be used (as described below) to inform the `strstreambuf`.

**strstreambuf(a,f)**

Constructs an empty buffer in dynamic mode. **a** is used as the allocator function in dynamic mode. If it is null, operator `new` will be used. **f** is used to free (or delete) areas returned by **a**. If it is null operator `delete` is used.

**strstreambuf(n)**

Constructs an empty buffer in dynamic mode. The initial allocation of space will be at least **n** bytes.

**strstreambuf(ptr,n,pstart)**

Constructs a buffer to use the bytes starting at **ptr**. If **n** is positive and the **n** bytes starting at **ptr** are used. If **n** is zero, **ptr** is assumed to point to the beginning of a null terminated strings and the bytes of that string (not including the terminating null character) will constitute the buffer. If **n** is negative the buffer is assumed to continue indefinitely. The get pointer is initialized to **ptr**. The put pointer is initialized to **pstart**. If **pstart** is null then stores will be treated as errors. If **pstart** is non null then the initial sequence (for fetching) consists of the bytes between **ptr** and **pstart**. If **pstart** is null then the initial sequence consists of the entire array.

Member functions:

**ssb->freeze(n)**

Inhibits (**n** nonzero) or permits (**n** zero) automatic deletion of the current array. Deletion normally occurs when more space is needed or when **ssb** is being destroyed. Only space obtained dynamic allocation is ever freed. It is an error (and the effect is undefined) to store characters into a buffer that was in automatic allocation mode and is now frozen. It is possible, however, to thaw (unfreeze) such a buffer and resume storing characters.

**ptr=ssb->str()**

Returns a pointer to the first char of the current array and freezes **ssb**. If **ssb** was constructed with an explicit array **init**, **ptr** will point to that array. If **ssb** is in automatic allocation mode, but nothing has yet been stored, **ptr** may be null. **str** freezes **ssb**.

**0b->setbuf(0,n)**

**ssb** remembers **n** and the next time it does a dynamic mode allocation, it makes sure that at least **n** bytes are allocated.

**SEE ALSO**

sbuf.pub(3C++) strstream(3C++)

**NAME**

stdio – standard buffered input/output package

**SYNOPSIS**

```
#include <stdio.h>
```

```
FILE *stdin;
```

```
FILE *stdout;
```

```
FILE *stderr;
```

**DESCRIPTION**

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros [getc\(3\)](#) and [putc](#) handle characters quickly. The higher level routines [fgets](#), [scanf](#), [fscanf](#), [fread](#), [puts](#), [fputs](#), [printf](#), [fprintf](#), [fwrite](#) all use [getc](#) and [putc](#); they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type **FILE**. [Fopen\(3\)](#) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

```
stdin      standard input file
stdout    standard output file
stderr    standard error file
```

A constant pointer `NULL` designates no stream at all.

An integer constant **EOF** is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file `<stdio.h>` of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following ‘functions’ are implemented as macros: [getc](#), [getchar](#), [putc](#), [putchar](#), [feof](#), [ferror](#), [fileno](#).

**SEE ALSO**

[printf\(3\)](#), [scanf\(3\)](#), [fopen\(3\)](#), [getc\(3\)](#), [fgets\(3\)](#), [fread\(3\)](#), [fseek\(3\)](#), [ungetc\(3\)](#), [popen\(3\)](#), [setbuf\(3\)](#), [ferror\(3\)](#)  
[open\(2\)](#), [read\(2\)](#), [fio\(3\)](#)

**DIAGNOSTICS**

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with [fopen](#), input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

**BUGS**

Buffering of output can prevent output data from being seen until long after it is computed – perhaps never, as when an abort occurs between buffer filling and flushing.

Buffering of input can cause a process to consume more input than it actually uses. This can cause trouble across [exec\(2\)](#) or [system\(3\)](#) calls.

BUffering may delay the receipt of a write error until a subsequent *stdio* writing, seeking, or file-closing call.

**NAME**

stdiobuf – ostream specialized to stdio FILE

**SYNOPSIS**

```
#include <iostream.h>
#include <strstream.h>
#include <stdio.h>
class stdiobuf : streambuf {
    FILE*          stdiobuf(FILE* f);
                  stdiofile();
}
```

**DESCRIPTION**

Operations on a `stdiobuf` are reflected on the underlying `FILE`. A `stdiobuf` is constructed in unbuffered mode, which causes all operations to be immediately reflected in the `FILE`. **seeks** are translated into **fseeks**. **setbuf** has its usual meaning. If it supplies a reserve area buffering will be turned back on.

**SEE ALSO**

filebuf(3C++) istream(3C++) ostream(3C++) sdbuf.pub(3C++)

**NAME**

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok, strdup  
 – string operations

**SYNOPSIS**

```
#include <libc.h>
```

```
char *strcat(s1, s2)
```

```
char *s1, *s2;
```

```
char *strncat(s1, s2, n)
```

```
char *s1, *s2;
```

```
int n;
```

```
int strcmp(s1, s2)
```

```
char *s1, *s2;
```

```
int strncmp(s1, s2, n)
```

```
char *s1, *s2;
```

```
int n;
```

```
char *strcpy(s1, s2)
```

```
char *s1, *s2;
```

```
char *strncpy(s1, s2, n)
```

```
char *s1, *s2;
```

```
int n;
```

```
int strlen(s)
```

```
char *s;
```

```
char *strchr(s, c)
```

```
char *s;
```

```
int c;
```

```
char *strrchr(s, c)
```

```
char *s;
```

```
int c;
```

```
char *strpbrk(s1, s2)
```

```
char *s1, *s2;
```

```
int strspn(s1, s2)
```

```
char *s1, *s2;
```

```
int strcspn(s1, s2)
```

```
char *s1, *s2;
```

```
char *strtok(s1, s2)
```

```
char *s1, *s2;
```

```
char *strdup(s)
```

```
char *s;
```

**DESCRIPTION**

The arguments *s1*, *s2* and *s* point to null-terminated strings. The functions *strcat*, *strncat*, *strcpy*, and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

*Strcat* appends a copy of string *s2* to the end of string *s1*. *Strncat* appends at most *n* characters. Each returns a pointer to the null-terminated result.

*Strcmp* compares its arguments and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*. *Strncmp* makes the same comparison but looks at at most *n* characters.

*Strcpy* copies string *s2* to *s1*, stopping after the null character has been copied. *Strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result will not be null-

terminated if the length of *s2* is *n* or more. Each function returns *s1*.

*Strlen* returns the number of characters in *s*, not including the terminating null character.

*Strchr* (*strrchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or `(char)` if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

*Strpbrk* returns a pointer to the first occurrence in string *s1* of any character from string *s2*, `(char)` if no character from *s2* exists in *s1*.

*Strspn* (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

*Strtok* considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call, with pointer *s1* specified, returns a pointer to the first character of the first token, having replaced the character after the token by 0. Subsequent calls, signified by *s1* being `(char *)0`, will scan from where the preceding call left off. The separator string *s2* may be different from call to call. When no token remains in *s1*, `(char)` is returned.

*Strdup* returns a pointer to a distinct copy of the null-terminated string *s* in space obtained from [malloc\(3\)](#) or `(char)` if no space can be obtained.

## SEE ALSO

[memory\(3\)](#)

## BUGS

*Strcmp* and *strncmp* use native character comparison, which may be signed or unsigned. The outcome of overlapping moves varies among implementations.

**NAME**

strstream – iostream specialized to arrays

**SYNOPSIS**

```
#include <strstream.h>
class ios {
public:
    enum        open_mode { in, out, ate, app };
                // and lots of other stuff...
};
class istrstream : public istream {
public:
                istrstream(char*);
                istrstream(char*, int);
    strstreambuf* rdbuf();
};
class ostrstream : public ostream {
public:
                ostrstream();
                ostrstream(char*, int, open_mode = 0);
    int          pcount();
    char*        str();
    strstreambuf* rdbuf();
};
```

**DESCRIPTION**

strstream specializes iostream for "incore" operations, that is, storing and fetching from arrays of bytes. The streambuf associated with a strstream is a strstreambuf.

Assume

- **iss** is a istrstream.
- **oss** is a ostrstream.
- **cp** is a char\*.
- **mode** is an open\_mode.
- **i** and **len** are int.
- **ssb** is a strstreambuf\*.
- **a** is a void\* (\*)(long).
- **f** is a void (\*)(void\*).

The constructors:

**istrsteam(cp)**

Characters will be fetched from the (null terminated) string **cp**erminating null character will not be part of the sequence. Seeks are allowed within that space.

**istrstream(cp,len)**

Characters will be fetched from the array beginning at **cp** and extending for **len** bytes. Seeks are allowed anywhere within that array.

**ostrstream()**

Space will be dynamically allocated to hold stored characters.

**ostrstream(cp,n,mode)**

Characters will be stored into the array starting at **cp** and continuing for **n** bytes. If `ios::ate` or `ios::append` is set in **mode**, **cp** is assumed to be a null terminated string and storing will begin at the null character. Otherwise storing will begin at **cp**. Seeks are allowed anywhere in the array. Members:

**cp=oss.str()**

Returns a pointer to the array being used. If **oss** was constructed with an explicit array, **cp** is just a pointer to the array. Otherwise, **cp** points to a dynamically allocated area. Until **str** is called, deleting the dynamically allocated area is the responsibility of **ss**. After **str** returns, the array

becomes the responsibility of the user program. Once **str** has been called the effect of storing more characters into **ss** is undefined.

**i=ss.pcount()**

The number of bytes that have been stored into the buffer. This is mainly of use when binary data has been stored and **ss.str()** does not point to a null terminated string.

**SEE ALSO**

strstreambuf(3C++), ostream(3C++)



**NAME**

swab – swap bytes

**SYNOPSIS**

**swab**(*from*, *to*, *nbytes*)  
**char** \**from*, \**to*;

**DESCRIPTION**

*Swab* copies *nbytes* bytes pointed to by *from* to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between machines with different byte orders. *Nbytes* should be even.

**NAME**

system – issue a shell command

**SYNOPSIS**

**system(string)**  
**char \*string;**

**DESCRIPTION**

*System* causes the command [CB]sh -c *string* to be executed. The current process waits until the shell has completed, then returns the exit status of the shell.

**SEE ALSO**

[popen\(3\)](#), [exec\(2\)](#), [fork\(2\)](#)

**DIAGNOSTICS**

Exit status 127 indicates the shell couldn't be executed.

**NAME**

tcp\_sock, tcp\_connect, tcp\_listen, tcp\_accept, tcp\_rcmd – tcp networking functions

**SYNOPSIS**

```
#include <sys/inet/tcp_user.h>

int tcp_sock();

int tcp_connect(fd, tp)
int fd;
struct tcpuser *tp;

int tcp_listen(fd, tp)
int fd;
struct tcpuser *tp;

int tcp_accept(fd, tp)
int fd;
struct tcpuser *tp;

int tcp_rcmd(host, port, locuser, remuser, cmd, fd2p)
char *host, *port, *locuser, *remuser, *cmd;
int *fd2p;
```

**DESCRIPTION**

These routines are loaded by the **-lin** option of *ld(1)*.

TCP is a protocol layered upon IP (internet protocol). It provides full-duplex byte stream connections between end points called sockets. The address of a socket is composed of the internet address of its host and the port number to which the socket is bound.

*Tcp\_sock* returns the file descriptor of an unbound socket. Once opened, a socket may be bound to a port number within the host and set up as the active or passive end of a connection.

Addresses and parameters are passed in **tcpuser** structures:

```
struct tcpuser {
    int code;
    tcp_port lport, fport;
    in_addr laddr, faddr;
    int param;
};
```

*Lport* and *laddr* refer to the port and address numbers of the local end of a connection. *Fport* and *faddr* refer to the port and address numbers of the foreign end of a connection.

*Tcp\_connect* binds socket *fd* to port *tp->lport* and attempts to set up a connection to the socket bound to port *tp->fport* on host *tp->faddr*. If *tp->lport* is 0, a local port number is automatically chosen. *Tcp\_connect* returns 0 if the connection is established, -1 otherwise. *tp->lport* and *tp->laddr* are filled in to reflect the local port and address numbers for the connection. Communication proceeds by performing *read(2)* and *write* on *fd*. If *tp->param* is non-zero, it specifies options to set for the connection. The only option supported is **SO\_KEEPALIVE** which causes empty messages to be sent periodically to detect dead connections.

*Tcp\_listen* binds socket *fd* to port *tp->lport* and configures the socket to listen for connection requests to that port. If *tp->faddr* and *tp->fport* are non-zero, only connections coming from sockets on machine *faddr* and bound to port *fport* are listened for. *Tcp\_listen* returns 0 on success, -1 otherwise. *tp->laddr* is filled in to reflect the local address number for the connection. *Select(2)* can be used with a listening socket to provide asynchronous polling of connection requests by selecting for pending input on the socket.

*Tcp\_accept* waits for and accepts a connection request sent to the listening socket *fd*. When a connection arrives, *tcp\_accept* returns a new file descriptor over which communications can proceed. *tp->faddr*, *tp->fport*, *tp->laddr*, and *tp->lport* are filled in to identify the two ends of the connection. *tp->param* is filled in with the minor device number of the tcp device used for the new connection. *Fd* is left open and continues listening for connections.

*Tcp\_rcmd* remotely executes a *cmd* on *host* as user *remuser*. Standard input is attached to *cmd*'s standard input and *cmd*'s standard output is attached to standard output. If *fd2p* is non-zero, it is filled with the file descriptor of a new TCP connection attached to *cmd*'s standard error. Otherwise, *cmd*'s standard error is attached to its standard output.

**FILES**

*/dev/tcp\**  
the socket devices

**SEE ALSO**

[ipc\(3\)](#), [internet\(3\)](#), [udp\(3\)](#)

**DIAGNOSTICS**

*Tcp\_sock* returns -1 if no sockets are available.

**NAME**

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – device-independent terminal screen control

**SYNOPSIS**

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char bp[1024], *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
char *cp;
int (*outc)();
```

**DESCRIPTION**

These functions are loaded by option **-ltermcap** of *ld(1)*. They extract and use capabilities from the terminal capability data base *termcap(5)*. These are low level routines; see *curses(3)* for a higher level package.

*Tgetent* extracts the entry for terminal *name* into the buffer at *bp*. *Bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum*, *tgetflag*, and *tgetstr*. *Tgetent* returns -1 if it cannot open the *termcap* file, if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a **TERMCAP** variable. If found, and the value does not begin with a slash, and the terminal type *name* is the same as the value of the environment variable **TERM**, the **TERMCAP** string is used instead of reading the *termcap* file. If it does begin with a slash, the string is used as a path name rather than

*Tgetnum* gets the numeric value of capability *id*, returning -1 if is not given for the terminal. *Tgetflag* returns 1 if the specified capability is present in the terminal's entry, if it is not. *Tgetstr* gets the string value of capability *id*, placing it in the buffer at *\*area*, advancing the *area* pointer. It decodes the abbreviations for this field described in *termcap(5)*, except for cursor addressing and padding information.

*Tgoto* returns a cursor addressing string decoded from **cm** to go to column *destcol* in line *destline*. It uses the external variables **UP** (from the **up** capability) and **BC** (if **bc** is given rather than **bs**) if necessary to avoid placing **en**, **^D**, or **^@** in the returned string. (Programs which call *tgoto* should be sure to turn off the **XTABS** bit(s), since *tgoto* may now output a tab. Note that programs using *termcap* should in general turn off **XTABS** anyway since some terminals use **[CB]^I for other functions, such as nondestructive space.**) If a **%** sequence is given which is not understood, then *tgoto* returns **`OOPS'**.

*Tputs* decodes the leading padding information of the string *cp*; *affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable; *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as in *tty(4)*. The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a null (**^@**) is inappropriate.

**FILES**

/etc/termcap

*TERMCAP(3X)*

*TERMCAP(3X)*

**SEE ALSO**

*vi(1), curses(3), termcap(5)*

**NAME**

timec, timegm, timelocal – convert ASCII to time

**SYNOPSIS**

```
#include <time.h>
```

```
long timec(string)
```

```
char *string;
```

```
long timegm(timep)
```

```
struct tm *timep;
```

```
long timelocal(timep, zone)
```

```
struct tm *timep;
```

```
char *zone;
```

**DESCRIPTION**

These routines are inverse to [ctime\(3\)](#) and its relatives. See [ctime\(3\)](#) for data layouts.

*Timec* converts to system format a date *string* as produced by *ctime*, [date\(1\)](#), or [ls\(1\)](#). An optional day of the week is ignored. A month name and day are required. A missing hour:min[:sec] field is taken to be **00:00:00**. An optional time zone (local time by default) may appear before or after the year. A missing year is assumed to be the past 12-month interval.

*Timegm* returns the system-format time corresponding to the broken-down GMT time pointed to by *timep*. In a copy of the broken-down time **tm\_mon** is reduced mod 12 by carrying (positively or negatively) to **tm\_year**. Next **tm\_mon** and **tm\_mday** are added to **tm\_yday** appropriately for **tm\_year**. Then **tm\_sec**, **tm\_min**, **tm\_hour**, **tm\_yday**, and **tm\_year** are adjusted by carrying. Finally the system-format date is calculated from these 5 fields.

*Timelocal* is like *timegm*, except that the broken-down time belongs to the specified time *zone*, or is local time if *zone* is zero.

Time zones and months are recognized by the first three characters, regardless of case. *Strings* for *ctime* may contain names longer than three characters and may contain extra white space and commas.

**EXAMPLES**

Set a date ahead one month:

```
struct tm brk_out = *localtime(&date);
brk_out.tm_yday = 0;
brk_out.tm_mon++;
date = timelocal(&brk_out, 0);
```

Convert a [date\(1\)](#) string to system format:

```
date = timec("Sat Sep 27 20:59:11 EDT 1986");
```

**SEE ALSO**

[ctime\(3\)](#), [time\(2\)](#)

**BUGS**

Unknown time zone names are taken to be GMT.

Times before the epoch yield nonsense.

**NAME**

*tolower*, *toupper* – force upper or lower case

**SYNOPSIS**

***tolower*(character)**

***toupper*(character)**

**DESCRIPTION**

If *character* is an upper case letter, *tolower* returns the same lower case letter, otherwise it returns the original character.

*Toupper* does the reverse.

**SEE ALSO**

[ctype\(3\)](#)



**NAME**

*ttyname*, *isatty*, *nametty* – find or set name of a terminal

**SYNOPSIS**

**char \**ttyname*(*fildes*)**

***isatty*(*fildes*)**

***nametty*(*fildes*, *file*)**

**char \**file*;**

**DESCRIPTION**

*Ttyname* returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fildes*.

*Isatty* returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

*Nametty* arranges that future opens of *file* will refer to the stream opened on *fildes*. *File* must exist before *nametty* is called. The arrangement is terminated when the other end of the stream is closed or hung up.

**FILES**

*/lib/ttydevs*            list of tty directories for *ttyname*

**SEE ALSO**

[fmount\(2\)](#), [ioctl\(2\)](#)

**DIAGNOSTICS**

*Ttyname* returns NULL if *fildes* does not describe an entry in any of the directories listed in

*Nametty* returns 1 for success, 0 for failure (*file* does not exist, *fildes* is not a stream).

**BUGS**

The return value of *ttyname* points to static data whose content is overwritten by each call.

If *fildes* is a network connection, *isatty* may produce answers having more to do with the network than to the file to which network data is copied. In particular, it always returns no for connections set up by *rx*, and always returns yes for connections arranged by *dcon*; see [con\(1\)](#).

**NAME**

udp\_connect, udp\_listen, udp\_datagram – udp networking functions

**SYNOPSIS**

```
#include <sys/inet/udp_user.h>

int udp_connect(sport, dhost, dport)
in_addr dhost;
udp_port sport, dport;

int udp_listen(sport, reply)
udp_port sport;
struct udpreply *reply;

int udp_datagram(sport)
udp_port sport;
```

**DESCRIPTION**

These routines are loaded by the **-lin** option of *ld(1)*.

UDP (universal datagram protocol) is a protocol layered upon IP (internet protocol). It provides datagram service between end points called sockets. A socket address is composed of the internet address of the host and the port number to which the socket is bound.

*Udp\_connect* returns the file descriptor of a UDP socket bound to port *sport*. Each *read(2)* from this file descriptor will only accept datagrams from the UDP socket at host *dhost*, port *dport*; a *write* on this file descriptor will be sent to that socket.

*Udp\_listen* returns the file descriptor of a UDP socket bound to port *sport* and waits for a datagram to be sent to that port. Once a message has been received from another socket, all writes will go to that socket and reads will only accept data from that socket.

*Udp\_datagram* returns the file descriptor of a UDP socket bound to port *sport*. Messages written to the file descriptor must start with a **struct udpaddr** which contains the destination of the message.

```
struct udpaddr {
    in_addr host;
    int     port;
};
```

Messages read from the file descriptor also start with a **struct udpaddr** and contain the address of the source socket.

**FILES**

/dev/udp\*  
the socket devices

**SEE ALSO**

*internet(3)*, *tcp(3)*

**DIAGNOSTICS**

All these routines returns  $-1$  on failure.

**NAME**

uname – get name of current system

**SYNOPSIS**

```
#include <sys/utsname.h>
```

```
int uname (name)
```

```
struct utsname *name;
```

**DESCRIPTION**

*Uname* stores information identifying the current system in the structure pointed to by *name*.

*Uname* uses the structure defined in <sys/utsname.h> whose members are:

```
char    sysname[9];
char    nodename[9];
char    release[9];
char    version[9];
```

*Uname* returns a null-terminated character string naming the current system in the character array *sysname*. Similarly, *nodename* contains the name that the system is known by on a communications network. *Release* and *version* further identify the operating system. Upon successful completion, a non-negative value is returned.

**FILES**

/etc/whoami

**DIAGNOSTICS**

If the routine fails, -1 is returned and *errno* is set to indicate the error.

**NAME**

ungetc – push character back into input stream

**SYNOPSIS**

```
#include <stdio.h>
```

```
ungetc(c, stream)
```

```
FILE *stream;
```

**DESCRIPTION**

*Ungetc* pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns *c*.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push **EOF** are rejected.

**SEE ALSO**

[getc\(3\)](#), [stdio\(3\)](#), [fseek\(3\)](#)

**DIAGNOSTICS**

*Ungetc* returns **EOF** if it can't push a character back.

**NAME**

`valloc` – aligned memory allocator

**SYNOPSIS**

```
char *valloc(size)  
unsigned size;
```

**DESCRIPTION**

*Valloc* allocates *size* bytes aligned on a boundary adequate for *vread(2)*. It is implemented by calling *malloc(3)* with a slightly larger request, saving the true beginning of the block allocated, and returning a properly aligned pointer.

**DIAGNOSTICS**

*Valloc* returns a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block.

**BUGS**

*Vfree* isn't implemented.

**NAME**

varargs – variable argument list

**SYNOPSIS**

```
#include <varargs.h>
function(va_alist)
va_dcl

va_list pvar;

va_start(pvar);

va_arg(pvar, type);

va_end(pvar);
```

**DESCRIPTION**

This set of macros allows portable procedures that accept variable argument lists to be written. Routines which have variable argument lists (such as [printf\(3\)](#)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

The literal identifier *va\_alist* is used in a function header to declare a variable argument list. It is declared by *va\_dcl*. Note that there is no semicolon after *va\_dcl*.

**Va\_list** is the type of the variable *pvar*, which is used to traverse the list. One variable of this type must always be declared.

*Va\_start* initializes *pvar* to the beginning of the list.

*Va\_arg* returns the next argument in the list pointed to by *pvar*. *Type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type is expected, since it cannot be determined at runtime.

*Va\_end* is used to finish up.

Multiple traversals, each bracketed by *va\_start* and *va\_end*, are possible.

**EXAMPLES**

How to define *execl* in terms of *execv*; see [exec\(2\)](#):

```
#include <varargs.h>
execl(va_alist)
va_dcl
{
    va_list ap;
    char *file;
    char *args[100];
    int argno = 0;
    va_start(ap);
    file = va_arg(ap, char*);
    while(args[argno++] = va_arg(ap, char*));
    va_end(ap);
    execv(file, args);
}
```

**NAME**

view2d, moviefil – movie of a function  $f(x, y, t)$

**SYNOPSIS**

**view2d**(*fd*, *nx*, *ny*, *time*, *u*, *v*, *fixuv*, *pmin*, *pmax*, *p*)

**short** *p*[];

**double** *time*;

**moviefil**(*fd*, *nx*, *ny*, *time*, *outside*, *f*)

**float** *time*, *outside*, *f*[];

**DESCRIPTION**

*View2d* writes a frame in the format [view2d\(5\)](#) onto the file specified by file descriptor *fd*. To load these routines, use the [ld\(1\)](#) option **-lview2d**. *Nx*, *ny* give the grid size. *Time* is a (nondecreasing) frame index, typically set to simulation time or iteration count. *U* and *v* describe the relation between pixel values and user function values:

$$p = u + f \times 2^{-v}.$$

*U* and *v* may vary from one frame to the next. When the global scaling is known beforehand, put *fixuv* = 1 and set *pmin* and *pmax* to the limits of the data. (Otherwise put *fixuv*=0; *pmin* and *pmax* will be ignored.)

*P* is the *nx* by *ny* array of pixel values, with the *x* index running fastest. There is a threshold for describing nonrectangular regions: any pixel value less than or equal to  $-32766$  is treated as an out-of-bounds marker and will appear as black. Other pixel values should lie in the range  $-32765..32765$  inclusive.

*Moviefil* is an alternate version that is somewhat less flexible but easier to use. It takes floats and scales automatically to shorts. An element of *f* less than *outside* is treated as undefined and will appear as black.

**SEE ALSO**

[view2d\(1\)](#), [view2d\(5\)](#)

**BUGS**

The array arguments are 2-D Fortran arrays.

**NAME**

jobs – summary of job control facilities

**SYNOPSIS**

```
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/vtimes.h>
#include <wait.h>

int fildes, signo;
short pid, pgrp;
union wait status;
struct vtimes vt;

ioctl(fildes, TIOCSPGRP, &pgrp)
ioctl(fildes, TIOCGPGRP, &pgrp)

setpgrp(pid, pgrp)
getpgrp(pid)
killpg(pgrp, signo)

sigset(signo, action)
sighold(signo)
sigelse(signo)
sigpause(signo)
sigsys(signo, action)

wait3(&status, options, &vt)

cc ... -ljobs
```

**DESCRIPTION**

The facilities described here are used to support the job control implemented in *cs(1)*, and may be used in other programs to provide similar facilities. Because these facilities are not standard in UNIX and because the signal mechanisms are also slightly different, the associated routines are not in the standard C library, but rather in the **-ljobs** library.

For descriptions of the individual routines see the various sections listed in SEE ALSO below. This section attempt only to place these facilities in context, not to explain the semantics of the individual calls.

**Terminal arbitration mechanisms.**

The job control mechanism works by associating with each process a number called a *process group*; related processes (e.g. in a pipeline) are given the same process group. The system assigns a single process group number to each terminal. Processes running on a terminal are given read access to that terminal only if they are in the same process group as that terminal.

Thus a command interpreter may start several jobs running in different process groups and arbitrate access to the terminal by controlling which, if any, of these processes is in the same process group as the terminal. When a process which is not in the process group of the terminal tries to read from the terminal, all members of the process group of the process receive a SIGTTIN signal, which normally then causes them to stop until they are continued with a SIGCONT signal. (See *sigsys(2)* for a description of these signals; *tty(4)* for a description of process groups.)

If a process which is not in the process group of the terminal attempts to change the terminal's mode, the process group of that process is sent a SIGTTOU signal, causing the process group to stop. A similar mechanism is (optionally) available for output, causing processes to block with SIGTTOU when they attempt to write to the terminal while not in its process group; this is controlled by the LTOSTOP bit in the tty mode word, enabled by "stty tostop" and disabled (the default) by "stty -tostop." (The LTOSTOP bit is described in *tty(4)*).

**How the shell manipulates process groups.**

A shell which is interactive first establishes its own process group and a process group for the terminal; this prevents other processes from being inadvertently stopped while the terminal is under its control. The shell then assigns each job it creates a distinct process group. When a job is to be run in the foreground,



the shell gives the terminal to the process group of the job using the TIOCSPGRP ioctl (See [ioctl\(2\)](#) and [tty\(4\)](#)). When a job stops or completes, the shell reclaims the terminal by resetting the terminal's process group to that of the shell using TIOCSPGRP again.

Shells which are running shell scripts or running non-interactively do not manipulate process groups of jobs they create. Instead, they leave the process group of sub-processes and the terminal unchanged. This assures that if any sub-process they create blocks for terminal i/o, the shell and all its sub-processes will be blocked (since they are a single process group). The first interactive parent of the non-interactive shell can then be used to deal with the stoppage.

Processes which are orphans (whose parents have exited), and descendants of these processes are protected by the system from stopping, since there can be no interactive parent. Rather than blocking, reads from the control terminal return end-of-file and writes to the control terminal are permitted (i.e. LTOSTOP has no effect for these processes.) Similarly processes which ignore or hold the SIGTTIN or SIGTTOU signal are not sent these signals when accessing their control terminal; if they are not in the process group of the control terminal reads simply return end-of-file. Output and mode setting are also allowed.

Before a shell *suspends* itself, it places itself back in the process group in which it was created, and then sends this original group a stopping signal, stopping the shell and any other intermediate processes back to an interactive parent. The shell also restores the process group of the terminal when it finishes, as the process which then resumes would not necessarily be in control of the terminal otherwise.

#### Naive processes.

A process which does not alter the state of the terminal, and which does no job control can invoke sub-processes normally without worry. If such a process issues a [system\(3\)](#) call and this command is then stopped, both of the processes will stop together. Thus simple processes need not worry about job control, even if they have “shell escapes” or invoke other processes.

#### Processes which modify the terminal state.

When first setting the terminal into an unusual mode, the process should check, with the stopping signals held, that it is in the foreground. It should then change the state of the terminal, and set the catches for SIGTTIN, SIGTTOU and SIGTSTP. The following is a sample of the code that will be needed, assuming that unit 2 is known to be a terminal.

```

short  tpgrp;
...
retry:
sigset(SIGTSTP, SIG_HOLD);
sigset(SIGTTIN, SIG_HOLD);
sigset(SIGTTOU, SIG_HOLD);
if (ioctl(2, TIOCGPGRP, &tpgrp) != 0)
    goto nottty;
if (tpgrp != getpgrp(0)) { /* not in foreground */
    sigset(SIGTTOU, SIG_DFL);
    kill(0, SIGTTOU);
    /* job stops here waiting for SIGCONT */
    goto retry;
}
...save old terminal modes and set new modes...
sigset(SIGTTIN, onstop);
sigset(SIGTTOU, onstop);
sigset(SIGTSTP, onstop);

```

It is necessary to ignore SIGTSTP in this code because otherwise our process could be moved from the foreground to the background in the middle of checking if it is in the foreground. The process holds all the stopping signals in this critical section so no other process in our process group can mess us up by blocking us on one of these signals in the middle of our check. (This code assumes that the command interpreter will not move a process from foreground to background without stopping it; if it did we would have no way of making the check correctly.)

The routine which handles the signal should clear the catch for the stop signal and *kill(2)* the processes in its process group with the same signal. The statement after this *kill* will be executed when the process is later continued with SIGCONT.

Thus the code for the catch routine might look like:

```

...
sigset(SIGTSTP, onstop);
sigset(SIGTTIN, onstop);
sigset(SIGTTOU, onstop);
...
onstop(signo)
int signo;
{
    ... restore old terminal state ...
    sigset(signo, SIG_DFL);
    kill(0, signo);
    /* stop here until continued */
    sigset(signo, onstop);
    ... restore our special terminal state ...
}

```

This routine can also be used to simulate a stop signal.

If a process does not need to save and restore state when it is stopped, but wishes to be notified when it is continued after a stop it can catch the SIGCONT signal; the SIGCONT handler will be run when the process is continued.

Processes which lock data bases such as the password file should ignore SIGTTIN, SIGTTOU, and SIGTSTP signals while the data bases are being manipulated. While a process is ignoring SIGTTIN signals, reads which would normally have hung will return end-of-file; writes which would normally have caused SIGTTOU signals are instead permitted while SIGTTOU is ignored.

### Interrupt-level process handling.

Using the mechanisms of *sigset(3)* it is possible to handle process state changes as they occur by providing an interrupt-handling routine for the SIGCHLD signal which occurs whenever the status of a child process changes. A signal handler for this signal is established by:

```
sigset(SIGCHLD, onchild);
```

The shell or other process would then await a change in child status with code of the form:

**recheck:**

```

sighold(SIGCHLD);           /* start critical section */
if (no children to process) {
    sigpause(SIGCHLD);      /* release SIGCHLD and pause */
    goto recheck;
}
sigrelse(SIGCHLD);         /* end critical region */
/* now have a child to process */

```

Here we are using *sighold* to temporarily block the SIGCHLD signal during the checking of the data structures telling us whether we have a child to process. If we didn't block the signal we would have a race condition since the signal might corrupt our decision by arriving shortly after we had finished checking the condition but before we paused.

If we need to wait for something to happen, we call *sigpause* which automatically releases the hold on the SIGCHLD signal and waits for a signal to occur by starting a *pause(2)*. Otherwise we simply release the SIGCHLD signal and process the child. *Sigpause* is similar to the PDP-11 *wait* instruction, which returns the priority of the processor to the base level and idles waiting for an interrupt.

It is important to note that the long-standing bug in the signal mechanism which would have lost a SIGCHLD signal which occurred while the signal was blocked has been fixed. This is because *sighold* uses the SIG\_HOLD signal set of *sigsys(2)* to prevent the signal action from being taken without losing

the signal if it occurs. Similarly, a signal action set with *sigset* has the signal held while the action routine is running, much as the interrupt priority of the processor is raised when a device interrupt is taken.

In this interrupt driven style of termination processing it is necessary that the *wait* calls used to retrieve status in the SIGCHLD signal handler not block. This is because a single invocation of the SIGCHLD handler may indicate an arbitrary number of process status changes: signals are not queued. This is similar to the case in a disk driver where several drives on a single controller may report status at once, while there is only one interrupt taken. It is even possible for no children to be ready to report status when the SIGCHLD handler is invoked, if the signal was posted while the SIGCHLD handler was active, and the child was noticed due to a SIGCHLD initially sent for another process. This causes no problem, since the handler will be called whenever there is work to do; the handler just has to collect all information by calling *wait3* until it says no more information is available. Further status changes are guaranteed to be reflected in another SIGCHLD handler call.

#### **Restarting system calls.**

In older versions of UNIX “slow” system calls were interrupted when signals occurred, returning EINTR. The new signal mechanism *sigset(3)* normally restarts such calls rather than interrupting them. To summarize: *pause* and *wait* return error EINTR (as before), *ioctl* and *wait3* restart, and *read* and *write* restart unless some data was read or written in which case they return indicating how much data was read or written. In programs which use the older *signal(2)* mechanisms, all of these calls return EINTR if a signal occurs during the call.

#### **SEE ALSO**

*csh(1)*, *ioctl(2)*, *killpg(2)*, *setpgrp(2)*, *sigsys(2)*, *wait3(2)*, *signal(3)*, *tty(4)*

#### **BUGS**

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of these system calls and even the calls themselves are thus subject to change.

**NAME**

plot: openpl et al. – graphics interface

**SYNOPSIS**

**openpl()**

**erase()**

**label(s)**

**char s[];**

**line(x1, y1, x2, y2)**

**circle(x, y, r)**

**arc(x, y, x0, y0, x1, y1)**

**move(x, y)**

**cont(x, y)**

**point(x, y)**

**linemod(s)**

**char s[];**

**space(x0, y0, x1, y1)**

**closepl()**

**DESCRIPTION**

These subroutines generate graphic output in a relatively device-independent manner. See [plot\(5\)](#) for a description of their effect. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

String arguments to *label* and *linemod* are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following [ld\(1\)](#) options:

**-lplot** device-independent graphics stream on standard output for [plot\(1\)](#) filters

**-l300** GSI 300 terminal

**-l300s** GSI 300S terminal

**-l450** DASI 450 terminal

**-l4014** Tektronix 4014 terminal

**/usr/jerq/lib/libjplot.a**

Blit terminal

**SEE ALSO**

[plot\(5\)](#), [plot\(1\)](#), [graph\(1\)](#)

**NAME**

intro – introduction to devices, line disciplines, and file systems

**DESCRIPTION**

This section describes drivers for devices, stream line disciplines, and file systems.

Devices are accessed through special files of type `S_IFBLK` (block devices) or `S_IFCHR` (character devices); see [stat\(2\)](#). Block devices use a block buffering scheme within the system, so that sectored devices like disks may be accessed a byte at a time. Character devices don't use the block buffers. Only block devices may be mounted as disk file systems. Most block devices have associated 'raw' character devices that bypass all buffering for fast direct I/O.

The device associated with a special file is identified by a pair of numbers: a major device number naming the driver, and a minor device number picking some particular device or subunit. Major numbers are listed in [mknod\(8\)](#). Minor numbers are specific to each driver; see the writeups in this section. Minor numbers are stored in a single unsigned byte; they are chosen from the range 0-255.

Some character devices are also stream devices. These use a different internal buffering mechanism to allow data to flow asynchronously. Various special operations are possible on streams; see [stream\(4\)](#).

Line disciplines are processing modules that may be inserted into streams. They are identified by integers passed to the calls that insert and remove them. The C library contains global variables initialized to the numbers for various line disciplines; [stream\(4\)](#) has a list.

There are several different types of file system: conventional disk volumes, remote file systems accessed by the system sending messages through a stream to a server as described in [netfs\(8\)](#), a file system containing a file representing each process in the system, and so on. All of these appear the same to ordinary processes, except that not all file systems implement all operations; for example, a process file has a name, and may be opened, read, and written like an ordinary file, but may not be renamed because [proc\(4\)](#) doesn't allow that.

File system types are identified by integers, used by and listed in [fmount\(2\)](#). They are just magic numbers at present.

**SEE ALSO**

[fmount\(2\)](#), [stream\(4\)](#), [mknod\(8\)](#)

**NAME**

bk – line discipline for machine-machine communication

**SYNOPSIS**

```
#include <sgtty.h>

int ldisc = NETLDISC, fildes; ...

ioctl(fildes, TIOCSETD, &ldisc);
```

**DESCRIPTION**

This line discipline provides a replacement for the old and new tty drivers described in [tty\(4\)](#) when high speed output to and especially input from another machine is to be transmitted over a asynchronous communications line. The discipline was designed for use by the Berkeley network *net(1)* but is well suited to uploading of data from microprocessors into the system. If you are going to send data over asynchronous communications lines at high speed into the system, you must use this discipline, as the system otherwise may detect high input data rates on terminal lines and disables the lines; in any case the processing of such data when normal terminal mechanisms are involved saturates the system.

A typical application program then reads a sequence of lines from the terminal port, checking header and sequencing information on each line and acknowledging receipt of each line to the sender, who then transmits another line of data. Typically several hundred bytes of data and a smaller amount of control information will be received on each handshake.

The old standard teletype discipline can be restored by doing:

```
ldisc = OTTYDISC; ioctl(fildes, TIOCSETD, &ldisc);
```

While in networked mode, normal teletype output functions take place. Thus, if an 8 bit output data path is desired, it is necessary to prepare the output line by putting it into RAW mode using [ioctl\(2\)](#). This must be done **before** changing the discipline with TIOCSETD, as most [ioctl\(2\)](#) calls are disabled while in network line-discipline mode.

When in network mode, input processing is very limited to reduce overhead. Currently the input path is only 7 bits wide, with newline the only recognized character, terminating an input record. Each input record must be read and acknowledged before the next input is read as the system refuses to accept any new data when there is a record in the buffer. The buffer is limited in length, but the system guarantees to always be willing to accept input resulting in 512 data characters and then the terminating newline.

User level programs should provide sequencing and checksums on the information to guarantee accurate data transfer.

**SEE ALSO**

[tty\(4\)](#)

**BUGS**

A standard program and protocol should be defined for uploading data from microprocessors, so that havoc doesn't result.

A full 8-bit input path should be provided with a mechanism for escaping newlines into an input packet.

**NAME**

buf\_ld – buffering line discipline

**DESCRIPTION**

*Buf\_ld* treasures up data for a while, then emits it in a burst. It is otherwise transparent. It is meant to reduce overhead of programs such as *cu(1)* and *uucp(1)* that read input from moderate-speed lines in raw or cbreak mode.

It saves characters until 16 have arrived, or until 1/20 sec has passed and no more characters have come.

**SEE ALSO**

*stream(4)*, *ttyld(4)*

**NAME**

conn\_ld – line discipline for unique stream connection

**SYNOPSIS**

```
#include <sys/filio.h>
```

**DESCRIPTION**

This line discipline provides unique connections to a server. The server process should push the line discipline on a pipe (see **FIOPUSHLD** in *stream(4)*) and *fmount(2)* the pipe end on a file. A subsequent attempt to *open(2)* or *creat* that file causes a new pipe to be created. A file descriptor for one end of the new pipe is passed on the mounted pipe to to the server process as if by **FIOSNDFD**; see *stream(4)*. The opening process is blocked until the server responds. The server should receive the passed file descriptor with **FIORCVFD** and respond in one of the following ways:

- Accept the new file descriptor by performing

```
ioctl(fd, FIOACCEPT, (void *)0);
```

The originating *open* completes and returns a file descriptor for the other end of the new pipe.

- Write some data on the new file descriptor. This performs an implicit **FIOACCEPT**.

- Pass a different file descriptor:

```
ioctl(fd, FIOSNDFD, &newfd);
```

The originator's end of the new pipe is closed, and a file descriptor for the open file designated by *newfd* is returned to the originating *open*.

- Reject the connection, by closing the new file descriptor or by performing

```
ioctl(fd, FIOREJECT, (void *)0);
```

The originating *open* fails with **ENXIO** and the new pipe is discarded.

**SEE ALSO**

*fmount(2)*, *stream(4)*



**NAME**

console – VAX console interface

**DESCRIPTION**

The console terminal is either in program mode (connected to the program running in the VAX) or in console mode (connected to the console interpreter, which prompts with >>>). On most VAXes, control-**p** switches to console mode. If the VAX CPU is still running, SET TERM PROG returns to program mode. If the CPU is halted, C restarts it and connects to the VAX. Hitting the break key in either mode may halt the console processor and produce an @ prompt; hit P to escape.

On an 11/750, switching to console mode always halts the VAX; only C will escape. Control-**d** while in console mode may induce micro-debugging mode, where the prompt is RDM>. Type RET to return to console mode.

On a MicroVAX, the break key (only) halts the VAX and switches to console mode.

All these modes are implemented by the VAX hardware. To the operating system, the console looks like an ordinary terminal as described in [tty\(4\)](#), except that the speed is fixed in the hardware.

**FILES**

/dev/console

**SEE ALSO**

[tty\(4\)](#), [ttyld\(4\)](#), [reboot\(8\)](#)

**NAME**

dh – DH-11 communications multiplexer

**DESCRIPTION**

Each line attached to the DH-11 communications multiplexer behaves as described in [tty\(4\)](#). Input and output for each line may independently be set to run at any of 16 speeds; see [tty\(4\)](#) for the encoding.

**FILES**

/dev/tty[hi][0-9a-f]

**SEE ALSO**

[tty\(4\)](#)

**NAME**

`dk`, `dkp_ld`, `unixp_ld`, `cmc_ld` – Datakit interface and protocols

**SYNOPSIS**

```
#include <sys/dkio.h>
```

**DESCRIPTION**

These device drivers and line disciplines are used to connect to a Datakit network. Normally the programs in [dkmgr\(8\)](#) do all the work.

Several combinations of hardware and software may be used to connect a system to Datakit:

The *dk* driver works with a DR11-C or DRV11-J connected through an adapter box to a Datakit CPM-422. The host computer does all the protocol work.

The *kdi* driver works with a KMC11-B and one of several line units (KDI, DUBHI, KMS11-P) connected to one of several Datakit or ISN interface boards. The KMC11 runs microcode that handles the URP protocol.

The experimental *kmcdk* driver works with a KMC11-B and a line unit, as above, but the KMC11 runs different microcode implementing a simple DMA engine, and the host does all the protocol work. This is slower, but rather more robust, than the *kdi* setup.

The experimental *cure* driver works with a custom-built microprocessor board connected to a Datakit, ISN, or Hyperkit fiber interface. The host does all the protocol work.

Each minor device number represents a Datakit channel; the device number is the channel number. The *kdi* driver allows only 96 channels per KMC11-line unit pair; devices 96-191 are channels 0-95 on a second pair, if present, and devices 192-255 are channels 0-63 on a third. For the other drivers, there may be only one hardware interface, which may have up to 256 channels.

Usually there is one interface, with files in directory See [dkmgr\(8\)](#) for more about naming conventions.

*Dkp\_ld* is a stream line discipline implementing the URP protocol. The *kdi* driver makes its own URP arrangements; other interfaces need the line discipline. A separate copy of *dkp\_ld* must be pushed on each active channel.

*Cmc\_ld* and *unixp\_ld* are line disciplines set up calls handle and controller handshake messages. *Cmc\_ld* runs a Research-only call setup protocol; *unixp\_ld* runs the standard one. One copy of the appropriate line discipline must be pushed on the common signaling channel to deal with occasional controller keep-alive and maintenance messages. Other copies of the line discipline come and go as calls are placed.

These *ioctl* calls are provided by the device drivers:

**DIOCNXCL** Allow this channel to be opened many times. By default, if a channel is open, it may not be opened again. The default is restored whenever the channel is completely closed.

**KIOCSHUT** Reset the *kdi* driver, hanging up all channels.

These *ioctl* calls are provided by the URP processors, *dkp\_ld* and *kdi*:

**DIOCSTREAM**

Don't generate a stream delimiter when this channel receives a BOT trailer.

**DIOCRECORD**

Insert a stream delimiter after receiving BOT; the default.

**DIOCSCTL** The third argument points to a byte; send that as a Datakit control envelope.

**DIOCRCTL** The third argument points to a byte; copy the most recently received non-URP control envelope there. Zero means no control has been received since the last call.

**DIOCXWIN** Set transmit window size. The third argument points to an array of two long integers. The first number is the maximum size of each URP block; the second is the number of blocks that may be outstanding. Blocks may be no more than 4096 bytes, and the protocol allows no more than eight blocks in a window. A **KIOCINIT** call should follow immediately, or things may go awry.

**KIOCISURP** Return success if some URP processor is active on this channel.

**KIOCINIT** Initialize URP.

These *ioctl* calls are provided by the call setup line disciplines:

**DIOCLHN** This is the common signalling channel.  
**DIOCHUP** Tell the controller to initialize, hanging up all channels.  
**DIOCSTOP** Temporarily hold back received data, so it won't be lost in call setup messages.  
**DIOCSTART** Release data held by **DIOCSTOP**.  
**DIOCCHAN** The third argument points to an integer; fill it in with the number of a free channel for calling out. This is a hint, not a promise; the channel may already be taken by the time it is opened. The caller should be prepared to try again.

**FILES**

/dev/dk

**SEE ALSO**

[dkmgr\(8\)](#), [kmc\(8\)](#)

A. G. Fraser and W. T. Marshall, 'Data Transport in a Byte Stream Network', *IEEE J-SAC*, (September, 1989)

*Datakit VSC Internal Interface Specification*, select code 700-283, AT&T Customer Information Center, Indianapolis

**BUGS**

*Dkp\_ld* and *kdi* insist on using exactly three blocks in a window, no matter what they are told in **DIOCXWIN**.

The *kdi* driver has only two block sizes, 28 and 252 bytes.

**NAME**

drum – paging device

**DESCRIPTION**

This file is the paging and swapping device. It usually refers to an indirect driver that allows swapping to be spread over several disk drives.

**FILES**

/dev/drum

**BUGS**

The indirect driver divides the swap area into interleaved sections of half a megabyte or so; reads may not span the boundary between sections. Since the system doesn't allocate blocks across sections, this probably doesn't matter.

**NAME**

dz – DZ-11 communications multiplexer

**DESCRIPTION**

Each line attached to the DZ-11 communications multiplexer behaves as described in [ttyld\(4\)](#). Each line may be set to run at any of 16 speeds; see [ttyld\(4\)](#) for the encoding.

**FILES**

/dev/tty#           where # is one or two digits  
/dev/ttyd[0-9a-f]

**SEE ALSO**

[ttyld\(4\)](#)

**NAME**

ethernet – Ethernet interface

**SYNOPSIS**

```
#include <sys/enio.h>
#include <sys/ethernet.h>
```

**DESCRIPTION**

There are drivers for several hardware interfaces to Ethernet. All have the same programming interface.

There are eight software channels for each hardware device. A channel sends and receives packets for a single Ethernet interface; hence eight protocols may be used independently on the same device. If a channel is open, it may not be opened again.

*Read* and *write* deal in Ethernet packets, consisting of a header followed by no less than 46 but no more than 1500 bytes of data. The header, defined in `<sys/ethernet.h>`, is as follows:

```
#define      ETHERALEN    6      /* bytes in an ethernet address */
struct etherpup {
    unsigned char dhost[ETHERALEN];    /* destination address */
    unsigned char shost[ETHERALEN];    /* source address */
    unsigned short type;    /* protocol type */
};
```

The protocol type is in the network's byte order, most significant byte first.

*Read* on a channel returns at most one complete packet. If only part of a packet fits in the *read* buffer, successive reads return the remainder. *Write* should be given a single complete packet; **dhost** and **type** must be filled in. The system supplies **shost**.

There are a few *ioctl* calls, defined in `<sys/enio.h>`:

**ENIOTYPE**

The third argument points to a short integer; use that as the protocol type for this channel.

**ENIOADDR**

The third argument points to a six-character buffer; copy the hardware address of this interface there.

Minor device numbers 0-7 are the eight channels of the first hardware device of a given type; 8-15 are the second device, and so on. File names usually end in two digits, like `/dev/i113` for the fourth channel of the second Interlan device.

**FILES**

```
/dev/i1??
    Interlan NI1010A devices
/dev/qe??
    DEQNA devices
/dev/bna??
    DEBNA devices
```

**SEE ALSO**

[internet\(3\)](#), [ipconfig\(8\)](#)

**BUGS**

The DEQNA driver fills in the protocol type field in transmitted packets; other drivers don't.

**NAME**

fd, stdin, stdout, stderr, tty – file descriptor files

**DESCRIPTION**

These files, conventionally called `/dev/fd/0`, `/dev/fd/1`, ... `/dev/fd/127`, refer to files accessible through file descriptors. If file descriptor *n* is open, these two system calls have the same effect:

```
fd = open( "/dev/fd/n", mode );
fd = dup( n );
```

On these devices *creat* (see [open\(2\)](#)) is equivalent to *open*, and *mode* is ignored. As with *dup*, subsequent reads or writes on *fd* fail unless the original file descriptor allows the operations.

**FILES**

```
/dev/fd/*
/dev/stdin
    linked to /dev/fd/0
/dev/stdout
    linked to /dev/fd/1
/dev/stderr
    linked to /dev/fd/2
/dev/tty
    linked to /dev/fd/3
```

**SEE ALSO**

[open\(2\)](#), [dup\(2\)](#)

**DIAGNOSTICS**

*Open* returns `-1` and **EBADF** if the related file descriptor is not open and in the appropriate mode (reading or writing).



**NAME**

fl – floppy interface

**DESCRIPTION**

This is a simple interface to the D.E.C. RX01 floppy disk unit, which is part of the console LSI-11 subsystem for VAX-11/780's. Access is given to the entire floppy consisting of 77 tracks of 26 sectors of 128 bytes.

All i/o is raw; the seek addresses in raw transfers should be a multiple of 128 bytes and a multiple of 128 bytes should be transferred. as in other “raw” disk interfaces.

**FILES**

/dev/floppy

**SEE ALSO**

arff(8)

**BUGS**

Multiple console floppies are not supported.

If a write is given with a count not a multiple of 128 bytes then the trailing portion of the last sector will be zeroed.

**NAME**

hp – RP06, RM03, RM-05 moving-head disk

**DESCRIPTION**

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc.

The origin and size of the pseudo-disks on each drive are as follows:

## RP03 partitions

disk	start	length
0	0	15884
1	15884	33440
2	40964	8360
3	0	0
4	0	0
5	0	0
6	49324	291346
7	0	0

## RM03 partitions

disk	start	length
0	0	15884
1	16000	33440
2	0	0
3	0	0
4	0	0
5	0	0
6	49600	82080
7	0	0

## RM05 partitions

disk	start	length
0	0	15884
1	16416	33440
2	0	500992
3	341696	15884
4	358112	55936
5	414048	36944
6	341696	159296
7	49856	291346

It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. Ordinarily devices 0 and 6 on rp06 and rm03 drives, and 0, 7, and either 6 or 5, 6, and 7 on rm05 drives. Note that the file system sizes are chosen to allow the partitions to be copied between the rp06's and rm05's. This is done so that systems with mixed drives will be able to rearrange file systems easily (see also [up\(4\)](#)). Device 2 is the entire pack, and is used in pack-to-pack copying.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

**FILES**

/dev/rp[0-3][a-h]	block files
/dev/rpp[0-3][a-h]	raw files

**SEE ALSO**

rp(4)

**BUGS**

In raw I/O *read* and *write(2)* truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read*, *write* and *lseek(2)* should always deal in 512-byte multiples.

**NAME**

ht – RH-11/TE-16 magtape interface

**DESCRIPTION**

The files *mt0*, ..., *mt15* refer to the DEC RH/TM/TE-16 magtape. The files *mt0*, ..., *mt7* are 800 bpi, and *mt8*, ..., *mt15* are 1600bpi. The files *mt0*, ..., *mt3* and *mt8*, ..., *mt11* are rewound when closed; the others are not. When a file open for writing is closed, a double end-of-file is written.

A standard tape consists of a series of 1024 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The *mt* files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files may be named *rmt0*, ..., *rmt15*, but the same minor-device considerations as for the regular files still apply.

Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. A zero count is returned when a tape mark is read; another read will fetch the first record of the next tape file.

**FILES**

/dev/mt\*, /dev/rmt\*

**SEE ALSO**

tp(1)

**BUGS**

The magtape system is supposed to be able to take 64 drives. Such addressing has never been tried. These bugs will be fixed when we get more experience with this device.

The driver is limited to four transports.

If any non-data error is encountered, it refuses to do anything more until closed. In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark explicitly.

**NAME**

`ip`, `ip_ld` – DARPA internet protocol

**SYNOPSIS**

```
#include <sys/inio.h>
#include <sys/inet/in.h>
#include <sys/inet/ip_var.h>
```

**DESCRIPTION**

The `ip_ld` line discipline and the `/dev/ip*` files together implement the DARPA IP datagram protocol. They are used by the programs described in [ipconfig\(8\)](#) and [route\(8\)](#).

Each Ethernet device, Datakit channel, or other stream that is to send and receive IP packets must be registered as an ‘IP interface’ by pushing `ip_ld` and setting local and remote addresses with `ioctl` calls. Thereafter, data received from the network are assumed to be IP packets, and are intercepted by the line discipline. Packets destined for the local address of an active IP interface are routed for reading on one of the `ip` device files. Other packets are routed to the IP interface with a matching remote address and retransmitted.

Data written on `ip` devices are taken to be IP packets, are handed to the IP interface with a matching address, and are sent. Packets destined for unreachable places are quietly dropped.

A packet consists of a single stream record, followed by a delimiter: at most one packet is returned by a `read` call; an entire packet must be presented in a single `write`. A packet includes the IP header. Numbers in the header are in host byte order.

Different `ip` devices handle different protocols atop IP. The minor device is the protocol number in the IP header; e.g. 6 for TCP or 17 for UDP. While an `ip` device is open, it may not be opened again. IP packets are often processed by pushing a line discipline such as `tcp_ld` on an `ip` device, rather than by explicit `read` and `write` calls; see [tcp\(4\)](#).

The following [ioctl\(2\)](#) calls, defined in `<sys/inio.h>`, apply to an IP interface. **IPIOLocal** and either **IPIOHOST** or **IPIONET** must be called on each interface before packets will be routed correctly. Type **in\_addr**, defined in `<sys/inet/in.h>`, is a 32-bit integer representing an IP address in host byte order.

**IPIOLocal**

The third argument points to an **in\_addr**: the local IP address for this interface.

**IPIOHOST**

The third argument points to an **in\_addr**: the remote IP address of the single host reachable through this interface.

**IPIONET**

The third argument points to an **in\_addr**: the remote IP address of the network of many hosts reachable through this interface. IP addresses are matched to the network address by applying an internal bit-mask: any IP address for which  $(address \& mask) == net\_address$  is part of the network. The default mask depends on the IP address class; see the IP protocol standard for details.

**IPIOMASK**

The third argument points to an **in\_addr** containing a new network mask for this interface.

**IPIOMTU**

The third argument points to an integer number of bytes. IP packets larger than this size (1500 by default) will be split into smaller ones before being sent through this interface.

**IPIOARP**

The network device for this interface is an Ethernet. Discard the Ethernet header from each incoming packet. When sending a packet, prefix an Ethernet header containing protocol type **0x8** and a destination address obtained by looking up the IP destination address in a table. If the IP address is not in the table, discard the packet, and make an **in\_addr** containing the offending address available for reading on this file descriptor (the one on which `ip_ld` was pushed).

**IPIORESOLVE**

The third argument points to a structure:

```
struct {
    in_addr inaddr;
    unsigned char enaddr[6];
```

```
};
```

Add an entry to the table consulted after **IPIOARP**, mapping IP address **inaddr** to Ethernet address **enaddr**.

The following *ioctl* calls, define in `<sys/inio.h>`, apply to the entire IP subsystem; they may be used on any file with *ip\_ld* pushed.

### IPIOROUTE

The third argument points to a structure:

```
struct route {
    in_addr dst;
    in_addr gate;
};
```

Arrange that henceforth, any IP packet destined for address **dst** will be routed as if destined for **gate**.

### IPIOGETIFS

The third argument points to a union as follows. The structure is defined in `<sys/inet/ip_var.h>`.

```
union {
    int index;
    struct ipif {
        struct queue *queue;
        int flags;
        int mtu;
        in_addr thishost;
        in_addr that;
        in_addr mask;
        in_addr broadcast;
        int ipackets, ierrors;
        int opackets, oerrors;
        int arp;
        int dev;
    } ipif;
};
```

Before the call, **index** should contain an integer naming an entry in the system's table of active interfaces. Interfaces are numbered in a continuous sequence starting at 0. Out-of-range numbers return an error. After the call, **ipif** is filled in with various numbers about that interface.

### FILES

`/dev/ip*`

### SEE ALSO

[ioctl\(2\)](#), [internet\(3\)](#), [ipconfig\(8\)](#), [route\(8\)](#)

DARPA standards RFC 791, RFC 1122

### BUGS

The ARP mechanism should be generalized to deal with networks other than Ethernet. There is only one ARP table for the entire system; there should be one for each interface.

The structures used by **IPIOROUTE** and **IPIORESOLVE** should appear in a header file somewhere.

**NAME**

kl – KL-11 or DL-11 asynchronous interface

**DESCRIPTION**

The discussion of terminal I/O given in [tty\(4\)](#) applies to these devices.

Since they run at a constant speed, attempts to change the speed via [ioctl\(2\)](#) are ignored.

**FILES**

/dev/ttyk[0-9a-f]

**SEE ALSO**

[tty\(4\)](#)

**BUGS**

Modem control for the DL-11E is not implemented.

**NAME**

lp – line printer

**DESCRIPTION**

*Lp* provides the interface to any of the standard DEC line printers. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

{	(
}	)
`	ˆ
	±

^ (.)PP The driver correctly interprets carriage returns, backspaces, tabs, and form feeds. A sequence of newlines which extends over the end of a page is turned into a form feed. Lines longer than 80 characters are truncated (This is a parameter in the driver). Another parameter allows indenting all printout if it is unpleasantly near the left margin.

**FILES**

/dev/lp

**SEE ALSO**

lpr(1)

**BUGS**

Half-ASCII mode, the indent and the maximum line length should be settable by an *ioctl(2)*.



**NAME**

*mem*, *kmem*, *kUmem*, *mtp*, *finclock* – memory and VAX processor registers

**DESCRIPTION**

*Mem* is a file that is an image of the main memory of the computer. It may be used to examine (and even to patch) the system. Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations return errors.

*Kmem* and *kUmem* are like *mem*, but access kernel-mode virtual memory. *KUmem* promises that reads and writes will be done in two-byte quantities; this is convenient for UNIBUS accesses.

*Mtp* accesses VAX internal processor registers. Each register is 4 bytes long; register *n* may be read or written at address *n\*4*.

*Finclock* reads a high-resolution clock. Reading four bytes returns a 32-bit unsigned integer representing the number of microseconds since the epoch 00:00:00 GMT, Jan. 1, 1970, with high-order bits discarded.

*Mem*, *kmem*, *kUmem*, and *mtp* have minor device numbers 0, 1, 3, and 5. *Finclock* is a separate driver; the minor device number is ignored.

**FILES**

*/dev/mem*  
*/dev/kmem*  
*/dev/kUmem*  
*/dev/mtp*  
*/dev/finclock*

**SEE ALSO**

[time\(2\)](#)  
VAX Hardware Handbook

**BUGS**

Examining and patching device and processor registers may give unexpected results when read-only or write-only bits are present.

An attempt to read a nonexistent processor register returns 0 instead of an error.

The *finclock* counter overflows every hour or so. It is only as precise as the hardware; hence it is inaccurate on a MicroVAX.

**NAME**

`mesg_ld`, `rmesg_ld` – message line discipline modules

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/stream.h>
```

**DESCRIPTION**

`Mesg_ld` and `rmesg_ld` turn internal stream messages into ordinary data and vice versa. They supply a way to splice a stream connection together through a process or across a network connection. The messages represent ordinary data and various control operations.

After `mesg_ld` has been pushed on a stream, `read(2)` on the stream returns encoded messages; `write` is expected to supply messages in the same coding. An encoded message consists of a header followed by zero or more bytes of associated data. The header, as defined in `<sys/stream.h>`, is of the form

```
struct mesg {
    char type;
    unsigned char magic;
    unsigned char losize, hisize;
};
#define MSGMAGIC 0345
#define MSGHLEN 4 /* true length of struct mesg in bytes */
```

The header is **MSGHLEN** bytes long; beware that this is not always the same as `sizeof(struct mesg)`. The **magic** field contains the constant **MSGMAGIC**, to help prevent interpreting bad data as a message header. There are **losize+(hisize<<8)** bytes of associated data.

Messages may be written in pieces, or several messages may be written at once. At most one message will be read at a time. If an impossible message is written, the stream may be shut down.

`Rmesg_ld` is exactly the opposite of `mesg_ld`. It is intended for use with devices that generate data containing encoded messages. Here is a list of message types, defined in `<sys/stream.h>`:

**M\_DATA**

(0) Ordinary data.

**M\_BREAK**

(01) A line break on an RS232-style asynchronous connection. No associated data.

**M\_HANGUP**

(02) When received, indicates that the other side has gone away. Thereafter the stream is useless. No associated data.

**M\_DELIM**

(03) A delimiter that introduces a record boundary in the data. No associated data.

**M\_IOCTL**

(06) An `ioctl(2)` request. The associated data is a four-byte integer containing the function code, least significant byte first, followed by some amount of associated data. An **M\_IOCACK** or **M\_IOCNAK** reply is expected.

**M\_DELAY**

(07) A real-time delay. One byte of data, giving the number of clock ticks of delay time.

**M\_CTL**

(010) Device-specific control message.

**M\_SIGNAL**

(0101) Generate signal number given in the one-byte message.

**M\_FLUSH**

(0102) Flush input and output queue if possible.

**M\_STOP**

(0103) Stop transmission immediately.

**M\_START**

(0104) Restart transmission after **M\_STOP**.

**M\_IOCACK**

(0105) Successful reply to **M\_IOCTL**. Associated data is to be written back to the caller.

**M\_IOCNAK**

(0106) Failed reply to **M\_IOCTL**. A single-byte message, if present, will be returned in by the failing *ioctl*.

**M\_PRCTL**

(0107) High-priority device-specific control message.

**SEE ALSO**

[stream\(4\)](#)

**BUGS**

The format of arguments to **M\_IOCTL** is machine dependent.  
The amount of associated data is limited, but large (>4K).

**NAME**

mt – magtape interface

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/mtio.h>
```

**DESCRIPTION**

The *rmt* files refer to magnetic tape drives. Filenames beginning with *rmt* are rewound when closed; those beginning with *nrmt* are not. When a file open for writing is closed, two file marks are written. If the tape is not to be rewound, it is positioned with the head between the two file marks.

Conventionally **rmt0** is 800, **rmt1** is 1600, and **rmt2** is 6250 bpi.

Each *read(2)* or *write* call reads or writes the next record on the tape. *Read* returns at most a single record; the return value is the record size. If the next record is larger than the read buffer, an error is returned. A file mark causes *read* to return 0; the next *read* will return the next record. Seeks are ignored.

An *ioctl(2)* call performs special operations:

**MTIOCTOP**

perform a suboperation encoded as below in the **mt\_op** field of a structure whose address is passed as the third argument of *ioctl*.

```
struct mtop {
    short    mt_op;        /* operation */
    daddr_t  mt_count;    /* repeat count */
};
```

**MTWEOF**

write an end-of-file record

**MTFSF**

forward space file

**MTBSF**

backward space file

**MTFSR**

forward space record

**MTBSR**

backward space record

**MTREW**

rewind

**MTOFFL**

rewind and put the drive offline

The files described above provide a ‘raw’ interface. There is also a ‘block’ interface which attempts to treat the tape like an ordinary file as much as possible. Block tapes are accessed through files with names beginning with *mt* or *nrmt*. Such a tape contains a single file, consisting of a series of 1024-byte records followed by a file mark. Seeks have their usual meaning, and it is possible to read and write a byte at a time, though writing in very small units may create enormous record gaps. The file always ends at the most recently written byte.

Conventions for minor device numbers vary among different hardware drivers:

For the TU78, the drive unit number is encoded in the two low-order bits. Adding 4 prevents the tape from rewinding at close time. Adding 8 selects 6250 bpi for writing; the default is 1600. The tape drive senses density automatically when reading.

The TE16 is like the TU78, except that the default density is 800 bpi, and adding 8 to the device number selects 1600.

For TMSCP tape drives such as the TU81 and the TK50, the unit number is encoded in the three low-order bits. Adding 128 prevents the tape from rewinding on close. Density is selected by octal bits 070; the eight possible values represent eight different device-dependent tape formats. For 9-track tape drives, add 0 for 800 bpi, 8 for 1600, 16 for 6250. For TK50 cartridge drives,

add 24 (old-style block format). For TK70 drives, add 8. The tape drives sense density automatically on reading, but if a drive doesn't support a particular density, the hardware may complain when the device is opened.

**FILES**

/dev/mt?  
/dev/rmt?  
/dev/nmt?  
/dev/nrmt?

**SEE ALSO**

*tape(1)*

**BUGS**

If any non-data error is encountered, the tape drivers generally refuse to do anything more until closed. The naming convention behaves poorly with multiple tape drives. Block tape has probably outlived its usefulness.

**NAME**

newtty – summary of the “new” tty driver

**SYNOPSIS**

**stty new**

**stty new crt**

**DESCRIPTION**

This is a summary of the new tty driver, described completely, with the old terminal driver, in [tty\(4\)](#). The new driver is largely compatible with the old but provides additional functionality for job control.

**CRTs and printing terminals.**

The new terminal driver acts differently on CRTs and on printing terminals. On CRTs at speeds of 1200 baud or greater it normally erases input characters physically with backspace-space-backspace when they are erased logically; at speed under 1200 baud this is often unreasonably slow, so the cursor is normally merely moved to the left. This is the behavior when you say “stty new crt”; to have the tty driver always erase the characters say “stty new crt crterase crtkill”, to have the characters remain even at 1200 baud or greater say “stty new crt –crterase –crtkill”.

On printing terminals the command “stty new prterase” should be given. Logically erased characters are then echoed printed backwards between a ‘\’ and an ‘/’ character.

Other terminal modes are possible, but less commonly used; see [tty\(4\)](#) and [stty\(1\)](#) for details.

**Input editing and output control.**

When preparing input the character # (normally changed to ^H using [stty\(1\)](#)) erases the last input character, ^W the last input word, and the character @ (often changed to ^U) erases the entire current input line. A ^R character causes the pending input to be retyped. Lines are terminated by a return or a newline; a ^D at the beginning of a line generates an end-of-file.

Control characters echo as ^x when typed, for some x; the delete character is represented as ^?.

The character ^V may be typed before *any* character so that it may be entered without its special effect. For backwards compatibility with the old tty driver the character ‘\’ prevents the special meaning of the character and line erase characters, much as ^V does.

Output is suspended when a ^S character is typed and resumed when a ^Q character is type. Output is discarded after a ^O character is typed until another ^O is type, more input arrives, or the condition is cleared by a program (such as the shell just before it prints a prompt.)

**Signals.**

A non-interactive program is interrupted by a ^? (delete); this character is often reset to ^C using [stty\(1\)](#). A quit ^\ character causes programs to terminate like ^? does, but also causes a *core* image file to be created which can then be examined with a debugger. This is often used to stop runaway processes. Interactive programs often catch interrupts and return to their command loop; only the most well debugged programs catch quits.

Programs may be stopped by hitting ^Z, which returns control to the shell. They may then be resumed using the job control mechanisms of the shell, i.e. the *fg* (foreground) command. The character ^Y is like ^Z but takes effect when read rather than when typed; it is much less frequently used.

See [tty\(4\)](#) for a more complete description of the new terminal driver.

**SEE ALSO**

[csh\(1\)](#), [newcsh\(1\)](#), [stty\(1\)](#), [tty\(4\)](#)

**NAME**

null – data sink

**DESCRIPTION**

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

**FILES**

/dev/null

**NAME**

proc – process file system

**SYNOPSIS**

```
#include <sys/types.h> #include <sys/proc.h>
#include <sys/pioctl.h>
```

**DESCRIPTION**

*Proc* is a file system that contains memory images of each running process in the system. The name of each entry in the `/proc` directory is the process id of the subject process, expressed in decimal with optional leading zeros. Each process file is owned by the `userid` of the subject process. The file mode includes read and write permission for the owner if that `userid` has read access to the associated text file; all other permission bits are zero. The file size is the sum of the sizes of virtual memory segments in the subject process.

The subject process is unaffected, except that `setuid` bits will be ignored if it does an *exec(2)*. (Setuid bits are also ignored if the *execing* process has traced signals, or stops on *exec*; see the description of **PIOCS-MASK** and **PIOCSEXEC** below.)

Data may be transferred from or to any locations in the subject's address space through *lseek(2)*, *read(2)*, and *write*. The *text segment* begins at address 0; the *data segment* starts above the text. The *user area* extends downward below address 0x80000000, and is **UPAGES\*NBPG** bytes long (see the header files listed below); the *stack segment* grows downward below the user area. The text, data, and stack sizes may be determined from the process's `proc` structure (see **PIOCGETPR** below). It is an error to access addresses between data and stack. No read or write may span a segment boundary; in the user area only the locations of saved user registers are writable.

*Ioctl(2)* calls control the subject process. The third argument usually points to an integer. The *ioctl* codes are:

**PIOCSTOP**

Send signal **SIGSTOP** to the process, and wait for it to enter the stopped state.

**PIOCWSTOP**

Wait for the process to stop.

**PIOCRUN**

Make the process runnable again after a stop.

**PIOCSMASK**

Define a set of signals to be traced. The process will stop when it receives any signal whose number, as given in *signal(2)*, corresponds to a 1-bit in the given integer, with the least significant bit counted as 1. The traced state and mask bits are inherited by the child of a *fork(2)*. When the process file is closed, the mask becomes zero, but the traced state persists.

**PIOCSEXEC**

Cause the process to stop after *exec(2)*. This condition is inherited across *fork(2)* and persists when the process file is closed.

**PIOCREXEC**

Reverse the effect of **PIOCSEXEC**.

**PIOCCSIG**

Clear the subject's currently pending signal (if any).

**PIOCKILL**

Set the subject's currently pending signal to a given number.

**PIOCOPENT**

Return a read-only file descriptor for the subject process's text file. (Thus a debugger can find the symbol table without knowing the name of the text file.)

**PIOCNICE**

Increment the priority of the subject process by a given amount as if by *nice(2)*.



**PIOCGETPR**

Copy the subject's **proc** structure (see `<sys/proc.h>`) from the kernel process table into an area pointed to the third argument. (This information, which resides in system space, is not accessible via a normal read.)

Any system call is guaranteed to be atomic with respect to the subject process, but nothing prevents more than one process from opening and controlling the same subject.

The following header files are useful in analyzing *proc* files:

**<signal.h>**

list of signal numbers

**<sys/param.h>**

size parameters

**<sys/types.h>**

special system types

**<sys/user.h>**

user structure

**<sys/proc.h>**

proc structure

**<sys/reg.h>**

locations of saved user registers

**<sys/pioctl.h>**

ioctl codes for *proc* files

**FILES**

`/proc/*`

**SEE ALSO**

[adb\(1\)](#), [ps\(1\)](#), [hang\(1\)](#), [fmount\(2\)](#), [signal\(2\)](#), [mount\(8\)](#), [pi\(9\)](#)

**DIAGNOSTICS**

These errors can occur in addition to the errors normally associated with the file system; see [intro\(2\)](#):

**ENOENT**

The subject process has exited.

**EIO** The subject process has attempted I/O at an illegal address.

**EBUSY**

The subject is in the midst of changing virtual memory attributes, or has pages locked for physical I/O.

**ENOSPC**

A write has been attempted on a shared text segment and there is no room on the swap space to make a copy.

**EPERM**

A non-super-user has attempted to better the subject's priority with **PIOCNICE**.

**BUGS**

A process must be swapped in for reading and writing (but not *ioctl*); this may cause a noticeable delay.

The spectrum of states which result in **EBUSY** is too conservative.

A process loaded from a text file with magic number 0407 does not have as a read-only text segment; in this (presumably rare) case **PIOCOPENT** does not work, and the process is accessible even if the text file is read-only.

The interface involves too many VAX-specific magic numbers.

**NAME**

ra – DEC MSCP disks (RA60, RA80, RA81, RA90)

**DESCRIPTION**

*Ra* devices occupy disk drives conforming to DEC's Mass Storage Control Protocol standard: drives such as the RA81 connected via controllers such as the UDA50. Files with minor device numbers 0 through 7 refer to different sections of drive 0, minor devices 8 through 16 refer to drive 1, and so on up to 63 (8 drives).

Normally the disk is accessed in 1024-byte blocks (1K). If 64 is added to the minor device number, 4096-byte blocks (4K) are used instead. A 4K device mounted as a file system is bitmapped; see [filsys\(5\)](#).

Conventionally the files are given names like *ra37* for section 7 of drive 3. There are no name rules distinguishing 1024-byte files from 4096-byte files; in practice the files are almost always the 4096-byte kind.

The start and size of the sections of each drive are as follows. Sizes are measured in 512-byte hardware sectors.

disk	start	length
0	0	10240
1	10240	20480
2	30720	249848
3	280568	249848
4	530416	249848
5	780264	arbitrarily large
6	30720	749544
7	0	arbitrarily large

The 'arbitrarily large' sections reach to the end of the disk. *Rarct* will display disk sizes; see [rarepl\(8\)](#). For example, an RA81 has 891072 sectors, so section 7 is that size, and section 5 is 891072–780264=110808 sectors. An RA90 has 2376153 sectors; section 7 is that size, section 5 is 2376153–780264=1595889 sectors. For other disks, run *rarct* and do the arithmetic.

The *ra* files discussed above access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with *rra* and end with a number which selects the same disk as the corresponding *ra* file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise [lseek\(2\)](#) calls should specify a multiple of 512 bytes.

Several [ioctl\(2\)](#) calls apply to the raw devices.

**UIOCHAR**

The third argument to *ioctl* points to an object to be filled with drive parameters:

```
struct ud_unit {
    daddr_t radsize; /* disk size, sectors */
    daddr_t rctsize; /* RCT size, including pad */
    long    medium; /* medium id */
    short   tracksz; /* sectors per track */
    short   groupsz; /* tracks per group */
    short   cylsz; /* groups per cylinder */
    char    rbns; /* RBNs per track */
    char    copies; /* number of RCT copies */
};
```

**UIORRCT**

The third argument points to an object of type

```
struct ud_rctbuf {
    caddr_t buf;
    int     lbn;
```

```
};
```

**buf** points to a 512-byte buffer, into which block **lbn** of the replacement and caching table (RCT) is read. As many copies of the RCT as necessary are examined to find a readable copy of the block.

### UIOWRCT

The third argument is like that of **UIORRCT**. Block **lbn** of the RCT is written in all copies.

### UIOREPL

The third argument points to an object of type:

```
struct ud_repl {
    daddr_t replbn;    /* good block */
    daddr_t lbn;      /* bad block */
    short  prim;      /* nonzero if primary replacement */
};
```

A 'replace' command is sent to the controller, requesting that attempts to access logical block *lbn* henceforth be revectorred to replacement block *replbn*. *Prim* should be set nonzero if and only if *replbn* is the primary replacement block for *lbn*.

### UIOSPDW

Arrange that the disk drive will spin down when the last file using it is closed.

### UIORST

Reset the controller to which this disk is connected. Any pending operations are abandoned and return an error.

### FILES

```
/dev/ra*
/dev/rra*
```

### SEE ALSO

[rarepl\(8\)](#)  
 MSCP Basic Disk Functions Manual  
 DEC Standard Disk Format Specification

### BUGS

In raw I/O [read\(2\)](#) and [write](#) truncate file offsets to 512-byte block boundaries, and [write](#) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, [read](#), [write](#) and [lseek\(2\)](#) should always deal in 512-byte multiples.

**UIORRCT** and **UIOWRCT** will misbehave if invoked on a section that doesn't start at the beginning of the disk. Section 7 (the whole disk) is the best choice.

The 1K/4K flag bit in the device number is unfortunate.

**NAME**

rk – RK11/RK07 disk driver

**DESCRIPTION**

Files with minor device numbers 0 through 7 refer to various portions of drive 0, minor devices 8 through 16 refer to drive 1, etc.

The range and size of the pseudo-drives for each drive are as follows:

RK07 partitions:

disk	start	length
0	0	15884
1	15906	10032
2	0	53780
3	0	0
4	0	0
5	0	0
6	26004	27786
7	0	0

On a dual RK07 system partition 0 is used for the root for one drive and partition 6 for the /usr file system. If large jobs are to be run, partition 1 on both drives provides a 10Mbyte paging area. Otherwise partition 2 on the other drive is used as a single large file system.

The *rk* files discussed above access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RK files begin with *rrk* and end with a number which selects the same disk as the corresponding *rk* file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *lseek(2)* calls should specify a multiple of 512 bytes.

**FILES**

/dev/rk?  
/dev/rrk? "

**BUGS**

In raw I/O *read* and *write(2)* truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read*, *write* and *lseek(2)* should always deal in 512-byte multiples.

**NAME**

rv – Racal/Vadic ACU interface

**DESCRIPTION**

The racal/vadic ACU interface is provided by the files */dev/cua[01]* which is a multiplexed file, and by the daemon *dnd* which monitors the file, simulating a standard DN dialer. To place an outgoing call one forks a sub-process trying to open */dev/cul?* and then opens the corresponding file */dev/cua?* file and writes a number on it. The daemon translates the call to proper format for the Racal/Vadic interface, and monitors the progress of the call recording accounting information for later use.

The codes for the phone numbers are the same as in the DN interface:

```
0-9    dial 0-9
:      dial *
;      dial #
-      delay for second dial tone
<      end-of-number
```

The entire telephone number must be presented in a single *write* system call.

It is require that an end-of-number code be given.

**FILES**

```
/dev/cua0    virtual dialer for 300 baud dialout
/dev/cua1    virtual dialer for 1200 baud dialout
/dev/cul0    the terminal which is connected to the 300 baud dialout
/dev/cul1    the terminal which is connected to the 1200 baud dialout
/usr/adm/dnacct  Accounting records for sucessfully completed calls.
```

**SEE ALSO**

cu(1), uucp(1)

**BUGS**

Locking problems.

The multiplexor seems to have rare-case bugs which occasionally crash the system taking trap type 9's, usually in the *sdata* system routine.

**NAME**

scsi – SCSI interface

**SYNOPSIS****#include** <scsi.h>**DESCRIPTION**

The special file `/dev/scsi` provides a low level interface to a SCSI bus. Commands are transmitted to the bus by *write*; the response to each command is received with *read(2)*. The format of a command is

```
unsigned long tran_id;    /* transaction id */
unsigned char target;    /* SCSI id of target device */
unsigned char flags;     /* flags for this transaction */
unsigned long nreturn;   /* number of bytes to be read back */
char cmd[10];           /* SCSI command */
char data[];            /* optional data */
```

Thus, to send *n* bytes of data, the byte count for *write* should be *n*+20. Possible flags are **SCSI\_WR** (data goes from host to SCSI), **SCSI\_RD** (data goes from SCSI to host), **SCSI\_BRESET** (reset the SCSI bus), **SCSI\_RESET** (reset the controller), and **SCSI\_SENSE** (return extended sense data on error). For most controllers, **SCSI\_BRESET** implies **SCSI\_RESET**. Flags are OR'ed together, and there must be exactly one of **SCSI\_WR** and **SCSI\_RD**. The SCSI command should terminate within a small time (currently 10 seconds); a longer limit (300 seconds) can be specified by using **SCSI\_LTMOUT**.

The data read is structured as

```
unsigned long tran_id;    /* transaction id */
unsigned char status;     /* scsi status byte */
unsigned char message;    /* scsi message byte */
unsigned char flags;     /* flags for this transaction */
unsigned char c_type;    /* 1=td 2=us */
unsigned short c_reg1;   /* td=sa, us=csr */
unsigned short c_reg2;   /* td=mscp, us=per */
unsigned char sense[22]; /* extended sense data */
char pad[2];
char data[];            /* any data */
```

Thus, to read *n* bytes of data. the byte count to *read* should be *n*+36. If *flags* has the **SCSI\_CERR** bit set, there was a controller error, which is described by the *c\_* fields. The values of **csr** (or **sa**) and **per** (or **mscp**) are documented in the interface manual for the U.S. Design 1158 Unibus controller (or T.D. Systems Viking controller). If the **SCSI\_SENSE** bit was set in the *write*, and the status byte shows a check condition, an attempt is made to get extended sense information. If the attempt succeeds the **SCSI\_SENSE** is set in *flags*. Otherwise, the status and message bytes for the failed attempt are placed in *sense[0]* and *sense[1]* respectively.

The transaction id identifies which *write* caused the results for this *read*. This will become more important when multiple simultaneous transactions are allowed.

**FILES**`/dev/scsi`**SEE ALSO**[\*scsish\(8\)\*](#)

**NAME**

stream – communication channels

**SYNOPSIS**

```
#include <sys/filio.h>
#include <sys/ttyio.h>
```

**DESCRIPTION**

A stream is a connection between two processes, or between a process and a device. It is referred to by a file descriptor, and ordinary read and write calls apply. When a write call is given on a stream whose other end has disappeared, for example because the process at other end of a pipe has terminated, or a device has hung up, signal **SIGPIPE** is generated; if the signal is ignored, the write call returns error **EPIPE**. The first 64 attempts to read such a disconnected stream return 0; subsequent attempts generate **SIGPIPE** signals.

Pipes are streams; so are most communication devices like terminals and networks.

Line disciplines may be inserted into a stream to do various sorts of processing. Line disciplines within a stream are identified by their position; level 0 is nearest the process. Line disciplines on one end of a pipe cannot be seen from the other. Line discipline types are integers; a list is given below.

These *ioctl* calls, defined in `<sys/filio.h>`, manipulate line disciplines:

**FIOPUSHLD**

The third argument points to an integer naming a line discipline; insert that line discipline at level 0.

**FIOINSLD** The third argument points to a structure:

```
struct insld {
    short ld;
    short level;
};
```

Insert the line discipline named by **ld** in the stream at depth **level**. If there are fewer than **level** line disciplines in the stream, an error is returned.

**FIOPOPLD** The third argument points to an integer; remove the line discipline at that level in the stream. A null pointer (the usual case) means level 0.

**FIOLOOKLD**

The third argument points to an integer; return the type of the line discipline at that level, both in the same integer and as the return value from *ioctl*. A null pointer means level 0.

These *ioctl* calls, also in `<sys/filio.h>`, perform other operations on streams:

**FIOSNDFD** The third argument points to an integer naming an open file descriptor. Send a reference to that file to the other end of the stream. This works only on pipes. The reference is unaffected by intervening line disciplines; in particular it cannot be intercepted or forged by *mesgld(4)*. **FIOSNDFD** returns immediately; it does not wait for the reference to be received.

**FIORCVFD** The third argument points to a structure:

```
struct passfd {
    int fd;
    short uid;
    short gid;
    short nice;
    char logname[8];
};
```

Receive a file reference sent by **FIOSNDFD**; fill in the structure with a file descriptor **fd** for the passed file, and the userid **uid**, groupid **gid**, login name **logname**, and scheduling priority **nice** of the sending process. The file reference must be the next message in the stream; if data precedes it, **EIO** is returned. If the stream is empty, **FIORCVFD** blocks until data or a file reference arrives.

**FIONREAD** The third argument points to an integer; fill it in with the number of characters that may be read from this stream without blocking.

These *ioctl* calls also work on streams, but are defined in `<sys/ttyio.h>` for historic reasons:

**TIOCSGRP**

The third argument points to a short integer. If the pointer is null (the usual case), make a new process group for the current process, and associate the group with this stream. If the pointer is not null, it points to a process group id; associate that group with this stream. When the lowest level of a stream receives a signal message (like **SIGINT** or **SIGQUIT** from *tyld(4)*), the signal is sent to processes in the associated process group. If the stream is shut down prematurely, the process group is sent **SIGHUP**.

**TIOCGGRP**

The third argument points to a short integer; fill it in with the process group id associated with this stream. 0 means no group.

**TIOCEXCL** Mark this stream so that future opens are forbidden except to the super-user or to processes in the associated process group.

**TIOCNXCL**

Remove the mark left by **TIOCEXCL**.

**TIOCSBRK** Send a break on a serial line.

**TIOCFLUSH**

Throw away queued data anywhere in the stream.

Here is a list of line discipline types. The names refer to global integers defined in the C library.

<b>tty_ld</b>	Regular terminal processing, <i>tyld(4)</i> .
<b>ntty_ld</b>	Restricted Berkeley ‘new tty’ terminal processing; see the Berkeley Users Manual.
<b>cdkp_ld</b>	Character-mode Datakit universal receiver protocol, <i>dk(4)</i> .
<b>dkp_ld</b>	Block-mode Datakit universal receiver protocol, <i>dk(4)</i> .
<b>rdk_ld</b>	
<b>uxp_ld</b>	Datakit protocols used in call setup, <i>dk(4)</i> .
<b>buf_ld</b>	Buffering mechanism, <i>bufld(4)</i> .
<b>mesg_ld</b>	
<b>rmesg_ld</b>	Turn stream controls into ordinary data and vice versa, <i>mesgld(4)</i> .
<b>conn_ld</b>	Make unique connections to a server, <i>connld(4)</i> .
<b>ip_ld</b>	
<b>tcp_ld</b>	
<b>udp_ld</b>	Internet protocols, <i>internet(3)</i> .

**SEE ALSO**

*ioctl(2)*, *signal(2)*, *mesgld(4)*, *ipc(3)*

D. M. Ritchie, ‘A Stream I/O System’, this manual, Volume 2



## NAME

tbl – kernel table file system

## DESCRIPTION

*Tbl* is a file-system mount point that provides access to kernel data tables. The name of each entry in the `/tbl` directory is the name of a directory containing files that describe a kernel data table. These files have the following names and contents:

**base**    system base address of the table  
**count**   number of elements in the table  
**data**    table contents  
**size**    size of a table element

The standard system-call interface is used to access *tbl*. *Open(2)* and *close(2)* behave as usual. Data may be transferred from or to any locations in the “data” file through *lseek*, *read*, and *write(2)*

The following header files are useful in analyzing *tbl* “data” files:

<sys/file.h>  
    “file” structure  
  
<sys/inode.h>  
    “inode” structure  
  
<sys/lnode.h>  
    “lnode” structure  
  
<sys/param.h>  
    size parameters  
  
<sys/proc.h>  
    “proc” structure  
  
<sys/stream.h>  
    “stream”, “block”, and “queue” structures  
  
<sys/text.h>  
    “text” structure  
  
<sys/types.h>  
    special system types

## FILES

`/tbl/*/base`  
`/tbl/*/count`  
`/tbl/*/data`  
`/tbl/*/size`

## SEE ALSO

[fmount\(2\)](#), [tblmount\(8\)](#),

## BUGS

The *super-user* may write on any file, despite the permissions.

**NAME**

tcp, tcp\_ld – DARPA transmission control protocol

**SYNOPSIS**

```
#include <sys/inio.h>
#include <sys/inet/tcp_user.h>
```

**DESCRIPTION**

The *tcp\_ld* line discipline and the */dev/tcp\** devices together implement the DARPA TCP circuit protocol. They are normally used through *tcpmgr(8)* and the routines in *ipc(3)*.

One instance of *tcp\_ld* should be pushed on an IP device stream, usually see *ip(4)*. Thereafter, data written on the *tcp* devices is turned into IP packets written to the IP device, and vice versa.

Different *tcp* devices represent different software channels. Files with odd minor device numbers are for placing calls; while such a file is open, it may not be opened again. Files with even device numbers receive calls.

To place a call, open an unused odd-numbered *tcp* file; write a **struct tcpuser** describing the address to be called; and read a **struct tcpuser** for status. The structure is defined in *<sys/inet/tcp\_user.h>*:

```
struct tcpuser {
    int code;
    tcp_port lport, fport;
    in_addr laddr, faddr;
    int param;
};

#define TCPC_LISTEN      1
#define TCPC_CONNECT    2
#define TCPC_OK         3
#define TCPC_SORRY      4    /* unknown error */
#define TCPC_BADDEV     5    /* tcp device is bad */
#define TCPC_NOROUTE    6    /* no routing to dest */
#define TCPC_BADLOCAL   7    /* bad local address */
#define TCPC_BOUND      8    /* address already bound */
#define SO_KEEPAALIVE   0x2  /* generate keepalives */
```

In the structure describing the call, **code** should be **TCPC\_CONNECT**; **faddr** and **fport** are the destination IP address and TCP port number; **laddr** is the IP address associated with a local IP interface, or **INADDR\_ANY** to let the system pick; **lport** is the local TCP port number, or to let the system pick; **param** is 0 or **SO\_KEEPAALIVE**.

In the structure returned for status, **code** is **TCPC\_OK** if the call completed correctly; henceforth data written to and read from the file is transported on the circuit. Other codes mean the circuit was not set up.

To listen for incoming calls, open an odd-numbered device and write a **struct tcpuser** with **code** set to **TCPC\_LISTEN**; **laddr** set to the local IP address for which calls should be taken, or **INADDR\_ANY** to catch any calls not explicitly taken by another listener; **lport** set to the port on which to listen, or 0 for any port; and **param** set to 0. Thereafter, reads return successive **tcpuser** structures, each describing a new call; **faddr** and **fport** identify the caller, **laddr** and **lport** the assigned local address. The local *tcp* device number, *n*, assigned to the call is returned in **param**. The corresponding device, */dev/tcpn*, should be opened; data read and written there is transported by the circuit.

Several *ioctl(2)* calls, defined in *<sys/inio.h>*, apply to *tcp* devices:

**TCPIOHUP**      When the remote end of the circuit is disconnected, send signal **SIGHUP** to the local process group associated with the stream.

**TCPMAXSEG**

The third argument points to an integer giving the maximum segment size for this connection: the greatest number of bytes to be packed into one IP packet.

**TCPGETADDR**

The third argument points to a **struct tcpuser**; fill in **laddr**, **lport**, **faddr**, and **fport** with the local and foreign addresses associated with the circuit.

**FILES**

/dev/tcp??

/dev/ip6

**SEE ALSO**

[ip\(4\)](#), [internet\(3\)](#), [tcpmgr\(8\)](#)

DARPA standards RFC 793, 1122

**NAME**

tty – serial line interface drivers

**SYNOPSIS**

```
#include <sys/ttyio.h>
```

**DESCRIPTION**

The files `/dev/tty*` refer to serial line devices such as the DZ11. They are normally used in conjunction with the terminal line discipline, [ttyld\(4\)](#).

Certain device-related parameters, such as parity and line speed, may be set by [ioctl\(2\)](#) calls:

**TIOCGDEV**

The argument points to a **ttydevb** structure to be filled in with current settings.

**TIOCSDEV**

The argument points to a **ttydevb** structure from which the parameters are set.

The **ttydevb** structure, as defined in `<sys/ttyio.h>`, is

```
struct ttydevb {
    char    ispeed; /* input speed */
    char    ospeed; /* output speed */
    short   flags; /* mode flags */
};
```

The speeds are encoded as follows. Impossible speeds are ignored.

B0	0	(hang up device)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

The flags are:

F8BIT	040	eight-bit input and output
ODDP	0100	odd parity
EVENP	0200	even parity

If F8BIT is set, all eight bits of each output character are transmitted without imposing parity, and all eight bits of each input character are passed back without parity checking or stripping. Otherwise, EVENP requests that even parity be accepted and generated, ODDP odd parity. If both EVENP and ODDP are set, or if both are clear, even parity is generated and any parity is accepted.

For DZ11 lines, 1200 baud and 8-bit mode are the defaults. The transmit and receive speeds are the same; **ospeed** is ignored.

**SEE ALSO**

[ioctl\(2\)](#), [ttyld\(4\)](#)

**BUGS**

Every hardware interface doesn't support every operation.

**NAME**

`tty_ld` – terminal processing line discipline

**SYNOPSIS**

```
#include <sys/ttyio.h>
```

**DESCRIPTION**

`Tty_ld` is usually inserted into a stream connected to a terminal device. It gathers input into lines, handles special characters like erase, kill, and interrupt, inserts output delays, and the like. It does not deal with hardware parameters such as speed and parity; see [tty\(4\)](#) for such matters.

Certain special characters have particular meaning on input. These characters are not passed to a program except in raw mode, where they lose their special character. It is possible to change these characters from the default.

The *erase* character (backspace by default) erases the last-typed character. It will not erase beyond the beginning of a line or an end-of-file character.

The *kill* character (default @) erases the entire preceding part of the line, but not beyond an end-of-file character.

The *end-of-file* character (default control-d) causes any characters waiting to be read to be passed immediately to the program, without waiting for newline. The end-of-file character itself is discarded. Thus if the end-of-file character occurs at the beginning of a line, there are no characters waiting, and zero characters will be passed back; this is the standard end-of-file indication.

The *escape* character (\) escapes a following erase, kill, or end-of-file character and allows it to be treated as ordinary data.

The *interrupt* character (default DEL) is not passed to a program but sends signal SIGINT to any processes in the process group of the stream; see [signal\(2\)](#) and [stream\(4\)](#).

The *quit* character (default FS, control-\) sends signal SIGQUIT.

The *stop* character (default DC3, control-s) delays printing on the terminal until something is typed in.

The *start* character (default DC1, control-q) restarts printing after a stop character without generating any input to a program.

Two [ioctl\(2\)](#) calls affect these characters:

**TIOCGETC**

The argument points to a **tchars** structure to be filled in with current settings.

**TIOCSETC**

The argument points to a **tchars** structure from which the characters are set.

The **tchars** structure, as defined in `<sys/ttyio.h>`, is

```
struct tchars {
    char    t_intrc; /* interrupt */
    char    t_quitc; /* quit */
    char    t_startc; /* start output */
    char    t_stopc; /* stop output */
    char    t_eofc; /* end-of-file */
    char    t_brkc; /* input delimiter (like nl) */
};
```

A character value of 0377 eliminates the effect of that character. The `t_brkc` character, by default 0377, acts like a new-line in that it terminates a line, is echoed, and is passed to the program. The stop and start characters may be the same, to produce a toggle effect. It is counterproductive to make other special characters (including erase and kill) identical.

Two [ioctl](#) calls affect other terminal processing parameters:

**TIOCGETP**

The argument points to a **sgttyb** structure to be filled in with the current settings.

**TIOCSETP**

The argument points to a **sgttyb** structure from which the parameters are set.

The **sgttyb** structure, as defined in `<sys/ttyio.h>`, is

```
struct sgttyb {
    char    sg_ispeed; /* unused */
    char    sg_ospeed; /* unused */
    char    sg_erase; /* erase character */
    char    sg_kill;  /* kill character */
    short   sg_flags; /* mode flags */
};
```

The flag bits are

```
ALLDELAY 0177400 Delay algorithm selection
VTDELAY  0040000 Form-feed and vertical-tab delays:
FF0      0
FF1      0040000
CRDELAY  0030000 Carriage-return delays:
CR0      0
CR1      0010000
CR2      0020000
CR3      0030000
TBDELAY  0006000 Tab delays:
TAB0     0
TAB1     0002000
TAB2     0004000
XTABS    0006000
NLDELAY  0001400 New-line delays:
NL0      0
NL1      0000400
NL2      0001000
NL3      0001400
RAW      0000040 Raw mode: wake up on all characters
CRMOD    0000020 Map CR into LF; echo LF or CR as CR-LF
ECHO     0000010 Echo (full duplex)
LCASE    0000004 Map upper case to lower on input
CBREAK   0000002 Return each character as soon as typed
TANDEM   0000001 Automatic flow control
```

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds; type 2 about .16 seconds; type 3 about .32 seconds.

New-line delay type 1 is supposed to be for the Teletype model 37; type 2 is about .10 seconds.

Tab delay type 1 is supposed to be for the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

In **RAW** mode, every character is passed immediately to the program without waiting until a full line has been typed. No erase or kill processing is done; the end-of-file, interrupt, and quit characters are not treated specially. There are no delays and no echoing, and no replacement of one character for another.

CRMOD causes input carriage returns to be turned into new-lines; input of either CR or LF causes CR-LF both to be echoed (for terminals without a new-line function).

CBREAK is a sort of half-cooked mode. Programs read each character as soon as typed, instead of waiting for a full line, but quit and interrupt work, and output delays CRMOD, XTABS, and ECHO work normally. On the other hand there is no erase or kill, and no special treatment of \ or end-of-file.

TANDEM mode causes the system to transmit the stop character whenever the input queue is in danger of

overflowing, and the start character when the input queue has drained sufficiently. It is useful for flow control when the 'terminal' is actually another machine that obeys the conventions.

**SEE ALSO**

*getty(8), stty(1), signal(2), ioctl(2), stream(4), tty(4)*

**BUGS**

The escape character cannot be changed.

**NAME**

up – emulex sc21/ampex 9300 UNIBUS moving head disk

**DESCRIPTION**

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc.

The origin and size of the pseudo-disks on each drive are as follows:

9300 partitions

disk	start	byte
0	0	15884
1	16416	33440
2	0	500992
3	341696	15884
4	358112	55936
5	414048	36944
6	341696	159296
7	49856	291346

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

**FILES**

/dev/up[0-3][a-h]	block files
/dev/rup[0-3][a-h]	raw files

**SEE ALSO**

rp(4)

**BUGS**

In raw I/O *read* and *write(2)* truncate file offsets to 512-byte block boundaries, and *write* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read*, *write* and *lseek(2)* should always deal in 512-byte multiples.



**NAME**

va – Benson-Varian interface

**SYNOPSIS**

```
#include <sys/vcmd.h>
```

**DESCRIPTION**

The Benson-Varian printer/plotter is normally used with the programs *vpr*(1), *vprint*(1) or *vtroff*(1). This description is designed for those who wish to drive the Benson-Varian directly.

The Benson-Varian at Berkeley uses 11" by 8" fan-fold paper. It will print 132 characters per line in print mode and 2112 dots per line in plot mode.

In print mode, the Benson-Varian uses a modified ASCII character set. Most control characters print various non-ASCII graphics such as daggers, sigmas, copyright symbols, etc. Only LF and FF are used as format effectors. LF acts as a newline, advancing to the beginning of the next line, and FF advances to the top of the next page.

In plot mode, the Benson-Varian prints one raster line at a time. An entire raster line of bits (2112 bits = 264 bytes) is sent, and then the Benson-Varian advances to the next raster line.

**Note:** The Benson-Varian must be sent an even number of bytes. If an odd number is sent, the last byte will be lost. Nulls can be used in print mode to pad to an even number of bytes.

To use the Benson-Varian yourself, you must realize that you cannot open the device, */dev/va0* if there is a daemon active. You can see if there is a daemon active by doing a *ps*(1), or by looking in the directory */usr/spool/vad*. If there is a file *lock* there, then there is probably a daemon */usr/lib/vad* running. If not, you should remove the *lock*.

In any case, when your program tries to open the device */dev/va0* you may get one of two errors. The first of these ENXIO indicates that the Benson-Varian is already in use. Your program can then *sleep*(2) and try again in a while, or give up. The second is EIO and indicates that the Benson-Varian is offline.

To set the Benson-Varian into plot mode you can use the following *ioctl*(2) call

```
ioctl(fileno(va), VSETSTATE, plotmd);
```

where **plotmd** is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *va* is the result of a call to *fopen* on *stdio*. When you finish using the Benson-Varian in plot mode you should advance to a new page by sending it a FF after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
```

...

```
fflush(va);
```

```
ioctl(fileno(va), VSETSTATE, prtmd);
```

```
write(fileno(va), "\f0", 2);
```

*N.B.:* If you use the standard I/O library with the Benson-Varian you **must** do

```
setbuf(vp, vbuf);
```

where *vbuf* is declared

```
char vbuf[BUFSIZ];
```

otherwise the standard I/O library, thinking that the Benson-Varian is a terminal (since it is a character special file) will not adequately buffer the data you are sending to the Benson-Varian. This will cause it to run **extremely** slowly and tends to grind the system to a halt.

**FILES**

*/dev/va0*

*/usr/include/sys/vcmd.h*

**SEE ALSO**

*vfont*(5), *vpr*(1), *vtroff*(1), *vp*(4)

**BUGS**

The 1's (one's) and l's (lower-case el's) in the Benson-Varian's standard character set look very similar; caution is advised.

**NAME**

vc – versatec model 122 interface

**SYNOPSIS**

```
#include <sys/vcio.h>
```

**DESCRIPTION**

The files *vc[0-9]* refer to the Versatec model 122 interface. Any plotter obeying the Versatec "green sheet" interface standard can be used with this interface.

Upon opening the device, a RESET\_ALL command is executed by the driver. The driver will only accept a *write* or an *ioctl(2)*; reads will fail and seeks are ignored.

*ioctl(2)* calls perform special operations. There are two basic calls: VGETSTATE and VSETSTATE. VSETSTATE can be used to send special commands to the plotter by setting bits in the control status register (CSR). VGETSTATE returns the CSR of the interface. The functions are specified in *vcio.h*. The bits are as follows:

**VC\_SPP**

is "Simultaneous Print/Plot"

**VC\_PP**

is print/plot mode. Data bytes sent in print mode are printed using the ROM in the plotter. Bytes written in plot mode are plotted. It is possible to use print mode to send commands to the plotter. See the manual for details.

**VC\_SWPBT**

swaps the bytes in the interface. Useful for the VAX.

The remote operations are sent one at a time. *is* is short for *Remote Line Terminate*. It sends a line terminator to the Versatec.

**VC\_CLEAR**

clears the buffer.

**VC\_RESET**

will reset the controller and the plotter.

**VC\_RFFED**

is short for *Remote Form Feed* and will advance the paper.

**VC\_REOTR**

is short for *Remote End of Transmission* which is equivalent to sending an EOF.

**VC\_RESET\_ALL**

is the ultimate reset.

**FILES**

/dev/vc?

**BUGS**

**NAME**

vp – Versatec interface

**SYNOPSIS**

```
#include <sys/vcmd.h>
```

**DESCRIPTION**

The Versatec printer/plotter is normally used with the programs *vpr*(1), *vprint*(1) or *vtroff*(1). This description is designed for those who wish to drive the Versatec directly.

The Versatec at Berkeley is 36" wide, and has 440 characters per line and 7040 dots per line in plot mode (this is actually slightly less than 36" of dots.) The paper used is continuous roll paper, and comes in 500' rolls.

To use the Versatec yourself, you must realize that you cannot open the device, */dev/vp0* if there is a daemon active. You can see if there is a daemon active by doing a *ps*(1), or by looking in the directory */usr/spool/vpd*. If there is a file *lock* there, then there is probably a daemon */usr/lib/vpd* running. If not, you should remove the *lock*.

In any case, when your program tries to open the device */dev/vp0* you may get one of two errors. The first of these ENXIO indicates that the Versatec is already in use. Your program can then *sleep*(2) and try again in a while, or give up. The second is EIO and indicates that the Versatec is offline.

To set the Versatec into plot mode you can use the following *ioctl*(2) call

```
ioctl(fileno(vp), VSETSTATE, plotmd);
```

where **plotmd** is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *vp* is the result of a call to *fopen* on *stdio*. When you finish using the Versatec in plot mode you should eject paper by sending it a EOT after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
```

...

```
fflush(vp);
```

```
ioctl(fileno(vp), VSETSTATE, prtmd);
```

```
write(fileno(vp), "\04", 1);
```

*N.B.:* If you use the standard I/O library with the Versatec you **must** do

```
setbuf(vp, vpbuf);
```

where *vpbuf* is declared

```
char vpbuf[BUFSIZ];
```

otherwise the standard I/O library, thinking that the Versatec is a terminal (since it is a character special file) will not adequately buffer the data you are sending to the Versatec. This will cause it to run **extremely** slowly and tends to grind the system to a halt.

**FILES**

*/dev/vp0*

**SEE ALSO**

*vfont*(5), *vpr*(1), *vtroff*(1), *va*(4)

**BUGS**

**NAME**

80.out assembler and link editor output

**DESCRIPTION**

*80.out* is the output file of the assembler *as80* and the link editor *ld80*. Both programs make *80.out* executable if there were no errors and no unresolved external references.

*80.out* has five sections: header, text, data, relocation information and a symbol table (in that order). The last two sections may be empty if the program was loaded with the “b”, “d” or “t” option of *ld80*.

(Constants beginning with ‘0’ are octal values.)

**HEADER**

The header always contains 040 bytes:

Address Contents

(octal)

0-1	Magic number (0413)
2-3	Size of text segment
4-5	Size of data segment
6-7	Size of bss segment
10-11	Size of symbol table
12-13	Load origin of text segment
14-15	Load origin of data segment
16-20	Load origin of bss segment
20-21	Size of relocation table
22-23	A word of flags
24-37	Padding

The size of each segment is in bytes. The size of the header is not included in any of the other sizes.

The flag values are:

Bit      Meaning

0          If set, no relocation information is present.

The start of the text segment in the file is 040, the start of the data is (040 + text size), the start of the relocation is (040 + text + data size), and the start of the symbol table is (040 + text size + data size + relocation size).

**RELOCATION INFORMATION**

The relocation information (if present) occupies one or two bytes for each byte or word of text or data. The bits of the relocation word (or byte) are:

Bit      Meaning

6-15      Symbol number in symbol table for external references. The first symbol is numbered 0.

5          High-byte flag: If set, the next byte of text or data is to be treated as the high order byte of a 16-bit quantity for relocation purposes.

4          Two-byte flag: If set, the next two bytes of text or data are to be treated as a 16-bit quantity for relocation purposes.

3          External flag: If set, bits 15-6 contain a symbol number, otherwise, only one byte of relocation information is present.

0-2      Segment information:

0          absolute

1          text

- 2 data
- 3 bss

**SEE ALSO**

"as80" (I), "ld80" (I), "nm80" (I)

**SYMBOL TABLE**

The symbol table entries consist of six words:

Word	Meaning
------	---------

The fifth word is a flag indicating the type of the symbol. The following values are possible:

- |     |  |
|-----|--|
| 00  | undefined                                  |
| 01  | absolute                                   |
| 02  | text                                       |
| 03  | data                                       |
| 04  | bss  |
| 05  | file name symbol (produced by ld80)        |
| 010 | undefined external                         |
| 011 | absolute external                          |
| 012 | text segment external                      |
| 013 | data segment external                      |
| 014 | bss segment external                       |
| 6   | The sixth word is the value of the symbol. |

The sixth word of a symbol table entry contains the value of the symbol.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in core when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation bits for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

**NAME**

a.out – object file format

**SYNOPSIS**

```
#include <a.out.h>
```

**DESCRIPTION**

*A.out* is the default name of the output file of the assembler *as(1)* or the link editor *ld(1)*. Both programs make *a.out* executable if there were no errors and no unresolved external references. An object file has five sections: a header, the program text and data, relocation information, a symbol table and a string table (in that order). The last three may be absent; see *strip(1)* and option *-s* of *ld(1)*. The header format, given in *<a.out.h>*, is

```
struct exec {
    long      a_magic;    /* magic number */
    unsigned  a_text;    /* size of text segment */
    unsigned  a_data;    /* size of initialized data */
    unsigned  a_bss;     /* size of uninitialized data */
    unsigned  a_syms;    /* size of symbol table */
    unsigned  a_entry;   /* entry point */
    unsigned  a_trsize;  /* size of text relocation */
    unsigned  a_drsize;  /* size of data relocation */
};
#define OMAGIC 0407      /* old impure format */
#define NMAGIC 0410     /* read-only text */
#define ZMAGIC 0413     /* demand load format */
```

Macros which take *exec* structures as arguments and tell whether the file has a reasonable magic number or return offsets:

```
#define N_BADMAG(x) ((x).a_magic)!=OMAGIC && \
    (x).a_magic!=NMAGIC && (x).a_magic!=ZMAGIC)
#define N_TXTOFF(x) \
    ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x) (N_TXTOFF(x) + (x).a_text+(x).a_data + \
    (x).a_trsize+(x).a_drsize)
#define N_STROFF(x) (N_SYMOFF(x) + (x).a_syms)
```

Sizes are expressed in bytes. The size of the header is not included in any of the other sizes.

When an *a.out* file is executed, a memory image of three segments is set up: the text segment, the data segment, and a stack. The text segment begins at virtual address 0. Following the text segment is the data segment, in which explicitly initialized data come first, then other data, initialized to 0. The stack occupies the highest possible locations in the core image, automatically growing downwards from **0x7fff400** as needed. The data segment may be extended by *brk(2)*.

If the magic number in the header is **OMAGIC**, the text segment is neither write-protected nor shared and the data segment is immediately contiguous with the text segment. This kind of executable program is rarely used. If the magic number is **NMAGIC** or **ZMAGIC**, the data segment is loaded at the first 0 mod 1024 address following the text segment, and the text segment is write-protected and shared among all processes that are executing the same file. **ZMAGIC** format, which *ld(1)* produces by default, supports demand paging: the text and data segments are multiples of 1024 bytes long and begin at byte offsets of 0 mod 1024 in the *a.out* file.

Macros are provided to compute the absolute offset of various parts of the file:

**N\_TXTOFF**

Text segment.

**N\_SYMOFF**

Symbol table.

**N\_STROFF**

String table.

The offsets of the data segment, text relocation information, and data relocation information are obtained by successively adding to **N\_TXTOFF** the size fields **a\_text**, **a\_data**, and **a\_trsize**. The first 4 bytes of the string table contain its size, including the 4 bytes.

The layout of a symbol table entry, as given in `<a.out.h>`, is

```
struct nlist {
    union {
        char      *n_name; /* for use when in-core */
        long      n_strx; /* index into file string table */
    } n_un;
    unsigned char n_type; /* type flag; see below */
    char          n_other;
    short         n_desc; /* see stab(5) */
    unsigned      n_value; /* value of this symbol (or struct offset) */
};
```

Basic values for **n\_type**:

```
#define N_UNDF      0x0    /* undefined */
#define N_ABS      0x2    /* absolute */
#define N_TEXT     0x4    /* text */
#define N_DATA     0x6    /* data */
#define N_BSS      0x8    /* bss */
#define N_COMM     0x12   /* common (internal to ld) */
#define N_FN       0x1f   /* file name symbol */
#define N_EXT      01     /* external bit, or'ed in */
#define N_TYPE     0x1e   /* mask for all the type bits */
```

Other permanent symbol table entries have some **N\_STAB** bits set. These are given in `<stab.h>`:

```
#define N_STAB      0xe0   /* if any of these bits set, keep */
```

The field **n\_un.n\_strx** gives an index into the string table; 0 indicates that no name is associated with the entry. The field **n\_un.n\_name** can be used to refer to the symbol name only if the program sets this up using **n\_strx** and appropriate data from the string table.

A symbol of type undefined external with nonzero value field names a common region; the value specifies its size.

Relocation information occupies eight bytes per relocatable datum:

```
struct relocation_info {
    int          r_address; /* address of datum to be relocated */
    unsigned     r_symbolnum:24, /* local symbol ordinal */
                r_pcrel:1, /* is referenced relative to pc */
                r_length:2, /* 0=byte, 1=word, 2=long */
                r_extern:1, /* symbol value unknown */
                :4; /* nothing, yet */
};
```

If **r\_extern** is 1, the datum designated by **r\_address** and **r\_length** will be relocated by adding to it the value of the associated external symbol. If **r\_extern** is 0, **r\_symbolnum** is encoded in the style of **n\_type** and the value will be relocated by adding the relocated base of the designated area (text, initialized data, or common data).

## SEE ALSO

[adb\(1\)](#), [as\(1\)](#), [ld\(1\)](#), [nm\(1\)](#), [stab\(5\)](#), [strip\(1\)](#)



**NAME**

acct – execution accounting file

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/acct.h>
```

**DESCRIPTION**

*Acct(2)* causes an entry to be appended to an accounting file for each process that terminates. The layout of an accounting file entry, as given in *<sys/acct.h>*, is

```
typedef unsigned short comp_t;
struct acct
{
    char    ac_comm[10]; /* command name */
    comp_t  ac_untime;   /* user time */
    comp_t  ac_stime;    /* system time */
    comp_t  ac_etime;   /* elapsed time */
    time_t  ac_btime;   /* beginning time */
    short   ac_uid;     /* user ID */
    short   ac_gid;     /* group ID */
    short   ac_mem;     /* average memory usage */
    comp_t  ac_io;      /* number of disk IO blocks */
    dev_t   ac_tty;    /* control typewriter */
    char    ac_flag;    /* flag */
};
```

Values in **ac\_flag**:

```
#define AFORK 01 /* has executed fork, but no exec */
#define ASU 02 /* used super-user privileges */
```

If the process does an *exec(2)*, the first 10 characters of the filename appear in *ac\_comm*.

The type **comp\_t** counts 60- or 50-cycle clock ticks in a private floating-point format: a three-bit base-8 exponent and a 13-bit unsigned mantissa. Thus the number of clock ticks that a process ran is expressed by  $(ac\_etime \& 017777) \ll ((ac\_etime \gg 13) \& 03)$ . The beginning time, **ac\_btime**, is recorded in the format of *time(2)*.

**SEE ALSO**

*acct(2)*, *sa(8)*

**NAME**

aliases – aliases file for delivermail

**SYNOPSIS**

**/usr/lib/aliases**

**DESCRIPTION**

This file describes user id aliases that will be used by */etc/delivermail*. It is formatted as a series of lines of the form

name:addr1,addr2,...addrn

The *name* is the name to alias, and the *addr*i are the addresses to send the message to. Lines beginning with white space are continuation lines. Lines beginning with ‘#’ are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag* using the program *newaliases(5)*. A *newaliases* command should be executed each time the aliases file is changed for the change to take effect.

**SEE ALSO**

*newaliases(1)*, *dbm(3)*, *delivermail(8)*

**BUGS**

Because of restrictions in *dbm(3)* a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by “chaining”; i.e. make the last name in the alias by a dummy name which is a continuation alias.

**NAME**

ar – archive (library) file format

**SYNOPSIS**

```
#include <ar.h>
```

**DESCRIPTION**

The archive command *ar(1)* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor *ld*.

A file produced by *ar* has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG    "!<arch>\n"
#define SARMAG   8
#define ARFMAG   "\n"
struct ar_hdr {
    char    ar_name[16];
    char    ar_date[12];
    char    ar_uid[6];
    char    ar_gid[6];
    char    ar_mode[8];
    char    ar_size[10];
    char    ar_fmag[2];
};
#define SAR_HDR 60
```

The name is a blank-padded string. The *ar\_fmag* field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for *ar\_mode*, which is octal. The date is the modification date of the file at the time of its insertion into the archive. The length of the header is SAR\_HDR. Because *struct* may be padded on some machines, SAR\_HDR should be used in preference to *sizeof(struct)* when reading and writing file headers.

Each file begins on an even (0 mod 2) boundary; a newline is inserted between files if necessary. Nevertheless *ar\_size* reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

If an archive contains only printable files, the archive itself is printable.

**SEE ALSO**

*ar(1)*, *ld(1)*, *nm(1)*

**BUGS**

File names lose trailing blanks.

Most software that deals with archives takes an embedded blank as a name terminator.

**NAME**

backup – incremental backup files

**DESCRIPTION**

The backup system consists of a number of client machines, and a ‘backup machine’, which has a database and a collection of backup copies of files. On clients files concerned with backup live in a directory, normally defined in the shell script. The file needed on both client and backup machine, has up to three lines, namely the backup machine name, the default backup device, and the directory, hereafter called **\$FM**, where the rest of the backup software lives on the backup machine. Client systems normally have just the first line. The rest of this description applies to the backup machine.

The databases live in **\$FM/db** and are maintained in *cbr(3)* form. The main database, called *stat*, stores two mappings. The first maps filename-time pairs to backup copy names, thus:

```
/n/bowell/usr/jim/goo//519487622 v/v22/17
```

The number after is the inode change date, expressed in seconds since the epoch; see *stat(2)*. If the backup copy is still on magnetic disk, it will be called **\$FM/v/v22/17**; otherwise it will be **v22/17** on some optical disk. (The mapping of backup copy name to optical disk name is kept in **\$FM/adm/volidmap**.) The second mapping maps filenames to the time of their most recently backed-up version:

```
/n/bowell/usr/jim/goo 520514116
```

The second database, **dir**, maps directoryname-time pairs to the contents of that directory. This allows quick recovery of file trees.

The third database, **fs**, maps filename-time pairs to (essentially) inodes. This allows efficient implementation of **backup mount**; see *backup(1)*.

The program **\$FM/bin/dbupdate** manages these databases. The *dir* and *fs* databases are optional; they will be updated only if they already exist. The program **\$FM/bin/sweep** also assigns the backup copy names into a flat directory structure. A new directory is used when the total size of the files in the current directory would exceed 20000K bytes, rounding each file size up to a multiple of 4K.

The backup copy of a file consists of a header that gives the original inode, pathname and owner (as a string), followed by the contents of the file. Directories are stored as a sequence of entry names.

To prevent multiple writers into a database, a lockfile is used. The content of this file is the process id of the process accessing the database. Locks are removed by **\$FM/bin/rmllocks** executed by *rc(8)* when the system boots.

The backup system supports multiple *filemap* databases (this allows the current database to be kept small). The list of database names is kept in one per line in order of increasing priority. The last name is assumed to be the active database; all the others are read-only.

Programs such as *sweep* and *dbupdate* leave droppings in the log file

Statistics of the numbers of files and bytes saved for users of a given system are kept in Each file consists of a sequence of records with a machine-independent structure; generally, one record per user per day. The records are maintained by which processes the file **\$FM/stat.log** that is maintained by *dbupdate*.

To allow quick searching for filenames with full regular expressions, a simple sorted list of all saved filenames is often kept, normally in

The device (and system) used for recovering files can be specified in many ways. In order of decreasing priority: a **-f** option in *backup recover* or *backup fetch* (see *backup(1)*), a default device on the client system (in line 2 of the default device on the backup system.

**FILES**

```
/usr/lib/backup/*
/usr/lib/backup/conf
/usr/backup/db
/usr/backup/locks
/usr/backup/log
/usr/backup/filenames
```

**SEE ALSO**

*backup(1), worm(8), backup(8), cbt(1), stat(2)*

A. Hume, 'The File Motel: an Owner's Manual', this manual, Volume 2

**NAME**

config – system configuration files

**DESCRIPTION**

These files are used as input by *config(A)*. Except as noted, they are kept in

The file named *files* lists the kernel source files. Each line consists of a filename (relative to followed by some magic words. For example:

```
sys/acct.c      standard
```

is a file used by any version of the system;

```
dev/uba.c      standard device-driver
```

is also always used, and contains device register references (which may require special compilation hacks);

```
dev/ju.c       optional
```

is included only if the *ju* device is expected;

```
dev/ttyld.c   optional
```

is included only if the *tty* pseudo-device is requested.

The file *devices* describes possible device drivers, file system handlers, and line disciplines; the information is used to generate handler dispatch tables. It consists of lines with the following blank-separated fields:

Type of handler:

**device** for character devices

**stream-device**

**block-device**

**file-system**

**line-discipline**

If the type is preceded by the word *standard* (e.g. *standard block-device*), the handler is always included; otherwise, it is included only if requested.

Table index: major device number, file system type, or line discipline number.

Driver name: used in and

*Config* writes a header file *name.h* for each device; if that device is configured, then the upper case *NAME* is defined to be the number of devices of that type.

Entry point name. Used as a prefix for data structure and driver entry points.

Entry points.

For block devices: some of **open**, **close**, **strategy**, **dump**, **B\_TAPE** (the last puts the flag *B\_TAPE* in the *d\_flags* entry in the block device switch).

For character devices: **open**, **close**, **read**, **write**, **ioctl**, **reset**.

For stream devices and line disciplines, **info** should be specified.

For file system handlers: **put**, **get**, **free**, **updat**, **read**, **write**, **trunc**, **stat**, **nami**, **mount**, **ioctl**.

As a special case, lines beginning with **:** are copied intact to This can be used for hacks like

```
: int mem_no = 3; /* major device number of memory special file */
```

Addenda to *files* and *devices* specific to a particular machine may be kept in */usr/sys/machine/files* and */usr/sys/machine/devices*. The addenda are treated as if appended to the general files.

**NAME**

core – format of memory image file

**DESCRIPTION**

A memory image of a terminated process is written into file `core` when any of various errors occur; see [signal\(2\)](#). The memory image is written in the process's working directory, if it can be; normal access controls apply.

A core file consists of a copy of the user block for the terminated process, followed by an image of its data and stack segments. The user block, which occupies 5K bytes, contains descriptive information about the process and a stack used by the operating system when serving the process; see the include file `<sys/user.h>`.

**SEE ALSO**

[adb\(1\)](#), [pi\(9\)](#) [bigcore\(1\)](#), [signal\(2\)](#)

**NAME**

cpio – format of cpio archive

**DESCRIPTION**

The archived files are recorded consecutively, each preceded by a **header**. The header structure, when the `-c` option of [cpio\(1\)](#) is not used, is:

```
typedef unsigned short ushort;
struct {
    short  h_magic,
           h_dev;
    ushort h_ino,
           h_mode,
           h_uid,
           h_gid;
    short  h_nlink,
           h_rdev,
           h_mtime[2],
           h_namesize,
           h_filesize[2];
    char   h_name[h_namesize rounded to word];
} Hdr;
```

When the `-c` option is used, the **header** information is printable, as described by the [printf\(3\)](#) call

```
printf(Chdr, "%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        Hdr.h_magic, Hdr.h_dev, Hdr.h_ino, Hdr.h_mode,
        Hdr.h_uid, Hdr.h_gid, Hdr.h_nlink, Hdr.h_rdev,
        Longtime, Hdr.h_namesize, Longfile, Hdr.h_name
```

Longtime and Longfile are equivalent to `Hdr.h_mtime` and `Hdr.h_filesize`, respectively. Every instance of `h_magic` contains the octal constant 070707. The items `h_dev` through `h_mtime` have meanings explained in [stat\(2\)](#). The length of the null-terminated path name `h_name`, including the null byte, is given by `h_namesize`.

The last element of the archive is a dummy entry for the name **TRAILER!!!**, with padding to a multiple of 512 bytes. Special files, directories, and the trailer are recorded with `h_filesize` equal to zero.

**SEE ALSO**

[cpio\(1\)](#), [find\(1\)](#), [stat\(2\)](#).



**NAME**

dir – format of directories

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/dir.h>
```

**DESCRIPTION**

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry; see [filsys\(5\)](#). The structure of a directory entry as given in the include file is:

```
#define DIRSIZ 14
struct direct
{
    ino_t    d_ino;
    char     d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are `.` for the directory itself and `..` for the parent directory. In the ultimate root directory `..` is the same as `.`

It is inadvisable to read directories using this structure. The routines in [directory\(3\)](#) and [dirread\(2\)](#) are more efficient and portable.

**SEE ALSO**

[filsys\(5\)](#), [directory\(3\)](#), [dirread\(2\)](#)

**NAME**

dump, ddate – incremental dump format

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ino.h>
#include <dumprestor.h>
```

**DESCRIPTION**

Tapes used by *dump* and *restor*(1) contain:

- a header record
- two groups of bit map records
- a group of records describing directories
- a group of records describing files

The format of the header record and of the first record of each description as given in the include file *<dumprestor.h>* is:

```
#define NTREC          10
#define MLEN          16
#define MSIZ          4096
#define TS_TAPE        1
#define TS_INODE       2
#define TS_BITS        3
#define TS_ADDR        4
#define TS_END         5
#define TS_CLRI        6
#define MAGIC          (int) 60011
#define CHECKSUM       (int) 84446
struct  spcl {
    int          c_type;
    time_t      c_date;
    time_t      c_ddate;
    int          c_volume;
    daddr_t     c_tapea;
    ino_t        c_inumber;
    int          c_magic;
    int          c_checksum;
    struct       dinode      c_dinode;
    int          c_count;
    char         c_addr[BSIZE];
} spcl;
struct  idates {
    char         id_name[16];
    char         id_incno;
    time_t      id_ddate;
};
#define  DUMPOUTFMT "%-16s %c %s"          /* for printf */
#define  DUMPINFMT  "% 16s %c %[\n]\n"     /* name, incno, ctime(date) */
#define  DUMPINFMT  "% 16s %c %[\n]\n"     /* inverse for scanf */
```

*NTREC* is the number of 1024 byte records in a physical tape block. *MLEN* is the number of bits in a bit map word. *MSIZ* is the number of bit map words.

The *TS\_* entries are used in the *c\_type* field to indicate what sort of header this is. The types and their meanings are as follows:

```
TS_TAPE      Tape volume label
TS_INODE     A file or directory follows. The c_dinode field is a copy of the disk inode and contains bits telling what sort of file this is.
```

TS_BITS	A bit map follows. This bit map has a one bit for each inode that was dumped.
TS_ADDR	A subrecord of a file description. See <i>c_addr</i> below.
TS_END	End of tape record.
TS_CLRI	A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.
MAGIC	All header records have this number in <i>c_magic</i> .
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

<i>c_type</i>	The type of the header.
<i>c_date</i>	The date the dump was taken.
<i>c_ddate</i>	The date the file system was dumped from.
<i>c_volume</i>	The current volume number of the dump.
<i>c_tapea</i>	The current number of this (1024-byte) record.
<i>c_inumber</i>	The number of the inode being dumped if this is of type <i>TS_INODE</i> .
<i>c_magic</i>	This contains the value <i>MAGIC</i> above, truncated as needed.
<i>c_checksum</i>	This contains whatever value is needed to make the record sum to <i>CHECKSUM</i> .
<i>c_dinode</i>	This is a copy of the inode as it appears on the file system; see <i>filsys(5)</i> .
<i>c_count</i>	The count of characters in <i>c_addr</i> .
<i>c_addr</i>	An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, <i>TS_ADDR</i> records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a *TS\_END* record and then the tapemark.

The structure *idates* describes an entry of the file */etc/ddate* where dump history is kept. The fields of the structure are:

<i>id_name</i>	The dumped filesystem is <i>'/dev/id_nam'</i> .
<i>id_incno</i>	The level number of the dump tape; see <i>dump(1)</i>
<i>id_ddate</i>	The date of the incremental dump in system format see <i>types(5)</i> .

## FILES

*/etc/ddate*

## SEE ALSO

*dump(8)*, *dumpdir(8)*, *restor(8)*, *filsys(5)*, *types(5)*

**NAME**

environ – user environment

**SYNOPSIS**

**extern char \*\*environ;**

**DESCRIPTION**

An array of strings called the ‘environment’ is made available by *exec(2)* when a process begins. By convention these strings have either the form *name=value*, defining a variable, or *name(){value}*, defining a function; see *sh(1)*. The following variables are used by various commands:

**PATH**

The sequence of directory prefixes that *sh*, *time(1)*, *nice(1)*, etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by `:`. *Login(8)* sets `PATH=/bin:/usr/bin`.

**HOME**

A user’s login directory, set by *login(8)* from the password file *passwd(5)*.

**TERM**

The kind of terminal for which output is to be prepared. This information is used by commands, such as *nroff* or *plot(1)*, which may exploit special terminal capabilities. Some customary values of **TERM** are **2621** (HP), **4014** (Tektronix), **5620** (Teletype), and **630** (Teletype). See described in *termcap(5)*, for a longer list.

**SHELL**

The name of the login shell.

The environment may be queried by *getenv(3)* or by the *set* or *whatism* commands of *sh(1)*. Names may be placed in the environment by the *export* command and by *name=value* arguments of *sh(1)*. Names may also be placed in the environment at the point of an *exec(2)*. It is unwise to conflict with certain *sh(1)* variables that are frequently exported by `.profile` files: **MAIL**, **PS1**, **PS2**, **IFS**.

**SEE ALSO**

*sh(1)*, *printenv(1)*, *exec(2)*, *getenv(3)*, *term(6)*

**BUGS**

Function definitions in the environment break some old programs, including old shells.

**NAME**

filsys, flblk, ino – format of a disk file system

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/flblk.h>
#include <sys/filsys.h>
#include <sys/ino.h>
```

**DESCRIPTION**

Every file system is divided into a certain number of blocks of 1K or 4K bytes, as determined by the predicate BITFS( ) applied to the minor device number where the file system is mounted. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the ‘super-block’. Its layout is defined in <sys/filsys.h>:

```
struct filsys {
    unsigned short s_isize;
    daddr_t      s_fsize;
    short        s_ninode;
    ino_t        s_inode[NICINOD];
    char         s_flock;
    char         s_ilock;
    char         s_fmod;
    char         s_ronly;
    time_t       s_time;
    daddr_t      s_tfree;
    ino_t        s_tinode;
    short        s_dinfo[2];
#define s_m      s_dinfo[0]
#define s_n      s_dinfo[1]
#define s_cylsize s_dinfo[0]
#define s_aspace s_dinfo[1]
    char         s_fsmnt[14];
    ino_t        s_lasti;
    ino_t        s_nbehind;
    union {
        struct {
            short    S_nfree;
            daddr_t  S_free[NICFREE];
        } R;
        struct {
            char     S_valid;
#define BITMAP     961
            long     S_bfree[BITMAP];
        } B;
        struct {
            char     S_valid;
            char     S_flag; /* 1 means bitmap not in S_bfree */
            long     S_bsize; /* size of bitmap blocks */
            struct buf *S_blk[BITMAP-1];
        } N;
    } U;
};
#define s_nfree  U.R.S_nfree
#define s_free   U.R.S_free
#define s_valid  U.B.S_valid
#define s_bfree  U.B.S_bfree
```

**s\_ise**

The address of the first block after the i-list, which starts in block 2. Thus the i-list is `s_ise-2` blocks long.

**s\_fsize**

The address of the first block not in the file system.

**s\_inode**

Array of free inode numbers.

**s\_ninode**

The number of free i-numbers in the `s_inode` array. Inodes are placed in the list in LIFO order. If the list underflows, it is replenished by searching the i-list to obtain the numbers of free inodes. When the list is full, freed inodes are not recorded in `s_inode`.

**s\_lasti**

Where the last search for free inodes ended.

**s\_nbehind**

Number of free inodes before `s_lasti` that are not listed in `s_inode`. The system will search forward for free inodes from `s_lasti` for more inodes unless `s_nbehind` is sufficiently large, in which case it will search the i-list from the beginning.

**s\_flock****s\_iloc**

Flags maintained in the core copy of the super-block while the file system while it is mounted. The values on disk are immaterial.

**s\_fmod**

Flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. The value on disk is immaterial.

**s\_ronly**

Flag for read-only file system. The value on disk is immaterial.

**s\_time**

Time of the last change to the super block.

**s\_dinfo**

Disk interleave information: `s_cylsize`= blocks per cylinder, `s_ospace`= blocks to skip; see [fsck\(8\)](#).

**s\_fsmnt**

Unused.

**s\_tfree****s\_tinode**

Numbers of free blocks and free inodes. Maintained for the benefit of `df` (see [du\(1\)](#)), these values are otherwise irrelevant.

Different data are used to manage free space in 1K and 4K file systems. These fields are for 1K file systems:

**s\_free** An array of free block numbers. `s_free[0]` is the block address of the next in a chain of blocks constituting the free list. The layout of these blocks is defined in `<sys/fblk.h>`:

```
struct fblk {
    int      df_nfree;
    daddr_t  df_free[NICFREE];
}
```

where `df_nfree` and `df_free` are exactly like `s_nfree` and `s_free`.

**s\_nfree**

Blocks given in `s_free[1]` through `s_free[s_nfree-1]` are available for allocation. Blocks are allocated in LIFO fashion from this list. If freeing would cause the array to overflow, it is cleared by copying into the newly freed block, which is pushed onto the free chain. If allocation would cause underflow, the array is replenished from the next block on the chain.

These are for 4K file systems:

**s\_bfree**

a bit array specifying the free blocks of a 4K file system. The bit  $(s\_bfree[i/w] \gg (i \% w)) \& 1$ , where  $w$  is the bit size of a long, is nonzero if the  $i$ th data block is free. If the file system is too large for the bitmap to fit here, then it is stored at the end of the file system, and locked into memory when the file system is mounted. The **N** variant of the union is used by the kernel in this case.

**s\_valid**

The bitmap of a mounted file system is maintained only in main memory; the bitmap on the medium is marked invalid by setting `s_valid` to zero. Unmounting updates the medium copy and sets `s_valid` to 1. A file system with invalid bitmap may be mounted read-only; its bitmap can be corrected by [chuck\(8\)](#).

I-numbers begin at 1, and the storage for inodes begins in block 2. Inodes are 64 bytes long. Inode 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each inode represents one file.

The layout of an inode is defined in `<sys/ino.h>`:

```
struct dinode {
    unsigned short di_mode;
    short         di_nlink;
    short         di_uid;
    short         di_gid;
    off_t         di_size;
    char          di_addr[40];
    time_t        di_atime;
    time_t        di_mtime;
    time_t        di_ctime;
};
```

**di\_mode**

The kind of file; it is encoded as `st_mode` [stat\(2\)](#), and is 0 for a free inode.

**di\_nlink**

The number of directory entries (links) that refer to this inode

**di\_uid**

Owner's userid.

**di\_gid**

Owner's groupid.

**di\_size**

Number of bytes in the file.

**di\_atime**

Time of last access; see [times\(2\)](#).

**di\_mtime**

Time of last modification.

**di\_ctime**

Time of last change to inode or contents.

**di\_addr**

For special files the first two bytes of `di_addr` contain the device code; see [intro\(4\)](#) and [types\(5\)](#).

For plain files and directories `di_addr` contains block numbers packed into 3 bytes each. The first 10 numbers specify device blocks directly. The last 3 are singly, doubly, and triply indirect and point to blocks of block pointers of type `daddr_t` (see [types\(5\)](#)). A zero pointer indicates a 'hole' where no data has been written. Holes read as if they contained all zeros.

A symbolic link is, aside from mode, a plain file whose sole content is the name of the file linked to.

**SEE ALSO**

*chuck(8), fsck(8), icheck(8), dir(5), mount(8), stat(2), types(5), l3tol(3)*



**NAME**

font – description files for troff

**DESCRIPTION**

Directories `/usr/lib/font/devdest` describe typesetters, where *dest* is as in the `-T` option of *troff(1)*. Such directories contain files named as in FILES below.

Lines of a typesetter description in file DESC have the following forms.

**res** *n* Resolution of device is *n* basic units per inch.

**hor** *n* Horizontal motion occurs in increments of *n* units.

**vert** *n* Vertical motion occurs in increments of *n* units.

**unitwidth** *n*

Widths are given for pointsize *n*.

**sizescale** *n*

Scaling for fractional pointsizes, not used.

**paperwidth** *n*

Width of paper is *n* units.

**paperlength** *n*

Length of paper is *n* units.

**biggestfont** *n*

Maximum number of characters in a font is *n*.

**sizes** *n n n ... 0*

Pointsizes *n ...* are available.

**fonts** *n name ...*

Number of initial fonts followed by their names, for example  
 fonts 4 R I B S

**charset**

This line comes last, followed by a list of special character names for the device, separated by spaces or newlines, as **bu** for `\(bu`

Lines of a font description file have the following forms.

**name** *name*

name of the font, such as **R** or **CW**

**internalname** *name*

The typesetter's name for the font, independent of the *troff* name or font position.

**special**

A *troff* special font, logically part of all non-special fonts.

**ligatures** *name ... 0*

The named ligatures are available. Legal names are **ff fi fl ffi ffl**.

**spacewidth** *n*

Space is *n* units wide (default 1/3 of an em).

**charset**

Must come last. Each line following **charset** describes one character thus:

*name width height code*

*Name* is either a single ASCII character or a special character listed in *Width* is in basic units. *Height* is 1 if the character descends below the baseline, 2 if it rises above the letter 'a', 3 if it both rises and descends, 0 for neither. *Code* is the number sent to the typesetter to produce the character. If a character name is a synonym for the preceding one, its width, height, and code may be replaced by one double quote [CB]."

Lines beginning with # are comments in both DESC and font description files.

*Troff* and its postprocessors use the binary versions as compiled by a program *makedev*.

**FILES**

`/usr/lib/font/dev*`

typesetter description directory

DESC typesetter description (ASCII)

DESC.out  
typesetter description (binary); created by *makedev*

*font* description of the named *font* (ASCII)

**font.out**  
description of the named *font* (binary); created by *makedev*

/n/bowell/usr/src/cmd/troff/makedev

## SEE ALSO

[troff\(1\)](#)

B. W. Kernighan, 'A Typesetter-Independent Troff', this manual, Volume 2

**NAME**

*fstab*, *mtab* – information about file systems

**SYNOPSIS**

```
#include <fstab.h>
```

**DESCRIPTION**

The file */etc/fstab* describes the normal configuration of file systems. It guides the default operation of *mount*, *umount*, *swapon*, and *fsck(8)*. The order of records in */etc/fstab* is important.

Each line of the file describes one file system. Fields separated by colons specify

```
pathname of block device or other mounted object
pathname of mount point
file system type number
integer mount flags
pass number for checking; see fsck(8)
```

File system type numbers and flags are listed in *fmount(2)*.

Two special non-numeric file system types signify things that aren't file systems: *xx* causes the line to be ignored, *sw* signifies a swap device.

Use *getfsent(3)* to read data from */etc/fstab*.

The file */etc/mtab* lists file systems currently mounted. Each entry is a structure of the form

```
#define FSNMLG 32
struct mtab {
    char file[FSNMLG];      mount point
    char spec[FSNMLG-1];   mounted object
    char type;              file system type
};
```

**EXAMPLES**

A simple *fstab*.

```
/dev/ra00:::0:0:1
/dev/ra02:/usr:0:0:2
/dev/ra05:/tmp:0:0:3
/dev/ra10:/ra10:0:1:1
/dev/ra11::sw:0:0
/dev/ra15:/ra15:0:1:3
/dev/null:/proc:2:0:0
```

**FILES**

```
/etc/fstab
/etc/mtab
```

**SEE ALSO**

*fmount(2)*, *getfsent(3)*, *mount(8)*

**BUGS**

Swap areas are not file systems, and should not be described in *fstab*.

For compatibility with old programs and habits, two deprecated magic file system types survive: *rw* means 'type 0, flag 0' (a disk file system, mounted for reading and writing); *ro* means 'type 0, flag 1' (a disk file system, mounted read-only).

Only file systems mounted with *mount(8)* are listed in *mtab*.

**NAME**

Inode – kernel user shares structure

**SYNOPSIS**

```
#include <sys/inode.h>
```

**DESCRIPTION**

The kernel *inode* structure is used to maintain per-user shares while a user has processes running. *Inodes* are installed by *login(8)* via the *limits(2)* system call when a new user logs into the system. *Dead* *Inodes* are removed by *sharer(8)* when the last process for a user exits. The layout as given in the include file is:

```
/*
 * Structure for active shares
 */
typedef short      uid_t;
struct Inode
{
    uid_t          l_uid;          /* real uid for owner of this node */
    u_short        l_flags;       /* (see below) */
    u_short        l_shares;      /* allocated shares */
    uid_t          l_group;       /* uid for this node's scheduling group */
    float          l_usage;       /* decaying accumulated costs */
    float          l_charge;      /* long term accumulated costs */
};
/*
 * Meaning of bits in l_flags
 */
#define  ACTIVENODE    001        /* this Inode is on active list */
#define  LASTREF      002        /* set for L_DEADLIM if last reference to this Inode */
#define  DEADGROUP    004        /* group account is dead */
#define  CHNGDLIMITS  020        /* this Inode's limits have changed */
#define  NOTSHARED    040        /* this Inode does not get a share of the m/c */
```

*Inodes* are grouped together in a tree. At any level in the tree, the share of resources allocated to an individual *Inode* is that proportion of the group's resources represented by the ratio of the *Inode*'s shares to the total shares of all the *Inodes* in the group. The *l\_group* field represents the *uid* of the group leader's *Inode*. The top of the tree is represented by *root*'s *Inode*, which is initialised at system boot time.

The LASTREF bit in *l\_flags* is set for the L\_DEADLIM request to the *limits(2)* system call if the last process referencing the *Inode* has exited. The DEADGROUP bit is set if this *Inode* was the last one referencing it's group. Dead groups are collected via the L\_DEADGROUP request to the *limits(2)* system call.

The *l\_charge* field is the long term accumulated charge for consumption of resources. For group leaders, it represents the charge for the whole group. The *l\_usage* field is a number representing recent usage of resources, and is used by the scheduler to determine current share of resources.

**kern\_Inode**

Each user's *Inode* is embedded in a larger structure to hold temporary values for use by the scheduler, known as a *kern\_Inode*. The layout as given in the include file is:

```
/*
 * Kernel user share structure
 */
typedef struct kern_Inode *    KL_p;
struct kern_Inode
{
    KL_p          kl_next;        /* next in active list */
    KL_p          kl_prev;       /* prev in active list */
    KL_p          kl_parent;     /* group parent */
    KL_p          kl_gnext;      /* next in parent's group */
    KL_p          kl_ghead;     /* start of this group */
    struct Inode  kl;           /* user parameters (as above) */
};
```

```

float      kl_gshares; /* total shares for this group */
float      kl_eshare; /* effective share for this group */
float      kl_norms; /* share**2 for this lnode */
float      kl_usage; /* kl.l_usage / kl_norms */
float      kl_rate; /* active process rate for this lnode */
float      kl_temp; /* temporary for scheduler */
float      kl_spare; /* <spare> */
u_long     kl_cost; /* cost accumulating in current period */
u_long     kl_muse; /* memory pages used */
u_short    kl_refcount; /* processes attached to this lnode */
u_short    kl_children; /* lnodes attached to this lnode */
};

```

Every process has a pointer to its owner's *kern\_lnode* called *p\_lnode*. Every time a process incurs a clock tick, the value *p\_lnode->kl\_usage* multiplied by *p\_lnode->kl\_rate* is added to its scheduling priority in *p\_sharepri*. *p\_sharepri* is decayed by the clock by an amount depending on the process's *p\_nice* value — the “nicer” the process, the slower the decay. This value is copied into the low-level scheduler's priority in *p\_pri* whenever the process is run in user space.

#### SEE ALSO

limits(2), share(5), sharer(8).

**NAME**

map – digitized map formats

**DESCRIPTION**

Files used by [map\(7\)](#) are a sequence of structures of the form:

```
struct {
    signed char patchlatitude;
    signed char patchlongitude;
    short n;
    union {
        struct {
            short latitude;
            short longitude;
        } point[n];
        struct {
            short latitude;
            short longitude;
            struct {
                signed char latdiff;
                signed char londiff;
            } point[-n];
        } highres;
    } segment;
};
```

Fields `patchlatitude` and `patchlongitude` tell to what 10-degree by 10-degree patch of the earth's surface a segment belongs. Their values range from  $-9$  to  $8$  and from  $-18$  to  $17$ , respectively, and indicate the coordinates of the southeast corner of the patch in units of 10 degrees.

Each segment of  $|n|$  points is connected; consecutive segments are not necessarily related. Latitude and longitude are measured in units of 0.0001 radian. If  $n$  is negative, then differences to the first and succeeding points are measured in units of 0.00001 radian. Latitude is counted positive to the north and longitude positive to the west.

The patches are ordered lexicographically by `patchlatitude` then `patchlongitude`. A printable index to the first segment of each patch in a file named *data* is kept in an associated file named *data.x*. Each line of an index file contains `patchlatitude`, `patchlongitude` and the byte position of the patch in the map file. Both the map file and the index file are ordered by patch latitude and longitude.

Shorts are stored in little-endian order, low byte first, regardless of computer architecture. To assure portability, *map* accesses them bitwise.

**SEE ALSO**

[map\(7\)](#), [proj\(3\)](#)

**NAME**

mpxio – multiplexed i/o

**SYNOPSIS**

```
#include <sys/mx.h>
```

```
#include <sgtty.h>
```

**DESCRIPTION**

Data transfers on mpx files (see [mpx\(2\)](#)) are multiplexed by imposing a record structure on the io stream. Each record represents data from/to a particular channel or a control or status message associated with a particular channel.

The prototypical data record read from an mpx file is as follows

```
struct input_record {
    short    index;
    short    count;
    short    ccount;
    char     data[];
};
```

where *index* identifies the channel, and *count* specifies the number of characters in *data*. If *count* is zero, *ccount* gives the size of *data*, and the record is a control or status message. Although *count* or *ccount* might be odd, the operating system aligns records on short (i.e. 16-bit) boundaries by skipping bytes when necessary.

Data written to an mpx file must be formatted as an array of record structures defined as follows

```
struct output_record {
    short    index;
    short    count;
    short    ccount;
    char     *data;
};
```

where the data portion of the record is referred to indirectly and the other cells have the same interpretation as in *input\_record*.

The control messages listed below may be read from a multiplexed file descriptor. They are presented as two 16-bit integers: the first number is the message code (defined in *<sys/mx.h>*), the second is an optional parameter meaningful only with M\_WATCH, M\_BLK, and M\_SIG.

**M\_WATCH**

a process ‘wants to attach’ on this channel. The second parameter is the 16-bit user-id of the process that executed the open.

**M\_CLOSE**

the channel is closed. This message is generated when the last file descriptor referencing a channel is closed. The *detach* command (see [mpx\(2\)](#)) should be used in response to this message.

**M\_EOT**

indicates logical end of file on a channel. If the channel is joined to a typewriter, EOT (control-d) will cause the M\_EOT message under the conditions specified in [tty\(4\)](#) for end of file. If the channel is attached to a process, M\_EOT will be generated whenever the process writes zero bytes on the channel.

**M\_BLK**

if non-blocking mode has been enabled on an mpx file descriptor *xd* by executing *ioctl(xd, MXNBLK, 0)*, write operations on the file are truncated in the kernel when internal queues become full. This is done on a per-channel basis: the parameter is a count of the number of characters not transferred to the channel on which M\_BLK is received.

**M\_UBLK**

is generated for a channel after M\_BLK when the internal queues have drained below a threshold.

**M\_SIG** is generated instead of a normal asynchronous signal on channels that are joined to typewriters. The parameter is the signal number.

Two other messages may be generated by the kernel. As with other messages, the first 16-bit quantity is the message code.

**M\_OPEN**

is generated in conjunction with 'listener' mode (see [mpx\(2\)](#)). The uid of the calling process follows the message code as with **M\_WATCH**. This is followed by a null-terminated string which is the name of the file being opened.

**M\_IOCTL**

is generated for a channel connected to a process when that process executes the *ioctl(fd, cmd, &vec)* call on the channel file descriptor. The **M\_IOCTL** code is followed by the *cmd* argument given to *ioctl* followed by the contents of the structure *vec*. It is assumed, not needing a better compromise at this time, that the length of *vec* is determined by *sizeof(struct sgtyb)* as declared in *<sgtty.h>*.

Two control messages are understood by the operating system. **M\_EOT** may be sent through an mpx file to a channel. It is equivalent to propagating a zero-length record through the channel; i.e. the channel is allowed to drain and the process or device at the other end receives a zero-length transfer before data starts flowing through the channel again. **M\_IOANS** can also be sent through a channel to reply to a **M\_IOCTL**. The format is identical to that received from **M\_IOCTL**.

**SEE ALSO**

[mpx\(2\)](#)



**NAME**

netnews – usenet news articles, utility files

**DESCRIPTION**

There are two formats of news articles: A and B. Format A is the only format that the older *netnews*(A) understands. *Readnews* and *postnews*(7) deal with both formats, but produce B by default.

Format A looks like this:

```
Aarticle-ID
newsgroups
path
date
title
body of article
```

Format B contains two extra pieces of information: receipt date and expiration date. A file in B format consists of a series of headers and then the body. A header is a line with a capital letter in the 1st column and a colon somewhere on the line. Unrecognized header fields are ignored. News is stored in whichever format it was created. The following fields are among those recognized:

```
From:
Newsgroups:
Subject:
Date:
Date-Received:
Expires:
Reply-To:
References: ID of article this is a follow-up to
Control: Text of a control message
```

Each line of the control file `/usr/lib/news/sys` file line has four fields, separated by colons:

```
system-name:subscriptions:flags:transmission command
```

Only the *system-name* and *subscriptions* need to be present.

The *system name* is the name of the system being sent to. The *subscriptions* are the newsgroups it gets. The *flags* are a set of letters describing how the article should be transmitted. The default is **B**. Valid flags include **A**, **B**, **N** (use *ihave/sendme* protocol), **U** (use *uux* and the name of the stored article in a `%s` string).

The *transmission command* is executed by the shell with the article to be transmitted as the standard input. The default is `uux - -z -r sysname!rnews`.

Somewhere in the control file, there must be a line for the host system. This line has no *flags* or *transmission commands*. A `#` as the first character in a line denotes a comment.

**FILES**

```
/usr/lib/news/*
/usr/spool/news/*
```

**SEE ALSO**

*postnews*(7), *readnews*(7)

M. Horton, *Standard for the Interchange of USENET Messages*, RFC850, DARPA Information Processing Techniques Office, Arlington VA, 1983

**NAME**

passwd, group – password and group files

**DESCRIPTION**

The file `/etc/passwd` has one line for each user with the following information:

- name (login name, contains no upper case)
- encrypted password
- numerical user ID
- numerical group ID
- comp center account number, box number, optional user-id
- initial working directory
- program to use as shell

Fields are separated by colons. The comp center field is used only when communicating with certain systems, and in other installations can contain any desired information. If the password field is null, no password is demanded; if the shell field is null, `/bin/sh` is used.

The file `/etc/group` defines the membership of users in permission groups. It contains one line for each group with the following colon-separated fields:

- group name
- encrypted password
- numerical group ID
- comma-separated list of all users allowed in the group

Each group is separated from the next by a new-line. If the password field is null, [newgrp\(1\)](#) will not demand a password.

Because the passwords are encrypted, these files can and do have general read permission and can be used, for example, to map numerical ids to names.

**FILES**

`/etc/passwd` `/etc/group`

**SEE ALSO**

[newgrp\(1\)](#), [getpwent\(3\)](#), [login\(8\)](#), [crypt\(3\)](#), [passwd\(1\)](#)

**BUGS**

[Passwd\(1\)](#) won't change passwords in the group file.

**NAME**

picfile – raster graphic image format

**DESCRIPTION**

Files in this format store images represented as two-dimensional arrays of multiple-channel pixels. A *picfile* consists of an ASCII header followed by binary data encoding the pixels in row-major order. The header is a list of attribute/value pairs separated by newlines, terminated by an empty line. Each header line has the form *name=value*. The name may not contain an ASCII NUL, newline or =; the value may not contain null or newline. The last line of a header is empty.

The standard attributes are described below; all but **TYPE** and **WINDOW** are optional. **TYPE** must come first; otherwise order is irrelevant. As any unrecognised attribute is passed over uninterpreted by all standard software, applications are welcome to include arbitrary annotations, like **SHOESIZE=10**, if they wish.

**TYPE=type**

How the pixels are encoded. Standard types are

**runcode**

A run-length encoding. The data are a sequence of (*nchan*+1)-byte records each containing a count *k* and *nchan* bytes giving a pixel value to be repeated *k*+1 times. A run may not span scanlines.

**dump  
bitmap**

A two-dimensional array of *nchan*-byte records in row major order.

One-bit pixels, packed into bytes high bit leftmost. Zero bits are white, one bits are black. Rows are padded with zeros to a multiple of 16 bits.

**ccitt-g4**

A black-and-white image under CCITT FAX Group 4 compression. This format is highly compressive on images of text and line art. Similarly, `ccitt-g31` and `ccitt-g32` for Group 3, 1-D and 2-D.

**pico  
ccir601**

A sequence of *nchan* two-dimensional arrays of single bytes.

Pixels are in dump order, 2 bytes per pixel encoded according to the IEEE digital component video standard.

**WINDOW=x0 y0 x1 y1**

The *x,y* coordinates of the upper left corner and the point just diagonally outside the lower right corner, *x* increasing to the right, *y* down.

**NCHAN=nchan**

The number of channels, default 1.

**CHAN=value**

The order of channels.

**RES=x y**

The digitizing resolution horizontally and vertically, in pixels/inch.

**CMAP=**

(The value is empty.) A color map, a 256×3-byte translation table for color values, follows the header. In a full-color picture, each color-map row maps pixel values of the corresponding channel. In a monochrome picture, pixel values index the color map to yield red, green and blue, like this:

```
unsigned char cmap[256][3];
red=cmap[pixel][0];
green=cmap[pixel][1];
blue=cmap[pixel][2];
```

**EXAMPLES**

**sed '/^\$/q' image**

Print a header. A sample header follows.

```
TYPE=dump
WINDOW=0 0 512 512
NCHAN=1
```

```
CHAN=m
RES=300 300
CMAP=
COMMAND= antiquantize 'halftone CLASSIC' 512.halftone LIBERTY.anticlassic
COMMAND= halftone CLASSIC 512.liberty 512.halftone 1.75 512.halftone
COMMAND= transpose IN OUT
COMMAND= resample 512 IN OUT
COMMAND= transpose IN OUT
COMMAND= resample 512 IN OUT
COMMAND= clip 400 400 LIBERTY OUT
```

**SEE ALSO**

[bcp\(1\)](#), [cscan\(1\)](#), [imscan\(1\)](#), [pico\(1\)](#), [flicks\(9\)](#) *mugs* in [face\(9\)](#) [rebecca\(9\)](#) [flickfile\(9\)](#)

T. Duff, 'The 10th Edition Raster Graphics System', this manual, Volume 2

**NAME**

plot – graphics interface

**DESCRIPTION**

Files of this format are produced by routines described in [plot\(3\)](#), and are interpreted for various devices by commands described in [plot\(1\)](#). A graphics file is an ASCII stream of instruction lines. Arguments are delimited by spaces, tabs, or commas. Numbers may be floating point. Punctuation marks (except `:`), spaces, and tabs at the beginning of lines are ignored. Comments run from `:` to newline. Extra letters appended to a valid instruction are ignored. Thus `...line`, `line`, `li` all mean the same thing. Arguments are interpreted as follows:

1. If an instruction requires no arguments, the rest of the line is ignored.
2. If it requires a string argument, then all the line after the first field separator is passed as argument. Quote marks may be used to preserve leading blanks. Strings may include newlines represented as `\n`.
3. Between numeric arguments alphabetic characters and punctuation marks are ignored. Thus `line` from `5 6 to 7 8` draws a line from (5, 6) to (7, 8).
4. Instructions with numeric arguments remain in effect until a new instruction is read. Such commands may spill over many lines. Thus the following sequence will draw a polygon with vertices (4.5, 6.77), (5.8, 5.6), (7.8, 4.55), and (10.0, 3.6).

```
move 4.5 6.77
vec 5.8, 5.6 7.8
4.55 10.0, 3.6 4.5, 6.77
```

The instructions are executed in order. The last designated point in a **line**, **move**, **rmove**, **vec**, **rvec**, **arc**, or **point** command becomes the ‘current point’ (X,Y) for the next command. Each of the following descriptions corresponds to a routine in [plot\(3\)](#).

**Open & Close**

- o** *string* Open plotting device. For *troff*, *string* specifies the size of the plot (default is 6i.)
- cl** Close plotting device.

**Basic Plotting Commands**

- e** Start another frame of output or erase the screen on CRT terminals without scroll.
- m** *x y* (move) Current point becomes *x y*.
- rm** *dx dy* Current point becomes *X+dx Y+dy*.
- poi** *x y* Plot the point *x y* and make it the current point.
- v** *x y* Draw a vector from the current point to *x y*.
- rv** *dx dy* Draw vector from current point to *X+dx Y+dy*
- li** *x1 y1 x2 y2* Draw a line from *x1 y1* to *x2 y2*. Make the current point *x2 y2*.
- t** *string* Place the *string* so that its first character is centered on the current point (default). If *string* begins with `\C` it is centered (right-adjusted) on the current point. A backslash at the beginning of the string may be escaped with another backslash.
- a** *x1 y1 x2 y2 xc yc r* Draw a circular arc from *x1 y1* to *x2 y2* with center *xc yc* and radius *r*. If the radius is positive, the arc is drawn counterclockwise; negative, clockwise. The starting point is exact but the ending point is approximate.
- ci** *xc yc r* Draw a circle centered at *xc yc* with radius *r*. If the range and frame parameters do not specify a square, the ‘circle’ will be elliptical.
- di** *xc yc r* Draw a disc centered at *xc yc* with radius *r* using the filling color (see **cfill** below). Only works on the 5620; on other devices is the same as **circle**.
- bo** *x1 y1 x2 y2* Draw a box with lower left corner at *x1 y1* and upper right corner at *x2 y2*.
- sb** *x1 y1 x2 y2* Draw a solid box with lower left corner at *x1 y1* and upper right corner at *x2 y2* using the filling color (see **cfill** below).

- par** *x1 y1 x2 y2 xg yg*  
 Draw a parabola from *x1 y1* to *x2 y2* ‘guided’ by *xg yg*. The parabola passes through the midpoint of the line joining *xg yg* with the midpoint of the line joining *x1 y1* and *x2 y2* and is tangent to the lines from *xg yg* to the endpoints.
- pol** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Draw polygons with vertices *x1 y1 ... xn yn* and *X1 Y1 ... Xm Ym*. If only one polygon is specified, the inner brackets are not needed.
- fi** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Fill a polygon. The arguments are the same as those for **pol** except that the first vertex is automatically repeated to close each polygon. The polygons do not have to be connected. Enclosed polygons appear as holes.
- sp** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Draw a parabolic spline guided by *x1 y1 ... xn yn* with simple endpoints.
- fsp** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Draw a parabolic spline guided by *x1 y1 ... xn yn* with double first endpoint.
- lsp** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Draw a parabolic spline guided by *x1 y1 ... xn yn* with double last endpoint.
- dsp** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Draw a parabolic spline guided by *x1 y1 ... xn yn* with double endpoints.
- csp** { {*x1 y1 ... xn yn*} ... {*X1 Y1 ... Xm Ym*} }  
 Draw a parabolic spline guided by *x1 y1 ... xn yn* with double endpoints.
- in** *filename*  
 (include) Take commands from *filename*.
- de** *string* { *commands* }  
 Define *string* as *commands*.
- ca** *string scale*  
 Invoke commands defined as *string* applying *scale* to all coordinates.

### Commands Controlling the Environment

- co** *string* Draw lines with color *string*. Available colors depend on the device. *String* may contain definitions for several devices separated by /. Colors possible for the various devices are:
- pen     **black, red, green, blue, Tblack, Tred, Tgreen, Tblue** (assumes default carousel,  
           **T=thick**  
           **1-8** (pen number)  
           **Snumber** character size as % of plotting area
- troff    **Ffont**  
           **Ppoint size**
- 2621    **Hcharacter** for plotting
- pe** *string* Use *string* as the style for drawing lines. Not all pen styles are implemented for all devices. *String* may contain definitions for several devices separated by /. The available pen styles are:
- pen     **solid, dott[ed], short, long, dotd[ashed], cdash, ddash**  
 4014    solid, **dott[ed], short, long, dotd[ashed], ddash**
- troff    **solid, dash** only straight lines will be dashed
- 5620    **Bnumber** line thickness
- 2621    **Hcharacter** for plotting
- cf** *string* Color for filling; may contain the definitions for several devices. separated by /. The following colors are available on the specified devices:
- pen     **black, red, green, blue, Tblack, Tred, Tgreen, Tblue**  
           **1-8** pen number
- 5620    **Btexture** string with octal numbers for texture; see [types\(9\)](#) The 16 words of texture should be followed by one word for the mode used by *texture()*; see [bitblt\(9\)](#)
- 2621    **Hcharacter** for filling
- All     /**Adegrees** slant of shading lines  
           /**Gnumber** gap between shading lines (in user units)

**ra** *x1 y1 x2 y2*

The data will fall between *x1 y1* and *x2 y2*. The plot will be magnified or reduced to fit the device as closely as possible.

Range settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *plot(1)*. The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices with nonsquare faces.

4014	range
troff	range
2621	range
5620	range dependent on layer size
pen	range dependent on paper size

**fr** *px1 py1 px2 py2*

Plot the data in the fraction of the display specified by *px1 py1* for lower left corner and *px2 py2* for upper right corner. Thus *frame* plots in the lower right quadrant of the display; *frame* uses the whole display but inverts the y coordinates.

**sa** Save the current environment, and move to a new one. The new environment inherits the old one. There are 7 levels.

**re** Restore previous environment.

**SEE ALSO**

*plot(1), plot(3), graph(1)*

**NAME**

poly – polyhedron database

**DESCRIPTION**

The directory `/usr/lib/polyhedra` contains an index file and many polyhedron description files, each describing a solid polyhedron and its (not necessarily unique) planar net. Each line of the index file consists of a polyhedron's number followed by a horizontal tab and the polyhedron's name. The polyhedron's number is also the name of its description file. The routines of `poly(3)` read such description files.

Each description file consists of a number of fields. Each field begins with a line consisting of `:` and the field name. The field continues until the next header line or end of file. Some fields contain *values*, which consist of a floating point number optionally followed by an algebraic expression enclosed in `[ ]`, or *angles*, which are a *value* in radians and optionally two more values (sin and cos) separated by `@`. The fields include, but are not limited to,

**number**

The polyhedron's number.

**name** The polyhedron's name, less than 128 characters long and not capitalized.

**symbol**

The Schlaefli symbol, a tab, and the Johnson symbol for the polyhedron, given in `eqn(1)` style with delimiters `@@`.

**dual** The name of the dual polyhedron optionally followed by a horizontal tab and the number of the dual.

**vertices**

The first line is the number of vertices, which follow, one per line. Each vertex has a coordinate (three *values* separated by spaces), a number *n*, and *n* face,edge pairs that surround the vertex.

**faces** The first line contains the number of faces and the maximum number of vertices in any face. The remaining lines are the faces, each with a vertex count *n*, followed by *2n* vertex numbers (planar, solid), *2n* edge numbers (planar, solid), and *n* angles. The vertices of each face are listed in the same order for both the planar and solid forms: counter-clockwise as viewed from above the planar net (i.e. from *z*>0) which generally corresponds to outside the solid polyhedron.

**edges** The first line contains the number of edges. Each edge is one line: *face1 vertex1 face2 vertex2 length angle*. The length is a *value*.

**summary**

The three lines summarise the different kinds of faces, vertices and edges respectively. Each line consists of a total and a list of *count example symbolic* triples; where *example* is an index into the appropriate list and *symbolic* is given in `eqn(1)` style with delimiters `@@`.

**EOF** The end of the polyhedron's description. (Another polyhedron description may follow in this file.)

An expression in a *value* gives the exact value in the syntax of `bc(1)` using these functions:  $a(x) = \tan^{-1}(x)$ ,  $b(x) = (x)^{1/3}$ ,  $c(x) = \cos(x)$ ,  $d(x) = \tan(x)$ ,  $p = \pi$ ,  $q(x) = x^2$ ,  $r(x) = \cos^{-1}(x)$ ,  $s(x) = \sin(x)$ ,  $t = \phi = (1 + \sqrt{5})/2$ . (.)if t .ig  $a(x) = \arctan(x)$ ,  $b(x) = \text{cubrt}(x)$ ,  $c(x) = \cos(x)$ ,  $d(x) = \tan(x)$ ,  $p = \text{pi}$ ,  $q(x) = x^2$ ,  $r(x) = \arccos(x)$ ,  $s(x) = \sin(x)$ ,  $t = \text{phi} = (1 + \text{sqrt}(5))/2$ . (.)The code may include assignments but does not include white space.

**FILES**

`/usr/lib/polyhedra/index` index file  
`/usr/lib/polyhedra/[0-9]*` description files

**SEE ALSO**

`poly(3)`, `poly(7)`



**NAME**

Share – Share Scheduling on Unix

**SYNOPSIS**

Scheduling for a share of the machine

**DESCRIPTION**

*Share* is a term covering those elements of the Unix kernel that affect the priority of a user's job. The basic scheduler in Unix schedules processes on a short term *per-process* basis. The *share scheduler* takes account of the history of a user's usage of the resources of the machine, and introduces a *per-user* long term scheduler. To do this, several variables are available to the scheduler in a per-user data structure known as an *Inode*. These record the intended share of the machine that the user should get, the recent history of resources consumed ("usage"), and the number of active (running) processes belonging to the user. Together, these affect the priority of each of the processes so that consumption of resources is adjusted toward the intended share.

A user's *usage* is calculated by accumulating the charges incurred by use of resources, and decaying the result over time. The share scheduler affects the low-level scheduling of a user's processes by adding the user's *usage* divided by the allocated share, and multiplied by the active process count, to the priority of a process every time that process incurs a clock tick. Since the larger its priority, the less often a process is scheduled, processes belonging to users with high *usage*, low share, or many active processes will get a smaller share of resources. Note that at any one time, a user can use all of the resources available provided there is no competition from others.

**SCHEDULING GROUPS**

*Inodes* are organised into a tree. For any particular sub-tree, the sub-tree's share of resources is divided up between the *Inodes* according to their relative shares. Sub-trees are also known as groups. The root *Inode* of the group is the group owner, and the leaf *Inodes* are its users. The total shares issued to the group include both those issued to the owner, as well as those issued to the users. Both owner and users are known as group members. The share of the group's resources allocated to any particular member of the group is in the proportion of the member's shares divided by the group's shares.

The most interesting group is the one at the top of the tree, whose owner is "root", and its group members are the primary scheduling groups. "root" gets 100% of the available resources, which is split as above between the primary scheduling groups.

Not all group owners represent real users, and in these cases there is no need to allocate them a share of resources. Such *Inodes* are indicated by the NOTSHARED flag, which causes the scheduler to ignore their shares when allocating their group's resources among its members. However, the long term *charge* of a group owner always includes all the charges levied on any member of its group.

For reasons of system management, "root" is always allocated 100% of the resources whenever it needs them. However, since all real users run on their own *Inodes*, the NOTSHARED flag is turned on for "root", and thus the primary scheduling groups have 100% of the available resources to share between themselves. However, for instrumentation purposes, "root"'s *charge* only represents its own consumption of resources, but the total consumption of resources is accumulated in the *kl\_temp* field of "root"'s *kern\_Inode* (see [Inode\(5\)](#) for details of a *kern\_Inode*.)

**CHARGES**

Charges making up the accumulating usage figure are levied by default as follows:-

cpu	100%
disk i/o	0%
terminal i/o	0%
system services	0%
memory	0%

Memory charges are levied every scheduler cycle, but note that *root* is never charged for the memory it uses. These charges can be varied at different times of the day to reflect their popularity by using the command [charge\(1\)](#).

Usage is decayed at an exponential rate intended to ensure that all users of the machine get an equal chance to compete for resources over a particular time period. The default decay results in a *half-life* of 2 minutes. Use [charges\(1\)](#) to find out the current decay rate and resource charges.

**NICE**

The *nice*(2) system call has a slightly different effect under Share. The *nice* parameter for a process now affects the rate at which its priority decays to a higher priority over time. *Nicing* a process will make it run slower, by reducing its effective share of the resources, but it may not run slower than another user's processes if that user has an even lower effective share of the resources. However, processes with a *nice* priority of 19 are guaranteed only to run when no other processes need the CPU. *Niced* processes are charged less for CPU time than normal processes, priority 19 processes are charged almost nothing for CPU time.

**MANAGEMENT**

There are three flags that control the operation of the share scheduler.

- NOSHARE** This turns off the scheduler. Since this will leave the parameters in a “frozen” state, it should probably only be done at system boot time. This flag is on by default when the system is booted, so the command *charge*(1) must be run to activate the scheduler. Note that the program *login*(8) won't attach users to their own *Inodes* while this flag is on, instead each user will remain attached to *root*'s *Inode*. [Value 01]
- ADJGROUPS** This flag turns on global group effective share adjustment. If any group is found to be getting less than MINGSHARE times its allocated share, then the costs incurred by its members are reduced proportionately. [Value 02]
- LIMSHARE** This flag deals with an “edge effect” that occurs when users first log in. It may be that their *usage* field has decayed to the point where they might temporarily be allocated nearly 100% of the machine. This flag limits any one user's share of the resources to no more than MAXUSHARE times their intended share. Of course, this still may be nearly 100%, if no other users are logged in, or the other users have very small shares. [Value 04]

The *charge*(1) command is used to manipulate these flags, and the charging parameters above. There are also other parameters which may be changed with *charge*:-

- DecayRate** The decay rate for users' *active process rates*. This parameter is calculated by counting the active processes per user every clock scan, and is decayed every clock scan. The usual value for this should result in a *half-life* for the rate of about 10 seconds.
- DecayUsage** The decay rate for users' usages. This may be altered to produce a *half-life* for usage ranging from a few seconds to many days.
- Delta** The run frequency of the share scheduler in seconds. The default value of 4 is fine.
- MAXGROUPS** This sets the maximum group nesting (depth of scheduling tree) allowed, not including “root”'s group.
- MAXPRI** Absolute upper bound for a process's priority.
- MAXUPRI** Upper bound for normal processes' priorities. *Idle* processes run with priorities in the range  $MAXUPRI < pri < MAXPRI$ .
- MAXUSAGE** Upper bound for “reasonable” usages. Users with usages larger than this are grouped together and given process priorities which prevent them from interfering with “normal” users. The usage (multiplied by the *active process rate*) is added to a running process's priority every time it incurs a clock tick, so the upper bound should be small enough not to overrun the value MAXUPRI in too short a time interval
- MAXUSERS** Sets the maximum number of users and groups that can be active. Note that this cannot exceed the maximum configured in the kernel.
- PriDecay** This is the decay rate for maximally niced processes. A reasonable minimum value for the *half-life* is about 100 seconds, but see the comment for MAXUSAGE above.

**PriDecayBase**      The base for calculating the decay rate for process priorities with normal *nice*. This should be set low enough so that the priorities of processes for users with low share don't decay too quickly. A reasonable minimum value for the *half-life* is about 2 seconds.

## FILES

*/usr/include/sys/share.h*      Definition of scheduler parameters.  
*/usr/include/sys/charges.h*    Default scheduler parameters.

## SEE ALSO

charge(1), pl(1), rates(1), shstats(1), ustats(1), lnode(5), shares(5), login(8), sharer(8).

## REFERENCES

"Scheduling for a Share of the Machine", J Larmouth, SP&E, Vol 5 1975 pp 29-49  
"Scheduling for Immediate Turnaround", J Larmouth, SP&E, Vol 8 1978 pp 559-578  
"A Fair Share Scheduler", J Kay & P Lauder, TM 11275-870319-01  
"Share Scheduler Administration", P Lauder

**NAME**

/etc/shares – shares data-base file for share system

**DESCRIPTION**

*/etc/shares* is an direct access data-base indexed on *uid* containing the uid, scheduling group and allocated shares for each user on the system. It also contains other scheduling data as defined in the files *<shares.h>* and *<sys/lnode.h>*.

Operations on the shares file are made via the shares routines described in section 3.

Data from the shares file are installed in kernel *lnode* structures for active users by *login(8)*. When users become inactive, the *lnode* structures are removed from the kernel and updated in the shares file by *sharer(8)*

The number of shares and the scheduling group of a user may be changed by using *passwd(1)*, with the *-a* or *-n* flags, or by *lim(1)*. Data in the shares file may be examined with either *pl(1)* or *pwintf(1)*.

**FILES**

<i>/etc/shares</i>	User data base.
<i>/usr/include/shares.h</i>	Format of an <i>/etc/shares</i> record.
<i>/usr/include/sys/lnode.h</i>	Format of an <i>lnode</i> structure in an <i>/etc/shares</i> record.

**SEE ALSO**

*lim(1)*, *pl(1)*, *pwintf(1)*, *closeshares(3)*, *getshares(3)*, *getshput(3)*, *openshares(3)*, *putshares(3)*, *setupshares(3)*, *sharesfile(3)*, *lnode(5)*, *share(5)*, *login(8)*, *sharer(8)*.

**NAME**

stab – symbol table types

**SYNOPSIS**

```
#include <stab.h>
```

**DESCRIPTION**

The include file `<stab.h>` defines some values of the `n_type` field of the symbol table of object files; see [a.out\(5\)](#). These are the types for permanent symbols used by the compilers [cc\(1\)](#) and [f77\(1\)](#) and the debugger [pi\(9\)](#). Symbol table entries are produced by assembler directives:

**.stabs**

specifies a name in quotes " ", a symbol type one char one short and an unsigned long usually an address).

**.stabd**

the same, referring to the current location without an explicit name.

**.stabn**

generates entries with no name.

The loader [ld\(1\)](#) preserves the order of symbol table entries produced by these directives.

The low bits of the `n_type` field place a symbol into at most one segment, according to the following masks, defined in `<a.out.h>`.

```
#define N_UNDF  0x0   /* undefined */
#define N_ABS   0x2   /* absolute */
#define N_TEXT  0x4   /* text */
#define N_DATA  0x6   /* data */
#define N_BSS   0x8   /* bss */
#define N_EXT   0x1   /* external bit, or'ed in */
```

The `n_value` field of a symbol is relocated by `ld` as an address within the appropriate segment, or is unchanged for a symbol not in any segment. In addition, the loader will discard certain symbols, according to rules of its own, unless the `n_type` field has one of the following bits set:

```
#define  N_STAB  0xe0
```

This allows up to 112 symbol types, split among the various segments. Some of these have already been claimed. Option `-g` of `cc` uses the following values, all 4 mod 16, for text symbols. Comments show the pertinent fields of the `.stabs` directive.

```
#define N_BFUN  0x24  /* procedure: name,,0,lineno,address */
#define N_FUN   0x24
#define N_NARGS 0x34  /* function call: ,,0,nbytes,address */
#define N_SLINE 0x44  /* src line: ,,0,lineno,address */
#define N_SO    0x64  /* source file: name,,0,lineno,address */
#define N_SOL   0x84  /* #include file: name,,0,lineno,address */
#define N_ESO   0x94  /* end source file: name,,0,lineno,address */
#define N_ENTRY 0xa4  /* alternate entry: name,,0,lineno,address */
#define N_RFUN  0xb4  /* return from function: ,,0,lineno,address */
#define N_LBRAC 0xc4  /* left bracket: ,,0,level,address */
#define N_RBRAC 0xd4  /* right bracket: ,,0,level,address */
#define N_EFUN  0xf4  /* end of function: name,,0,lineno,address */
```

These values, all 8 mod 16, are used for data symbols:

```
#define N_LCSYM 0x28  /* .lcomm symbol: name,,0,type,address */
#define N_ECOML 0xe8  /* end common (local name): ,,address */
```

And these for non-relocated symbols:

```
#define N_GSYM  0x20  /* global symbol: name,,0,type,0 */
#define N_FNAME 0x22  /* procedure name (f77 kludge): name,,0 */
#define N_STFUN 0x32  /* static function: name,,0,type,0 */
#define N_RSVM  0x40  /* register sym: name,,0,type,register */
```

```

#define N_BSTR 0x5c /* begin structure: name,,0,type,length */
#define N_ESTR 0x5e /* end structure: name,,0,type,length */
#define N_SSYM 0x60 /* structure elt: name,,0,type,offset */
#define N_SFLD 0x70 /* structure field: name,,0,type,offset */
#define N_LSYM 0x80 /* local sym: name,,0,type,offset */
#define N_PSYM 0xa0 /* parameter: name,,0,type,offset */
#define N_BCOMM 0xe2 /* begin common: name,, */
#define N_ECOMM 0xe4 /* end common: name,, */
#define N_VER 0xf0 /* symbol table version number */
#define N_TYID 0xfa /* struct, union, or enum name */
#define N_DIM 0xfc /* dimension for arrays */

```

Field `n_desc` holds a type specifier in the form used by `cc(1)`, by up to 6 qualifiers, with **q1** most significant:

```

struct desc {
    short q6:2, q5:2, q4:2, q3:2, q2:2, q1:2;
    short basic:5;
};

```

The qualifiers are coded thus: 0 none 1 pointer 2 function 3 array

The basic types are coded thus: 0 undefined 1 function argument 2 character  
 3 short 4 int 5 long 6 float 7 double 8 struc-  
 ture 9 union 10 enumeration 11 member of enumeration 12 unsigned  
 character 13 unsigned short 14 unsigned int 15 unsigned long 16 void

The Pascal compiler, `pc(A)`, uses the following `n_type` value:

```

#define N_PC 0x30/* global pascal symbol: name,,0,subtype,line */

```

and uses the following subtypes to do type checking across separately compiled files: 1 source  
 file name 2 included file name 3 global label 4 global constant  
 5 global type 6 global variable 7 global function 8 global proce-  
 dure 9 external function 10 external procedure

## SEE ALSO

[a.out\(5\)](#), [pi\(9\)](#) [as\(1\)](#), [ld\(1\)](#)

## BUGS

The loader's relocation conventions limit the number of useful `n_type` values.

**NAME**

termcap – terminal capability file

**DESCRIPTION**

*Termcap* describes terminals as used, for example, by *vi(1)* and *curses(3)* or in the **TERMCAP** environment variable. A *termcap* entry is a line containing fields separated by `:`. Lines may be broken; `\` at the end of a line signifies continuation. Empty fields are ignored.

The first field for each entry gives names for a terminal separated by `|`. The first name is conventionally two characters long for the benefit of older systems; the second name is the customary abbreviation; and the last name fully identifies the terminal.

There are three types of capability: Boolean for the presence of a feature, numeric for sizes and time delays, and strings for performing operations. Some string fields may be preceded by a number, which specifies padding, a time delay required with the operation. These capabilities are marked ‘P’ or ‘P\*’ below. Padding is measured in milliseconds; P\* signifies that the padding is proportional to the number of lines (or characters) affected, for example, `3.5*` specifies 3.5 milliseconds per unit.

Name	Type	Pad	Description
ae	str	P	End alternate character set
al	str	P*	Add new blank line
am	bool		Automatic margin
as	str	P	Start alternate character set
bc	str		Backspace char if not <code>^H</code>
bs	bool		Terminal can backspace
bt	str	P	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	P*	Clear to end of display
ce	str	P	Clear to end of line
ch	str	P	Like cm but horizontal motion only
cl	str	P*	Clear screen
cm	str	P	Cursor motion
co	num		Number of columns
cr	str	P*	Carriage return, default <code>^M</code>
cs	str	P	Change scrolling region ( <code>vt100</code> ), like cm
cv	str	P	Like ch, but vertical only
da	bool		Display may be retained above
dB	num		Backspace delay
db	bool		Display may be retained below
dC	num		Carriage return delay
dc	str	P*	Delete character
dF	num		Form feed delay
dl	str	P*	Delete line
dm	str		Enter delete mode
dN	num		Newline delay
do	str		Down one line
dT	num		tab delay
ed	str		End delete mode
ei	str		End insert mode; give <code>:ei=:</code> if ic
eo	str		Can erase overstrikes with blank
ff	str	P*	Hard copy page eject, default <code>^L</code>
hc	bool		Hardcopy terminal
hd	str		Half line down
ho	str		Home cursor if no cm
hu	str		Half line up
hz	str		Hazeltine, can't print
ic	str	P	Insert character
if	str		Name of file containing initializationis

im	bool	Enter insert mode; give :im=: if ic
in	bool	Insert mode distinguishes nulls on display
ip	str	P*Insert pad after character insert
is	str	Terminal initialization string
k0-k9	str	Other function key codes
kb	str	Backspace key code
kd	str	Down arrow key code
ke	str	Leave keypad transmit mode
kh	str	Home key code
kl	str	Left arrow key code
kn	num	Number of function keys
ko	str	Termcap entries for other non-function keys
kr	str	Right arrow key code
ks	str	Enter keypad transmit mode
ku	str	Up arrow key code
l0-9	str	Labels on other function keys
li	num	Number of lines on screen or page
ll	str	Last line, first column, if no cm
ma	str	Arrow key map
mi	bool	Safe to move in insert mode
ml	str	Memory lock above cursor
ms	bool	Safe to move in standout or underline mode
mu	str	Turn off memory lock
nc	bool	No correctly working CR (DM2500, H2000)
nd	str	Nondestructive space (cursor right)
nl	str	P*Newline character, default \n
ns	bool	Nonscrolling CRT
os	bool	Terminal overstrikes
pc	str	Pad character, default NUL
pt	bool	Has hardware tabs (possibly set by is)
se	str	Leave standout mode
sf	str	PScroll forward
sg	num	Number of blanks left by so, se
so	str	Enter standout mode
sr	str	PScroll reverse (backward)
ta	str	PTab, if not ^I or if padded
tc	str	Entry of similar terminal, must be last
te	str	String to end programs that use cm
ti	str	String to begin programs that use cm
uc	str	Underscore one char and move past it
ue	str	Leave underscore mode
ug	num	Number of blanks left by us, ue
ul	bool	Terminal underlines but doesn't overstrike
up	str	Cursor up one line
us	str	Enter underscore mode
vb	str	Visible bell, (may not move cursor)
ve	str	Leave open/visual mode
vs	str	Enter open/visual mode
xb	bool	Beehive (f1=escape, f2=^C)
xn	bool	Newline ignored after wrap (Concept)
xr	bool	Return acts like ce \r \n (Delta Data)
xs	bool	Standout not erased by writing over (HP264?)
xt	bool	Tabs are destructive, magic so (Telera 1061)

The following example is one of the more elaborate *termcap* entries. (Do not believe it; see the file for current facts.)



```
co|c100|concept 100:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200\Eo\47\E:\
:al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
:dc=16\E^A:d1=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:\
:nd=\E=:se=\Ed\E:so=\ED\EE:ta=8\t:ul:up=\E;vb=\Ek\Ek:xn:
```

Among the Boolean capabilities shown for the Concept are automatic margins `am`: automatic return and linefeed at the end of a line. Numeric capabilities are indicated by #; `co#80` means the Concept has 80 columns. String capabilities are indicated by =; to clear to end of line on the Concept, issue `<escape>` `<control-C>` and pad with 16 milliseconds delay.

In strings the ASCII ESC character is represented by `\E` and control characters are represented by `^c`, where character `c` has ASCII code 0100 greater than the desired control character. Newline, return, tab, backspace, form feed, `\` and `^` are represented by `\n` `\r` `\t` `\b` `\f` `\|` `^`. Backslash `\` followed by 3 digits specifies a byte in octal. A null character is encoded `\200`: the routines that use *termcap* information mask out the high bit of all bytes.

Local cursor motions are undefined if they run off the left or top of the screen; the *curses* routines refrain from issuing such motions. It is assumed that the screen will scroll up upon running off the bottom; this assumption is negated by `ns`. Capability `am` (automatic margin) describes the handling of the right margin.

Cursor addressing is described by capability `cm`, which contains *printf(3)*-like format codes for line and column positions. The leftmost column is column 0.

```
%d    as in printf
%2     like %2d
%3     like %3d
%.     like %c
%+x    adds x before converting
%>xy  if value exceeds x, add y, no output
%r     reverse order of line and column, no output
%i     increment line/column (1-origin)
%%     single %
%n     exclusive or row and column with 0140 (DM2500)
%B     BCD: 16*(x/10) + (x%10), no output
%D     Reverse coding: x-2*(x%16), no output
```

For example, to go to line 3 column 12, a HP2645 terminal must get `\E&a12c03Y` padded for 6 milliseconds: `:cm=6\E&%r%2c%2Y:`

Capability `a1` adds an empty line before the line where the cursor is and leaves the cursor on the new line. This will always be done with the cursor at column 0. Capability `d1` deletes the line where the cursor is and is also done with the cursor at column 0. Capabilities `da` and `db` warn that off-screen lines may appear at the top or bottom of the screen upon scrolling or deleting lines. The *curses* routines do not use this feature, but do guard against its effects.

Insert-character operations usually affect only the current line and shift characters off the end of the line rigidly. Some terminals, such as the Concept 100, distinguish typed from untyped blanks on the screen, shifting upon insertion only up to an untyped blank, i.e. a space caused by cursor motion; these terminals have capability `in` (insert null).

Some terminals have an insertion mode; others require a special sequence to open up a blank position on the current line. Insertion mode is entered and left by `im` and `ie`, which should be null strings if there is no insertion mode. String `ic` is sent just before each character to be inserted, and padding `ip` is sent after. Capability `mi` says it is possible to move around without leaving insertion mode. Delete mode works similarly: enter with `dm`, leave with `de`, and issue `dc` before each character.

Highlighting, or 'standout' mode is entered by `so` and left by `se`. Underline mode is entered by `us` and left by `ue`. Terminals that underline characters individually have capability `uc`. The visual bell capability `vb` flashes the screen without moving the cursor.

A terminal with a keypad that transmits cursor motions may be described by capabilities `kl` `kr` `ku` `kd` `kh` that give the codes for left, right, up, down, and home. Up to ten function keys may be described by `k0`

through `k9`. Special labels for the function keys may be given as `l0` through `l9`.

The initialization string `is` is expected to set tabs if that is necessary. That string may come from a file if both are present `is` is done first.

The entry for a terminal may be continued by jumping to another entry given by `tc`. Duplicate capabilities are resolved in favor of the first.

**FILES**

`/etc/termcap`

**SEE ALSO**

[curses\(3\)](#), [termcap\(3\)](#), [vi\(1\)](#), [ul\(1\)](#)

**BUGS**

*Termcap* entries, including `tc` continuations, are limited to 1024 characters.

**NAME**

tp – DEC/mag tape formats

**DESCRIPTION**

*Tp* dumps files to and extracts files from DECTape and magtape. The formats of these tapes are the same except that magtapes have larger directories.

Block zero contains a copy of a stand-alone bootstrap program. See [reboot\(8\)](#).

Blocks 1 through 24 for DECTape (1 through 62 for magtape) contain a directory of the tape. There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

```
struct {
    char          pathname[32];
    unsigned short mode;
    char          uid;
    char          gid;
    char          unused1;
    char          size[3];
    long          modtime;
    unsigned short tapeaddr;
    char          unused2[16];
    unsigned short checksum;
};
```

The path name entry is the path name of the file when put on the tape. If the pathname starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, size and time modified are the same as described under i-nodes (see file system [filsys\(5\)](#)). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies  $(size+511)/512$  blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks above 25 (resp. 63) are available for file storage.

A fake entry has a size of zero.

**SEE ALSO**

[filsys\(5\)](#), [tp\(1\)](#)

**BUGS**

The *pathname*, *uid*, *gid*, and *size* fields are too small.

**NAME**

troff – device-independent output

**DESCRIPTION**

*Troff(1)* produces an ASCII representation of a typeset document, expressed in the following syntax. Strings inside [ ] are optional. The string `\n` represents newline. White space (spaces or newlines) may occur between commands and is sometimes necessary to terminate numbers.

- sn** Set point size to *n*.
- fn** Use font in position *n*. Normally fonts are mounted starting at position 1; 0 is reserved. *troff*.
- cx** Place character *x* at the current location on the page; *x* is a single ASCII character.
- Cname** Place special character. The *name* of the character is delimited by white space.
- Hn** Go to horizontal location *n*, expressed in basic units.
- hn** Add *n* to the current horizontal location (relative goto).
- Vn** Go to vertical location *n*, measured positive downward.
- vn** Add *n* to the current vertical location.
- mnx** A two-digit number followed by an ASCII character; equivalent to **hmc***x*.
- nb a** End of line. No action is required; *troff* will explicitly reset the location. Number *b* is the amount of space before the line, *a*, the amount of space after the line.
- w** A **w** appears between words of the input document. No action is required.
- pn** Begin a new page with page number *n*. The vertical location on the page becomes 0.
- # ... \n** Comment.
- Dl x y \n** Draw a line from the current location by *x,y*.
- Dc d \n** Draw a circle of diameter *d* with the leftmost edge at the current location, *x,y*. The current location becomes *x+d,y*.
- De dx dy \n** Draw an ellipse with *x*-axis *dx* and *y*-axis *dy*. The leftmost edge of the ellipse will be at the current location. The current location becomes *x+dx,y*.
- Da x y u v \n** Draw an arc counterclockwise from the current location to *x+u, y+v*, with center offset *x,y* from the current location. The end of the arc becomes the current location.
- D x y x y ... \n** Draw a spline curve (wiggly line) from the current location, moving by *x,y* each time. The end of the curve becomes the current location.
- x i[nit] \n** Initialize the typesetting device. The actions required depend on the device.
- x T dest \n** The name of the typesetter is *dest*, as in option **-T** of *troff(1)*.
- x r[es] n h v \n** The resolution of the typesetting device is *n* units per inch. Horizontal motions must be multiples of *h* units, vertical motions *v* units.
- x p[ause] \n** Pause. Cause the current page to finish but do not relinquish the typesetter.
- x s[top] \n** Stop. Cause the current page to finish and then relinquish the typesetter.
- x t[railer] \n** Generate a trailer if necessary.
- x f[ont] n name \n** Load font *name* into position *n*.
- x H[eight] n \n** Set the character height to *n* points. This causes the letters to be elongated or shortened. It does not affect the width of a letter. Not all typesetters can do this.
- x S[lant] n \n** Set the slant to *n* degrees, if possible.

`x...\n` Arbitrary; may be used for device-specific functions.

**SEE ALSO**

[troff\(1\)](#), [d202\(1\)](#), [apsend\(1\)](#), [lp\(1\)](#), [proof\(9\)](#)

B. W. Kernighan, *A Typesetter-Independent Troff* this manual, volume 2.

**NAME**

ttys – terminal initialization data

**DESCRIPTION**

The file `/etc/ttys` directs [init\(8\)](#) in associating login processes with terminal ports. It contains one line per port.

If the first character of a line is 0 the line will be ignored; if it is 1 the line will be effective. The second character is used as an argument to [getty\(8\)](#), which performs such tasks as baud-rate recognition, reading the login name, and calling [login\(8\)](#). For normal lines, the character is 0. Other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics; see [getty\(8\)](#) for a list. The remainder of the line is the terminal's entry in the device directory,

**FILES**

`/etc/ttys`

**SEE ALSO**

[init\(8\)](#), [getty\(8\)](#), [login\(8\)](#)

**NAME**

ttytype – data base of terminal types by port

**SYNOPSIS**

/etc/ttytype

**DESCRIPTION**

*Ttytype* is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in [termcap\(5\)](#)), a space, and the name of the tty, minus `/dev/`.

This information is read by [tset\(1\)](#) and by [login\(1\)](#) to initialize the TERM variable at login time.

**SEE ALSO**

[tset\(1\)](#), [login\(1\)](#)

**BUGS**

Some lines are merely known as “dialup” or “plugboard”.

**NAME**

types – primitive system data types

**SYNOPSIS**

```
#include <sys/types.h>
```

**DESCRIPTION**

The data types defined in the include file are used in the operating system. Some data of these types are useful in user code.

```
typedef long          daddr_t;      disk block number, see filsys(5)
typedef char *       caddr_t;      general memory pointer
typedef unsigned short ino_t;       inode number, filsys(5)
typedef long         size_t;        file size, stat(2)
typedef long         time_t;        time, time(2)
typedef unsigned short dev_t;       device code, stat(2)
typedef long         off_t;         file offset, lseek(2)
```

The following macros analyze and synthesize device numbers; see [intro\(4\)](#).

```
#define major(x)      (((int)(((unsigned)(x)>>8)&0377))
#define minor(x)      (((int)((x)&0377))
#define makedev(x,y)  ((dev_t)(((x)<<8) | (y)))
```

The file contains other definitions as well, internal to the system or specific to particular system calls. Pages in section 2 tell which calls need `<sys/types.h>`.

**SEE ALSO**

[filsys\(5\)](#), [time\(2\)](#), [intro\(4\)](#)



**NAME**

utmp, wtmp – login records

**SYNOPSIS**

```
#include <utmp.h>
```

**DESCRIPTION**

The *utmp* file allows one to discover information about who is currently logged in. The file is a sequence of entries with the following structure declared in the include file:

```
struct utmp {
    char  ut_line[8]; /* tty name */
    char  ut_name[8]; /* user id */
    long  ut_time;   /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of *time(2)*.

The *wtmp* file records logins and logouts. Its format is exactly like *utmp* except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names | and } indicate the system-maintained time just before and just after a *date(1)* command changed the system's idea of the time.

*Wtmp* is maintained by *login(8)* and *init(8)*. Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by *ac(8)*.

**FILES**

```
/etc/utmp
/usr/adm/wtmp
```

**SEE ALSO**

*login(8)*, *init(8)*, *who(1)*, *ac(8)*

**NAME**

uuencode – format of an encoded uuencode file

**DESCRIPTION**

Files output by *uuencode(1)* consist of a header line, followed by a number of body lines, and a trailer line. *Uudecode(1)* will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters “begin ”. The word *begin* is followed by a mode (in octal), and a string which names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing newline). These consist of a character count, followed by encoded characters, followed by a newline. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of “end” on a line by itself.

**SEE ALSO**

uuencode(1), uusend(1), uucp(1), mail(1)

**NAME**

vfont – font formats for the Benson-Varian or Versatec

**SYNOPSIS**

`/usr/lib/vfont/*`

**DESCRIPTION**

The fonts for the printer/plotters have the following format. Each file contains a header, an array of 256 character description structures, and then the bit maps for the characters themselves. The header has the following format:

```
struct header {
    short      magic;
    unsigned short size;
    short      maxx;
    short      maxy;
    short      xtnd;
} header;
```

The *magic* number is 0436 (octal). The *maxx*, *maxy*, and *xtnd* fields are not used at the current time. *Maxx* and *maxy* are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. The *size* is the size of the bit maps for the characters in bytes. Before the maps for the characters is an array of 256 structures for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
    unsigned short addr;
    short      nbytes;
    char      up;
    char      down;
    char      left;
    char      right;
    short     width;
};
```

The *nbytes* field is nonzero for characters which actually exist. For such characters, the *addr* field is an offset into the rest of the file where the data for that character begins. There are *up+down* rows of data for each character, each of which has *left+right* bits, rounded up to a number of bytes. The *width* field is not used by *vcat*, although it is used by *vwidth(1)* to make width tables for *troff*. It represents the logical width of the glyph, in raster lines, and shows where the base point of the next glyph would be.

**FILES**

`/usr/lib/vfont/*`

**SEE ALSO**

*troff(1)*, *pti(1)*, *vpr(1)*, *vtroff(1)*, *vwidth(1)*, *vfontinfo(1)*, *fed(1)*

**NAME**

view2d – movie of a function  $f(x, y, t)$

**DESCRIPTION**

Files of this format are produced by functions in [view2d\(3\)](#), and displayed by commands in [view2d\(1\)](#). A movie file consists of one or more frames, each consisting of a header and a sequence of 16-bit signed integer values for each pixel, scanned left to right and bottom to top. (Left-to-right is the inner loop.)

The header consists of the 32-bit magic number 0135246, then eight 16-bit integers:

*VER* The version number.

*NX, NY*

The number of pixels in the frame. These may not vary from frame to frame.

*u, v* relate pixel values  $p$  in the file to user function values  $f$  by

$$p = u + f \times 2^{-v}.$$

*FIXUV*

normally 0; 1 if  $u, v, PMIN, PMAX$  of first frame give a bound on the data in the entire file.

*PMIN, PMAX*

limits of the data; only used when  $FIXUV=1$ .

and finally a 16-byte ASCII representation of a floating point value:

*TIME* is a frame index, typically set to simulated time or to an iteration counter. This need not be uniformly spaced from frame to frame, but should be nondecreasing.

The range of displayable pixel values is  $[-32765, 32765]$ . Values below this range are deemed out of bounds and not plotted; values above are reserved.

**SEE ALSO**

[view2d\(1\)](#), [view2d\(3\)](#)

**NAME**

whoami – computer name

**DESCRIPTION**

The file `/etc/whoami` contains one line of information – the name of the computer, as used in [mail\(1\)](#) and [uucp\(1\)](#).

**FILES**

`/etc/whoami`

**NAME**

worm – format of worm disks

**SYNOPSIS**

```
#include <worm.h>
```

**DESCRIPTION**

A WORM disk is a linked list of ‘superblocks’, roughly one for every *worm write* on the WORM. The ‘governing’ superblock is at block zero if it exists and has a valid magic number; otherwise the governing superblock is the last superblock in the linked list starting at block 1. (The link to the next superblock is preallocated and thus the last superblock in the list will be unwritten.) Each superblock has some status information and a pointer to a set of ‘inodes’ describing a set of files. The status information for the WORM is that of the governing superblock; the set of files on the WORM is the accumulation of all the superblocks taken in order. The structure of a superblock as given in the include file is:

```
#define SMAGIC    0x21746967
#define VLINK     1          /* linked list superblock */
#define VBTREE   2          /* cbt superblock */
typedef struct superblock
{
    long magic;          /* magic number for superblock */
    unsigned short blocksize; /* physical size of blocks */
    short version;      /* type of superblock */
    long nblocks;       /* number of blocks on device */
    long zero;          /* first logical data block */
    long nfree;         /* number of free blocks */
    long ninodes;       /* number of inodes */
    long ninochars;     /* number of bytes of inode names */
    long binodes;       /* start of inodes */
    long nextffree;     /* next free file block */
    long nextsb;        /* next superblock */
    short fd;           /* fildes for device (in core) */
    char vol_id[128];   /* name the disk can be mounted as */
    char comment[128]; /* comments */
    long myblock;       /* where this superblock is */
    long nF;            /* bytes for .F (VBTREE) */
    long nT;            /* bytes for .T (VBTREE) */
    long ctime;         /* create time for this superblock */
} superblock;
```

superblocks are padded with zeros to **blocksize**.

Following each **VLINK** superblock is a set of inodes, a string table, and then the data blocks for the files described by the inodes. Following a (there is at most one) **VBTREE** superblock there is a set of inodes, a string table and the **.F** and **.T** files for a *cbt(1)* database where the keys are filenames and the data is an inode number.

```
#define DMAGIC    0x3A746967
typedef struct Inode
{
    long magic;          /* magic number for Dirent */
    long block;          /* starting block of file */
    long nbytes;         /* bytes in file */
    long ctime;         /* creation time */
    union {
        char *n;        /* core - name */
        long o;         /* disk - offset into chars block */
    } name;             /* filename */
    long pad1           /* to 32 bytes */
    short mode          /* as in stat(2) */
    short uid           /* owner */
}
```

```
        short gid          /* owner */
        short pad2        /* to 32 bytes */
    } Inode
```

If the **block** field of an inode is negative, the file has been deleted.

**SEE ALSO**

[worm\(8\)](#)

**NAME**

prtx – format of prtx commands

**DESCRIPTION**

Commands are in ascii characters and are terminated by either a ";" or a newline character. Besides commands there are "prefixes" that modify the following a command.

Most commands and prefixes take arguments. These are often numbers or point positions. The latter being a pair of numbers. Numbers can be arbitrary arithmetic expressions involving "+", "-", "\*", "/" , numeric constants, and parenthesis. A numeric constant is a decimal number (optionally including a decimal point and fraction) followed by a scale. A scale is "i", for inches, "c" for characters, or "d" for dots. (A dot is the smallest resolution on the printer.) If the scale is omitted "c" is assumed. Constants used in multiplication or division should omit the scale.

Point positions are specified by a row position and a column position, separated by white space (blanks or tabs). The upper left hand corner of the page is 0 0. Positions increase to the right and downward. If a page contains 60 characters vertically and 80 horizontally. the last horizontal dot is at 80c-1d and the last vertical dot is at 60c-1d.

The commands are:

# comment

This command has no effect. Note that it is terminated by a ";" just like any other command.

line *point1 point2 ...*

Line Command: Straight lines are drawn starting at *point1* going to *point2* and then to further points. At least two points must be given but there is no maximum. Points may be separated by a "," for readability.

interp *point1 point2 ...*

Interpolate command: A curved line is drawn determined by the points. A point may be surrounded by curly braces ( '{' and '}' ) in which case it is a "guide" point. The curve is a second degree spline that satisfies the following constraints:

It passes through all normal points.

If two guide points are adjacent in the list it passes through the point midway between them and is tangent to the line connecting them.

If a normal point is adjacent in the list to a guide point the curve as it passes through the normal point is tangent to the line connecting it to the guide.

If a normal point is not adjacent on the list to a guide point the curve, as it passes through the normal point, has the same tangent as the circle that goes through that point and two neighboring points on the list. This will be the preceding and the following points for a point in the middle of the list.

Unfortunately these rules can result in cusps. They seem to work best in the cases where there are three normal points (an approximation to a circular segment) or in the case where guide points are used to connect two normal points.

ellipse *center radius*

Ellipse Command: An ellipse is drawn centered at *center*, with its shape determined by *radius*. The ellipse's axes are parallel to the picture's. *radius* is two expressions. The first determines the "vertical radius" (i.e. half the length of the ellipse's vertical axis) and the second determines the "horizontal radius". If they are equal (as distances, not as number of dots or characters) the ellipse is a circle.

text *point text*

Text Command: Characters specified by *text* are placed on the output starting at *point*. (The position is the upper left hand corner of the first character.) *text* begins at the first non blank character following the point and continues to the end of the line. Embedded ";" are allowed, and "C" type escapes are processed. An escaped newline (i.e. one immediately preceded by a backslash) is treated as text and does not end the command. A leading ":" is stripped off. This is necessary if the first character desired is a space or arithmetic operator.



**stext point point text**

Slanted text command: Text is placed in the output as with the simple text command except the base line (the line running across the top of the characters) is the line through the two points. The text starts at the first point and may proceed any distance towards or beyond the second point. The second point provides a direction only, the characters are not stretched or compressed.

**mtext point terminator**

Multi line text command: Lines following the one containing the command upto (but not including) the first line that begins with the terminator character (default '.') are put in the output, with the first line positioned at *point* and the following lines positioned vertically below the first.

**smtext point [point] [ terminator ] [text ]**

General form of the text command: The modifiers "s", "m", and "N" may be combined. If both "s" and "m" are present "s" must come first. The second point is required present if "s" is present and "terminator" is required if "m" is present. Otherwise they must be omitted. If "m" is omitted then text must be present.

**\$X [ point ]**

Macro Command: Invoke macro *X*. The the body of macro *X* is executed with an offset of *point*. That is, positions in the body of the macro are taken as relative to *point*. If *point* is omitted it is take to be the origin. The color and solidity in effect at the point invocation are used in drawing unless they are explicitly overridden by a prefix within the macro body. An invoke command is legal within a macro body but care should be taken to avoid a macro invoking itself.

Prefixes modify the following command, which is separated from the prefix by white space only. However, commands can be grouped with braces (i.e the characters '{' and '}'). If the opening brace is in the same line as the prefix, commands may be on different lines, and the prefix applies to all commands within the braces ; Braces can be nested. More than one prefix can modify a single command The following prefixes are defined:

**at point** Translation Prefixes: Locations in the prefixed command are taken as relative to the argument of *at*.

**expand factor**

Expand Prefix: The prefix command is expanded about the origin by *factor*, which is an unscaled numeric expression. That is, all positions and shapes are multiplied by *factor*.

**rotate angle**

Rotation Prefix: Locations in the prefixed command are rotated about the origin by *angle* degrees. Text will still be horizontal, unless created with an *stext* command.

**size N**

Character Size Prefix: Characters in the prefixed command have their size multiplied by *N*. *N* must be an integer.

**style solidity**

Line Style Prefix: The line drawing commands (line, ellipse, interp) will normally draw solid lines but this can be changed by a *style* prefix. Solidity is one of the following literals: solid, longdashed, dashed, dotdashed, dotted, invisible. Invisible lines are not drawn at all, so this style is useful in taking up space in connection with the stack and shelf prefixes.

**color colorname**

Color prefix: Every thing drawn by the following command will be in the named color. Legal colors are: yellow, orange, red, green, blue, violet. If the output device does not have colors this prefix is ignored.

**boxit space**

Boxit Prefix: Space consists of two numeric expressions, a row and a column spacing. Boxit draws the prefixed command and an enclosing box. The box is drawn to leave the specified spaces at the top and bottom (row space), and sides (column space) around the prefixed command.

- define X** Define macro: *X* is any alphabetic character (lower and upper case are considered identical) and the macro is defined as equivalent to the following command. The prefixed command, which must be a block (i.e. surrounded by braces), becomes the body of the macro and is not executed until the macro is invoked. Once defined a macro character may not be redefined. More elaborate macros can be obtained using a general macro processor such as *m4* and the general arithmetic expressions provided by *prtx*.
- stack** Stack commands: The prefixed command must be a block. Each command of the block is interpreted as if the origin had been moved to below the earlier commands. The effect is what would be achieved if each command were prefixed by "at max 0", where max is the maximum row contained in previous commands. A row is contained in a command if the command puts a dot in the row. Text commands also contain some blank space below the characters. (In some circumstances, e.g. slanted text, *prtx* cannot easily compute the contained rows and makes a guess. But, arbitrary lines, and horizontal text work properly.)
- shelf** Lay commands left to right. This is like the stack prefix except commands are moved to the right rather than down.

## **HISTORY**

The current version of *prtx* also accepts an old form of the command language. It is compatibility with this old form that makes the macro prefix require the prefixed command to be a block. In the future this restriction will disappear.

## **FUTURE PLANS**

### **SEE ALSO**

*prtx(1)*

### **AUTHOR**

Jerry Schwarz (harpo!jerry)

**NAME**

prtxfont – character sets for prtx

**DESCRIPTION**

A font file describes for each character the dots that *prtx* should use to construct that character. Any character not described in the file is treated as a blank. Only characters 0 through 127 are allowed.

The first line of a file contains an integer NR giving the number of rows of dots in each character.

Except for the first each line contains a character designator followed by the description of that character. The character designator is either the character itself or "0 where n is the octal representation of the character.

A character description consists of NR strings. Each string consisting of some number of "dot descriptions", and separated from the preceding string or the character designator by spaces. Conceptually a character is formed by placing the strings on top of each, left aligned, with the first on top. For normal size characters each dot description corresponds to a potentially printed dot. For larger characters each description corresponds to several dots and indicates a pattern for the dots.

The possible dot descriptions are:

1            Fill in the entire area. This is the only description that causes a dot to be printed at the normal size.

Leave the entire area blank

a            Fill in the upper right hand corner

b            Fill in the lower right hand corner

c            Fill in the lower left hand corner

d            Fill in the upper left hand corner

A            Fill in the the area but leave a wedge blank on the left hand side. The descriptions with omitted wedges are useful for constructing pointed characters. E.g. 'A' might be used at the right hand part of a '>'

B            Fill in the area but leave a wedge blank at the top.

C            Fill in the area but leave a wedge blank at the right.

D            Fill in the area but leave a wedge blank at the bottom.

**AUTHOR**

Jerry Schwarz (harpo!jerry)

**NAME**

adventure, zork, rogue, wump – dungeon-exploration games

**SYNOPSIS**

**/usr/games/adventure**

**/usr/games/zork**

**/usr/games/rogue**

**/usr/games/wump**

**DESCRIPTION**

*Adventure* is the granddaddy of dungeon-exploration games, part of the object of which is to puzzle out the object and the rules. *Zork* marks the zenith of the genre.

*Rogue* requires a cursor-addressed terminal to show fragments of the cave map. Indicia published at the bottom of the screen are the cave level being explored, the amount of gold accumulated, armor class, and measures of your potency: ‘hit points’, strength, and experience level. Type ? for more help.

The *wump* cave is inhabited by a Wumpus and by Super Bats that like to pick you up and drop you somewhere else. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits.

**FILES**

adv.susp

rogue.save

/usr/games/lib/rogue\_roll

**SEE ALSO**

**readnews -n net.games.rogue**

**BUGS**

You take these games where and as you find them.

**NAME**

arithmetic – drill in number facts

**SYNOPSIS**

`/usr/games/arithmetic [ +-x/ ] [ range ]`

**DESCRIPTION**

*Arithmetic* types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back `Right!`, and a new problem. If the answer is wrong, it replies `What?` and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; `+x/` get addition, subtraction, multiplication, and division problems respectively. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is `+-`

*Range* is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

**NAME**

ascii, latin1 – character set maps

**DESCRIPTION**

The file `/usr/pub/ascii` is a map of the ASCII character set, to be printed as needed. It contains:

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dc1	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 \$	045 %	046 &	047 '
050 (	051 )	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135 ]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174	175 }	176	177 del

The file `/usr/pub/latin1` is a map of characters 0200-0377 in the ISO Latin-1 extension to the ASCII code. Escape sequences are given for typing the characters in [mux\(9\)](#)

**FILES**

`/usr/pub/ascii`  
`/usr/pub/latin1`

**NAME**

atc – air traffic controller

**SYNOPSIS**

*/usr/games/atc*

**DESCRIPTION**

*Atc* presents air traffic on a cursor-controlled screen. As the controller, you must shepherd it safely through the air space. At the beginning of the game *atc* displays the takeoff/landing direction for each airport and prompts for the game duration with: < >. Enter a number from 16 simulated minutes (hard) to 99 (easier)

Options are

**-u=***file*

Take airspace description from *file*.

**-a=***name*

use the named airspace; default is `Apple1`.

**-s=***seed*

for a 32-bit random number generator

**-t=***time*

Preset the game duration.

**-p=***file*

save the play of the game in the named file

**-m=***file*

play a ‘movie’ of the saved game

In the display of the airspace **%** and **#** denote airports; **\*** and **!** denote navigational aids (navaids); and commas denote airways that link numbered entry/exit ‘fixes’, airports, and navaids. Dots are separated by one mile, horizontally, vertically, and diagonally. An airplane appears as a letter followed by its height in thousands of feet.

There are two kinds of planes: jets flying 1 mile per tick (15 seconds) and props flying 1/2 mile per tick.

You must prevent various misfortunes. Running out of fuel is serious. So is a close encounter – less than 3 miles horizontal separation at a given altitude. A plane changing altitude is considered to be at both its old and new altitudes. A ‘boundary error’, leaving the airspace at the wrong place, not on an airway, or at the wrong height, is also serious, but not as likely to be fatal.

The right side of the screen shows flight plans. A typical flight strip looks like:

**Fj 7->3 4 NE +**

The first letter is the aircraft name, the next letter is **j** for jet or **p** for prop. The next field gives the plane’s intentions: this one is entering at (or is now at) fix 7 and leaving at fix 3. The origin character tells where the plane is (or will be when it enters), the destination is a fix it wishes to go to. (It will, however, continue on a straight path unless instructed otherwise.) Next is the altitude, in this case 4000 feet. The bearing is a compass direction: N, NE, etc. The final character is the amount of fuel left, **+** for more than 10 minutes, otherwise the number of minutes of fuel remaining. Jets begin with 15 minutes of fuel, props 21.

At the top of the flight plans are listed planes that will appear in the next minute, preceded by how many ticks (0-4) they are away. Planes may be cleared for takeoff as soon as they are listed.

Commands are terminated by newline. Backspace may be used to correct errors. The following kinds of commands can be issued.

**\$**

End the game (game normally ends after 26 planes)

**W** Print flight plan for airplane **W**

**XA3** **X** will change altitude to 3000 feet

**QA0** **Q** will land (go to 0 feet altitude)

**HRE** **H** will turn right until it is heading east

**ALNW**

**A** will turn left until it is heading northwest

**CTS** C will turn south through the smallest angle  
**T\*7** T will take exit bearing for fix 7 at next navaid  
**P\*%** P will take landing bearing for % at next navaid  
**DH** D will circle (hold) at next navaid  
**MR0** Abort pending hold, clearance, or turn for plane M  
**J?** Cancel delayed commands for J  
 space Speed up the game by advancing 15 seconds

*Climbing/descending.* Planes climb or descend 1000 feet per mile. Climbing from 0 is a takeoff; descending to 0 is a landing. The takeoff/landing direction for each airport is given. A landing airplane must reach altitude 0 headed in the right direction 1 mile before the runway. No further commands may be given after a descent to 0, as control then rests with the tower. If a plane lands from the wrong direction, it will climb to 1000 feet and issue a 'go around' error. While changing altitude, a flight strip reads like

**Dp :->2 7v3 S 9**

which means at 7000 feet descending to 3000.

*Turning.* Planes turn 45 degrees per mile. Turns may be left L right R or to a specified direction T. Thus ULNE tells plane U to turn to his left until it is heading northeast. Changes of direction are indicated in the flight strip:

**Nj :->5 5 S r W +**

indicates that jet N is heading south, and will turn 90 degrees to the right. To cancel the remaining part of this turn, give the command N) NR0.

*Nav aids.* A plane may be directed to turn at a navaid or hold (circle) there, Thus command 'AH' holds plane A at the next navaid. The flight strip for a plane that is to hold looks like

**Ap :->2 5 S \* 7**

During the hold, the \* will become h. Every incoming plane that will be landing holds at a navaid unless the controller gives it other instructions.

The command \* clears a plane to turn sharply to any known fix at the next navaid. The flight strip for a plane cleared through a navaid (to fix 5, for example) looks like:

**Hj .->2 5 S \*5 +**

A holding aircraft given a clearance will continue around to the navaid, then immediately assume the specified bearing. Turns cancel clearances.

*Delayed commands.* Commands of the form

@location,command[,command ...]

stack up activities. A location may be any fix or a point offset from a fix, e.g.

@#sw3s2,ARE

which means at the point which can be reached by going three miles SW from airport #, then two miles S, plane A should begin a right turn until heading E. The information command shows all delayed commands pending for that plane. Note that delayed commands allow one to specify actions more than one navaid ahead.



**NAME**

back – backgammon

**SYNOPSIS**

**/usr/games/back**

**DESCRIPTION**

This program does what you expect. It will ask whether you need instructions.

**NAME**

banner, rot, rnd, bigp – print in large type

**SYNOPSIS**

`/usr/games/banner [ -font ] text [ | /usr/games/rot | /usr/games/rnd height width ]`

`/usr/games/bigp [ text ]`

**DESCRIPTION**

*Banner* prints its arguments, one per line, in

```

#                                     #
#                                     #
#   ### # ##  ## #   ###           ##### #  # # ##   ###
#         # ## # #  ## #   #       #  #  # ## # #  #
#   ##### #   #  # #####          #  #  #  #  # #####
#  #  # # #   #  #  #           # # # ## #  #  # ##
#   ### #   #####   ##          ##  ## # #####   ## ##
#                                     #
#                                     #
#                                     #####

```

The *font* may be selected from see [font\(9\)](#)

*Rot* rotates its input clockwise ninety degrees.

```
banner Sideways | rot
```

runs down the page, and

```
banner Upside Down | rot | rot
```

is disconcerting.

*Rnd* scales the non-white-space characters in its input by integral *height* and *width*.

*Bigp* generates banners suitable for printing on a line-printer. Its output is approximately the same as

```
banner -defont text | rot | rnd 2 4
```

If no arguments are supplied, *bigp* creates a banner from the standard input.

**NAME**

bcd, ppt, morse – convert to antique media

**SYNOPSIS**

*/usr/games/bcd text*

*/usr/games/ppt*

*/usr/games/morse*

**DESCRIPTION**

*Bcd* converts the literal *text* into a tangible form familiar to old-timers.

*Ppt* converts the standard input into yet another old standard.

*Morse* converts the standard input into a pronounceable two-symbol code.

**SEE ALSO**

[dd\(1\)](#)

**NAME**

bianchi – espresso, steamed milk, hot water

**SYNOPSIS**

**make mess**

**DESCRIPTION**

This machine makes espresso, hot water, and steamed milk. It is attached to the cold water supply. The shut off valve is actuated by the yellow handle under the sink.

The power switch is a two position switch on the lower left portion of the front face of the cabinet. At the 0 position, the machine is powered off.

The push button switch near the center of the upper front face enables water to flow through the coffee receptacles. It does not automatically shut off.

The handle on the right side of the base enables cold water to enter the machine should the pump fail. It should not normally be used.

Users accustomed to the previous machines should be warned that this one requires considerably less coffee per cup. The bayonet mounted ground coffee holders will be very hard to attach if overfilled.

**WARNING!**

The steam and hot water supplies are both extremely hot and capable of high pressure. Turn them on slowly. The unwary user may suffer loss of skin or eyesight.

**BUGS**

When first powered on it takes 15 minutes to warm up. The last person leaving at night should turn the machine off.

**NAME**

boggle, hangman, scrabble – word games

**SYNOPSIS**

**/usr/games/boggle** [ + ] [ ++ ] [ *word word word word* ]

**/usr/games/scrabble**

**/usr/games/hangman** [ *arg* ]

**DESCRIPTION**

*Boggle* provides practice for the Parker Brothers game. If invoked with 4 arguments of 4 letters each, the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If invoked without arguments, it will generate a board, let you enter words for 3 minutes, and then tell how well you did relative to **/usr/dict/words**. Words may be formed from any sequence of 3 or more adjacent letters in the grid. Letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

Enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting interrupt. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Option + removes the restriction that positions can only be used once in each word. Option ++ causes a position to be considered adjacent to itself as well as to its (at most) 8 neighbors.

*Scrabble* plays the Selchow and Righter game on a cursor-addressed terminal against a single opponent. To place a letter first move the cursor by typing 2, 4, 6, 8 for down, left, right, up respectively (1, 3, 7, 9 are diagonals) then type the letter in place. Type ? to cycle through a set of helpful tables.

*Hangman* chooses a word at least seven letters long from a dictionary. You then guess letters one at a time.

The optional argument *arg* names an alternate dictionary. The special name -a gets a particular very large dictionary.

**FILES**

/usr/dict/words

/usr/dict/web2

alternate dictionary for hangman

**BUGS**

*Hangman* runs hyphenated compounds together.

*Scrabble* rubs in its brilliance with merciless play and an inhumane interface.

**NAME**

bridge – card game

**SYNOPSIS**

**/usr/games/bridge** [ *arg ...* ]

**DESCRIPTION**

*Bridge* manages bridge games among four players. A master process mediates the flow of information between player processes. Each player process is either a ‘robot’ player or a cursor-controlled screen interface with a human player.

If several humans wish to play each invokes *bridge* and a rendezvous protocol hooks them together in a common game. Once the game is set up *bridge* displays a diagram similar to those in newspaper bridge columns.

Bids are coded **p** for pass, **d** for double, **3n** for three notrump, and so on. Plays are coded **c3** for the club three, **ht** for the heart ten, and so on. A menu of common commands appears at the bottom of the screen; further help may be obtained by typing +.

Arguments take several forms:

**-h** *nhumans*

**humans**=*nhumans*

Join (or set up and join) a game with indicated number of human players.

**-s** *seed*

**seed**=*seed*

Initialize the random number generator. Useful for duplicate play.

**-f** *file*

**deck**=*file*

Take the initial shuffled card deck from named file.

**-r** *file*

**script**=*file*

Make a record of the game in the named file.

**-d** *dealership*

**dlr**=*dealership*

Specify dealership with a one-letter direction code.

**-v** *vulnerability*

**vuln**=*vulnerability*

Specify vulnerability with one of these codes: **none both n-s e-w**.

**-t**

**tough** Do not display bidding history on player’s screen.

**FILES**

rendezvous files

/etc/termcap

DEBUG

recipient of debugging messages

**SEE ALSO**

J. A. Reeds and L. A. Shepp, *Bridge: An exciting new card game*, TM 11217-840119-02, TM 11218-840119-01.

**BUGS**

Occasionally the whole program goes dead.

The robots’ bridge technique has subtle bugs.

**NAME**

canfield, fish – card games

**SYNOPSIS**

**/usr/games/fish**

**/usr/games/canfield**

**DESCRIPTION**

The object of the children's card game *fish* is to accumulate 'books' of 4 cards of equal rank. At each turn one player selects a card from his hand, and asks the other player for all cards of that rank. If the other player has some, he hands them all over and the first player makes another request. The turn ends when the second player has no card of the rank requested; he replies, 'Go fish!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again.

The ranks are called a, 2, ..., 10, j, q, k. Hitting return requests information about the state of the game. Typing p as a first guess gets 'pro' level play.

*Canfield* is a solitaire game. It requires a cursor-addressed terminal.

**NAME**

chess – board game

**SYNOPSIS**

`/usr/games/chess`

**DESCRIPTION**

*Chess* is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

**FILES**

`/usr/lib/book`                    opening 'book'

**DIAGNOSTICS**

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

**WARNING**

Over-use of this program will cause it to go away.

**BUGS**

Pawns may be promoted only to queens.



**NAME**

ching – the book of changes

**SYNOPSIS**

`/usr/games/ching` [ *hexagram* ]

**DESCRIPTION**

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams*, each symbolized by a particular arrangement of six straight --- and broken - - lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each hexagram consists of two major sections. The 'Judgement' relates specifically to the matter at hand (E.g., 'It furthers one to have somewhere to go.') while the 'Image' describes the general attributes of the hexagram and how they apply to one's own life ('Thus the superior man makes himself strong and untiring.').

When any of the lines have the values six or nine, they are moving lines; for each there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second hexagram (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow-stalks or tossed coins. The resulting hexagram will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, the Unix oracle simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with other indicia which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin-toss divination. The answer appears on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try *fortune*(1)

**SEE ALSO**

It furthers one to see the great man.

**DIAGNOSTICS**

The great prince issues commands,  
Founds states, vests families with fiefs.  
Inferior people should not be employed.

**BUGS**

Waiting in the mud  
Brings about the arrival of the enemy.  
  
If one is not extremely careful,  
Somebody may come up from behind and strike him.  
Misfortune.

**NAME**

doctor, tso – psychiatric consultation

**SYNOPSIS**

**/usr/games/doctor**

**/usr/games/tso**

**DESCRIPTION**

*Doctor* will understandingly explore most anything with you. Just type your thoughts followed by double carriage returns.

*Tso*, on the other hand, has a will of its own.

**NAME**

eqnchar – special character definitions for eqn

**SYNOPSIS**

**eqn** /usr/pub/eqnchar [ *file ...* ] | **troff** [ *option ...* ]

**neqn** /usr/pub/eqnchar [ *file ...* ] | **nroff** [ *option ...* ]

**DESCRIPTION**

*Eqnchar* contains *nroff* and *troff(1)* character definitions for constructing characters that are not available on standard fonts. These definitions are primarily intended for use with *neqn* and *eqn(1)*. It contains definitions for the following characters.

B	Times bold	BI	Times bold italic
CH	Chess (APS-5)	CS	ConstantWidth Slanted
CW	Constantwidth (ASCII)	CX	News Gothic condensed
GB	Greek bold (APS-5)	GR	Greek
GS	German Script (APS-5)	HB	Helvetica bold
HI	Helvetica italic	HK	Helvetica black
HX	Helvetica bold italic	M1	Universal [sic]
I	Times italic	M2	Universal
Math 2	Math 2	M3	Universal
Math 3	Math 3	OE	Old English
OK	Helvetica outline black	PA	Palatino
PB	Palatino bold	PX	Palatino bold italic
PI	Palatino italic	PO	Printout (ASCII constant)
S1	Times Roman	S	Special (math symbols)
SC	Script (APS-5)	SM	Stymie medium (APS-5)
TB	Techno bold (APS-5)	TX	Techno bold italic
US	USA state maps	X1	Universal Newspaper

defont Default *mux* font

**FILES**

/usr/lib/font

/usr/lib/font/dev202/DESC.out  
description of 202 typesetter

/usr/lib/font/dev202/R.out  
tables for font R

/usr/jerq/font

**SEE ALSO**

*troff(1)*, *jf(9)*, *font(9)*

**NAME**

say – proverbs

**SYNOPSIS**

`/usr/games/say` [ *N* ]

**DESCRIPTION**

*Say* constructs *N* proverbs out of old parts. *N*=1 by default.

**NAME**

hangman, ana, word\_clout – word games

**SYNOPSIS**

**/usr/games/hangman** [ *arg* ]

**/usr/games/ana** [ *n* ]

**/usr/games/word\_clout**

**DESCRIPTION**

*Hangman* chooses a word at least seven letters long from a dictionary. You then guess letters one at a time.

The optional argument *arg* names an alternate dictionary. The special name `-a` gets a very large dictionary.

*Ana* reads words, one per line, from standard input and prints anagrams on standard output. The number *n*, which also may be given in standard input, limits the number of words in the anagrams.

*Word\_clout* traces connections in a thesaurus to find just words. Need to express unpleasant feelings so they seem auspicious? *Word\_clout* suggests calling them ‘warm’. The program will give instructions in its uses, which include service as a thesaurus.

**FILES**

/usr/dict/words

/usr/dict/web2

alternate dictionary for hangman /usr/lib/spell/aspell

**BUGS**

*Hangman* runs hyphenated compounds together.

**NAME**

`imp` – interactive mail program

**SYNOPSIS**

`/usr/games/imp`

**DESCRIPTION**

*Imp* attempts to avoid the complexity of using modern mail programs by asking for control information interactively rather than by expecting the user to supply it on the command line or by using defaults that are sometimes not exactly what the user has in mind. Thus, *imp* prompts for the userid of the mail target, the target's home machine, the userid and home machine of the sender, and the postmark. The message can be entered from the standard input as in [mail\(1\)](#) or taken from a file.

A particularly useful application of *imp* is to cause mail that you send from some borrowed account to appear as if it came from you on your home machine, thereby reducing the possibility of confusing the recipient.

**FILES**

`/dev/tty`

**SEE ALSO**

[mail\(1\)](#), [upas\(8\)](#)

**BUGS**

Probably. Try sending mail to yourself before using *imp* to send mail to others.

**NAME**

ipa – international phonetic alphabet font and preprocessor

**SYNOPSIS**

**ipa** [ *file ...* ]

**DESCRIPTION**

*Ipa* copies the named *files* to the standard output, translating text delimited in either two ways into *troff* character codes for IPA graphics:

**@ipa(...)**

**@ipa{...}**

To generate IPA characters in a table, *ipa* should come before *tbl(1)* in a pipeline of processes.

The following table shows the correspondence between ASCII characters and IPA graphics. Only phonemes that occur in American English are handled. The reference tells how to access other IPA graphics that exist in the font.

a	@ipa{a}	b	@ipa{b}	c	@ipa{c}	d	@ipa{d}	e	@ipa{e}
f	@ipa{f}	g	@ipa{g}	h	@ipa{h}	i	@ipa{i}	j	@ipa{j}
k	@ipa{k}	l	@ipa{l}	m	@ipa{m}	n	@ipa{n}	o	@ipa{o}
p	@ipa{p}	q	@ipa{q}	r	@ipa{r}	s	@ipa{s}	t	@ipa{t}
u	@ipa{u}	v	@ipa{v}	w	@ipa{w}	x	@ipa{x}	y	@ipa{y}
z	@ipa{z}	A	@ipa{A}	B	@ipa{B}	C	@ipa{C}	D	@ipa{D}
E	@ipa{E}	F	@ipa{F}	G	@ipa{G}	H	@ipa{H}	I	@ipa{I}
J	@ipa{J}	K	@ipa{K}	L	@ipa{L}	M	@ipa{M}	N	@ipa{N}
O	@ipa{O}	P	@ipa{P}	Q	@ipa{Q}	R	@ipa{R}	S	@ipa{S}
T	@ipa{T}	U	@ipa{U}	V	@ipa{V}	W	@ipa{W}	X	@ipa{X}
Y	@ipa{Y}	Z	@ipa{Z}	!	@ipa{!}	@	@ipa{@}	#	@ipa{#}
\$	@ipa{\$}	%	@ipa{%}	^	@ipa{^}	&	@ipa{&}	*	@ipa{*}
_	@ipa{_}	-	@ipa{-}	+	@ipa{+}	=	@ipa{=}	,	@ipa{,}
<	@ipa{<}	.	@ipa{.}	>	@ipa{>}	:	@ipa{:}	;	@ipa{;}
"	@ipa{"}	'	@ipa{'}		@ipa{ }		@ipa{ }	'	@ipa{'}
?	@ipa{?}								

*address.fc* (.)if t .ig

TABLE NOT PRINTABLE IN NROFF (.)SH SEE ALSO M. Y. Liberman, *An IPA Preprocessor for Troff*, 11225-860915-15TMS

*troff(1)*

**BUGS**

*Ipa* mounts the IPA fonts in *troff* font positions 5 and 6, which may conflict with other font assignments.

**NAME**

latex, slutex, bibtex – tex macro package and bibliographies

**SYNOPSIS**

**latex** *file* [.tex]

**slutex** *file* [.tex]

**bibtex** *auxname*

**DESCRIPTION**

*Latex* is a standard set of macros for [tex\(1\)](#) inspired by, but not identical to, Scribe. The command *latex file* processes *file.tex* and produces *file.dvi*, which should be printed with [lp\(1\)](#). It will probably be necessary to run *latex* twice, to get all of the cross-referencing done properly. *Latex* writes cross-referencing information in *file.aux*. *Slutex* is version of *latex* for making slides.

*Bibtex* reads the top-level auxiliary (**.aux**) file output by *latex* and creates a bibliography (**.bbl**) file to be included in the source file. The *auxname* on the command line should be given without an extension. Each **\cite** in the source file is looked up in bibliography files to gather together those used in the document. Then a bibliography style file is executed to write a **\thebibliography** environment.

The source file should have defined the bibliography (**.bib**) files to search with the **\bibliography** command, and the bibliography style (**.bst**) file to execute with the **\bibliographystyle** command. *Bibtex* searches the **TEXINPUTS** path (see [tex\(1\)](#)) for **.bst** files, and the **BIBINPUTS** path for **.bst** files. The manual describes how to make bibliography files.

See files in `/usr/lib/tex/macros/doc` for more documentation. In particular, `local.tex` is the *Local Guide* referred to in the manual.

**SEE ALSO**

Leslie Lamport, *LATEX: A Document Preparation System*, Addison Wesley, 1986  
Howard Trickey, *Latex User Guide*, this manual, Volume 2,  
[tex\(1\)](#), [lp\(1\)](#)



**NAME**

mail – mail addresses

**DESCRIPTION**

*Mail(1)* uses the programs of *upas(8)* to interpret mail addresses.

**Network addresses**

A general network mail address has the form *machine!...!name*, with one or more machines mentioned. A machine in the middle of the list gets the mail marked ‘from’ the preceding part of list and forwards it to the next to handle the rest of the list.

Rules for converting addresses among the conventions of different networks are given by rewrite rules; see *upas(8)*. A rough description of the rewrite rules for the local research network follows.

A simple name, containing no punctuation, is translated according to ‘Local addresses’ below to produce more addresses, which get rewritten in turn.

The conventional network address **local!name** is delivered to the mailbox **/usr/spool/mail/ name** if it exists or if *name* is registered as a login name in the password file *passwd(5)*. Otherwise the mail is undeliverable.

Mail to another machine is forwarded.

Addresses in other forms are rewritten recursively.

**Local addresses**

‘Alias files’ specify local name translation. Each line of an alias file begins with # (comment) or with a name. The rest of a name line gives the translation. The translation may contain multiple addresses and may be continued to another line by appending a backslash. Items are separated by white space.

In translating a name, the sender’s personal alias file is checked first. Then the system alias files, listed one per line in are checked in order. If the name is not found, the translation is taken to be **local!name**.

On research network machines, the first system alias file is **/usr/lib/upas/names.local**; it is never touched from afar. Alias files for various organizations, e.g **/usr/lib/upas/names.1127**, are maintained, often by users themselves, at selected sites and sent around the network when changed by *ship(8)*. The master alias file for center 1122 is kept on ‘alice’, those for other centers on ‘bowell’.

**Addresses to/from major networks**

A ‘from’ address is automatically supplied as a return postmark on outgoing mail addressed ‘to’ the several networks. Respondents should be able to send to these addresses. For non-research AT&T machines that use the research gateway, ‘from’ addresses with @ should be replaced by **person%machine@research.att.com**.

UUCP:

(to) **machine!person**  
(from) **research!person**

CSNET:

(to) **csnet!machine-domain-name!person**  
(from) **person@research.att.com**

ARPANET:

(to) **arpa!machine-domain-name!person**  
(from) **person@research.att.com**

ACSNET:

(to) **acsnet!machine-domain-name!person**  
(from) **person@research.usa**

BITNET:

(to) **bitnet!machine!person**  
(from) **person@research.att.com**

**FILES**

`/usr/lib/upas/namefiles`  
`/usr/lib/upas/names.*`

**SEE ALSO**

*uucp(1), mail(1), upas(8)*

**NAME**

man – macros to typeset manual

**SYNOPSIS**

**nroff -man** *file* ...

**troff -man** *file* ...

**DESCRIPTION**

These macros are used to lay out pages of this manual.

Except in `.LR` and `.RL` requests, any text argument denoted *t* in the request summary may be zero to six words. Quotes " ... " may be used to include blanks in a ‘word’. If *t* is empty, the special treatment is applied to the next text input line (the next line that doesn’t begin with dot). In this way, for example, `.I` may be used to italicize a line of more than 6 words, or `.SM` followed by `.B` to make small letters in ‘bold’ font.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

The fonts are

- R** roman, the main font, preferred for diagnostics
- I** italic, preferred for parameters, short names of commands (use **F** for full path names), names of manual pages, and naked function names
- B** ‘bold’, actually the constant width font `CW`, preferred for examples, declarations, keywords, names of **struct** members, and literals (numbers are rarely literals)
- F** also font `CW`; used for filenames to help cross-indexing
- L** also font `CW`. In *troff* **L=B**; in *nroff* arguments of the macros `.L`, `.LR`, and `.RL` are printed in quotes; preferred only where quotes really help (e.g. lower-case literals and punctuation).

Type font and size are reset to default values before each paragraph, and after processing font- or size-setting macros.

The **-man** macros admit equations and tables in the style of *eqn(1)* and *tbl(1)*, but do not support arguments on `.EQ` and `.TS` macros.

These strings are predefined by **-man**:

`\*R` ‘@’, ‘(Reg)’ in *nroff*.

`\*S` Change to default type size.

**FILES**

`/usr/lib/tmac/tmac.an`

`/usr/man/man0/xx`

**SEE ALSO**

*troff(1)*, *man(1)*

**REQUESTS**

Request	Cause	If no Break Argument	Explanation
<code>.B t</code>	no	<code>t=n.t.l.*</code>	Text <i>t</i> is ‘bold’.
<code>.BI t</code>	no	<code>t=n.t.l.</code>	Join words of <i>t</i> alternating bold and italic.
<code>.BR t</code>	no	<code>t=n.t.l.</code>	Join words of <i>t</i> alternating bold and Roman.
<code>.CT c x...</code>	no		Chapter <i>c</i> , topics <i>x</i> in topic index; see <code>/usr/man</code> for topic codes.
<code>.DT</code>	no		Restore default tabs.
<code>.EE</code>	yes		End displayed example
<code>.EX</code>	yes		Begin displayed example
<code>.F t</code>	no	<code>t=n.t.l.</code>	Text <i>t</i> is filename.
<code>.FR t</code>	no	<code>t=n.t.l.</code>	Join words of <i>t</i> alternating filename and Roman.
<code>.HP i</code>	yes	<code>i=p.i.*</code>	Set prevailing indent to <i>i</i> . Begin paragraph with hanging indent.
<code>.I t</code>	no	<code>t=n.t.l.</code>	Text <i>t</i> is italic.
<code>.IB t</code>	no	<code>t=n.t.l.</code>	Join words of <i>t</i> alternating italic and bold.
<code>.IP x i</code>	yes	<code>x=""</code>	Same as <code>.TP</code> with tag <i>x</i> .

.IR	<i>t</i>	no	<i>t</i> =n.t.l.	Join words of <i>t</i> alternating italic and Roman.
.L	<i>t</i>	no	<i>t</i> =n.t.l.	Text <i>t</i> is literal.
.LP		yes		Same as .PP.
.LR	<i>t</i>	no		Join 2 words of <i>t</i> alternating literal and Roman.
.PD	<i>d</i>	no	<i>d</i> = <i>4v</i>	Interparagraph distance is <i>d</i> .
.PP		yes		Begin paragraph. Set prevailing indent to default.
.RE		yes		End of relative indent. Set prevailing indent to amount of starting .RS.
.RF	<i>t</i>	no	<i>t</i> =n.t.l.	Join words of <i>t</i> alternating Roman and filename.
.RI	<i>t</i>	no	<i>t</i> =n.t.l.	Join words of <i>t</i> alternating Roman and italic.
.RL	<i>t</i>	no		Join 2 or 3 words of <i>t</i> alternating Roman and literal.
.RS	<i>i</i>	yes	<i>i</i> =p.i.	Start relative indent, move left margin in distance <i>i</i> . Set prevailing indent to default for nested indents.
.SH	<i>t</i>	yes	<i>t</i> =""	Subhead; reset paragraph distance.
.SM	<i>t</i>	no	<i>t</i> =n.t.l.	Text <i>t</i> is small.
.SS	<i>t</i>	no	<i>t</i> =""	Secondary subhead.
.TF	<i>s</i>	yes		Prevailing indent is wide as string <i>s</i> in font <b>L</b> ; paragraph distance is 0.
.TH	<i>n c x</i>	yes		Begin page named <i>n</i> of chapter <i>c</i> ; <i>x</i> is extra commentary, e.g. 'local', for page head. Set prevailing indent and tabs to default.
.TP	<i>i</i>	yes	<i>i</i> =p.i.	Set prevailing indent to <i>i</i> . Restore default indent if <i>i</i> =0. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.
.1C		yes		Equalize columns and return to 1-column output
.2C		yes		Start 2-column nofill output

\* n.t.l. = next text line; p.i. = prevailing indent

## BUGS

There's no way to fool *troff* into handling literal double quote marks " in font-alternation macros, such as .BI.

There is no direct way to suppress column widows in 2-column output; the column lengths may be adjusted by inserting .sp requests before the closing .1C.

**NAME**

mars – memory array redcode simulator

**SYNOPSIS**

**mars** [ **-dfhmp** ] [ **-cqsvalue** ] *file* ...

**DESCRIPTION**

*Mars* is a simulator for the ‘Redcode’ machine from Kee Dewdney,s ‘Computer Recreations,’ *Scientific American* , May, 1984, coded by Michael Mauldin, CMU. The easiest way to create an object file is to use the [redcode\(6\)](#) command to assemble the object file from a redcode source file.

An object file contains three header lines: the name of the program, its length, and its starting location. Here is a sample redcode object file, for the Dwarf program:

```
name dwarf
length 4
start 1
00000007999
20000517999
10000027998
41799800000
```

The instruction format is an 11 digit decimal string, packed thus:

```
struct {
    char[1] opcode;
    char[1] mode1; char[4] arg1;
    char[1] mode2; char[4] arg2;
}
```

Options allow for tracing execution, for graphically displaying the progress of each program, and for analyzing and dumping memory before and after execution. For example

```
mars -s1234 -f -c20000 dwarf.obj gemini.obj imp.obj
```

specifies that *imp*, *dwarf*, and *gemini* are to be run together, with a fullscreen display for 20000 cycles using a random number seed of 1234.

- cN** the maximum number of cycles for this run. The default is 10000.
- d** (debug) execution to be traced in excruciating detail.
- f** (fullscreen) execution will be displayed graphically on any terminal supported by [curses\(3\)](#).
- h** (holes) description of memory usage will be printed after execution terminates.
- m** memory will be dumped before and after execution terminates.
- p** similar to **-m**, except only memory near each program counter is dumped.
- qN** *quit as soon as there are fewer than N* programs still alive. Default is **q1**.
- sN** seed for random number generator; *N=0* seeds from the clock.

**SEE ALSO**

[redcode\(6\)](#)

**NAME**

mbits – macros to typeset bitmaps

**SYNOPSIS**

**troff** [ *option ...* ] **-mbits** [ *option ...* ] *file ...*

**DESCRIPTION**

These macros are used to typeset bitmaps. They are compatible with other *troff* macro packages, so that bitmap figures may be included in documents.

- .BM** *f*           Set the format for subsequent **.BM** requests to *f*. The default is **b** for [blitbtl\(9\)](#) output. The other possibility is **i** for faces and large icons in ASCII format.
- .BM** *f s*         Insert the bitmap from file *f*; each pixel will be *s* basic units square. The bitmap origin is placed at the current point, which is left unchanged.
- .BM** *f s dX dY*   Set the number registers *dX* and *dY* to the width and height (in basic units) of the bitmap in file *f*, assuming pixel size *s*. This form may be used to calculate positioning.

**EXAMPLES**

This sequence centers the bitmap and spaces past it to continue with the text:

```
.BM i
.BM /n/face/48x48x1/pjw 6 dX dY
.sp
.in (\n(.lu-\n(dXu)/2u
.BM /n/face/48x48x1/pjw 6
.in
.sp \n(dYu

(.).EE
```

**FILES**

/usr/lib/bttroff

**SEE ALSO**

[blitbtl\(9\)](#) [troff\(1\)](#), [bitfile\(9\)](#) [ms\(6\)](#)

**BUGS**

At time of writing, *mbits* does not work with PostScript output devices. See [mpictures\(6\)](#) for an alternative.

**.BM** does not work inside a diversion.

The concept of ‘pixel size’ varies among typesetting devices.

**NAME**

mcs – macros for formatting cover sheets

**SYNOPSIS**

**troff -mcs file ...**

**DESCRIPTION**

The *mcs* package of *troff(1)* macros generates cover sheets for Bell Labs documents. The macros were not meant for human production. Use *docgen(1)* to write them:

```
docgen
```

The *mcs* macros automatically load the *ms(6)* macros.

In the following description macros marked \* are mandatory; all others are optional. Optional arguments are enclosed in square brackets. The macros must be used in the order presented:

**Request****Explanation**

- \* **.TI** [*draft*]  
Title text follows. Unless there is an argument, the cover sheet will be shipped automatically to the Bell Labs library ITDS. Duplicate shipments are harmless.
- \* **.AH** *author loc dept ext rm e-mailaddre co*
- .AP** *name*  
Responsible AT&T person
- \* **.SA** Begin Abstract
- \* **.SE** End Abstract
- .KW** [*k1 ... k9*]  
Keywords
- \* **.TY** *type software*  
Memo *type*: TM technical memorandum, IM internal memorandum, TC technical correspondence;  
*software*: *y* if memo is software related
- \* **.NU** *org-date-seq filing\_case work\_project*  
Document Number
- .ED** *doc\_number*  
Earlier document number.
- \* **.MY** [*a1 ... a8*]  
Mercury Code, positional arguments, **y-n**, at most 3 **y**: 1 Chemistry and Materials, 2 Communications, 3 Computing, 4 Electronics, 5 Life Science, 6 Mathematics and Statistics, 7 Physics, 8 Manufacturing
- .RL** *code*  
*y* release to any AT&T employee; *n* release only on approval of each request.
- .PR** [**BR**]  
Proprietary Marking, default ATT-BL Proprietary, 0 unmarked, BR restricted
- .GS** Government Security
- .CO** Complete Copy Distribution List follows
- .CE** End distribution lists
- .CV** Cover Sheet Only Distribution List follows
- \* **.SC** *pages*  
Total *pages* ; do not include coversheet pages (if old format of *pages* and *otherpages* , the arguments are added).

To turn the paper into released paper format put a **.RP** before the title macro. To make it into a CSTR add a **.TR** before the title and **.AI .MH** after the author macro.

**FILES**

```
/usr/lib/tmac/tmac.cs
/usr/lib/tmac/tmac.rscover
```

**SEE ALSO**

*docgen(1)*, *ms(6)*, *sendcover(8)*

**NAME**

mille – card game

**SYNOPSIS**

`/usr/games/mille` [ *file* ]

**DESCRIPTION**

*Mille* plays a two-handed game reminiscent of the Parker Brother's game of Mille Bournes. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

- P** Pick a card from the deck. This card is placed in the P slot in your hand.
- D** Discard a card from your hand. To indicate which card, type the number of the card in the hand (or P for the just-picked card) followed by a newline or space.
- U** Use a card. The card is again indicated by its number.
- O** Sort the cards in your hand. This command toggles on and off.
- Q** Quit the game.
- S** Save the game in a file. You will be asked for a file name. A newline without a name terminates the command, but not the game.
- R** Redraw the screen from scratch.
- W** Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. The end-of-game window saves time by eliminating the switch at the end of the game to show the final score.

**Cards**

The number of such cards appears after the card name:

Hazard	Repair	Safety
Out of Gas (2)	Gasoline (6)	Extra Tank (1)
Flat Tire (2)	Spare Tire (6)	Puncture Proof (1)
Accident (2)	Repairs (6)	Driving Ace (1)
Stop (4)	Go (14)	Right of Way (1)
Speed Limit (3)	End of Limit (6)	

25 – (10), 50 – (10), 75 – (10), 100 – (12), 200 – (4)

**Rules**

The object of the game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does, making points on the way.

The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled: 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards prevent your opponent from putting down Distance cards. They can only be played if your opponent has a *Go* card on top of the Battle pile. The hazards are 'Out of Gas,' 'Accident', 'Flat Tire', 'Speed Limit and 'Stop'. *Remedy* cards fix hazards: 'Gasoline', 'Repairs', 'Spare Tire', 'End of Limit', and 'Go'. *Safety* cards prevent your opponent playing Hazard cards: 'Extra Tank', 'Driving Ace', 'Puncture Proof', 'Right of Way'.

The board is split into several areas. From top to bottom, they are: SAFETY AREA: (unlabeled) where the safeties will be placed. HAND: The cards in your hand. BATTLE: This is the Battle pile. where the Hazard and Remedy Cards are played, except Speed Limit and End of Limit. Only the top card is displayed, as it is the only effective one. SPEED: The Speed pile. Speed Limit and End of Limit cards are played here to control the speed at which the player is allowed to put down miles. MILEAGE : Miles are placed. The total of the numbers shown here is the distance traveled so far. Not more than two 200-mile cards may be played in one turn.

The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches exactly 700 points or the deck runs out. If someone reaches 700, they have the option of going for an 'Extension', which means that the play continues until 1000 miles.



**Hazards and Remedies**

**Go** (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played Right of Way.

**Stop** is played on your opponent's Go card.

**Speed Limit**

is played on your opponent's Speed pile. Until they play an End of Limit they can only play 25 or 50 mile cards.

**End of Limit**

is played on your Speed pile to nullify a Speed Limit.

**Out of Gas**

is played on your opponent's Go. They must play Gasoline and then Go before they can play any more mileage.

Flat Tire and Accident are played similarly.

**Safety Cards**

prevent your opponent from playing the corresponding Hazards for the rest of the hand. It cancels the hazard, and entitles the player to an extra turn.

**Right of Way**

prevents both Stop and Speed Limit cards and acts as a permanent Go.

A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

**Coup Fourre:** This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, eve before you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points.

**Scoring:** Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window are:

Milestones Played: sum of miles

Each Safety: 100 points

All 4 Safeties: 300 points

Each Coup Fourre:

300 points for each Coup Fourre accomplished.

Trip Completed: 400 points

Safe Trip: 300 points bonus for completing trip without 200 mile

Delayed Action:

300 points for finishing after the deck was exhausted.

Extension: 200 points bonus for a 1000 mile trip.

Shut-Out: 500 points for completing while opponent has 0 miles

**AUTHOR**

Ken Arnold

**SEE ALSO**

[curses\(3\)](#)

**NAME**

monop – monopoly game

**SYNOPSIS**

/usr/games/monop [ file ]

**DESCRIPTION**

*Monop* is reminiscent of the Parker Brother's game Monopoly, and monitors a game among 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must fix the problem, *i.e.*, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, e.g., a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

*A Summary of Commands:*

- quit** quit game: This allows you to quit the game. It asks you if you're sure.
- print** print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):
- Name  
The first ten characters of the name of the square
  - Own  
The *number* of the owner of the property.
  - Price  
The cost of the property (if any)
  - Mg This field has a '\*' in it if the property is mortgaged
  - # If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
  - Rent  
Current rent on the property. If it is not owned, there is no rent.
- where** where players are: Tells you where all the players are. A '\*' indicates the current player.
- own holdings**  
List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.
- holdings** holdings list: Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type 'done'.
- shell** shell escape: Escape to a shell. When the shell dies, the program continues where you left off.
- mortgage**  
mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.
- unmortgage**  
unmortgage property: Unmortgage mortgaged property.
- buy** buy houses: Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks

you to re-input things.

- sell** sell houses: Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to *buy*
- card** card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.
- pay** pay for jail: Pay \$50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.
- trade** This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.
- resign** Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.
- save** save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.
- restore** restore game: Read in a previously saved game from a file. It leaves the file intact.
- roll** Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

## FILES

/usr/games/lib/cards.pck Chance and Community Chest cards

## BUGS

No command can be given an argument instead of a response to a query.

**NAME**

mpictures – picture inclusion macros

**SYNOPSIS**

**troff -mpictures** [ *options* ] *file* ...

**DESCRIPTION**

*Mpictures* macros insert PostScript pictures into *troff*(1) documents. The macros are:

**.BP** *source height width position offset flags label*

Define a frame and place a picture in it. Null arguments, represented by "", are interpreted as defaults. The arguments are:

*source*

Name of a PostScript picture file, optionally suffixed with (*n*) to select page number *n* from the file (first page by default).

*height* Vertical size of the frame, default **3.0i**.

*width* Horizontal size of the frame, current line length by default.

*position*

l (default), c, or r to left-justify, center, or right-justify the frame.

*offset* Move the frame horizontally from the original *position* by this amount, default **0i**.

*flags* One or more of:

**ad** Rotate the picture clockwise *d* degrees, default *d*=90.

**o** Outline the picture with a box.

**s** Freely scale both picture dimensions.

**w** White out the area to be occupied by the picture.

**l,r,t,b** Attach the picture to the left right, top, or bottom of the frame.

*label* Place *label* at distance **1.5v** below the frame.

If there's room, **.BP** fills text around the frame. Everything destined for either side of the frame goes into a diversion to be retrieved when the accumulated text sweeps past the trap set by **.BP** or when the diversion is explicitly closed by **.EP**.

**.PI** *source height,width,yoffset,xoffset flags*.

This low-level macro, used by **.BP**, can help do more complex things. The two arguments not already described are:

*xoffset*

Offset the frame from the left margin by this amount, default **0i**.

*yoffset*

Offset the frame from the current baseline, measuring positive downward, default **0i**.

**.EP** End a picture started by **.BP**; **.EP** is usually called implicitly by a trap at frame bottom.

If a PostScript file lacks page-delimiting comments, the entire file is included. If no **%%BoundingBox** comment is present, the picture is assumed to fill an 8.5×11-inch page. Nothing prevents the picture from being placed off the page.

**FILES**

/usr/lib/tmac/tmac.pictures

**SEE ALSO**

*troff*(1)

**DIAGNOSTICS**

A picture file that can't be read by the PostScript postprocessor is replaced by white space.

**BUGS**

A picture and associated text silently disappear if a diversion trap set by **.BP** isn't reached. Call **.EP** at the end of the document to retrieve it.

Macros in other packages may break the adjustments made to the line length and indent when text is being placed around a picture.

A missing or improper **%%BoundingBox** comment may cause the frame to be filled incorrectly.

**NAME**

mpm, mspe – macros for page makeup

**SYNOPSIS**

**troff -mpm** *file ...*

**troff -mspe** *file ...*

**DESCRIPTION**

These *troff*(1) macros, largely compatible with *ms*(6), make better pages. They silently invoke and provide information to a postprocessor that moves floating figures, avoids widows, and justifies pages vertically by stretching vertical spaces that result from **.PP**, **.LP**, **.JP**, **.QP**, **.SH**, **.NH**, **.DS/.DE**, **.EQ/.EN**, **.TS/.TE**, **.PS/.PE**, **.P1/.P2**, and **.QS/.QE**. The packages support different styles:

**-mpm** generic

**-mspe** *Software—Practice and Experience*

The following macros are different from or not part of **-ms**. Values denoted *n* have default value **1v**.

**.BP** Begin a new page.

**.FL** Flush: force out previous keeps.

**.FC** Finish a two-column region and start a new one.

**.KF** *m*

Floating keep, with preferred center at vertical position *m*. Special values `top` (default) and `bottom` are permitted.

**.NE** *n* Start new page if remaining vertical space on this page is less than *n*.

**.P1** Begin a program display (Courier font).

**.P2** End a program display.

**.P3** Insert optional break point in program display.

**.SP** *n exactly*

Insert vertical space of height *n*, stretchable unless **exactly** is present.

**.Tm** *text*

Place page number and *text* on the standard error output.

**.X** *text*

Present *text* to the hidden page-makeup program as part of a device-dependent output sequence **x X text**. Equivalent to `\X'text'`.

Useful number registers:

**HM** Header margin; default 1 inch.

**FM** Footer margin; default 1 inch.

**FO** Footer position; default 10 inches.

**%#** Page number of current page.

**dP,dV**

Shrinkage of point size and vertical spacing for **.P1**, in points.

Useful strings:

**%e,%o**

Even and odd page title commands, as **.tl ""**.

**FILES**

`/usr/lib/tmac/tmac.pm`

`/usr/lib/tmac/pm`

**SEE ALSO**

*ms*(6), *troff*(1)

B. W. Kernighan and C. J. Van Wyk, 'The `-mpm` Macro Package', this manual, Volume 2

**BUGS**

These features of **-ms** are missing:

Document styles other than the default **.RP**.

Space between front matter and first paragraph. Recover it with **.SP 2**.

Separating rule above footnotes.

Keeps assigned to a separate page.

Pages with more than two columns.

*Troff* option **-o** doesn't work with **-mpm** because only the postprocessor knows the page numbers.

**NAME**

ms – macros for formatting manuscripts

**SYNOPSIS**

**nroff** -ms [ options ] file ...

**troff** -ms [ options ] file ...

**DESCRIPTION**

This package of *nroff* and *troff*(1) macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*; see *column*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, but the following requests may be used with impunity after the first **.PP**: *.bp*, *.br*, *.sp*, *.ls*, *.na*.

Output of the *eqn*(1), *neqn*, *tbl*(1), *pic*(1), *refer*(1), and *prefer*(1) preprocessors for equations, tables, pictures, and references is acceptable as input.

Diacritical marks may be applied to letters, as in these examples:

\*'e	\*'a	\*'e	\*^e	\*^o	\*:u	\* n	\*,c	\*vc
è	à	é	ê	ô	ü	n	ç	ç <sup>v</sup>

**FILES**

/usr/lib/tmac/tmac.s

**SEE ALSO**

M. E. Lesk, ‘Typing Documents on the UNIX System: Using the -ms Macros with Troff and Nroff’, this manual, Volume 2

*eqn*(1), *troff*(1), *refer*(1), *prefer*(1), *tbl*(1), *pic*(1), *mcs*(6)

**REQUESTS**

Request	Initial Value	Cause	Explanation
.1C	yes	yes	One column format on a new page.
.2C	no	yes	Two column format.
.AB	no	yes	Begin abstract.
.AE	-	yes	End abstract.
.AI	no	yes	Author’s institution follows. Suppressed in <b>.TM</b> .
.AT	no	yes	Print ‘Attached’ and turn off line filling.
.AU <i>x y</i>	no	yes	Author’s name follows. <i>x</i> is location and <i>y</i> is extension, ignored except in <b>TM</b> .
.B <i>x y</i>	no	no	Print <i>x</i> in boldface, append <i>y</i> ; if no argument switch to boldface.
.B1	no	yes	Begin text to be enclosed in a box.
.B2	no	yes	End boxed text.
.BI <i>x y</i>	no	no	Print <i>x</i> in bold italic and append <i>y</i> ; if no argument switch to bold italic.
.BT	date	no	Bottom title, automatically invoked at foot of page. May be redefined.
.BX <i>x</i>	no	no	Print <i>x</i> in a box.
.CW <i>x y</i>	no	no	Constant width font for <i>x</i> , append <i>y</i> ; if no argument switch to CW.
.CT	no	yes	Print ‘Copies to’ and turn off line filling.
.DA <i>x</i>	nroff	no	‘Date line’ at bottom of page is <i>x</i> . Default is today.
.DE	-	yes	End displayed text. Implies <b>.KE</b> .
.DS <i>x</i>	no	yes	Start of displayed text, to appear verbatim line-by-line: I indented (default), L left-justified, C centered, B (block) centered with straight left margin. Implies <b>.KS</b> .
.EG	no	-	Print document in BTL format for ‘Engineer’s Notes.’ Must be first.
.EN	-	yes	Space after equation produced by <i>neqn</i> or <i>eqn</i> (1).
.EQ <i>x y</i>	-	yes	Display equation. Equation number is <i>y</i> . Optional <i>x</i> is I, L, C as in <b>.DS</b> .
.FE	-	yes	End footnote.
.FP <i>x</i>	-	no	Set font positions for a family, e.g., <i>.FP</i>
.FS	no	no	Start footnote. The note will be moved to the bottom of the page.
.HO	-	no	‘AT&T Bell Laboratories, Holmdel, New Jersey 07733’.
.I <i>x y</i>	no	no	Italicize <i>x</i> , append <i>y</i> ; if no argument switch to italic.

.IH	no	no	'AT&T Bell Laboratories, Naperville, Illinois 60540'
.IM	no	no	Print document in BTL format for an internal memorandum. Must be first.
.IP <i>x y</i>	no	yes	Start indented paragraph, with hanging tag <i>x</i> . Indentation is <i>y</i> ens (default 5).
.KE	-	yes	End keep. Put kept text on next page if not enough room.
.KF	no	yes	Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS	no	yes	Start keeping following text.
.LG	no	no	Make letters larger.
.LP	yes	yes	Start left-blocked paragraph.
.LT <i>x r p</i>	no	yes	Start a letter with today's date. If <i>x</i> is missing, use letterhead paper; otherwise print letterhead with room <i>r</i> and phone <i>p</i> ; address follows.
.MF	-	-	Print document in BTL format for 'Memorandum for File.' Must be first.
.MH	-	no	'AT&T Bell Laboratories, Murray Hill, New Jersey 07974'.
.MR	-	-	Print document in BTL format for 'Memorandum for Record.' Must be first.
.ND <i>date</i>	troff	no	Use date supplied (if any) only in special BTL format positions; omit from page footer.
.NH <i>n</i>	-	yes	Same as <b>.SH</b> , with automatic section numbers like '1.2.3'; <i>n</i> is subsection level (default 1).
.NL	yes	no	Make letters normal size.
.PE	-	yes	End picture; see <a href="#">pic(1)</a> .
.PF	-	yes	End picture; restore vertical position.
.PP	no	yes	Begin paragraph. First line indented.
.PS <i>h w</i>	-	yes	Start picture; height and width in inches.
.PY	-	no	'AT&T Bell Laboratories, Piscataway, New Jersey 08854'
.QE	-	yes	End quoted material.
.QP	-	yes	Begin quoted paragraph (indent both margins).
.QS	-	yes	Begin quoted material (indent both margins).
.R	yes	no	Roman text follows.
.RE	-	yes	End relative indent level.
.RP	no	-	Cover sheet and first page for released paper. Must precede other requests.
.RS	-	yes	Start level of relative indentation from which subsequent indentation is measured.
.SG <i>x</i>	no	yes	Insert signature(s). In <b>.TM</b> <i>x</i> is initials of author and typist; in <b>.LT</b> <i>x</i> is author's name.
.SH	-	yes	Section head follows, font automatically bold.
.SM	no	no	Make letters smaller.
.TA <i>x...</i>	5...	no	Set tabs in ens. Default is 5 10 15 ...
.TE	-	yes	End table; see <a href="#">tbl(1)</a> .
.TH	-	yes	End heading section of table.
.TL	no	yes	Title follows.
.TM <i>x...</i>	no	-	Print document in BTL technical memorandum format. Arguments are TM number, (quoted list of) case number(s), and file number. Must precede other requests.
.TR <i>x</i>	-	-	Print in BTL technical report format; report number is <i>x</i> . Must be first.
.TS <i>x</i>	-	yes	Begin table; if <i>x</i> is <b>H</b> table heading is repeated on new pages.
.UL <i>x</i>	-	no	Underline argument (even in troff).
.UX <i>y z</i>	-	no	'zUNIXy'; first use gives registered trademark notice.
.WH	-	no	'AT&T Bell Laboratories, Whippany, New Jersey 07981'.
.[	-	no	Begin reference; see <a href="#">refer(1)</a> .
.]	-	no	End reference.



**NAME**

number – convert Arabic numerals to English

**SYNOPSIS**

**number**

**DESCRIPTION**

*Number* copies the standard input to the standard output, replacing all decimal numbers by their spelled-out equivalent. Punctuation is added to make the output sound well when played through voice synthesizers.

**NAME**

ogre – war game

**SYNOPSIS**

`/usr/games/ogre` [ *type* ]

**DESCRIPTION**

Ogre is a game of tank warfare in the 21st century. You command a force of infantry, armor, and howitzers pitted against a giant cybernetic tank, the Ogre. Your mission is to destroy the Ogre, or at least render it immobile, before it reaches and destroys your command post.

A more complete reference on how to play can be found in the Ogre rule book for the Metagaming MicroGame, now distributed by Steve Jackson's company. Here's some very sketchy and incomplete documentation for Ogre players:

The game has the following phases:

1) Initialization. The player's armor units, infantry, and command post are placed on the map. Nothing can be placed on the leftmost 7 columns of hexes, or on craters (\*'s), or on any unit already placed. Valid commands are:

```
w e
a d (hex movement keys)
z x
place a:
H howitzer
T heavy tank
M missile tank
G GEV
I Infantry unit (attack strength 3)
C Command Post
```

on the space currently pointed at by the cursor. Note that these are capital letters.

Units are displayed as these characters, except infantry, which appear as '1', '2', or '3' depending on their attack strength.

2) The Ogre (an O) now appears.

3) You are given the opportunity to move all your vehicles and infantry that can move. The cursor motion keys are used to move the unit indicated by the cursor. Additionally, 's' or ' ' can be used to let a vehicle stay motionless. No vehicle can move through a crater hex, or into a hex occupied by another friendly unit on its last turn, although it can move through a friendly hex on its way elsewhere. Moving through the hex occupied by the Ogre is an attempt to ram the Ogre. This reduces the Ogre's treads by some amount, and destroys the unit.

4) You now fire all your vehicles in range at designated targets on the Ogre. The following commands are used:

```
m fire at missiles
b fire at main batteries
s fire at secondary batteries
a fire at anti-personnel guns
t fire at treads
```

The odds of destroying the target are displayed, but no action is taken until 'r' is used, or until you run out of attack points (except for attacks on treads – see below). (In the odds display, '+' means a sure thing.)

**p** Pass. The unit is passed over, and given the opportunity to fire later.

**r** resolve all allocations so far, and display the results. This is implied by 't', as tread attacks cannot be grouped. A resolve is done automatically when you run out of attacking units.

5) Second movement phase for GEVs. Just like step 3, except that only GEVs can move.

6) The Ogre moves. If it runs over any of your units, they are damaged or destroyed.

7) The Ogre fires at all units in range. Destroyed units are removed from the map. Disabled units are displayed in lower case, and may not move or fire until the end of the NEXT Ogre attack.

Steps 3 through 7 are repeated until either a) the Ogre has no movement points left, in which case you win, or b) your command post is destroyed, in which case the Ogre wins.

### **MISCELLANEOUS**

The display "a/r Dd Mm" means the unit concerned attacks at a, at range r, defends at d, and moves m hexes per turn.

The Ogre by default is a Mark III. An argument of '5' on the command line makes it a Mark V, and gives you more armor points.

The game can be interrupted at any point with a control-C. There's now no way to restart.

The paper game is copyright (c) 1977 by Steve Jackson. This computer implementation is copyright (c) 1984 by Michael Caplinger.

### **AUTHOR**

Michael Caplinger, Rice University (mike@rice.ARPA), from a Microgame of the same name published by Metagaming of Austin, Texas, and written by Steve Jackson. This implementation is not authorized in any way by Mr. Jackson, and should not be sold for profit.

### **SEE ALSO**

[\*termcap\(5\)\*](#)

### **BUGS**

The Ogre sometimes gets confused and doesn't know where to go, so it oscillates from one hex to another, and then back.

**NAME**

quiz – test your knowledge

**SYNOPSIS**

`/usr/games/quiz` [ `-i file` ] [ `-t` ] [ *category1 category2* ]

**DESCRIPTION**

*Quiz* gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, *quiz* gives instructions and lists the available categories.

*Quiz* tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, it reports a score and terminates.

The `-t` flag specifies ‘tutorial’ mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The `-i` flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```

line      = category newline | category ‘:’ line
category = alternate | category ‘|’ alternate
alternate = empty | alternate primary
primary   = character | ‘[’ category ‘]’ | option
option    = ‘{’ category ‘}’

```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash `\` is used as with *sh(1)* to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

**FILES**

`/usr/games/lib/quiz.k/*`

**BUGS**

The construct `a|ab` doesn’t work in an information file. Use `a{b}` or `ab|a`. Case distinctions cannot be checked even when they count.

COMMANDS.SH NAME redcode – assembler for mars game

## SYNOPSIS

*redcode file ...*

## DESCRIPTION

*Redcode* is an assembler for the assembly language given by Kee Dewdney in the ‘Computer Recreations’, *Scientific American*, May, 1984. The command line lists of source file names ending in `.red`. They are assembled into corresponding object files with names ending in `.obj`.

A source file consists of a name directive giving the program’s name, then any number of program and data statements, then an end directive that gives the starting location of the program. Statements have the following syntax:

```
[label] opcode arg1 [arg2] [; comment]
```

There are three addressing modes; all address calculations are done modulo 8000.

<i>Syntax</i>	<i>Meaning</i>
<code>#[0-9]+</code>	immediate
<code>[0-9]+</code>	relative
<code>@[0-9]+</code>	indirect, relative

The following opcodes are implemented, along with the corresponding semantics specified in pseudo-C:

<i>Instruction</i>	<i>Mnem</i>	<i>Opcode</i>	<i>Args</i>	<i>Explanation</i>
Move	mov	1	A B	B=A
Add	add	2	A B	B+=A
Subtract	sub	3	A B	B-=A
Jump	jmp	4	A	PC=A
Jump if zero	jmz	5	A B	PC=(B==0)?A:PC+1
Jump if greater	jmg	6	A B	PC=(B<4000)?A:PC+1
Dec, Jmp if 0	djz	7	A B	PC=(--B==0)?A:PC+1
Compare	cmp	8	A B	PC=(A==B)?PC+1:PC+2

The following non-executable directives may be used to reserve and initialize data space:

<i>Directive</i>	<i>Mnem</i>	<i>Arg</i>	<i>Explanation</i>
Buffer space	bss	n	Reserve n words
Data	data	A	Initialize 1 word
Name	name	't'	Name of program
End	end	start	Specify starting location

The program was written by Paul Milazzo at Rice.

For documentation on the object code format, see

[mars\(6\)](#).

## EXAMPLES

```
name    'dwarf'
site    data    -1          ; address of last 0 'bomb'
start   add     #5         site ; move site forward
        mov     #0         @site ; write 0 'bomb'
        jmp     start      ; loop
end     start
```

## SEE ALSO

[mars\(6\)](#)

**NAME**

snake, worm – display chase games

**SYNOPSIS**

**/usr/games/snake**

**/usr/games/worm**

**DESCRIPTION**

*Snake* must be played on a HP2621 terminal or equivalent. The object of the game is to make as much money as possible without getting eaten by the snake.

You are represented on the screen by **I**. The snake is 6 squares long, each marked **S**. The money is **\$** and an exit is **#**. Your score is posted in the upper left hand corner.

You can move around using keys h for left, up, down, right. To earn money, move to the same square the money is on. A new **\$** will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit.

*Worm* also requires a 2621-compatible terminal. Once started, with h keys as for *snake*, the worm moves forward unless directed otherwise. The object is to collect points displayed on the screen without running into the wall or any part of the worm itself. The points are added to the worm's length.

**NAME**

terminals – conventional names

**DESCRIPTION**

These names are used by certain commands and are maintained as part of the shell environment; see [sh\(1\)](#), [environ\(5\)](#).

- 2621** Hewlett-Packard HP262? series terminals
- 1620** DIABLO 1620 (and others using HyType II)
- 33** Teletype Model 33
- 37** Teletype Model 37
- 43** Teletype Model 43
- 5620** Teletype Model 5620 dotmap display
- dumb** terminals with no special features
- 4014** Tektronix 4014
- vt52** Digital Equipment Corp. VT52

The list goes on and on. Consult `/etc/termcap` (see [termcap\(5\)](#)) for the whole truth.

Commands whose behavior may depend on the terminal typically consult `TERM` in the environment or accept arguments of the form `-Tterm`, where *term* is one of the names given above.

**SEE ALSO**

[stty\(1\)](#), [tabs\(1\)](#), [plot\(1\)](#), [sh\(1\)](#), [environ\(5\)](#), [ul\(1\)](#), [column\(1\)](#), [termcap\(5\)](#), [nroff](#) in [troff\(1\)](#)

**BUGS**

The programs that ought to adhere to this nomenclature do so only fitfully.

**NAME**

warp – war games

**SYNOPSIS**

**/usr/games/warp**

**DESCRIPTION**

*Warp* is a space-war game; it volunteers instructions.

*Battle* is the classic grid game of battleship. It needs a cursor-addressed terminal.



**NAME**

worms, hanoi, rain – silly demos

**SYNOPSIS**

**/usr/games/worms**

**/usr/games/hanoi**

**/usr/games/rain**

**DESCRIPTION**

These games draw various moving patterns on a cursor-addressed terminal.

**NAME**

apnews, ap.keys – present AP wire stories

**SYNOPSIS**

**apnews** [ **-f** *dir* ] [ **-r** ]

**DESCRIPTION**

*Apnews* presents news from the AP wire on a cursor-addressed screen. The top half of the screen contains 20 story slugs (two-word labels). *Apnews* responds to these commands:

- n** Print story for slug *n*; page through it by typing newlines.
- m** Present more slugs.
- .** Return to current slug list.
- t** Top. Return to first list of slugs
- s** *keywords*  
Present slugs for stories containing these keywords.
- y** Present slugs for stories containing words from the last story read.
- c** *file*  
Copy. Add story being read to named file or directory.
- ?** Print some help.

To suggest interest, slugs may be followed by a bracketed number that shows the average number of pages (up to 5) that readers have perused. Option **-r** turns this feature off.

Option **-f** directs the attention of *apnews* to a specified directory of AP stories, as may be collected by

To monitor news automatically, put a file *ap.keys* in your home directory. This file contains instructions marked by \*, each followed by one or more search lines. Instructions specify what to capture:

- S** whole story
- P** first paragraph
- H** heading

then what to do with it:

| *command*

specifying a command (often mail) to be executed with the story as standard input

> *file* specifying a file or directory to add the story to; pathnames are relative to your home directory

If no instruction is present, the default is

\*S

Search lines may contain:

- (1) a sequence of blank-separated words; these words must occur in this order
- (2) a sequence of words separated by commas; these words must appear in the same sentence
- (3) a sequence of words separated by periods; these words may occur anywhere in the story, but all must appear

Combinations are allowed, e.g. *x . y, z* specifies *y* and *z* in the same sentence and *x* somewhere in the same story. The character **!** means not, so that **!chocolate** means *chip* not preceded by *chocolate*. Some suffixes are removed; and capitals are ignored except when entire words are capitalized. Thus *ERA* and *era* are distinguished, but *Waters* and *waters* are not. Special 'words' specify story types:

- #f** flash
- #b** bulletin
- #u** urgent news
- #r** regular news
- #d** deferred news

**EXAMPLES**

```
*S > stuff
bell laboratories
FCC . telephone, regulation
*P | mail joe
#b
AM-NewsDigest
```

**FILES**

```
ap.keys
/usr/spool/ap/*
```

**BUGS**

*Apnews* can fail to work well in a [mux\(9\)](#) window, for two reasons.

- (1) The window needs a terminal emulator. Before invoking *apnews*, do `exec term 5620` (or **2621**); see [term\(9\)](#)
- (2) Remote execution needs a transparent connection. If logged in elsewhere make the connection to the serving machine by doing, for example, `ndcon alice` or `nrx alice apnews` (after downloading an emulator, if necessary); see [dcon\(1\)](#).

**NAME**

dist, dme, plan, path, cross – aviation navigation

**SYNOPSIS**

`/usr/ken/bin/dist obj obj ...`

`/usr/ken/bin/dme obj radial dist`

`/usr/ken/bin/plan [ -dist1 [ -dist2 ]] obj obj`

`/usr/ken/bin/path [ -dist ] obj obj ...`

`/usr/ken/bin/cross [ -dist ] obj`

**DESCRIPTION**

These routines provide navigation services using an aviation database. *Objects* in the database are of four types: VORs ending in **.v**, airports ending in **.a**, NDBs ending in **.n**, and intersections ending in **.i**. An ambiguous object specified without suffix is interpreted in the above order.

The canonical program in this series, *dist*, prints the magnetic bearing and distance in nautical miles between a set of two or more specified objects. The magnetic correction is applied at the first of a pair of objects, so the bearing from Morristown to San Diego is 281 degrees while the reverse is 50 degrees. *Dist* also prints a frequency for objects. For VORs and NDBs, the frequency is obvious. For airports, the frequency is sometimes the tower frequency, sometimes UNICOM and sometimes zero. Intersections have no frequency.

*Dme* prints the latitude and longitude of a point that is a bearing and distance from an object. The format printed is the source form of the database of objects. *Dme* is used to create new objects for the database – usually intersections.

*Plan* finds a shortest distance from one object to another traveling along a route of VORs. The optional argument *dist1* is the maximum allowable distance between VORs en route (default 100 nautical miles) and *dist2* is the maximum allowable distance between the starting object and the first VOR and the last VOR and terminal object (default 50).

*Path* lists all objects in the database that lie within *dist* (default 10 nautical miles) of the great circle route between two objects. If more than two arguments are given, routes are calculated for each pair of objects. The list is ordered by distance along the route. The tangential distance to each object is given with negative to the left and positive to the right.

*Cross* prints all objects in the database within *dist* (default 50 nautical miles) from the given object.

**FILES**

objects

**SEE ALSO**

[avw\(7\)](#)

**BUGS**

The database is old and should not be used for navigation purposes.

Frequencies for airports are inconsistent.

*Plan* uses low power (terminal) VORs when very often these cannot be tracked from 50 miles.

*Plan* will cheerfully plan routes through prohibited areas, over open water and over high mountains. The only criterion is the distance between objects.

*Cross* has bugs.

**NAME**

w, fp, ft, fd, rad – aviation weather

**SYNOPSIS**

**/usr/ken/bin/w** [ *station ...* ]

**/usr/ken/bin/fp** [ *obj ...* ]

**/usr/ken/bin/ft** [ *obj ...* ]

**/usr/ken/bin/fd** [ *obj ...* ]

**/usr/ken/bin/rad**

**DESCRIPTION**

*W* looks up each of its arguments in a set of weather files. The weather files are constantly maintained by a daemon that reads the National Weather Service Wire. If *w* is called with no arguments, it reads the standard input and looks up each line.

The active weather files are:

**AA** airport name and runway direction and length  
**NT** NOTAM – Notices to Airmen – special cautions at airports  
**FT** 24-hour terminal forecasts at airports issued three times a day  
**SA** hourly surface observations taken at airports  
**FD** daily winds aloft forecasts taken at certain reporting points  
**FP** daily area forecasts taken at certain reporting points  
**SD** hourly radar precipitation taken at certain reporting points

If *w* is given an airport name, it will print the latest AA, NT, FT, and SA data. The other files are printed by providing the reporting point name. In most cases this is a meaningless string of characters that are supplied by other programs.

*Fp* takes a series of navigation stations as arguments; see *av(A)* for a description of these objects. If *fp* has no argument, it uses the default here which specifies Murray Hill, NJ. If *fp* is given one argument, it will print the name of the nearest FP (area forecast) station to the argument. If *fp* is given two arguments, it will print the names of all FP stations that are nearest to some point on the great circle route between the objects. If more than two arguments are given then the stations are printed for each pair of arguments. The output of *fp* is meant to be piped into *w*.

*Ft* and *fd* behave the same as *fp* but print the station names reporting winds aloft and terminal forecasts respectively.

*Rad* creates a radar summary weather map and prints the map on the laser printer.

**FILES**

**/usr/weather/\***  
 weather files

**/usr/ken/lib/obj**  
 navigation aids

**/usr/weather/rd**  
 weather wire daemon

**SEE ALSO**

*av(7)*

**NAME**

cal – print calendar

**SYNOPSIS**

cal [ *month* ] *year*

**DESCRIPTION**

*Cal* prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

**BUGS**

The year is always considered to start in January even though this is historically naive. Beware that cal 90 refers to the early Christian era, not the 20th century.

**NAME**

dict – look up words in English dictionaries

**SYNOPSIS**

**dict** [ **-p** ] [ *dictionary* [ *word* ] ]

**DESCRIPTION**

*Dict* looks up words in the specified *dictionary*, **webster** by default. Words are read, one per line from the standard input, and entries are written on the standard output. Characters other than letters, digits, and space are ignored. A single word may be specified in the command. The options are

- p** Find all entries of which the specified *word* is a prefix.
- r** Print raw form, including diacriticals, font marks, etc. Different for each dictionary to which it applies.

*Dictionary* is one of

**webster**

Merriam-Webster Collegiate Dictionary, Seventh Edition, full text. No option **-p**.

**web7** Same, words only.

**etym[ology]**

Inverted index to *webster* by root words.

**web2** Merriam-Webster New International Dictionary, Second Edition, unabridged, words only.

**oed** The Oxford New English Dictionary, or OED, full text. No option **-p**.

**oxford**

The Oxford Advanced Learner's Dictionary of Contemporary English, full text. No **-p**.

**slang** New Dictionary of American Slang (Harper). Field identifiers: **me** main entry (perhaps flagged @), **nu** sense number (also given as *\*n\**), **pr** pronunciation, **ps** part of speech, **vr** variations **la** provenance, **df** definition, **dx** definition by example, **ex** example, **ed** editorial note, **et** etymology, **xr** cross reference, **xx** indirect address, **sq** sequence number in original text.

**names**

The Oxford Dictionary of British surnames.

**thesaurus**

Collins Thesaurus.

**thesaurusa**

Same, augmented with complete backreferences among words.

**places**

USGS Gazetteer of populated places in US and possessions, with standard county code, latitude, longitude, year of listing, altitude (feet), 1980 population, topo sheet code. For full name search, use thus: **dict places 'new york, ny'**; without state, use prefix search: **dict -p places 'new york'**.

**towns** A shorter gazetteer (PICADAD) of US populated places with latitude, longitude, zip code, and population class (0:0-1000, 1:1000-2500, 2:2500-5000, 3:5000-10,000, 4:10,000-25,000, 5:25,000-50,000, 6:50,000-100,000, 7:100,000-250,000, 8:250,000-500,000, 9:500,000+), and something else.

**spell** Word list of [spell\(1\)](#).

**acro[nym]**

17000 AT&T acronyms

anything else

Print list of available dictionaries.

The dictionaries are copyrighted and must not be copied without permission, except for *web2*, *spell*, and *acro*.

**FILES**

/usr/dict/\*  
    (**n/bowell**)

/usr/dict/words  
    **spell** or **web7**, depending on machine

/usr/dict/oed  
    (**n/kwee**)

/usr/lib/dict/\*

/usr1/maps/usplaces  
    (**n/bowell**)

/usr/spool/town/ustowns\*  
    (**n/alice**)

**SEE ALSO**

*look(1)*, *town(7)*

**BUGS**

In **webster**, **web7**, and **pron**, diacriticals are done right – by overstrikes – which means they disappear on most screen terminals.

In **towns**, all data are lower case; missing zip codes look like normal codes ending in ‘000’. Latitude and longitude denote the center of population of the containing political entity; unincorporated places are spotted at the county center.

In **slang**, some entries contain extra trash; option **-p** helps overcome the trouble.



**NAME**

dkname – map system name to Datakit address

**SYNOPSIS**

**dkname** *sysname* ...

**DESCRIPTION**

*Dkname* looks up machine names and prints the corresponding full Datakit address on the standard output. The addresses are not necessarily printed in the same order as the arguments. The addresses are obtained by the first possible match in the file

**FILES**

/usr/lib/uucp/Systems.dk

**NAME**

`greek` – graphics for extended TTY-37 type-box

**SYNOPSIS**

`cat /usr/pub/greek [ | greek -Tterminal ]`

**DESCRIPTION**

*Greek* gives the mapping from ascii to the ‘shift out’ graphics in effect between SO and SI on model 37 Teletypes with a 128-character type-box. These are the default greek characters produced by *nroff*. The filters of *greek(1)* attempt to print them on various other terminals. The file contains:

alpha	$\alpha$	A	beta	$\beta$	B	gamma	$\gamma$	\
GAMMA	$\Gamma$	G	delta	$\delta$	D	DELTA	$\Delta$	W
epsilon	$\epsilon$	S	zeta	$\zeta$	Q	eta	$\eta$	N
THETA	$\Theta$	T	theta	$\theta$	O	lambda	$\lambda$	L
LAMBDA	$\Lambda$	E	mu	$\mu$	M	nu	$\nu$	@
xi	$\xi$	X	pi	$\pi$	J	PI	$\Pi$	P
rho	$\rho$	K	sigma	$\sigma$	Y	SIGMA	$\Sigma$	R
tau	$\tau$	I	phi	$\phi$	U	PHI	$\Phi$	F
psi	$\psi$	V	PSI	$\Psi$	H	omega	$\omega$	C
OMEGA	$\Omega$	Z	nabla	$\nabla$	[	not	$\neg$	-
partial	$\partial$	]	integral	$\int$	^			

**SEE ALSO**

[greek\(1\)](#)

[troff\(1\)](#)

**NAME**

hier – file system hierarchy

**DESCRIPTION**

The following outline gives a quick tour through a representative directory hierarchy.

```

/          root
/vmunix   the kernel binary (UNIX itself)
/lost+found
           directory for connecting detached files for fsck(8)
/dev/     devices (4)
           console  main console, tyld(4)
           tty*     terminals, tyld(4)
           ra*     disks, ra(4)
           rra*    raw disks, ra(4)
           ...
/bin/     utility programs, cf /usr/bin/ (1)
           as      assembler
           cc      C compiler executive, cf /lib/ccom, /lib/cpp, /lib/c2
           csh     C shell
           ...
/lib/     object libraries and other stuff, cf /usr/lib/
           libc.a  system calls, standard I/O, etc. (2,3,3S)
           ...
           ccom   C compiler proper
           cpp    C preprocessor
           c2     C code improver
           ...
/etc/     essential data and maintenance utilities; sect (8)
           dump   dump program dump(8)
           passwd password file, passwd(5)
           group  group file, group(5)
           motd   message of the day, login(8)
           whoami system name, uname(3)
           termcap description of terminal capabilities, termcap(5)
           ttytype table of what kind of terminal is on each port, ttytype(5)
           mtab   mounted file table, mtab(5)
           dumpdates
                   dump history, dump(8)
           fstab  file system configuration table fstab(5)
           ttys  properties of terminals, ttys(5)
           getty  part of login, getty(8)
           init  the parent of all processes, init(8)
           rc    shell program to bring the system up
           cron  the clock daemon, cron(8)
           mount mount(8)
           wall  wall(1)
           ...
/tmp/     temporary files, usually on a fast device, cf /usr/tmp/
           e*    used by ed(1)
           ctm*  used by cc(1)
           ...
/usr/     general-purpose directory, usually a mounted file system
           adm/  administrative information
                   wtmp  login history, utmp(5)
                   messages
                           hardware error messages

```

```

        tracct  phototypesetter accounting, troff\(1\)
        lpacct  line printer accounting lpr\(1\)
/usr  /bin
      utility programs, to keep /bin/ small
      tmp/   temporaries, to keep /tmp/ small
            stm*  used by sort\(1\)
            raster used by plot\(1\)
      dict/  word lists, etc.
            words  principal word list, used by look\(1\)
            spellhist history file for spell\(1\)
      games/
            hangman
            lib/   library of stuff for the games
            quiz.k/ what quiz\(6\) knows
                   index  category index
                   africa  countries and capitals
                   ...
            ...
      include/ standard #include files
            a.out.h  object file layout, a.out\(5\)
            stdio.h  standard I/O, stdio\(3\)
            math.h   (3M)
            ...
            sys/    system-defined layouts, cf /usr/sys/h
      lib/   object libraries and stuff, to keep /lib/ small
            atrun  scheduler for at\(1\)
            lint/  utility files for lint
                   lint[12] subprocesses for lint\(1\)
                   llib-lc  dummy declarations for /lib/libc.a, used by lint\(1\)
                   llib-lm  dummy declarations for /lib/libc.m
                   ...
            struct/ passes of struct\(1\)
            ...
            tmac/  macros for troff\(1\)
                   tmac.an macros for man\(7\)
                   tmac.s  macros for ms\(7\)
                   ...
            font/  fonts for troff\(1\)
                   ftR    Times Roman
                   ftB    Times Bold
                   ...
            uucp/  programs and data for uucp\(1\)
                   L.sys  remote system names and numbers
                   uucico  the real copy program
                   ...
            units  conversion tables for units\(7\)
            eign  list of English words to be ignored by ptx\(1\)
/usr/  man/
      volume 1 of this manual, man\(1\)
            man0/  general
                   intro  introduction to volume 1, ms\(7\) format
                   xx    template for manual page
            man1/  chapter 1
                   as.1

```

```

        mount.lm
        ...
    ...
    cat1/    preformatted pages for section 1
    ...
    spool/   delayed execution files
    at/      used by at(1)
    lpd/     used by lpr(1)
            lock    present when line printer is active
            cf*    copy of file to be printed, if necessary
            df*    daemon control file, lpd(8)
            tf*    transient control file, while lpr is working
    uucp/    work files and staging area for uucp(1)
            LOGFILE
                summary log
            LOG.*  log file for one transaction
    mail/    mailboxes for mail(1)
            name   mail file for user name
            name.lock
                lock file while name is receiving mail
    secretmail/
            like mail/
    uucp/    work files and staging area for uucp(1)
            LOGFILE
                summary log
            LOG.*  log file for one transaction
    wd       initial working directory of a user, typically wd is the user's login name
            .profile
                set environment for sh(1), environ(5)
    calendar
            user's datebook for calendar(1)
    doc/     papers, mostly in volume 2 of this manual, typically in ms(7) format
    as/      assembler manual
    c        C manual
    ...
    /usr/    src/
            source programs for utilities, etc.
    cmd/     source of commands
            as/      assembler
            ar.c     source for ar(1)
            ...
            troff/   source for nroff and troff(1)
            font/   source for font tables, /usr/lib/font/
                    ftR.c   Roman
                    ...
            term/   terminal characteristics tables, /usr/lib/term/
                    tab300.c DASI 300
                    ...
            ...
    games/   source for /usr/games
    libc/    source for functions in /lib/libc.a
            crt/     C runtime support
            csu/     startup and wrapup routines needed with every C program
                    crt0.s   regular startup
                    mcrt0.s  modified startup for cc -p

```

	sys/	system calls (2)
		access.s
		alarm.s
		...
	stdio/	standard I/O functions (3S)
		fgets.c
		fopen.c
		...
	gen/	other functions in (3)
		abs.c
		...
	local/	source which isn't normally distributed
	new/	source for new versions of commands and library routines
	old/	source for old versions of commands and library routines
	sys/	system source
	h/	header (include) files
		acct.h <a href="#">acct(5)</a>
		stat.h <a href="#">stat(2)</a>
		...
	sys/	system source proper
		main.c
		pipe.c
		sysent.c system entry points
ucb/		binaries of programs developed at UCB
		...
	edit	editor for beginners
	ex	command editor for experienced users
		...
	mail	mail reading/sending subsystem
	man	on line documentation
		...
	pi	Pascal translator
	px	Pascal interpreter
		...
	vi	visual editor

**SEE ALSO**

[ls\(1\)](#), [du\(1\)](#), [icheck\(8\)](#), [find\(1\)](#), [grep\(1\)](#)

**BUGS**

The position of files is subject to change without notice.

**NAME**

library, bellcat – bell labs library services

**SYNOPSIS**

**library** [ *option ...* ] [ *item ...* ]

**bellcat** [ **-q** ] [ *database* ]

**DESCRIPTION**

*Library* mails orders to the Bell Labs library network for books, technical reports, etc. Its use is self-explanatory.

A long response may be inserted with **e** or **r** as in *Mail(A)* Interaction may be forestalled by answering questions on the command line (see example) and in a personal identity file, named in environment variable **LIBFILE** (**.lib** by default), which contains one or more lines like these:

**ID:** PAN or SSN

**LIBNAME:** last name

**LIBLOG:** log file, readable with **mail -f**

**LIBCNTL:** concatenated search control codes: **a** acknowledge, **mnumber** max on retrieved items

**LIBLOCAL:** interaction control code: **x** brief prompts

Most **LIBFILE** items may be entered as environment variables by the same names.

*Bellcat* places a call to on-line library databases. Once entered, *bellcat* is self-explanatory. to exit. Option **-q** gets a ‘quick search’, which uses no special terminal features and does not offer help.

Some of the databases are

**books** (default)

**journals**

**released**

released papers by Bell Labs authors

**tech\_reports**

non-AT&T technical reports

*xxx* unknown name causes a list of databases to be printed

The databases of *Bellcat* and more are available through the LINUS service of the library network. You may use the *library* command to sign up for LINUS.

**EXAMPLES**

**library -1 123456-851234-56tm**

Order a technical memorandum, giving answer 1 for kind of query and specifying a document number.

**library -4 -p waldstein, r k**

Consult the people file.

**FILES**

`$HOME/.lib`

`/usr/lib/bellcat`

**BUGS**

Except under option **-q**, *bellcat* requires a native-mode Teletype 5620 or a (possibly simulated) HP2621 terminal. Under *mux(9)* *bellcat* invokes a simulator if necessary. This introduces an extra level of shell, which can be avoided thus: `exec bellcat`.

**NAME**

map – draw maps on various projections

**SYNOPSIS**

**map** *projection* [ *param ...* ] [ *option ...* ]

**DESCRIPTION**

*Map* prepares on the standard output a map suitable for display by any plotting filter described in *plot(1)*. A menu of projections is produced in response to an unknown *projection*. For the meanings of *params* pertinent to particular projections see *proj(3)*.

The default data for *map* are world shorelines. Option **-f** accesses the higher-resolution World Data Bank II.

**-f** [ *feature ...* ]

Features are ranked 1 (default) to 4 from major to minor. Higher-numbered ranks include all lower-numbered ones. Features are

**shore**[1-4]

seacoasts, lakes, and islands; in the absence of **-m**, option **-f** automatically includes **shore1**

**ilake**[1-2]

intermittent lakes

**river**[1-4]

rivers

**iriver**[1-3]

intermittent rivers

**canal**[1-3]

**3**=irrigation canals

**glacier**

**iceshelf**[12]

**reef**

**saltpan**[12]

**country**[1-3]

**2**=disputed boundaries, **3**=indefinite boundaries

**state** states and provinces (US and Canada only)

In other options coordinates are in degrees, with north latitude and west longitude counted as positive.

**-l S N E W**

Set the southern and northern latitude and the eastern and western longitude limits. Missing arguments are filled out from the list  $-90, 90, -180, 180$ .

**-k S N E W**

Set the scale as if for a map with limits **-l S N E W** and no **-w** option.

**-o lat lon rot**

Orient the map in a nonstandard position. Imagine a transparent gridded sphere around the globe. Turn the overlay about the North Pole so that the Prime Meridian (longitude 0) of the overlay coincides with meridian *lon* on the globe. Then tilt the North Pole of the overlay along its Prime Meridian to latitude *lat* on the globe. Finally again turn the overlay about its 'North Pole' so that its Prime Meridian coincides with the previous position of meridian *rot*. Project the map in the standard form appropriate to the overlay, but presenting information from the underlying globe. Missing arguments are filled out from the list  $90, 0, 0$ . In the absence of **-o**, the orientation is  $90, 0, m$ , where *m* is the middle of the longitude range.

**-w S N E W**

Window the map by the specified latitudes and longitudes in the tilted, rotated coordinate system. Missing arguments are filled out from the list  $-90, 90, -180, 180$ . (It is wise to give an encompassing **-l** option with **-w**. Otherwise for small windows computing time varies inversely with area!)



**-d** *n*

For speed, plot only every *n*th point.

**-r**

Reverse left and right (good for star charts and inside-out views).

**-s1****-s2**

Superpose. Outputs for a **-s1** map (no closing) and a **-s2** map (no opening) may be concatenated.

**-g** *dlat dlon res*

Grid spacings are *dlat*, *dlon*. Zero spacing means no grid. Missing *dlat* is taken to be zero. Missing *dlon* is taken the same as *dlat*. Grid lines are drawn to a resolution of *res* (2° or less by default). In the absence of **-g**, grid spacing is 10°.

**-p** *lat lon extent*

Position the point *lat*, *lon* at the center of a square plotting area. Scale the map so that a side of the square is *extent* times the size of one degree of latitude at the center. By default maps are scaled and positioned to fit within the plotting area. An *extent* overrides option **-k**.

**-c** *x y rot*

After all other positioning and scaling operations, rotate the image *rot* degrees counterclockwise about the center and move the center to position *x*, *y*, of the plotting area, whose nominal extent is  $-1 \leq x \leq 1$ ,  $-1 \leq y \leq 1$ . The map is clipped to this area. Missing arguments are taken to be 0.

**-m** [*file ...*]

Use map data from named files. If no files are named, omit map data. Files that cannot be found directly are looked up a standard directory, which contains, in addition to the data for **-f**,

**world** World Data Bank I from CIA (default)

**states** US map from Census Bureau

**counties**

US map from Census Bureau

The environment variables **MAP** and **MAPDIR** change the default map and default directory.

**-b** [*lat1 lon1 lat2 lon2 ...*]

Suppress the drawing of the normal boundary (defined by options **-l** and **-w**). Coordinates, if present, define the vertices of a polygon to which the map is clipped. If only two vertices are given, they are taken to be the diagonal of a rectangle. To draw the polygon, give its vertices as a **-u** track.

**-t** *file ...*

The arguments name ASCII files that contain lists of points, given as latitude-longitude pairs in degrees. If the first file is named **-**, the standard input is taken instead. The points of each list are plotted as connected 'tracks'.

Points in a track file may be followed by label strings. A label breaks the track. A label may be prefixed by **"**, **:**, or **!** and is terminated by a newline. An unprefixed string or a string prefixed with **"** is displayed at the designated point. The first word of a **:** or **!** string names a special symbol (see option **-y**). An optional numerical second word is a scale factor for the size of the symbol, 1 by default. A **:** symbol is aligned with its top to the north; a **!** symbol is aligned vertically on the page.

**-u** *file ...*

Same as **-t**, except the tracks are unbroken lines. (**-t** tracks are dot-dash lines.)

**-y** *file*

The *file* contains *plot(5)*-style data for **:** or **!** labels in **-t** or **-u** files. Each symbol is defined by a comment *:name* then a sequence of *m* and *v* commands. Coordinates (0,0) fall on the plotting point. Default scaling is as if the nominal plotting range were *ra* -1 -1 1 1; *ra* commands in *file* change the scaling.

**EXAMPLES**

map perspective 1.025 -o 40.75 74 A view looking down on New York from 100 miles (0.025 of the 4000-mile earth radius). The job can be done faster by limiting the map so as not to 'plot' the invisible part of the world: map perspective 1.025 -o 40.75 74 -l 20 60 30 100. A circular border can be forced by adding option **-w** 77.33. (Latitude 77.33° falls just inside a

polar cap of opening angle  $\arccos(1/1.025) = 12.6804^\circ$ .)

map mercator -o 49.25 -106 180 A map whose 'equator' is a great circle passing east-west through New York. The pole of the map is placed  $90^\circ$  away ( $40.75+49.25=90$ ) on the other side of the earth. A  $180^\circ$  twist around the pole of the map arranges that the Prime Meridian of the map runs from the pole of the map over the North Pole to New York instead of down the back side of the earth. The same effect can be had from map mercator -o 130.75 74

map albers 28 45 -1 20 50 60 130 -m states A customary curved-latitude map of the United States.

map albers 28 45 -1 20 50 60 130 -y yfile -t tfile An example of tracks, labels, and symbols. Arrows at New York and Miami are 8% and 12% as long as the map is wide. The contents of yfile and tfile are

```
ra -50 -50 50 50          25.77 80.20 :arrow 12
:arrow                   25.77 80.20 Miami
m -1 0                   25.77 80.20
v 0 0                    35.00 74.02
v -.6 .3                 40.67 74.02 !arrow 8
m -.6 -.3                 40.67 74.02 " New York
v 0 0                    34.05 118.25 Los Angeles
```

map harrison 2 30 -1 -90 90 120 240 -o 90 0 0

A fan view covering  $60^\circ$  on either side of the Date Line, as seen from one earth radius above the North Pole gazing at the earth's limb, which is  $30^\circ$  off vertical.

Option

**-o**

overrides the default

**-o 90 0 180,**

which would rotate

the scene to behind the observer.

## FILES

All files in directory \$MAPDIR

[1-4]??

World Data Bank II for option **-f**

**world,states,counties**

default and other maps for option **-m**

\*.x map indexes

map the program proper

## SEE ALSO

[map\(5\)](#), [proj\(3\)](#), [plot\(1\)](#)

## DIAGNOSTICS

'Map seems to be empty'—a coarse survey found zero extent within the **-l** and **-w** bounds; for maps of limited extent the grid resolution, *res*, or the limits may have to be refined.

## BUGS

The syntax of range specifications in **-y** files differs from that in options.

Windows (option **-w**) cannot cross the Date Line.

No borders appear along edges arising from visibility limits.

Segments that cross a border are dropped, not clipped.

Certain very long line segments are dropped on the assumption that they were intended to go the other way around the world.

Automatic scaling may miss the extreme points of peculiarly shaped maps; use option **-p** to recover.

Although *map* draws grid lines dotted and **-t** tracks dot-dashed, many plotting filters cannot cope and make them solid.

**NAME**

me – macros for formatting papers

**SYNOPSIS**

**nroff** –me [ options ] file ...

**troff** –me [ options ] file ...

**DESCRIPTION**

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col(1)*

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .pp:

```
.bp    begin new page
.br    break output line here
.sp n  insert n spacing lines
.ls n  (line spacing) n=1 single, n=2 double space
.na    no alignment of right margin
.ce n  center next n lines
.ul n  underline next n lines
.sz +n add n to point size
```

Output of the *eqn*, *neqn*, *refer*, and *tbl(1)* preprocessors for equations and tables is acceptable as input.

**FILES**

/usr/lib/tmac/tmac.e

/usr/lib/me/\*

**SEE ALSO**

*eqn(1)*, *troff(1)*, *refer(1)*, *tbl(1)*

–me Reference Manual, Eric P. Allman

Writing Papers with Nroff Using –me

**REQUESTS**

In the following list, “initialization” refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The –me Reference Manual* for interesting details.

Request	Initial Value	Cause	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x <i>x</i>	-	no	Begin indexed item in index <i>x</i>
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ <i>m H</i>	-	no	Define paper section. <i>m</i> defines the part of the paper, and can be <b>C</b> (chapter), <b>A</b> (appendix), <b>P</b> (preliminary, e.g., abstract, table of contents, etc.), <b>B</b> (bibliography), <b>RC</b> (chapters renumbered from page one each chapter), or <b>RA</b> (appendix renumbered from page one).
.+c <i>T</i>	-	yes	Begin chapter (or appendix, etc., as set by .++). <i>T</i> is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by <i>eqn</i> or <i>neqn</i> .

.EQ $x y$	-	yes	Precede equation; break out and add space. Equation number is $y$ . The optional argument $x$ may be $I$ to indent equation (default), $L$ to left-adjust the equation, or $C$ to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS $x$	-	yes	Begin table; if $x$ is $H$ table has repeated heading.
.ac $A N$	-	no	Set up for ACM style output. $A$ is the Author's name(s), $N$ is the total number of pages. Must be given before the first initialization.
.b $x$	no	no	Print $x$ in boldface; if no argument switch to boldface.
.ba $+n$	0	yes	Augments the base indent by $n$ . This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi $x$	no	no	Print $x$ in bold italics (nofill only)
.bx $x$	no	no	Print $x$ in a box (nofill only).
.ef 'x'y'z' '''	no	no	Set even footer to $x y z$
.eh 'x'y'z' '''	no	no	Set even header to $x y z$
.fo 'x'y'z' '''	no	no	Set footer to $x y z$
.hx	-	no	Suppress headers and footers on next page.
.he 'x'y'z' '''	no	no	Set header to $x y z$
.hl	-	yes	Draw a horizontal line
.i $x$	no	no	Italicize $x$ ; if $x$ missing, italic text follows.
.ip $x y$	no	yes	Start indented paragraph, with hanging tag $x$ . Indentation is $y$ ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form $.*x$ . Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of 'x'y'z' '''	no	no	Set odd footer to $x y z$
.oh 'x'y'z' '''	no	no	Set odd header to $x y z$
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh $n x$	-	yes	Section head follows, font automatically bold. $n$ is level of section, $x$ is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz $+n$	10p	no	Augment the point size by $n$ points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u $x$	-	no	Underline argument (even in <i>troff</i> ). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp $x$	-	no	Print index $x$ .

**NAME**

netlib – retrieve public-domain software

**SYNOPSIS**

**mail research!netlib**

**DESCRIPTION**

*Netlib* retrieves files by electronic mail from a set of libraries of public-domain software, mostly mathematical. Netlib responds to mail messages containing one or more of the requests described below.

**send index**

**send** [ *option ...* ] *file ...* [ **but not** *file ...* ] **from** *library ...*

Retrieve files from specified libraries. The *index* lists all libraries and gives other helpful information. A file is delivered together with all files it depends on from its library, unless *option only* is present. Files are retrieved in upper case for requests written in upper case.

**whois names**

Retrieve addresses and telephone numbers from a database of applied mathematicians.

**find** *word ...* [ **from** *library ...* ]

Retrieve one-line index descriptions by content from all or any directories.

**mailsize size**

Limit the length of mail messages to *size*. The size may be given in kilobytes, e.g. **100k**.

For information about a particular library, retrieve its *index*, which lists routines with one-line descriptions, or its *directory*. The library *core* contains machine constants and basic linear algebra modules that are needed with many other libraries.

**EXAMPLES****send index from eispack**

What's in **eispack**?

**send directory from eispack**

Get file names and sizes.

**send dgeco from linpack**

Retrieve a routine and all it depends on.

**send list of dgeco from linpack**

How big would that retrieval be?

**find cubic spline**

What does *netlib* have about 'cubic' or 'spline'?

**NAME**

netnews – send or receive news articles

**SYNOPSIS**

**netnews** [ *option ...* ]

**netnews -s** [ *newsgroup ...* ]

**netnews -i** *title* [ **-n** *newsgroup ...* ]

**DESCRIPTION**

*Netnews* is an intercomputer news service. Used now only locally, it has been replaced by *postnews(7)* and *readnews(7)* for outside connections. When invoked without options it prints recent articles. Normally the articles printed are restricted to newsgroups you have signed up for and are newer than your last use of *netnews*. After each article a command is read from the standard input:

newline

Go on to next article.

**p** Print article again.

**-** Go back to previous article.

**c** Cancel (restricted to contributor and super-user).

**w** *file*

Append a copy of the article to the named *file*.

**r** Reply to author (via mail).

**q** Exit.

**x** Exit without update.

The options are:

**-p** Print with no questions asked.

**-r** Print in reverse time order.

**-l** Print titles only.

**-a** *date*

Print articles received after *date*; no date means the beginning of time.

**-n** *newsgroup ...*

Print articles from named newsgroups.

**-t** *string ...*

Print only titles containing one of the *strings*.

**-s** *newsgroup ...*

Subscribe to named newsgroups. If no *newsgroups* are given, list your current subscriptions. Newsgroup *all* receives all articles; *net.all* receives all newsgroups that begin with *net.*, etc. All users subscribe to the newsgroup *general*.

**-i** Insert an article (read from standard input) with title *title* to the newsgroups specified by **-n** (default *general*).

**FILES**

`$HOME/.newsrc`

user's subscription list

`/usr/spool/news/sys.nnn`

news articles

`/usr/spool/news/.bitfile`

bit map of users with news

`/usr/spool/news/.ngfile`

list of legal newsgroups

`/usr/spool/news/.uindex`

index of netnews users

`/usr/spool/news/.nindex`  
index of news articles

`/usr/spool/news/.seq`  
sequence number of last article

`/usr/spool/news/.history`  
list of all articles ever seen

`/usr/spool/news/.sys`  
system subscription list

**SEE ALSO**

[news\(7\)](#), [readnews\(7\)](#), [netnews\(5\)](#)

**NAME**

news – print news items

**SYNOPSIS**

news [ -a ] [ -n ] [ -s ] [ *item ...* ]

**DESCRIPTION**

When invoked without options, this simple local news service prints files that have appeared in **/usr/news** since last reading, most recent first, with each preceded by an appropriate header. The time of reading is recorded. The options are

- a** Print all items, regardless of currency. The recorded time is not changed.
- n** Report the names of the current items without printing their contents, and without changing the recorded time.
- s** Report the number of current items.

Other arguments select particular news items.

If an interrupt is received during a news item, the next item is started immediately. Another interrupt within a second of the first causes the program to terminate.

To post a news item, create a file in

You may arrange to receive news automatically by registering your mail address in A daemon mails newly posted news items to all addresses on the list.

**FILES**

/usr/news/\*  
\$HOME/news\_time date of last read news  
/usr/lib/subscribers

**SEE ALSO**

*calendar(1), readnews(7)*



**NAME**

findauthor, papers, makepaper – consult database of locally authored papers

**SYNOPSIS**

**findauthor** *person*

**papers** *person*

**makepaper** *papername*

**DESCRIPTION**

*Findauthor* produces a shell *cd* command to set the current directory to the place where papers by *person* (a login name or a last name) are stored.

*Papers* lists the names and titles of the papers stored under *person*.

*Makepaper* produces *troff*(1) output for *papername*.

The database is stored in file system File `/n/bowell/pap/Titles` lists titles, authors and installation date of papers in the database. Many papers have been preprocessed for quick presentation of figures and equations with *reader*(9) These papers are stored in directories named *papername.d*; other papers are stored as single files.

**EXAMPLES**

**‘findauthor aho‘**

Change to directory of Aho’s works.

**FILES**

`/n/bowell/pap/Titles`

titles, authors and installation dates

`/n/bowell/pap/*org`

membership list

`/n/bowell/pap/center/department/author/papername[d]`

**SEE ALSO**

*reader*(9) *docsubmit*(1), *troff*(1), *doctype*(1)

**BUGS**

*Makepaper* depends on *doctype*(1) to determine what preprocessors to run.

*Makepaper* does not know the author’s original arguments to *refer*, so papers are produced with *refer*’s default arguments.

*Makepaper* does not work with non-*troff* formatters such as *tex*(1) or *monk*(1).

**NAME**

polypic, polypr – database of polyhedra

**SYNOPSIS**

**polypic** [ **-phfis** ] *solid* ...

**polypr** [ **-cfhnpsifile** ] *solid* ...

**DESCRIPTION**

*Polypic* outputs a picture of the planar nets for the specified solids in *plot(5)* format. The options are:

- f** Print the face numbers.
- h** Print the hinge numbers.
- i file** Use *file* instead of the normal database.
- s** Produce an orthogonal view of the 3D solid rather than the net.
- p** Produce *pic(1)* output.

*Polypr* gives more general access to the data. By default, all the data is output. If no solid is specified, all the solids in the database are output. The options are:

- n** Print only the database index number and the name.
- h** Print only a header line with name and number of faces and hinges.
- f** Print only the data for the flat net.
- s** Print only the data for the 3D solid.
- i file** Use *file* instead of the normal database.
- c** Produce output for Tom Duff's polygon renderer (implies **-s**).
- p** Produce useful parameters for a perspective view; implies **-c**.

For both *polypic* and *polypr*, the solid may be specified by either its index number or by any prefix of the name. Ambiguities are resolved by database order.

**FILES**

/usr/include/poly.h

/usr/lib/polyhedra

**SEE ALSO**

*plot(1)*, *poly(5)*

A. G. Hume, 'Exact Descriptions of Regular and Semi-regular Polyhedra and Their Duals' *Computing Science Technical Report 130*, AT.T Bell Laboratories, November, 1986

**BUGS**

Not all the polyhedra have valid 3D data.

Option **-s** doesn't work yet.

**NAME**

postnews – submit netnews articles

**SYNOPSIS**

**postnews** [ *file* ]

**DESCRIPTION**

*Postnews* submits the *file* as a [readnews\(7\)](#) article. It prompts for title, primary newsgroup, and other newsgroup distribution. Good manners decree an informative title and accurate, minimal, distribution.

The names of newsgroups are relative pathnames of directories depending from with slashes replaced by dots.

If no file is specified, *postnews* invokes an editor specified by the environment variable **EDITOR** (default `vi`). The editor's buffer is initialized with header information, which may be changed. The text of the article may be appended.

**SEE ALSO**

[readnews\(7\)](#)

BSD manual for more sophisticated uses, such as posting news from a program.

**BUGS**

The editor default is distinctly nonclassical.

**NAME**

pq – telephonet directory assistance

**SYNOPSIS**

**pq** [ *option ...* ] *query*

**DESCRIPTION**

*Pq* queries 'Post' directory assistance computers for information from the AT.T phone book. A normal query is a name, in the form *first.middle.middle2.last.suffix*, where only *last* is required. Earlier names may be truncated, even to a null string, and extra punctuation may be dropped. Other fields can be specified, in the form *attribute=value*, with fields separated by */*. The attributes are

**pn**

**name** personal (full) name; for prefix match on last name, append \* or ...; for phonetic search, append or prepend ?

**first, middle, middle2, last, suffix**

parts of name, prefix-matched except for suffix

**pid, ssn**

personnel identification number, social security number

**org**

organization code

**tl**

title: abbreviated or prefix-matched; e.g. **tl=dh, tl=dep.he**

**tel**

phone number: (908)582-6050, punctuation optional, parts may be omitted from left

**area, exch, ext**

parts of phone number

**loc**

location code; for prefix match, append \*

**room, street, city, state, zip**

parts of address, prefix-matched except for state

**ema**

email address; for prefix match, append \*

**multi**

which of multiple addresses for one employee; e.g. **multi=2**

The options are

**-l** The query is a location code.

**-o format**

Provide output in the specified *format*, a string like that of *printf(3)*, with format codes being attribute names.

**EXAMPLES**

**pq penzias**

**pq a.a.penzias**

**pq ema=research!aap**

Three ways to find a person.

**pq loc=mh/room=2b519**

Find members of an office.

**pq -l mt**

Find information about a location.

**pq -o "24pn 10org 6loc 6room 12tel ema" a.a.penzias**

The default output format.

**FILES**

\$POST/lib/dispatch

directory configuration file

**SEE ALSO**

tel (7)

**NAME**

qns – query name server

**SYNOPSIS**

**qns** [ **-n** *server* ] [ *request* ]

**DESCRIPTION**

*Qns* retrieves information from a database of naming information. It is used by *rsh* and *rlogin* (see [dcon\(1\)](#)) to translate names to internet addresses and by *mail(1)* to route electronic mail.

Entries in the database consist of one or more *value,type* pairs or simple *values*. A simple *value* declares the name of some entity. An entry may contain no name or several, and different entries containing the same name need not refer to the same entity.

These types are used:

- dk**     *Value* is a Datakit address.
- in**     *Value* is a numeric IP address.
- dom**    *Value* is an internet domain name.
- tel**    *Value* is a telephone number, possibly prefixed by a *uucp Dialcodes* name.
- org**    *Value* is an organization name.
- svc**    *Value* names a service.
- origin** *Value* must be **local**, for sorting by ‘distance’; see below.

The following entries describe an entity **research** with a Datakit address, an IP address and domain name, belonging to organization **att**, and offering the **uucp** service:

```
192.11.4.55,in research research.astro.nj.att.com.,dom att,org
research nj/astro/research,dk uucp,svc att,org
```

*Qns* prints database entries that match *requests*. If a *request* is supplied on the command line, *qns* prints the answer and exits; otherwise it reads and answers requests from the standard input until end-of-file. The possible requests are:

- set** *key...*  
Print every entry matching all *keys*.
- value** *tlist key...*  
Examine entries matching the *keys* until a pair with type *tlist* is found; print the matching value and stop. *Tlist* may be a single type, or several separated by |.
- reset** Cause the name server to reinitialize its database.
- help** Print a list of requests.
- quit** Exit *qns*.

A *key* is a *value,type* pair; an entry matches if it contains that pair. If *,type* is omitted, any pair with the specified *value* will do. A \* at the end of a *value* stands for an arbitrary suffix.

When a database search returns several entries with **dk** or **dom** types, and the database contains an entry with the conventional pair **local,origin**, the entries are sorted by increasing ‘distance’ from the **dk** or **dom** pairs in the **local,origin** entry. Datakit names in the same exchange are nearer than names in different exchanges in the same area, which are nearer than names in different areas. Domain names matching to four levels of domain hierarchy are nearer than names matching to only three levels, and so on.

*Qns* expects to reach the name server [ns\(8\)](#); option **-n** points it at service *server* instead.

**SEE ALSO**

[ipc\(3\)](#), [ns\(8\)](#)

**NAME**

checknews, readnews – read netnews articles

**SYNOPSIS**

**readnews** [ **-a** *date* ] [ **-n** *newsgroup ...* ] [ **-t** *title ...* ] [ **-lprxhfUM** ] [ **-c** [ *command* ] ]

**readnews -s**

**checknews** [ **ynqve** ] [ *readnews-options* ]

**DESCRIPTION**

*Readnews* prints unread articles that have arrived via the informal, worldwide ‘netnews’ network. Without arguments it prints unread articles from newsgroups to which you subscribe. The options are:

**-M** An interface to *Mail(A)* *mail(1)*-like interface.

**-c** *command*

Articles are written to a temporary ‘mailbox’ and the *command* (e.g. *mail -f* ) invoked, with the mailbox in place of *.*. A missing *command* gets something like *mail(1)*.

**-p** Articles are sent to the standard output, no questions asked.

**-l** Titles only. The file *.newsrc* will not be updated.

**-r** Print articles in reverse order.

**-f** No followup articles.

**-h** Printed in a briefer format.

**-u** Update file *.newsrc* every 5 minutes.

**-n** *newsgroup ...*

Select articles that belong to *newsgroups*.

**-t** *titles*

Select articles whose titles contain one of the *title* strings.

**-a** *date*

Select articles that were posted since *date*; a missing *date* means the beginning of time.

**-x** Ignore select previously read as well as unread articles.

**-s** Print subscription list.

The file or a file specified by environment variable **NEWSRC**, tells what topics you are interested in and what you have read. If *.newsrc* contains a line starting with options (left justified, continued by trailing \), or if the environment variable **NEWSOPTS** is present, options are taken from there as well as the command line. In case of conflict, an option on the command line take precedence, followed by *.newsrc* and finally **NEWSOPTS**.

*Readnews* invokes some other programs to perform services. To reply to a news item it uses *mail(1)* or an alternate in environment parameter **MAILER**. It paginates with *p(1)*, or an alternate in **PAGER**. **PAGER** is a command, perhaps containing *as* in option **-c**, or empty for no pagination.

The default and *mail* interfaces support the following commands, and prompt with common alternatives: a newline accepts the first one. For example, [*ynq*] proposes yes, no, and quit; newline gets yes.

**y** Yes. Print current article and go on to next.

**n** No. Skip the current article. (In *mail* interface, it means **y**.)

**q** Quit; update

**c** Cancel the article. Only the author or the super-user can do this.

**r** Reply. Reply to article’s author via mail. You are placed in *mail*, or an alternate in environment parameter **EDITOR**, with a header constructed from the article. You may change or add headers. Add your reply after the blank line. Upon exit the message is mailed.

**rd** Reply directly. You are placed in **MAILER** (*mail* by default). Type the text of the reply and then control-D.

**f** *title* Submit a followup article. If you omit the title, *readnews* generates an appropriate one. You will be placed in your **EDITOR** to compose the followup.

- fd** Follow up directly. This is like **f**, but does not construct headers.
- N** *newsgroup*  
Go to the named *newsgroup*, or the next newsgroup if none is named
- s** *file* Save. Append the article to *file*. The default is If *file* is not a full pathname, it is taken relative to **HOME**, overridden by environment parameter **NEWSBOX**. If the first character of *file* is |, the rest is taken as the name of a program, into which the article is piped.
- #** Report the name and size of the newsgroup.
- e** Erase. Forget that this article was read.
- h** Print a more verbose header.
- H** Print a very verbose, complete header.
- U** Unsubscribe from this newsgroup and go on to the next.
- d** Read a digest. Presents a digest as separate articles.
- D** *number*  
Decrypt a Caesar cipher (usually used to obscure off-color material in net . jokes). The rotation is normally determined line-by-line from character frequencies. If this fails, an explicit *number* (usually 13) may be given.
- v** Print the current version of the news software.
- !** Shell escape.
- number*  
Go to article *number*.
- ±n** Skip *n* articles, 1 by default. The articles skipped are recorded as 'unread'.
- Go back to last article. This is a toggle, typing it twice returns you to the original article.
- x** Exit. Like **q** except that .newsrc is not updated.
- X** *system*  
Transmit article to the named system.

A – following **c**, **f**, **fd**, **r**, **rd**, **e**, **h**, **H**, or **s** refers to the previous article: **r -** is the normal way to reply to a just-read article when the next one is being offered.

*Checknews* reports whether there is news present, with options:

- y** Report only if news is present (default).
- n** Report only if news is absent.
- q** Turn off reports; nonzero exit status indicates news.
- v** Show the name of the first newsgroup containing unread news.
- vv** Explain any claim of new news, useful if checknews and readnews disagree.
- e** Execute *readnews* if there is news.

## EXAMPLES

```
readnews Read all unread articles.
readnews -n net.langs.c -a last thursday Print every unread article about C since last Thursday.
readnews -p >/dev/null . Discard all unread news: useful after returning from a long trip.
readnews -c "ed" -l Invoke ed(1) on a file containing the titles of all unread articles.
```

## FILES

```
/usr/spool/news/newsgroup/number
    News articles
/usr/lib/news/active
    Active newsgroups and numbers of articles
/usr/lib/news/help
    Help file for default interface
$HOME/.newsrc
```

## SEE ALSO

*postnews(7)*, *Mail(A)*

## BUGS

*Readnews* is baroque; many users prefer to browse among the files in

**NAME**

scat – sky catalogue

**SYNOPSIS**

scat

**DESCRIPTION**

*Scat* looks up items in catalogues of objects outside the solar system. Items are read, one per line, from the standard input, and their descriptions or cross-index listings (suitable for input to *scat*) are printed on the standard output. An item is in one of the following formats:

**ngc1234**

Number 1234 in the Revised New General Catalogue of Nonstellar Objects. The output identifies the type **pn**=planetary nebula, **gc**=globular cluster, **oc**=open cluster, **dn**=diffuse nebula or **nc**=nebular cluster), possibly contained within the Large Magellanic Cloud (**in lmc**) or Small Magellanic Cloud (**in smc**), its position in 2000.0 coordinates and galactic coordinates, and a brief description.

**sao12345**

Number 12345 in the Smithsonian Astrophysical Star Catalogue. Output identifies the visual and photographic magnitudes, 2000.0 coordinates, proper motion, spectral type, multiplicity and variability class, and HD number.

**m4** Catalog number 4 in Messier's catalog. The output is the NGC number.

"alpha umi"

Star names are provided in double quotes. Known names are the Greek letter designations, proper names such as Betelgeuse, and bright variable stars. Constellation names must be the three-letter abbreviations. The output is the SAO number. For non-Greek names, SAO numbers and names are listed for all stars with names for which the given name is a prefix.

**12h34m -16**

Coordinates in the sky are translated to the nearest 'patch', approximately one square degree of sky. The output is the coordinates identifying the patch, the constellations touching the patch, and the NGC and SAO objects in the patch.

**c umi** Gives voluminous output consisting of the patches covering the named constellation.

**FILES**

/n/kwee/usr/rob/sky2/\* .sky

**SEE ALSO**

[sky\(7\)](#)

/n/kwee/usr/rob/sky2/constelnames for the three-letter abbreviations of the constellation names.

**BUGS**

The database is fine, but the program is feeble.

Coordinates printed by the program in the listings for SAO and NGC objects are not understood as input.



**NAME**

sky – astronomical ephemeris

**SYNOPSIS**

sky [ -l ]

**DESCRIPTION**

*Sky* predicts the apparent locations of the sun, moon, visible planets, and stars brighter than magnitude 2.5. It reads one line from the standard input to obtain the desired time expressed as five numbers: year, month, day, hour, and minute in GMT. An empty line means now. Each object is printed with astronomical coordinates, azimuth-elevation coordinates relative to Murray Hill, NJ, and magnitude. For variable stars the maximum magnitude is printed with \*.

Option **-l** causes *sky* to prompt for another viewing location.

Standard astronomical effects are accounted for: nutation and precession of the equinox, annual aberration, diurnal parallax, and proper motion. Atmospheric effects (extinction and refraction) are not calculated, nor is perturbation of the earth by other bodies.

In ephemeris (slightly different from civil) time, the program yields positions of sun, moon, and stars good to a few tenths of an arc-second. Planets are good to a few seconds.

**FILES**

/usr/lib/startab

**SEE ALSO**

[scat\(7\)](#)

*American Ephemeris and Nautical Almanac* and *Explanatory Supplement to the American Ephemeris and Nautical Almanac*

**NAME**

submit – install document in database

**SYNOPSIS**

**submit** *name* [ *option ...* ]

**DESCRIPTION**

*Submit* ships a paper to from whence the paper will be installed under the given *name* in a database of locally-authored papers. The paper then becomes available for printing with *makepaper* (see [papers\(7\)](#)) or for on-line inspection with [reader\(9\)](#)

*Submit* expects to be run from a directory that contains the files for the paper. If only the *name* argument is given, *submit* assumes that the paper is maintained by [make\(1\)](#) and follows instructions in the makefile, which must be present. The options are

**-mk** Use [mk\(1\)](#) instead of *make*.

**-M** *makefile*  
Use the given makefile.

**-m** *target*  
Make the given target.

**-s** *script arg ...*  
Use a shell *script* instead of a makefile.

**-i** *file ...*  
Format the paper with [troff\(1\)](#) from the listed files, using macro packages and preprocessors as determined by [doctype\(1\)](#). Do not use a makefile or a shell script.

**-t** *file ...*  
Format the paper with [tex\(1\)](#) from the listed files. Do not use a makefile or a shell script. Only the basenames of the files need be given.

**-tatex** The paper uses *tatex* (see [latex\(7\)](#)). This option is used only in conjunction with **-t**.

**FILES**

/n/bowell/pap/spool/\*

**DIAGNOSTICS**

*Submit* complains if it can't find input or if it encounters a *troff* `.sy` command.

**SEE ALSO**

[reader\(9\)](#) [troff\(1\)](#), [papers\(7\)](#)

**BUGS**

*Submit* fails if the *troff* preprocessors are run in a 'hidden' way by calling a program (or shell script).

When *troff* preprocessor output (i.e. *pic*, *grap*, *tbl*, *tped*, *ideal*) is included directly in the paper or with `.so` commands, an attempt is made to find the preprocessor input file. If this attempt fails, *submit* refuses to ship the paper.

Automatic installation fails if the system can't find the author's name or the author is not in the file containing the center organization.

Doesn't work with [monk\(1\)](#).



**NAME**

telno – retrieve from bell labs phone book

**SYNOPSIS**

**telno** [ *datum ... datum* ]

**DESCRIPTION**

If there are no arguments on the command line, *telno* reads its arguments, one to a line, from the standard input. Arguments are names, phone numbers, or organization numbers, possibly preceded by a keyword and an equal sign. The first character of a name must be a letter, of a phone number or organization number a digit. Otherwise regular expressions *a la grep(1)* are accepted. Names are prefixes of last names, or the last name followed by a comma and one initial.

Giving a name, a phone number, or an organization number (*org=127*) produces white pages information.

**SEE ALSO**

[tel\(7\)](#)

**DIAGNOSTICS**

*too long* means that the given name has more letters than any name in the phone book. (All the names are truncated to 8 characters before the data arrives on the machine.) *None* is printed when there are no matches.

**BUGS**

The phone book is badly out of date.

**NAME**

town – gazetteer of US places

**SYNOPSIS**

**town** [ *place* ]

**DESCRIPTION**

*Town* produces information about the *place*, which is the name of a US town, possibly followed by a comma and a two-letter state abbreviation. If no *place* is given, place names are read one per line from the standard input.

The information produced is latitude, longitude, approximate population, a forecast from [weather\(7\)](#), and a recent item from [apnews\(7\)](#).

**FILES**

/usr/spool/town/ustown\* gazetteer in [cbt\(1\)](#) format

**SEE ALSO**

[av\(A\)](#), [dict\(7\)](#)

**NAME**

units – conversion program

**SYNOPSIS**

**units**

**DESCRIPTION**

*Units* converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
* 2.54
/ 0.393701
```

Quantities are specified using the following grammar:

```
Unit:  Empty | Unit Term | Unit / Term
Term:  Number | Name | (Unit) |
         square Term | sq Term | cube Term | cu Term |
         Term ^ Number
```

Numbers are specified in the form expected by [atof\(3\)](#). Names are maximal strings of non-numeric, non-punctuation characters. Powers are indicated by the ^ operator or by the words `square` and `cube`. Parentheses alter grouping. The empty unit has value 1. Terms are multiplied together unless connected by / for inversion, e.g. 15 pounds force/sq in.

Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

The pound is a unit of mass. Compound names are run together, e.g. `lightyear`. British units that differ from their US counterparts are prefixed thus: `brgallon`. Currency is denoted `belgiumfranc`, `britainpound`, etc.

A response of ? to 'You want:' displays all known units conformable with the 'You have:' quantity.

The complete list of units can be found in `/usr/lib/Units` and

**FILES**

```
/usr/lib/Units
/n/alice/usr/td/Monetary.units
/usr/lib/Units.bin
```

**BUGS**

Since *units* does only multiplicative scale changes, it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit.

Currency conversions are only as accurate as the most recent report of foreign exchange prices from the AP wire.

**NAME**

weather – conditions and forecast by town

**SYNOPSIS**

**weather** [ **-h** ] [ **-m** ]

**DESCRIPTION**

*Weather* reports current weather conditions and a forecast for towns in the contiguous United States. It reads town names from the standard input, one per line. Each input line should be in the style of

walla

where the state is given as the 2-letter Post Office abbreviation.

The information provided is the temperature, humidity, whether or not it is raining or snowing, and an indication of cloudiness and visibility if it is not. If the wind is above 10 knots the wind speed and direction are given. Up to three forecasts are given (assuming they can be found within a 4×4 degree latitude-longitude square): a forecast of high and low temperatures plus probability of precipitation taken from some numerical model of the atmosphere, a general area forecast and a marine forecast (if near the coast and if the **-m** argument is given).

The command **weather -h** is equivalent to *weather* followed by a single input line of *murray hill, nj*.

**SEE ALSO**

*avw(A)*

**NAME**

wx - get weather information

**SYNOPSIS**

wx [ **-cemnoOs** ] [ **-a state** ]

**DESCRIPTION**

*Wx* writes the local (greater New York and Philadelphia) weather forecast.

The following options are available:

- e**                Print only the extended forecasts.
- m**                Print the marine weather and forecast, which gives coastal and offshore information (winds, tides, wave heights, visibilities, etc.) for an area from Watch Hill, R.I. to Manasquan, N.J.
- c**                Print a summary of temperatures and weather for selected U.S. cities.
- n**                Print the national weather summary.
- s**                Print ski conditions for New York and New England.
- oO**              Print the previous (**-o**) and oldest available (**-O**) national weather summaries.
- a state**        (where *state* is a two-letter, lower case state abbreviation) Print (cat) an area forecast (if available) for any state requested in the continental U.S.; e.g. *wx -a mi* will print the state forecast for Michigan. The required 2-letter abbreviation for any state may be found by typing, *wx -a ?*.

Except for *wx -a*, all information is printed under the *p* command. For details of *p*, refer to the manual page.

**FILES**

/usr/sbin/wx

/usr/pub/weather/\*                      weather and forecast files, updated daily

**BUGS**

What comes out is only as good as what went in. If the information you are requesting was not sent over the weather line, you will get no meaningful output. If information over the line came in with faulty separator codes, you will get strange, often cryptic, output.



**NAME**

11as, 11cc, 11ld, 11nm, 11ranlib, 11reloc, 11size, 11strip – pdp11 support

**DESCRIPTION**

These programs do cross-compiling and related support functions for the DEC PDP-11 family of computers. Their descriptions correspond closely with those of similarly named programs in Section 1 of this manual.

**FILES**

11a.out output file

**NAME**

ac – login accounting

**SYNOPSIS**

*/etc/ac* [ *option ...* ] [ *person ...* ]

**DESCRIPTION**

*Ac* prints the total connect time recorded in the accounting file, If *persons* are named, only those login names are considered. The options are

**-w** *file* Use *file* instead of – means the standard input.

**-p** Print total connect time for each user.

**-d** Print totals for each day.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login(8)*, provided it exists. To start accounting, create it with length 0. When accounting is turned on, the file grows without limit. It is prudent periodically to process the data and truncate the file.

**FILES**

*/usr/adm/wtmp*

**SEE ALSO**

*init(8)*, *sa(8)*, *login(8)*, *utmp(5)*, *who(1)*

**NAME**

adduser – procedure for adding new users

**DESCRIPTION**

A new user must choose a login name, which must not already appear in */etc/passwd*. An account can be added by editing a line into the *passwd* file; this must be done with the *passwd* file locked e.g. by using *vipw(8)*.

A new user is given a group and user id. User id's should be distinct across a system, since they are used to control access to files. Typically, users working on similar projects will be put in the same group. Thus at UCB we have groups for system staff, faculty, graduate students, and a few special groups for large projects. System staff is group "10" for historical reasons, and the super-user is in this group.

A skeletal account for a new user "ernie" would look like:

```
ernie::235:20:. Kovacs,508E,7925,6428202:/mnt/grad/ernie:/bin/csh
```

The first field is the login name "ernie". The next field is the encrypted password which is not given and must be initialized using *passwd(1)*. The next two fields are the user and group id's. Traditionally, users in group 20 are graduate students and have account names with numbers in the 200's. The next field gives information about ernie's real name, office and office phone and home phone. This information is used by the *finger(1)* program. From this information we can tell that ernie's real name is "Ernie Kovacs" (the . here serves to repeat "ernie" with appropriate capitalization), that his office is 508 Evans Hall, his extension is x2-7925, and this his home phone number is 642-8202. You can modify the *finger(1)* program if necessary to allow different information to be encoded in this field. The UCB version of *finger* knows several things particular to Berkeley – that phone extensions start "2-", that offices ending in "E" are in Evans Hall and that offices ending in "C" are in Cory Hall.

The final two fields give a login directory and a login shell name. Traditionally, user files live on a file system which has the machines single letter *net(1)* address as the first of two characters. Thus on the Berkeley CS Department VAX, whose Berknet address is "csvax" abbreviated "v" the user file systems are mounted on "/va", "/vb", etc. On each such filesystem there are subdirectories there for each group of users, i.e.: "/va/staff" and "/vb/prof". This is not strictly necessary but keeps the number of files in the top level directories reasonably small.

The login shell will default to "/bin/sh" if none is given. Most users at Berkeley choose "/bin/csh" so this is usually specified here.

It is useful to give new users some help in getting started, supplying them with a few skeletal files such as *.profile* if they use "/bin/sh", or *.cshrc* and *.login* if they use "/bin/csh". The directory "/usr/skel" contains skeletal definitions of such files. New users should be given copies of these files which, for instance, arrange to use *tset(1)* automatically at each login.

**FILES**

<i>/etc/passwd</i>	password file
<i>/usr/skel</i>	skeletal login directory

**SEE ALSO**

*passwd(1)*, *finger(1)*, *chsh(1)*, *chfn(1)*, *passwd(5)*, *vipw(8)*

**BUGS**

User information should be stored in its own data base separate from the password file.

**NAME**

`analyze` – Virtual UNIX postmortem crash analyzer

**SYNOPSIS**

```
/etc/analyze [ -s swapfile ] [ -f ] [ -m ] [ -d ] [ -D ] [ -v ] corefile [ system ]
```

**DESCRIPTION**

*Analyze* is the post-mortem analyzer for the state of the paging system. In order to use *analyze* you must arrange to get a image of the memory (and possibly the paging area) of the system after it crashes (see [crash\(8\)](#)).

The *analyze* program reads the relevant system data structures from the core image file and indexing information from **/vmunix** (or the specified file). to determine the state of the paging subsystem at the point of crash. It looks at each process in the system, and the resources each is using in an attempt to determine inconsistencies in the paging system state. Normally, the output consists of a sequence of lines showing each active process, its state (whether swapped in or not), its *p0br*, and the number and location of its page table pages. Any pages which are locked while raw i/o is in progress, or which are locked because they are *intransit* are also printed. (Intransit text pages often diagnose as duplicated; you will have to weed these out by hand.)

The program checks that any pages in core which are marked as not modified are, in fact, identical to the swap space copies. It also checks for non-overlap of the swap space, and that the core map entries correspond to the page tables. The state of the free list is also checked.

Options to *analyze*:

- D** causes the diskmap for each process to be printed.
- d** causes the (sorted) paging area usage to be printed.
- f** which causes the free list to be dumped.
- m** causes the entire coremap state to be dumped.
- v** (long unused) which causes a hugely verbose output format to be used.

In general, the output from this program can be confused by processes which were forking, swapping, or exiting or happened to be in unusual states when the crash occurred. You should examine the flags fields of relevant processes in the output of a [pstat\(8\)](#) to weed out such processes.

It is possible to look at the core dump with *adb* if you do

```
adb /vmunix /vmcore
/m 80000000 #ffffff
```

which fixes the map of *vmcore* so that symbols in data space will work. Note that the debugger is looking at the physical memory at the point of crash; you will have to determine which pages of physical memory virtual pages are in if you wish to look at them. If *analyze* says that a processes page tables are in page 218 (hex of course), then you can look at them by looking at address 0x80043000 in the dump, i.e. “80043000,80/X” will print the page of page tables.

**FILES**

`/vmunix` default system namelist

**SEE ALSO**

[ps\(1\)](#), [crash\(8\)](#), [pstat\(8\)](#)

**AUTHORS**

Ozalp Babaoglu and William Joy

**DIAGNOSTICS**

Various diagnostics about overlaps in swap mappings, missing swap mappings, page table entries inconsistent with the core map, incore pages which are marked clean but differ from disk-image copies, pages which are locked or intransit, and inconsistencies in the free list.

It would be nice if this program analyzed the system in general, rather than just the paging system in particular.

**NAME**

arcv – convert archives to new format

**SYNOPSIS**

*/etc/arcv* file ...

**DESCRIPTION**

*Ar*cv converts archive files (see *ar(1)*, *ar(5)*) from 32v and Third Berkeley editions to a new portable format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177545 at the start; new archives have a first line “!<arch>”.

**FILES**

*/tmp/v\**, temporary copy

**SEE ALSO**

*ar(1)*, *ar(5)*

**NAME**

*arff* – read RT11 files

**SYNOPSIS**

**arff** [ *key* ] [ *name ...* ]

**DESCRIPTION**

*Arff* saves and restores files from an RT11 volume, such as the VAX console floppy or tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file names specifying which files are to be dumped or restored.

RT11 filenames must be chosen from the character set **a-z0-9.**. Unix filenames are trimmed to the last pathname element; upper-case letters are folded to lower-case.

The *key* must include one of the following letters:

- r** The named files are replaced where found in the RT11 volume, or added taking up the minimal possible portion of the first empty spot.
- x** The named files are extracted from the RT11 volume.
- d** The named files are deleted from the volume. The resulting empty spots are coalesced where possible.
- t** A table of contents for the RT11 volume is printed. If filenames are given, they are echoed if found.

The following modifiers may be added to the *key*:

- v** Normally *arff* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the file than just the name.
- f** causes *arff* to use the next argument as the name of the archive instead of */dev/floppy*.
- m** causes *arff* not to use the mapping algorithm employed in interleaving sectors around a floppy disk. In conjunction with the **f** option it may be used for extracting files from non-floppy sources, such as the VAX-11/750 console cassette.

**FILES**

*/dev/floppy*

**AUTHORS**

Keith Sklower, Richard Tuck

**BUGS**

Floppy errors are handled ungracefully.

The program is too floppy-dependent. Mapping belongs in the device driver, or at least shouldn't be the default.

**NAME**

asd – automatic software distribution

**SYNOPSIS**

**/usr/lib/asd/cdaemon**  
**/usr/lib/asd/dkinstall**  
**/usr/lib/asd/mkspool**  
**/usr/lib/asd/rmllocks**  
**/usr/lib/asd/udaemon**

**DESCRIPTION**

These programs constitute the innards of the automatic software distribution system invoked by [ship\(8\)](#).

*Mkspool* creates an ASD spool directory for its invoker (if necessary), puts a file named *dummy* in the directory to prevent *cdaemon* from deleting it, and prints the directory's name.

*Udaemon* examines its invoker's ASD spool directory in lexical order for things to do. To forestall multiple daemons, it first makes an empty file named *L.pid* in the spool directory and tries to link to it a file. If the link fails, *L.pid* is removed and *udaemon* exits.

Shipping instructions appear in pairs of files with *.s* and *.d* suffixes. *Udaemon* examines each status (*.s*) file for destination names and places a network call to send the corresponding data (*.d*) file to *dkinstall* at each destination. *Dkinstall* verifies the data and gives it to *inspkg*, see [mkpkg\(8\)](#).

A status file contains zero or more entries, each of which has one or more lines. The first line of an entry is a network address, with default network **dk** and default service **asd**, possibly followed by a blank and a failure report. An entry with destination # is a comment and is ignored. Lines after the first begin with a tab, and contain output generated by the entry. If an entry has output, it is considered complete and will be processed no further.

*Udaemon* scans each status file once, and attempts to send the corresponding data file to the destination for each incomplete entry. If the attempt fails, a failure report is appended to the entry, and it remains incomplete. If the attempt succeeds, or the failure is severe, an error message or output from *dkinstall* is appended to the entry, which makes the entry complete. If an entry is completed successfully but there are no output lines, the entry is deleted.

Each instance of *udaemon* remembers which network addresses failed with non-severe errors; entries with the same address in later status files are given failure status 'deferred for sequence.'

If at least one additional entry was completed, *udaemon* sends the new status file to the owner by [mail\(1\)](#) after the whole file has been scanned. If no incomplete entries remain, both status and data files are removed.

*Cdaemon* examines every subdirectory of deletes empty directories and, impersonating the owner, invokes *udaemon* for nonempty directories. *Cdaemon* should be run regularly by [cron\(8\)](#) with super-user permissions.

*Rmllocks* removes all lock files in subdirectories of It should be called from [rc\(8\)](#).

Entries in an ASD spool directory may be made without regard to locks provided that everything is done in the right sequence: (1) call *mkspool*; (2) create the data file; (3) create the status file under a temporary name; (4) rename the status file to end with *.s*; (5) remove the *dummy* file, if present; and (6) call *udaemon*.

Because *mail* will not send an empty file, a status file must have a comment entry if acknowledgment is desired after a successful *udaemon* run.

**FILES**

**/usr/spool/asd/userid** user's ASD spool directory

**SEE ALSO**

[mkpkg\(8\)](#), [ship\(8\)](#)

**NAME**

backup – backup client administration

**SYNOPSIS**

**/usr/lib/backup/sel**

**/usr/lib/backup/fcheck** *maxsize maxdays files ...*

**/usr/lib/backup/act** [ *stat* ]

**DESCRIPTION**

These programs select and back up files to the incremental file backup system, [backup\(1\)](#).

*Sel* prints on the standard output a list of filenames that might need to be backed up. The initial version picks out files that have been changed in the past few days, skipping huge files and eliding boring names like **core**. *Sel* is a shell script; the local administrator is expected to customize it.

*Fcheck* is a fast, specialized file scanning program, used by *sel*. It examines each of the *files*, descending into directories, and prints the name of each file that has been changed in the last *maxdays* days and is smaller than *maxsize* kilobytes. Symbolic links are followed when presented as arguments, examined but not followed otherwise.

*Act* reads a list of filenames from the standard input. It searches the backup database [backup\(5\)](#) for the current version of each file, and backs up files that aren't registered.

By default, *act* sends errors by [mail\(1\)](#) to user **backup**. If the *stat* argument is non-empty, errors and additional comforting chatter are printed on the standard output instead.

One way to request automatic backups is to run

```
/usr/lib/backup/sel | /usr/lib/backup/act
```

regularly from [cron\(8\)](#). Particular files may be backed up by hand at any time by running *act* with a list of filenames. There are no special permissions involved; any user may run *act*.

**SEE ALSO**

[backup\(1\)](#), [backup\(5\)](#)

A. Hume, 'The File Motel: An Owner's Manual', this manual, Volume 2



**NAME**

chown – change owner or group

**SYNOPSIS**

*/etc/chown* *owner,group file ...*

**DESCRIPTION**

*Chown* changes the owner of the *files* to *owner* and the groupid to *group*. Either *owner* or *group* may be omitted to leave the owner or groupid unchanged.

*Owner* may be either a decimal userid or a login name found in *Group* may be either a decimal groupid or a group name found in

The owner of a file may change its group to that of the current process. Other changes are restricted to the super-user.

**FILES**

*/etc/passwd*

*/etc/group*

**SEE ALSO**

*chown* in *chmod(2)*, *passwd(5)*, *chmod(1)*, *chdate(1)*

**NAME**

chuck – a file system checking program

**SYNOPSIS**

*/etc/chuck* [ *option ...* ] *device*

*/etc/chuck -M* *blocks device*

*/etc/upchuck* [ *-w* ] [ *-p program* ]

**DESCRIPTION**

*Chuck* checks and optionally repairs the file system on the named *device*. It is normally invoked by *upchuck* by *rc(8)* during reboots. The *-w* flag to *upchuck* is passed on to *program*. If *program* is not present, the default is (Try, as super-user, */etc/upchuck -p /bin/echo* to see the normal arguments to *chuck*.) If *upchuck* can read the raw version of *device*, it will, except for the root file system.

The options are

*-w* Try to do standard repairs.

*-b* *blocksize*  
Specify file system block size; default is 4096.

*-i* Interactive. Ask approval for each change.

*-I* *inode ...*

*-B* *block ...*  
Report on inodes or blocks specified by number.

*-v* Verbose. Give more commentary.

*Chuck* can also make a new file system: *chuck -M size device* makes a bitmapped file system (only) of *size* 4096-byte blocks on *device*. It asks approval before writing.

**FILES**

*/etc/fstab*

**SEE ALSO**

*fstab(5)*, *filsys(5)*, *fsck(8)*, *reboot(8)*

**BUGS**

*Chuck* does not replace real expertise. It will not automatically repair a file system with duplicate blocks. In complicated situations it may have to be run several times to get complete consistency.

It will not recover from I/O errors in reading the inodes, and does not yet extend `lost+found` when necessary.

It uses memory freely (about 12 bytes per file system block and 84 bytes per inode).

*Chuck* is still experimental, and acts the part. Error messages are usually inscrutable.

It believes even postposterous super-blocks and consequently can get core images.

**NAME**

`clri` – clear inode

**SYNOPSIS**

*/etc/clri special i-number ...*

**DESCRIPTION**

*Clri* writes zeros on the inodes with the decimal *i-numbers* on the file system in file *special*. After *clri*, any blocks in the affected file will show up as 'missing' in *icheck(8)*.

The inode becomes allocatable.

The primary purpose of this program is to remove a file which for some reason appears in no directory. If it is used to clear an inode which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the cycle is likely to be repeated.

*Clri* is a last resort; normally *fsck(8)* can do the necessary repairs.

**SEE ALSO**

*fsck(8)*, *icheck(8)*

**BUGS**

If the file is open, *clri* is likely to be ineffective.

**NAME**

config – configure a Unix kernel

**SYNOPSIS**

**/etc/config** [ *machine* ]

**DESCRIPTION**

*Config* generates files used to build a Unix kernel for the named *machine*. The working directory should be `/usr/sys/conf` or `/usr/sys/machine`; if the latter, *machine* may be omitted from the command.

A machine description is expected in *machine/conf*; a makefile, a number of header files, and some C and assembler programs are generated from the description.

The usual way to configure a new system is:

```
mkdir /usr/sys/newmach
cd /usr/sys/newmach
(create conf)
/etc/config
make
```

**FILES**

All these files are in the configuration directory.

`../conf/makefile`

makefile template

`../conf/files`

list of kernel source files

`files`

more sources specific to this machine

`../conf/devices`

list of device handlers

`devices`

more devices specific to this machine

`conf` machine description

**SEE ALSO**

[config\(5\)](#)

**BUGS**

At the moment, it's also necessary to create

**NAME**

cpp – C language preprocessor

**SYNOPSIS**

**/lib/cpp** [ *option ...* ] [ *ifile* [ *ofile* ] ]

**DESCRIPTION**

*Cpp* interprets preprocessor directives and does macro substitution for *cc(1)* and other compilers. The input *ifile* and output *ofile* default to standard input and standard output respectively.

The options are:

- P** Do not place line number markings in output.
- C** Do not remove comments.
- Uname**
- Dname**
- Dname=def**
- Idir** Same as in *cc(1)*. **-U** overrides **-D**.
- H** Report all included files on standard error file,.
- M** Attach modification date to file names in line number directives thus: *file@modtime*, where *modtime* is the integer number of seconds since the epoch.
- T** Truncate preprocessor symbols to eight characters.
- Ydir** Use *dir* instead of */usr/include* as the last resort in searching for include files.

The output file contains processed text sprinkled with lines that show the original input line numbering:

```
# linenumber "ifile"
```

The input language is as described in the reference, with a few additions:

The *# linenumber* marks placed in the output are accepted as an alternative to the official **#line** directive.

These symbols are predefined in various implementations:

```
ibm gcos os tss unix
interdata pdp11 u370 u3b u3b5 vax
RES RT
lint
```

Preprocessor formal parameters are recognized within quoted strings in the replacement text.

When comments are removed they are replaced by null strings; this unofficial feature makes it possible to construct identifiers by concatenation.

**FILES**

*/usr/include*  
standard directory for include files

**SEE ALSO**

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1988

**NAME**

crash – what happens when the system crashes

**DESCRIPTION**

This section explains what happens when the system crashes and how you can get a crash dump for analysis of non-transient problems.

When the system crashes voluntarily it prints a message of the form

panic: why i gave up the ghost

on the console, and then invokes an automatic reboot procedure as described in [reboot\(8\)](#). If the auto-reboot switch is off on the console, then the processor will simply halt at this point. Otherwise the registers and the top few locations of the stack will be printed on the console, and then the system will check the disks and (unless some unexpected inconsistency is encountered), resume multi-user operations.

The system has a large number of internal consistency checks; if one of these fails, then it will panic with a very short message indicating which one failed. In the absence of a dump, little can be done about one of these. If the problem recurs, you should arrange to get a dump for further analysis by running with auto-reboot disabled during normal working hours and then following the procedure described below.

The most common cause of system failures is hardware failure, which can reflect itself in different ways. Here are the messages which you are likely to encounter, with some hints as to causes. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

IO err in push

hard IO err in swap

The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

Timeout table overflow

ran out of bdp's

ran out of uba map

These really shouldn't be panics, but until we fix up the data structures involved, running out of entries causes a crash. If the timeout table overflows, you should make it bigger. If you run out of bdp's or uba map you probably have a buggy device driver in your system, allocating and not releasing UNIBUS resources.

KSP not valid

SBI fault

Machine check

CHM? in kernel

These indicate either a serious bug in the system or, more often, a glitch or failing hardware. For the machine check, the top part of the resulting stack frame gives more information. You can refer to a VAX 11/780 System Maintenance Guide for information on machine checks. If machine checks or SBI faults recur, check out the hardware or call field service. If the other faults recur, there is likely a bug somewhere in the system, although these can be caused by a flakey processor. Run processor microdiagnostics.

trap type d, code=d

A unexpected trap has occurred within the system; the trap types are:

reserved addressing mode

- |   |                                    |
|---|------------------------------------|
| 1 | privileged instruction             |
| 2 | BPT                                |
| 3 | XFC                                |
| 4 | reserved operand                   |
| 5 | CHMK (system call)                 |
| 6 | arithmetic trap                    |
| 7 | reschedule trap (software level 3) |
| 8 | segmentation fault                 |
| 9 | protection fault                   |

## 10 trace pending (TP bit)

The favorite trap type in system crashes is trap type 9, indicating a wild reference. The code is the referenced address. If you look down the stack, just after the trap type and the code are the pc and the ps of the processor when it trapped, showing you where in the system the problem occurred. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes, e.g. we have trapped with code 80000800 three times in six months as an instruction fetch went across this page boundary in the kernel but have been unable to find any reason for this to have happened.

## init died

The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

That completes the list of panic types you are likely to see. Now for the crash dump procedure:

At the moment a dump can be taken only on magnetic tape. Before you do anything, be sure that a clean tape is mounted with a ring-in on the tape drive if you plan to make a dump.

Write the date and time on the console log. Use the console commands to examine the registers, program status long word, and the top several locations on the stack. A suggested command sequence, which is executed by the “@DUMP” console command script, is:

```
E PSL<return>
E R0/NE:F<return>
E SP<return>
E/V @ /NE:40<return>
```

If hardware problems dictate a special set of commands be executed when the system crashes, a sequence of commands can be saved using the console command “LINK” to be reexecuted with “PERFORM” (which can be abbreviated “P”). If a dump is to be taken on magnetic tape (this is a good idea in most any case where the cause of the crash is not immediately obvious) then the following commands will (should) be executed:

```
D PSL 0<return>
D PC 80000200<return>
C<return>
```

These commands are actually part of the standard “@DUMP” script. This should write a copy of all of memory on the tape, followed by two EOF marks. Caution: Any error is taken to mean the end of memory has been reached. This means that you must be sure the ring is in, the tape is ready, and the tape is clean and new.

If there are not 40(hex) locations active on the kernel stack when the procedure is begun, then the console may begin to print error diagnostics. You can stop this by hitting “C” (control-C), and then give the last three commands above.

If the dump fails, you can try again, but some of the registers will be lost. See below for what to do with the tape.

To restart after a crash, follow the directions in [reboot\(8\)](#); if the virtual memory subsystem is suspected as the cause of the crash, then a version of the system other than “vmunix” should be booted which will leave the paging areas temporarily intact for use by the post-mortem analysis program *analyze*. After checking your root file system consistency with [fsck\(8\)](#), you can read the core dump tape into the file /vmcore with

```
dd if=/dev/rmt0 of=/vmcore bs=20b
```

It does not work to use just [cp\(1\)](#), as the tape is blocked. With the system still in single-user mode, run the analysis program *analyze*, e.g.:

```
analyze -s /dev/drum /vmcore /vmunix
```

and save the output. Then boot up “vmunix” and let it do the automatic reboot, i.e. to boot multi-user from an RM03/RM05/RP06 on the MASSBUS

```
>>> BOOT RPM
```

After rebooting, to analyze a dump you should execute *ps -alx* to print the process table at the time of

the crash. Use [adb\(1\)](#) to examine */vmcore*. The location *dumpstack-80000000* is the bottom of a stack onto which were pushed the stack pointer **sp**, **PCBB** (containing the physical address of a *u\_area*), **MAPEN**, **IPL**, and registers **r13-r0** (in that order). *r13(fp)* is the system frame pointer and the stack is used in standard **calls** format. Use [adb\(1\)](#) to get a reverse calling order. In most cases this procedure will give an idea of what is wrong. A more complete discussion of system debugging is impossible here. See, however, [analyze\(8\)](#) for some more hints.

**SEE ALSO**

[analyze\(8\)](#), [reboot\(8\)](#)

*VAX 11/780 System Maintenance Guide* for more information about machine checks.

**BUGS**



**NAME**

cron – clock daemon

**SYNOPSIS**

*/etc/cron*

**DESCRIPTION**

*Cron* executes commands at specified dates and times according to the instructions in the file It should be run once from [rc\(8\)](#).

*Crontab* entries are lines of seven fields separated by blanks or tabs:

*login minute hour day month weekday command*

*Login* is the login name under whose userid and groupid the command should be executed. The next five fields are integer patterns for

minute	0-59
hour	0-23
day of month	1-31
month of year	1-12
day of week	0-6; 0=Sunday

Each pattern may contain a number in the range above; two numbers separated by a hyphen meaning an inclusive range; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values.

The last field is a string to be executed by the shell, after replacing `\n` with newline.

It is wise to spread the times of activities to avoid bogging the system down at favorite hours.

*Cron* examines *crontab* every minute.

**EXAMPLES**

```
daemon 17 3 * * * /usr/bin/calendar - # reminders at 0317
adm 15 4 1,15 * * ac -p | mail adm # bimonthly accounts
root 0 12 22-28 11 4 /etc/wall Time for Thanksgiving dinner
```

**FILES**

*/etc/crontab*

**SEE ALSO**

[at\(1\)](#)

**BUGS**

The behavior of `\n` in *crontab* entries is nonstandard. Strings following `\n` should be delivered to the command as standard input.

**NAME**

dcheck – file system directory consistency check

**SYNOPSIS**

`/etc/dcheck [ -i numbers ] [ filesystem ]`

**DESCRIPTION**

**N.B.:** *Dcheck* is obsoleted for normal consistency checking by *fsck(8)*.

*Dcheck* reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The `-i` flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

**FILES**

Default file systems vary with installation.

**SEE ALSO**

*fsck(8)*, *icheck(8)*, *filsys(5)*, *clri(8)*, *ncheck(8)*

**DIAGNOSTICS**

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

**BUGS**

Since *dcheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

*Dcheck* is obsoleted by *fsck* and remains for historical reasons.

**NAME**

delivermail – deliver mail to arbitrary people

**SYNOPSIS**

`/etc/delivermail [-fr] address ] [-a ] [-ex ] [-n ] [-m ] [-s ] [-i ] [-h N ] address ...`

**DESCRIPTION**

*Delivermail* delivers a letter to one or more people, routing the letter over whatever networks are necessary. *Delivermail* will do inter-net forwarding as necessary to deliver the mail to the correct place.

*Delivermail* is not intended as a user interface routine; it is expected that other programs will provide user-friendly front ends, and *delivermail* will be used only to deliver pre-formatted messages.

*Delivermail* reads its standard input up to a control-D or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. If the `-i` flag is given, single dots are ignored. It determines the network to use based on the syntax of the addresses. Addresses containing the character '@' or the word "at" are sent to the ARPANET; addresses containing '!' are sent to the UUCP net, and addresses containing ':' or '.' are sent to the Berkeley network. Other addresses are assumed to be local.

Local addresses are looked up in a file constructed by *newaliases*(1) from the data file `/usr/lib/aliases` and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash or using the `-n` flag. Normally the sender is not included in any alias expansions, e.g., if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'. The `-m` flag disables this suppression.

*Delivermail* computes the person sending the mail by looking at your login name. The "from" person can be explicitly specified by using the `-f` flag; or, if the `-a` flag is given, *delivermail* looks in the body of the message for a "From:" or "Sender:" field in ARPANET format. The `-f` and `-a` flags can be used only by the special users *root* and *network*, or if the person you are trying to become is the same as the person you are. The `-r` flag is entirely equivalent to the `-f` flag; it is provided for ease of interface only.

The `-ex` flag controls the disposition of error output, as follows:

- e** Print errors on the standard output, and echo a copy of the message when done. It is assumed that a network server will return the message back to the user.
- m** Mail errors back to the user.
- p** Print errors on the standard output.
- q** Throw errors away; only exit status is returned.
- w** Write errors back to the user's terminal, but only if the user is still logged in and write permission is enabled; otherwise errors are mailed back.

If the error is not mailed back, and if the mail originated on the machine where the error occurred, the letter is appended to the file *dead.letter* in the sender's home directory.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *delivermail* from suppressing the blanks from between arguments.

The message is normally edited to eliminate "From" lines that might confuse other mailers. In particular, "From" lines in the header are deleted, and "From" lines in the body are prepended by '>'. The `-s` flag saves "From" lines in the header.

The `-h` flag gives a "hop-count", i.e., a measure of how many times this message has been processed by *delivermail* (presumably on different machines). Each time *delivermail* processes a message, it increases the hop-count by one; if it exceeds 30 *delivermail* assumes that an alias loop has occurred and it aborts the message. The hop-count defaults to zero.

*Delivermail* returns an exit status describing what it did. The codes are defined in `<sysexits.h>`

`EX_OK` Successful completion on all addresses.

EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.

**FILES**

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.dir	data base of alias names
/usr/lib/aliases.pag	
/bin/mail	to deliver uucp mail
/usr/net/bin/v6mail	to deliver local mail
/usr/net/bin/sendmail	to deliver Berknet mail
/usr/lib/mailers/arpa	to deliver ARPANET mail
/tmp/mail*	temp file
/tmp/xscript*	saved transcript

**SEE ALSO**

biff(1), binmail(1), mail(1), newaliases(1), aliases(5)

**BUGS**

*Delivermail* sends one copy of the letter to each user; it should send one copy of the letter to each host and distribute to multiple users there whenever possible.

*Delivermail* assumes the addresses can be represented as one word. This is incorrect according to the ARPANET mail protocol RFC 733 (NIC 41952), but is consistent with the real world.

**NAME**

*dkhup*, *dkmgr*, *dkzap* – manage Datakit interface

**SYNOPSIS**

```
/usr/ipc/mgrs/dkhup [ -N prefix ]
/usr/ipc/mgrs/dkmgr [ -N prefix ] [ -m outname ] [ -n service ]
/usr/ipc/mgrs/dkzap [ -N prefix ]
```

**DESCRIPTION**

*Dkhup* starts the common signaling channel protocol for a Datakit interface. Initially it sends several reset messages, and tells the controller to hang up all outstanding calls; thereafter it simply keeps the signaling protocol running.

*Dkmgr* receives and places Datakit calls. Outbound calls may be placed by calling *ipcopen* (*ipc*(3)) with the *outname* specified by option **-m**; the default is **dk**. *Dkmgr* announces itself to the Datakit network with the *service* name specified by option **-n**; the default is taken from Inbound calls to *service* are connected to the local login service; inbound calls to Datakit address *service.serv* are handed to local service *serv*.

*Dkhup* and *dkmgr* are normally run once from *rc*(8). Both programs must be running to make the network available.

*Dkzap* arranges for a KMC11-assisted Datakit interface to be reset, reloaded, and restarted. This should be done only if things are badly broken, as it hangs up all existing calls through that interface.

Datakit devices are expected to have names of the form */dev/dk/dknn* with *nn* a two-digit channel number. If there are more than 100 channels, the first digit overflows to lower-case letters: channel 100 is a0. The common signaling control channel is named */dev/dk/dkctl*. All three programs accept an option **-N prefix** to change the naming convention; for example, **-N /dev/kb/kb** means that the files have names like */dev/kb/kb32* and */dev/kb/kbctl*.

Support also exists for a less general naming convention: there may be two sets of files, named */dev/dk/dk0nn* and */dev/dk/dk2nn*, with control channels */dev/dk/dkctl0* and */dev/dk/dkctl2*. *Dkhup* and *dkzap* take the extra argument **0** or **2** to point at one or the other of the control names. A separate copy of *dkhup* is needed for each name. *Dkmgr* takes an option **-u c**, where *c* is **0** or **2** to use one set of files, or **b** to use both simultaneously; in the latter case, *service* is announced to both networks. This scheme is obsolete and overdue for replacement; the missing piece is something to pick an interface for outcalls.

*Dkmgr* records its activity in file *service* in directory */usr/ipc/log*, default */usr/ipc/log/dk*.

**FILES**

*/dev/dk*

**SEE ALSO**

*con*(1), *kmc*(8), *svcmgr*(8), *ipc*(3)

**BUGS**

*Dkhup* should be folded into *dkmgr*; it is separate for historic reasons.

**NAME**

`dmesg` – system diagnostic messages

**SYNOPSIS**

`/etc/dmesg [ - ] [ -i ] [ core namelist ]`

**DESCRIPTION**

*Dmesg* looks in a system buffer for recent console messages from the operating system and reproduces them on the standard output. Under option `-`, *dmesg* produces only those messages printed by the system since the last time `dmesg -` was run. It is normally run periodically by [cron\(8\)](#) to produce the error log

Option `-i` prints messages produced since the last `dmesg -` without changing any records.

If *core* and *namelist* are specified, they are used in place of `/dev/kmem` and

**FILES**

`/usr/adm/messages`  
error log

`/usr/adm/msgbuf`  
record of option `-`

**BUGS**

Since the system error message buffer is small, not all error messages are guaranteed to be logged. Error messages generated immediately before a system crash may not be logged.

**NAME**

dump – incremental file system dump

**SYNOPSIS**

**/etc/dump** [ key [ *argument ...* ] filesystem ]

**DESCRIPTION**

*Dump* copies to magnetic tape all files changed after a certain date in the *filesystem*. The *key* specifies the date and other options about the dump. *Key* consists of characters from the set **0123456789fuJsdWn**.

**0–9**

This number is the ‘dump level’. All files modified since the last date stored in the file */etc/dump-dates* for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire filesystem to be dumped.

**f** Place the dump on the next *argument* file instead of the tape.

**u** If the dump completes successfully, write the date of the beginning of the dump on file */etc/dump-dates*. This file records a separate date for each filesystem and each dump level. The format of */etc/dumpdates* is readable by people, consisting of one free format record per line: filesystem name, increment level and *ctime(3)* format dump date. */etc/dumpdates* may be edited to change any of the fields, if necessary. Note that */etc/dumpdates* is in a format different from that previous versions of *dump* maintained in */etc/ddate*, although the information content is identical.

**J** This option is intended to be invoked only when the old format */etc/ddate* files are updated to the new format */etc/dumpdates* format. The effect of this option is to convert between the old, obsolete format and to the new format. If the **J** option is invoked, all other options are ignored, and *dump* terminates immediately.

**s** The size of the dump tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, *dump* will wait for reels to be changed. The default tape size is 2300 feet.

**d** The density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per reel. The default is 1600.

**W** *Dump* tells the operator what file systems need to be dumped. This information is gleaned from the files */etc/dumpdates* and */etc/fstab*. The **W** option causes *dump* to print out, for each file system in */etc/dumpdates* the most recent dump date and level, and highlights those file systems that should be dumped. If the **W** option is set, all other options are ignored, and *dump* exits immediately.

**w** Is like **W**, but prints only those filesystems which need to be dumped.

**n** Whenever *dump* requires operator attention, notify by means similar to a *wall(1)* all of the operators in the group “operator”.

If no arguments are given, the *key* is assumed to be **9u** and a default file system is dumped to the default tape.

*Dump* requires operator intervention on these conditions: end of tape, end of dump, tape write error, tape open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** key, *dump* interacts with the operator on *dump*'s control terminal at times when *dump* can no longer proceed, or if something is grossly wrong. All questions *dump* poses **must** be answered by typing “yes” or “no”, appropriately.

Since making a dump involves a lot of time and effort for full dumps, *dump* checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, *dump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

*Dump* tells the operator what is going on at periodic intervals, including usually low estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time to the tape change. The output is verbose, so that others know that the terminal controlling *dump* is busy, and will be for some time.

Now a short suggestion on how to perform dumps. Start with a full level 0 dump

**dump 0un**

Next, dumps of active file systems are taken on a daily basis, using a modified Tower of Hanoi algorithm, with this sequence of dump levels:

3 2 5 4 7 6 9 8 9 9 ...

For the daily dumps, a set of 10 tapes per dumped file system is used on a cyclical basis. Each week, a level 1 dump is taken, and the daily Hanoi sequence repeats with 3. For weekly dumps, a set of 5 tapes per dumped file system is used, also on a cyclical basis. Each month, a level 0 dump is taken on a set of fresh tapes that is saved forever.

**FILES**

/dev/rrp1g	default filesystem to dump from
/dev/rmt8	default tape unit to dump to
/etc/ddate	old format dump date record (obsolete after <b>-J</b> option)
/etc/dumpdates	new format dump date record
/etc/fstab	Dump table: file systems and frequency
/etc/group	to find group <i>operator</i>

**SEE ALSO**

restor(1), dump(5), dumpdir(1), fstab(5)

**DIAGNOSTICS**

Many, and verbose.

**BUGS**

Sizes are based on 1600 BPI blocked tape; the raw magtape device has to be used to approach these densities. Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

It would be nice if *dump* knew about the dump sequence, kept track of the tapes scribbled on, told the operator which tape to mount when, and provided more assistance for the operator running *restor*.



**NAME**

dumpdir – print the names of files on a dump tape

**SYNOPSIS**

*/etc/dumpdir* [ *f filename* ]

**DESCRIPTION**

*Dumpdir* is used to read magtapes dumped with the *dump* command and list the names and inode numbers of all the files and directories on the tape.

The *f* option causes *filename* as the name of the tape instead of the default.

**FILES**

default tape unit varies with installation  
rst\*

**SEE ALSO**

dump(1), restor(1)

**DIAGNOSTICS**

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

**BUGS**

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, *dumpdir* doesn't use it.

**NAME**

expire – remove outdated news articles

**SYNOPSIS**

```
/usr/lib/news/expire [ -n newsgroups ] [ -i ] [ -I ] [ -v [ level ] ] [ -edays ]  
[ -a ]
```

**DESCRIPTION**

*Expire* is normally started up by [cron\(8\)](#) every night to remove all expired news. If no newsgroups are specified, the default is to expire **all**.

Articles whose specified expiration date has already passed are considered expirable. The **-a** option causes *expire* to archive articles in `/usr/spool/oldnews`. Otherwise, the articles are unlinked.

The **-v** option causes *expire* to be more verbose. It can be given a verbosity level (default 1) as in **-v3** for even more output. This is useful if articles aren't being expired and you want to know why.

The **-e** flag gives the number of days to use for a default expiration date. If not given, an installation dependent default (often 2 weeks) is used.

The **-i** and **-I** flags tell **expire** to ignore any expiration date explicitly given on articles. This can be used when disk space is really tight. The **-I** flag will always ignore expiration dates, while the **-i** flag will only ignore the date if ignoring it would expire the article sooner. *WARNING*: If you have articles archived by giving them expiration dates far into the future, these options might remove these files anyway.

**SEE ALSO**

[checknews\(1\)](#), [inews\(1\)](#), [readnews\(1\)](#), [recnews\(8\)](#), [sendnews\(8\)](#), [uurec\(8\)](#)

**NAME**

`fsck` – file system consistency check and interactive repair

**SYNOPSIS**

`/etc/fsck -p [ special ... ]`

`/etc/fsck [ -y ] [ -n ] [ -sX ] [ -SX ] [ -t filename ] [ special ... ]`

**DESCRIPTION**

`fsck` inspects the disk filesystems in the named *special* files and repairs inconsistencies. If no files are named, every file system listed in *fstab*(5) with type 0 and a nonzero pass number is checked.

Under option **-p**, `fsck` runs without intervention, repairing minor inconsistencies and aborting on major ones. This form is usually called from *rc*(8). If no special files are named, file systems in *fstab* are checked in parallel passes: all file systems with pass number 1 are checked simultaneously, then all file systems with pass number 2, and so on until *fstab* is exhausted.

Here are the minor ailments repaired automatically under **-p**:

- unreferenced inodes;
- wrong link counts in inodes;
- missing blocks in the free list;
- blocks in the free list also in files; and
- counts wrong in the super-block.

Other inconsistencies cause `fsck` to abandon the inconsistent file system, and exit with a nonzero status when the current pass finishes.

Without the **-p** option, `fsck` inspects one file system at a time, interactively. Each inconsistency causes `fsck` to print a message and ask permission to fix the problem. The operator may require arcane knowledge to guide `fsck` safely through repair of a badly damaged file system.

Here are the remaining options. They are allowed only if **-p** is absent.

- y** Assume a yes response to all questions. This should be used with great caution.
- n** Assume a no response to all questions; do not open the file system for writing. This option is assumed if the file system cannot be opened for writing.
- sX** Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the superblock of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system. If the file system has a bitmap free list (see *filesystems*(5)), the free list is always reconstructed unless the **-n** option is enabled.  
Parameter *X* allows free-list parameters to be specified: `-sblocks-per-cylinder:blocks-to-skip`. If *X* is not given, the values used when the file system was created are used; see *mkfs*(8). If these values were not specified, *X* is assumed to be **400:9**.
- SX** Conditionally reconstruct the free list. This option is like **-sX** except that the free list is rebuilt only if no discrepancies were found. **-S** implies **-n**.
- t** If `fsck` cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without **-t**, `fsck` will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked. If it did not already exist, it is removed when `fsck` completes.

Inconsistencies checked are:

- Blocks claimed more than once.
- Blocks designated outside the file system.
- Incorrect link counts.
- Directory size not 16-byte aligned.
- Bad inode format.
- Blocks not accounted for anywhere.
- Directory entry pointing to unallocated inode.
- Inode number out of range.

- More than 65536 inodes.
- More blocks for inodes than there are in the file system.
- Bad free block list format.
- Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are reconnected by placing them in the directory `lost+found` in the root of the file system being checked. The name assigned is the inode number, prefixed by #.

Checking the raw device is almost always faster, but *fsck* distinguishes bitmapped from non-bitmapped file systems by examining the minor device number, so the block device is safer.

## FILES

`/etc/fstab`

## SEE ALSO

[fstab\(5\)](#), [filsys\(5\)](#), [mkfs\(8\)](#), [reboot\(8\)](#)

T. J. Kowalski, 'Fscck—the UNIX File System Check Program', this manual, Volume 2

## BUGS

Inode numbers for `.` and `..` in each directory should be checked for validity.

Some systems save core images after a crash in the swap area; on such machines, checking many large file systems in parallel may cause swapping, overwriting the crash dump. It is best just to write crash dumps in a safer place. If disk space for dumps and swapping is scarce, avoid checking more than three 120-megabyte file systems in parallel on a machine with four megabytes of physical memory.

Examining the minor device number is a botch; there should be an explicit flag somewhere.

*Fsck* does not have supernatural powers.

**NAME**

`fstat` – file status

**SYNOPSIS**

`/etc/fstat` [ **-u** user ] [ **-p** pid ] [ **-f** filename ]

**DESCRIPTION**

*Fstat* identifies open files. A file is considered open if a process has it open, if it is the working directory for a process, or if it is an active pure text file. Under default options, *fstat* reports on all open files.

Options:

- u** Report all files open by a specified user.
- p** Report all files open by a specified process id.
- f** Restrict reports to the specified file. If the file is a character special file, *fstat* additionally reports on any open files on that device, treating it as a mounted file system.

**SEE ALSO**

`ps(1)`, `pstat(8)`

**DIAGNOSTICS**

Yet to be determined.

**BUGS**

*Fstat* tries to be clever if you elide the **-u**, **-f**, or **-p** flags for the argument. Like any expert system, it is sometimes wrong.

**NAME**

getty – set terminal mode

**SYNOPSIS**

*/etc/getty* [ *char* ]

**DESCRIPTION**

*Getty* is invoked by *init*(8) after a terminal is opened. While reading the user's name *getty* attempts to adapt the system to the speed and type of terminal being used.

*Init* calls *getty* with an argument specified by the *ttys*(5) entry for the terminal line. The argument *char* determines the line speed and other characteristics; see below. *Getty* then types a banner identifying the system (from and the `login:` message. The user's name is then read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is then changed to another baud rate and `login:` is typed again. Successive breaks cycle through a set of speeds.

The user's name is terminated by a newline or carriage return. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

Finally, *login*(8) is called with the user's name as argument.

Here are the possible values for *char*. If more than one speed is given, the first speed is used initially; others are selected by successive depressions of the BREAK key. Other things, like delays and tab expansion, are set in various ways as well.

char	speed
0	300-1200-150-110
2	9600
3	1200-300
4	300
5	300-1200
6	2400
7	4800
8	9600-1200-300
9	300-9600-1200
a	2400-1200
b	1200-2400
j	exta (usually 19200)

**SEE ALSO**

*init*(8), *login*(8), *ioctl*(2), *ttys*(5)

**NAME**

halt – halt the processor

**SYNOPSIS**

**/etc/halt** [ **-n** ]

**DESCRIPTION**

*Halt* writes out sandbagged information to the disks and then halts the processor. The machine does not reboot, even if the auto-reboot switch is set on the console.

The **-n** option prevent the sync before the reboot.

**SEE ALSO**

reboot(8)

**BUGS**

**NAME**

icheck, dcheck, ncheck – file system consistency check

**SYNOPSIS**

**/etc/icheck** [ option ... ] *filesystem* ...

**/etc/dcheck** [ option ... ] *filesystem* ...

**/etc/ncheck** [ option ... ] *filesystem* ...

**DESCRIPTION**

These programs perform consistency checks on file systems. For normal file system maintenance, see [fsck\(8\)](#). Common options are

**-B** The file system is bitmapped. If *filesystem* is a special file, this option is set automatically from the minor device number.

**-i number ...**

Report only on specified inode *numbers* (*dcheck* and *ncheck* only).

*Icheck* examines each *filesystem*, builds a list of used blocks, and compares this list against the free list maintained on the file system. The normal output of *icheck* includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; *i.e.* not in any file nor in the free list.

Other *icheck* options are

**-s** Ignore the free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent. The words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The normal output reports are suppressed.

**-b number ...**

Report each appearance of the selected block *numbers* in a file or on the free list.

**-d** Report each duplicate block.

**-m** Report each missing block.

**-e** Print at most one diagnostic per file; useful for badly curdled file systems.

*Dcheck* reads the directories in each *filesystem* and compares the link count in each inode with the number of directory entries by which it is referenced.

*Ncheck* generates a list of pathname vs i-number for each named *filesystem*. Other *ncheck* options are

**-a** Report . and . . , which are normally ignored.

**-s** Report only special files, and files with set-userid or set-groupid mode; helpful in finding security breaches.

**SEE ALSO**

[filsys\(5\)](#), [chuck\(8\)](#), [fsck\(8\)](#), [clri\(8\)](#)

**DIAGNOSTICS**

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. 'Dups in free' means that blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.



When a file turns up for which the link-count and the number of directory entries disagree, *dcheck* reports the relevant facts. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

When the filesystem structure is improper, *ncheck* prints ?? to denote the 'parent' of a parentless file. A pathname beginning with . . . denotes a loop.

## **BUGS**

Extraneous diagnostics may be produced if these commands are applied to active file systems.

They believe even preposterous super-blocks and consequently can get core images.

*Ncheck's* report is in no useful order, and probably should be sorted. *Ncheck* fails to report the root inode.

**NAME**

`inews` – submit news articles

**SYNOPSIS**

`inews` [ `-h` ] `-t` title [ `-n` newsgroup ... ] [ `-e` expiration ]

`inews` `-p` [ file ]

`inews` `-C` newsgroup

**DESCRIPTION**

*Inews* submits netnews articles. It is not intended for people; see *postnews*(1) for routine use. The first form is for submitting articles; the second for receiving articles from other machines; the third for creating newsgroups.

In the first form, the article is read from the standard input. A *title* must be specified, one or more *newsgroups* (default 'general') may be specified, and a nonstandard *expiration* date may be specified. Option `-f` substitutes another sender's name instead of the user. Option `-h` specifies that headers are present at the beginning of the article and should be included with the article header instead of as text.

The sender's full name is taken from the environment variable NAME, or from the system index (often *passwd*(5)). The environment variable ORGANIZATION overrides the system default.

In the second form *inews* reads the article from the named *file*.

The third form is for creating new newsgroups. This may be limited to specific users such as the super-user or news administrator.

**FILES**

`/usr/spool/news/.sys.nnn` temporary articles

`/usr/spool/news/newsgroup/article_no`

`/usr/spool/news/`

`knowlib/newsgroups` and highest article number in each

`seqlib/newsgroups` of last article

`listlib/newsgroups`

`system/news/` subscription list

**SEE ALSO**

*news*(5), *newsrsrc*(5), *postnews*(1), *readnews*(1), *recnews*(8), *sendnews*(8) *uurec*(8)

**NAME**

*init* – process control initialization

**SYNOPSIS**

*/etc/init*

**DESCRIPTION**

*Init* is invoked by the operating system as the last step in the boot procedure. It is always process 1.

When started normally, *init* calls *rc(8)* with parameter **autoboot**. If this succeeds, *init* begins multi-user operation. If *rc* fails, *init* commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When the single user shell terminates, *init* runs *rc* without the parameter, and begins multi-user operation.

*Rc* performs housekeeping such as checking and mounting file systems and starting daemons; see *rc(8)*.

In multi-user operation, *init*'s role is to create a process for each directly connected terminal port on which a user may log in. To begin such operations, it reads the *ttys(5)* file and forks to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing on file descriptors 0, 1, 2, and 3 (the standard input and output, the diagnostic output and Opening the terminal will usually involve a delay, since the *open* is not completed until someone dials and carrier is established on the channel. Then *getty(8)* is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login(8)* to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file or as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp(5)*, which records current users, and makes an entry in *wtmp*, which maintains a history of logins and logouts. Then the appropriate terminal is reopened and *getty* is invoked again.

*Init* catches signal **SIGHUP** and interprets it to mean that the *ttys* file should be read again. The shell process on each line that has become inactive according to *ttys* is terminated; a new process is created for each line added; lines unchanged in the file are undisturbed. Thus it is possible to drop or add terminal lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use `kill -1 1`.

*Init* will terminate multi-user operations, kill all outstanding processes, and resume single-user mode if sent signal **SIGTERM**: use `kill 1`. *Init* will wait at most 30 seconds for outstanding processes to die, to avoid waiting forever.

If, at bootstrap time, the *init* program cannot be executed, the system will loop in user mode at a low address.

**FILES**

*/dev/console*  
*/dev/tty*  
*/etc/utmp*  
*/usr/adm/wtmp*  
*/etc/ttys*  
*/etc/rc*

**SEE ALSO**

*login(8)*, *kill(1)*, *sh(1)*, *ttys(5)*, *getty(8)*, *rc(8)*, *reboot(8)*

**BUGS**

*Init*'s multi-user functions should be integrated with the world of *svcmgr(8)*.

**NAME**

install – place files in their proper homes

**SYNOPSIS**

**/etc/install** [ **-c** ] [ **-s** ] file dest

**DESCRIPTION**

*Install* moves or copies the *file* to *dest*. If *dest* is a directory, the file is installed in the directory. Its main use is in makefiles subsidiary to the primary source directory /src.

Option **-c** causes the file to be copied, otherwise it is moved. Option **-s** invokes [strip\(1\)](#) on the file first.

If possible, the group and owner of *dest* are changed to 'bin'.

**BUGS**

Only one option can appear.

**NAME**

`ipconfig`, `dkipconfig`, `udpconfig` – set up DARPA Internet protocols

**SYNOPSIS**

`/usr/ipc/mgrs/ipconfig` [ `-m mask` ] [ `-df` ] *ip-device localhost network* [*arp-device* ] .

`/usr/ipc/mgrs/dkipconfig` *gatemachine localhost remotehost* .

`/usr/ipc/mgrs/udpconfig` *udp-device* .

**DESCRIPTION**

*Ipconfig* activates the DARPA Internet protocol on a communications device, with Internet address *localhost* for the host and network address *network* for the device.

If *arp-device* is specified, the ARP address resolution protocol is started on that device. Option `-d` causes *ipconfig* to print ARP requests on the standard output as they are received.

Option `-m` declares a subnet mask for the network reached through *ip-device*. *Mask* may be a four-piece IP address like `255.255.255.0` or a 32-bit hexadecimal number like `ffffff00`.

Option `-f` is a special workaround for networks with obsolete hosts. It causes *ipconfig* to answer illegal ARP requests for the subnet's broadcast address with an illegal Ethernet address, to prevent broadcast storms.

*Dkipconfig* places a network call to *gatemachine* and activates the IP protocol on the connection, so that the remote machine becomes a gateway for the caller's IP traffic. *Localhost* becomes the calling machine's Internet address through this IP interface; *remotehost* is the Internet address to which local IP packets should be sent to reach the gateway.

*Ipconfig* and *dkipconfig* record unusual events and errors in log files `/usr/ipc/log/ipconfig` and

*Udpconfig* activates the UDP datagram protocol on the named *udp-device*, usually Only one *udpconfig* is needed for the entire collection of IP networks.

These programs are usually run once from [rc\(8\)](#).

**EXAMPLES**

The following calls start IP on system `fs` on the first Interlan Ethernet controller, with ARP active; arrange for machine `nj/astro/research` to pass IP packets to `fs`; and activate UDP.

```
/usr/ipc/mgrs/ipconfig /dev/il100 fs mh-astro-net /dev/il101 .
```

```
/usr/ipc/mgrs/dkipconfig nj/astro/research fs-dk research-dk127 .
```

```
/usr/ipc/mgrs/udpconfig /dev/ipudp .
```

**FILES**

`/usr/ipc/log/ipconfig`

`/usr/ipc/log/dkipconfig`

**SEE ALSO**

[con\(1\)](#), [qns\(7\)](#), [route\(8\)](#), [tcpmgr\(8\)](#)

**NAME**

kmc, kdiload, kmcdump – control KMC11 input/output processors

**SYNOPSIS**

*/etc/kdiload* [ *dev* [ *file* ] ]

*/etc/kmcdump* [ *dev* ]

**DESCRIPTION**

These commands control the KMC11-B microprocessors used for Datakit protocol processing.

*Kdiload* resets KMC device *dev*, copies the microcode in *file* into the KMC's memory, and starts the KMC. *Dev* may be a pathname or a single character key identifying the KMC; the default is 2. *File* defaults to that specified in */etc/kmctab* if a single character key is used, */etc/dkk.dubhi* otherwise.

*Kmcdump* stops the KMC and copies its state into files in the working directory. *Dev* may be a single character key or a pathname; the default is 2. The KMC's memory is copied to the file **core.k.nnn**, where *k* is the keyletter and *nnn* is some number; the state of the KMC's registers and some trace information from Unix is written to **regs.k.nnn**.

These commands search the file */etc/kmctab* for KMC devices and microcode files. The file contains lines of three blank-separated fields:

- single character identifying this KMC
- full pathname of the KMC device file
- full pathname of the microcode to be used in this KMC

The KMC with key *K* uses Datakit special files with names like **/dev/dk/dkK03**. If there is only one KMC for Datakit, its key is 2. If the only KMC is the only Datakit interface in a machine, its key is 2, and its special files look like **/dev/dk/dk03**.

**FILES**

- /etc/kmctab*
- /bin/kasb*

**SEE ALSO**

[dkmgr\(8\)](#)

**BUGS**

For the moment, the only permissible keys are 0 and 2. The KMC and Datakit filename conventions are arcane, and based on obsolete notions; they should be replaced.

**NAME**

*ldpcs* – load comet microcode

**SYNOPSIS**

*/etc/ldpcs* [ **-f** ] [ **-v** ] *pcsfile*

**DESCRIPTION**

*Ldpcs* loads microcode from *pcsfile* into the VAX-11/750 patchable control store. Normally, the hardware ID register is checked to see that the system is an 11/750 and that its base microcode revision level is appropriately high; the **-f** option removes the checks. The **-v** option causes the microcode version number to be printed after loading.

*Ldpcs* is usually called from [rc\(8\)](#) to load the most recent DEC microcode patches from

The patch file consists of 1024 bytes of patch bits, followed by 10240 bytes of actual patches. Each patch bit represents a 20-bit microcode word; the patches themselves are 20-bit words packed together. The format is the same as that distributed by DEC.

**FILES**

*/dev/mem*

*/dev/mtpr*

**BUGS**

Calling *ldpcs* is a good idea, but it is not mandatory; the system will run without the patches.

**NAME**

login – sign on

**SYNOPSIS**

*/etc/login name*

*/etc/login -f name [ cmd ]*

*/etc/login -p passwd-line [ cmd ]*

**DESCRIPTION**

*Login* is executed by [getty\(8\)](#). See the Introduction to this volume for how to dial up initially.

*Login* asks for a password if appropriate. Echoing is turned off during the typing of the password. The **-f** option forces login of the named user, without a password. **-p** is similar to **-f**, but an entire line of password file information is supplied.

*Login* initializes the userid, the groupid, and the working directory according to specifications found in the password file; see [passwd\(5\)](#). It also initializes environment variables PATH and HOME. Finally it executes a command interpreter (usually [sh\(1\)](#)). Argument 0 of the command interpreter is its name with a dash prepended. If a *cmd* argument was present, two additional arguments **-c cmd** are passed, and environment variable REXEC is set to 1.

Upon a successful login, accounting files are updated and, if no options are present, the message of the day is printed and the user is informed of the existence of mail.

Successful logins are recorded in */etc/utmp* and if *cmd* was present, \* is appended to the login name in *wtmp*, and no record is made in *utmp*.

Only the super-user may execute *login*.

**FILES**

*/etc/utmp*

accounting

*/usr/adm/wtmp*

accounting

*/usr/spool/mail/\**

mail

*/etc/motd*

message-of-the-day

*/etc/passwd*

password file

*/etc/group*

groups file

**SEE ALSO**

[newgrp\(1\)](#), [passwd\(1\)](#), [environ\(5\)](#), [passwd\(5\)](#), [getty\(8\)](#), [init\(8\)](#), [su\(8\)](#), [svcmgr\(8\)](#).

**DIAGNOSTICS**

'Login incorrect': the name or the password is bad.

'No Shell' or 'no directory': the initial shell or home directory specified in the password file does not exist.

'Cannot open password file': things are badly curdled.

**BUGS**

Information passed to options **-p** and **-f** is not checked. Only trusted programs should run *login*. Only trusted programs may usefully do so anyway; *login* has no privileges.



**NAME**

makekey – generate encryption key

**SYNOPSIS**

**/usr/lib/makekey**

**DESCRIPTION**

*Makekey* improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e. to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, . and /. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

*Makekey* is intended for programs, such as [crypt\(1\)](#), that perform encryption. Usually its input and output will be pipes.

**SEE ALSO**

[crypt\(1\)](#)

**NAME**

mkfs, mkbitfs, mklost+found – construct a disk file system

**SYNOPSIS**

*/etc/mkfs special size [ spacing cylsize ]*

*/etc/mkbitfs special size [ spacing cylsize ]*

*/etc/mklost+found*

**DESCRIPTION**

*Mkfs (mkbitfs)* constructs a regular (bitmapped) file system on the block device *special*. *Size* is the number of 1KB (4KB) blocks on the special file. Values of *size* for various special files, e.g. *ra(4)*, are given in Section 4. The new file system has a single empty directory and a number of inodes that depends on the size.

*Spacing* and *cylsize* are used for block allocation. *Spacing* is the preferred distance between successive blocks of a file; *cylsize* is the number of blocks the system can use without causing a disk seek. Suboptimal default values are built into the programs; some reasonable values are suggested below.

*Mklost+found* creates a directory called **lost+found** in the working directory with several empty slots. This directory is used as a repository for disconnected files recovered by *fsck(8)* and *chuck(8)*.

**EXAMPLES**

```
/etc/mkbitfs /dev/ra03 31231
/etc/mount /dev/ra03 /mnt
cd /mnt
/etc/mklost+found
```

Notice the change of units between *ra(4)* and here:  $31231 = 249848 * 512 / 4096$ .

**SEE ALSO**

*filsys(5)*, *fsck(8)*

**BUGS**

*Fsck* should make *lost+found* automatically when needed.

**MACHINE DEPENDENCIES**

Here are empirically good values for *spacing* and *cylsize* for VAX hardware. Minor changes, such as a new version of a controller, can invalidate them. To be certain, run your own experiments. Cylinder size doesn't seem to matter much on RA81 disks. On RA90s, it doesn't matter as long as it is large.

## Bitmapped (4K) file systems

space	cyl	disk, controller, CPU
2	500	RA90, KDB50, VAX 8550
4	500	RA90, UDA50, VAX 8550
2	40	RA81, KDB50, VAX 8550
3	40	RA81, old KDB50, VAX 8550
3	40	RA81, UDA50, VAX 8550
3	105	Wren VI, Viking UDD, VAX 8550
4	40	RA81, UDA50, VAX-11/750
3	40	RA60, UDA50, VAX-11/750
3	40	RA81, KDA50, MicroVAX II
1	40	RD54, RQDX3, MicroVAX II
3	40	RA81, KDA50, MicroVAX III

## Old-style (1K) file systems

space	cyl	disk, controller, CPU
5	357	RA81, KDB50, VAX 8550
7	357	RA81, UDA50, VAX-11/780
7	357	RA81, KDA50, MicroVAX II
7	128	RD54, RQDX3, MicroVAX II
6	357	RA81, KDA50, MicroVAX III

**NAME**

mknod – construct special file

**SYNOPSIS**

*/etc/mknod name [ c ] [ b ] major minor*

**DESCRIPTION**

*Mknod* makes a special file. The first argument is the *name* of the entry. The second is **b** to make a file for a block device, **c** to make a character device. The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number).

Minor devices are specific to each device driver; see the writeups in Section 4. The assignment of major device numbers varies with the implementation. Here are some conventional ones in the system used on VAXes:

**Block devices**

0	MASSBUS SMD disk
1	block TE16 tape, <i>mt</i> (4)
2	UNIBUS SMD disk
3	VAX 8550 console disk
7	MSCP disk, <i>ra</i> (4)
8	block TU78 tape, <i>mt</i> (4)
10	block TMSCP tape, <i>mt</i> (4)

**Character devices**

0	console terminal, <i>console</i> (4)
1	DZ11 serial interface
2	VAX-11/750 console tape
3	memory, null device, <i>mem</i> , <i>null</i> (4)
4	raw MASSBUS SMD disk
5	TE16 tape, <i>mt</i> (4)
7	interleaved swap devices, <i>drum</i> (4)
8	VAX-11/780 console floppy
12	raw VAX 8550 console disk
13	raw UNIBUS SMD disk
14	DEBNA Ethernet controller, <i>ethernet</i> (4)
17	DR11-C Datakit interface, <i>dk</i> (4)
22	TU78 tape, <i>mt</i> (4)
26	KMC11-B control interface, <i>kmc</i> (4)
28	raw MSCP disk, <i>ra</i> (4)
31	KMC11-B protocol-support Datakit interface, <i>dk</i> (4)
40	file descriptor device, <i>fd</i> (4)
42	IP protocol endpoints
43	TCP protocol endpoints
44	Interlan NI1010A Ethernet controller, <i>ethernet</i> (4)
50	UDP protocol endpoints
57	DHV11 serial line controller
58	DEQNA Ethernet controller, <i>ethernet</i> (4)
59	TMSCP tape, <i>mt</i> (4)

**SEE ALSO**

[mknod\(2\)](#)

**NAME**

mkpkg, inspkg, seal, unseal – package files for automatic software distribution

**SYNOPSIS**

**mkpkg** [ *option ...* ] *file ...*

**inspkg** [ *option ...* ] [ *file ...* ]

**seal** [ *option* ] [ *file ...* ]

**unseal** [ *option* ] [ *file ...* ]

**DESCRIPTION**

These programs are used by [ship\(8\)](#) to keep files identical across machines.

*Mkpkg* packages *files* and writes the result on the standard output. *Inspkg* installs the named packages or the standard input.

Non-existent files given to *mkpkg* are deleted upon installation. Directories are copied with all their contents. Hard links are reproduced. Symbolic links and special files are reproduced with the same inode contents. File modification and access times and owner and group names are reproduced as far as possible. Old versions of files are removed before installation: *inspkg* needs write permission in containing directories.

Options for both *mkpkg* and *inspkg*:

**-v** Place running commentary on the standard error file.

**-D** *path1=**path2*

Pretend that any pathname beginning with *path1* really begins with *path2*. Relative pathnames are extended to full pathnames before comparison.

Options for *mkpkg*; only one may occur:

**-x** *command*

Include in the package instructions to execute the shell *command* after all files have been installed. *Command* is unaffected by option **-D**.

**-X** *file* Include in the package instructions to run the shell script *file* after all files have been installed. The file name is subject to option **-D**.

Options for *inspkg*:

**-n** Skip the actual installation, but verify the input packages and produce a backup if requested.

**-b** Write on the standard output a backup package that contains whatever was destroyed.

A package is an [ar\(1\)](#) archive containing an extra ASCII file named

*Seal* concatenates the named *files* or the standard input onto the standard output in an error-detecting form suitable for shipment by [mail\(1\)](#). *Unseal* reverses the process, concatenating copies of all the original inputs onto the standard output. When [asd\(8\)](#) uses [uucp\(1\)](#), it sends sealed packages.

A sealed file is printable, has fewer than 128 characters per line, and has no lines consisting of a single period. The first line is `!<seal>` and the last one begins with `!end`. Other lines, such as mail headers, can be added to either end of a sealed file without hindering *unseal*.

Options for *seal* and *unseal*:

**-k** A key will be demanded to encrypt the checksum calculation.

**-K** *keyfile*

Same, but taking the first line of *keyfile* as the key.

**SEE ALSO**

[ar\(1\)](#), [cpio\(1\)](#), [tar\(1\)](#), [bundle\(1\)](#), [ship\(8\)](#), [ar\(5\)](#), [asd\(8\)](#)

**BUGS**

The pipeline `mkpkg` fails if input and output files overlap.

*Inspkg* fills any holes in files.

**NAME**

mount, umount – mount and dismount file system

**SYNOPSIS**

**/etc/mount** [ *special name* [ *fstype* [ *flags* ] ] ]

**/etc/mount -a**

**/etc/mount** [ *special name* [ **-r** ] ]

**/etc/umount** *name*

**/etc/umount -a**

**DESCRIPTION**

*Mount* announces to the system that a removable file system of type *fstype* is present on the file *special*. The file *name* must exist already; it becomes the name of the newly mounted root. *Fstype* and *flags* are integers; if omitted, they default to 0. Type 0 is an ordinary disk file system. Other types and possible flag values are listed in [fmount\(2\)](#).

The shorthand **mount special file -r** is equivalent to **mount special file 0 1**: mount an ordinary file system read-only.

If option **-a** is present, *mount* attempts to mount, in order, every file system listed in [fstab\(5\)](#).

*Umount* announces to the system that the file system mounted on file *name* is to be removed.

These commands maintain a table of mounted file systems in see [fstab\(5\)](#). If invoked without an argument, *mount* prints the table. If option **-a** is present, *umount* attempts to remove, in reverse order, each file system listed in *mtab*.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, even if no explicit write is attempted.

**EXAMPLES**

**/etc/mount /dev/ra02 /usr**

Mount the file system on disk /dev/ra02 on directory /usr.

**/etc/mount /dev/null /proc 2**

Mount the process file system.

**FILES**

/etc/mtab

mount table

/etc/fstab

file system table

**SEE ALSO**

[fmount\(2\)](#), [fstab\(5\)](#), [netfs\(8\)](#)

**BUGS**

Mounting file systems full of garbage may crash the system.

Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

**NAME**

netfs – network file system

**SYNOPSIS**

`/usr/netb/setup.go`

**DESCRIPTION**

The network file system is conventionally a set of directories contained in and a set of files and programs in Connections in the network file system are asymmetric: files on a ‘server’ system are made accessible on a ‘client’ system, usually in directory `/n/server-name`.

**Client**

The client runs to maintain connections; it is started by invoking `/usr/netb/setup.go` from `rc(8)`. `Setup` uses `/usr/netb/friends` to control the connections to servers. Each line in `friends` contains six fields:

- network address
- network call argument
- mount point
- protocol id
- unique identifier
- debugging flag
- network calling username

The network address and argument give the location of the server. They are interpreted differently according to the protocol id, which should be one of

- d** Call the server on the named network address, with default network `dk` and default service name `fsb`. The network call argument is ignored. The server machine should respond by calling `zarf`, described below; see `svcmgr(8)`.
- t** Call the named network address, with default network `tcp`, and invoke the program named in the network call argument using the protocol of `rsh`; `con(1)`.

`Setup` calls `setlogname` (see `getuid(2)`) to make the network call appear to have been placed by the calling username. The username may be omitted; `daemon` is the default.

The mount point is the directory on which the remote file system is to appear. The unique identifier is a integer in the range 0-255; it is used internally to distinguish connections, and must be unique among all active remote file systems (including those not maintained by `setup`, e.g. `faced(9)`) The debugging flag is usually 0; nonzero numbers increase the chatter in various logfiles.

`Setup` reads the `friends` file when it starts, and checks for changes once a minute. Each remote file system is probed once a minute; if there is no response to several consecutive probes, the connection is torn down and restarted. Failed connections are retried every minute.

**Server**

The server program is A separate `zarf` process exists for each client.

When a connection is started, the client sends the server a list of valid user and group names and the corresponding numerical IDs on the client system. The `userid` and `groupid` of user and group names that exist on both machines are mapped so that client and server see IDs under the same names. Unmapped IDs on the server appear as `-1` on the client. Client processes with unmapped IDs are denied access.

`Zarf` is subject to access control on the server. It will have access only to files that its own `userid` and `groupid` admit. Unless run as super-user, it will create files with its own, not mapped, `userid`.

`Zarf` reads configuration information from `/usr/netb/except.local` and The files are read only once, when `zarf` starts, **except.local** first. Usually **except** is the same on all machines in some administrative cluster, **except.local** contains things specific to a particular server system.

The files contain sections beginning with the line **client origin**. *Origin* is the name of the calling client, as provided by the network; `*` matches any client. The first matching section is used.

Within each section, lines have of one of the following forms. Lines beginning with `#` are ignored.

**uid** *cname=sname*

Regardless of the contents of password files, map client user name *cname* to server user name *sname*. If *cname* is not announced as valid by the client, the line is ignored. If *sname* is not a valid name on the server, any previous mapping for *cname* is discarded.

**gid** *cname=sname*

Map client group name *cname* to server group name *sname*, as above.

**param otherok=***val*

If *val* is 1, client processes with unmapped userids are granted world access to existing files on the server. Unmapped userids may never create files (who would own them?). If *val* is anything else, no access is permitted to unmapped client userids.

**param root=***pathname*

Use *pathname* rather than / as the root of the filename hierarchy made visible on the client.

**EXAMPLES**

A *friends* file for a connection to **alice** over Datakit, **shamash** over TCP/IP, and **bebop** over TCP/IP without administrative help:

```
alice          -                /n/alice d  0  0
tcp!shamash!400 -            /n/sun   d  1  0
bebop         /usr/pjw/netb/zarf /n/bebop t  2  0  pjw
```

Some *except* file rules:

```
client dk:nj/astro/research
param otherok=1
client *
uid root=
gid mail=other
param otherok=0
param root=/usr/spool
```

If the *research* machine calls as a client, the whole file system tree is visible, all userids including the super-user are permitted normal access, and user names unknown to the server are permitted world access. If any other machine calls, only the contents of **/usr/spool** are visible, *root* and unknown users are explicitly denied access, and processes in group *mail* on the client are treated as if in group *other* on the server.

**FILES**

```
/n/*
/usr/netb/friends
    client connection info
/usr/netb/except.local
/usr/netb/except
/usr/netb/setup1
    log file for setup
/usr/netb/zarf.log
    log file for zarf server control info
```

**SEE ALSO**

S. A. Rago, 'A Look at the Version 9 Network File System', this manual, Volume 2

**BUGS**

The scheme works only in a modest-sized, friendly community, as it requires a process per client, trust of clients' security, and common login names.

File modification times are adjusted for clock-time differences between machines. Thus, when viewed across the network, identical files installed on different machines by [asd\(8\)](#) may appear to have different modification times, and symbol tables of random libraries ([ar\(1\)](#)) may appear to be out of date.

**NAME**

`netstat`, `dkstat` – show network status for internet and datakit networks

**SYNOPSIS**

`netstat` [ **-acCirRst** ] [ *system* ] [ *core* ]

`dkstat` [ *interval* [ *count* ] ]

**DESCRIPTION**

`Netstat` displays internet (TCP/IP and UDP/IP) traffic and configuration data. Without options, it lists all TCP and UDP connection assignments. A single option changes the listing:

- a** known mappings between internet names and Ethernet addresses.
- c** all TCP and UDP connection assignments (the default).
- C** detailed state of active TCP connections.
- i** active IP interfaces.
- n** display numeric internet addresses rather than host and network names.
- s** protocol statistics.
- r** routing tables.
- R** routing tables, including deleted entries (for debugging).
- tbuf** running trace of packets passing through *buf*: `il` for the Interlan Ethernet controller, `qe` for the DEQNA, `tcp` (the default) for all packets passing through TCP.

The arguments *system* and *core* are substitutes for the defaults `/unix` and

`Dkstat` reports the number of bytes received and sent over the Datakit network, together with error reports if any occurred. The first report is cumulative since a reboot. Further reports may be requested every *interval* seconds; these reports are incremental.

The optional *count* argument restricts the number of reports.

**DIAGNOSTICS**

**nlist /unix failed:** `netstat` could not find pertinent system information, perhaps because this system isn't set up for TCP/IP.



**NAME**

ns – name server database

**SYNOPSIS**

**/usr/IPC/mgrs/ns** [ **-m** *server* ] [ **-d** ]

**DESCRIPTION**

*Ns* maintains a database of naming information, accessed by [qns\(7\)](#) and other programs. It should be run once from [rc\(8\)](#).

The database is accessed through local service **ns**, or service *server* if option **-m** was specified.

The file `/usr/IPC/lib/ns.db` contains instructions for building the database. These instructions are lines of one of the following forms:

**#uusys file**      Read the named *uucp Systems file*. For entries using caller ACU, add a database entry containing

*system telephone-number,tel uucp,svc*

For entries using caller DK or DKH, add an entry containing

*system datakit-address,dk uucp,svc*

For any other entry, add

*system uucp,svc*

**#inhost file**      For each line in the named 4BSD-style internet hosts *file*, add a database entry of the form

*ip-address,in hostname host-domain-name,dom*

**#innet file**      For each line in the named 4BSD-style internet networks *file*, add a database entry of the form

*ip-net-address,in netname*

**#include file**      Interpret the contents of *file* in the same format as **ns.db**.

In all cases, *file* may be followed by a list of *value,attribute* pairs to be included with any database entries caused by that file. If the filename doesn't begin with `/`, it is prefixed with `/usr/IPC/lib`.

Blank lines and lines beginning with `#` followed by a space or tab are ignored.

Any other lines are taken as literal database entries: a collection of *value,attribute* pairs separated by spaces. Each line is a single entry.

The database is ephemeral; it is rebuilt whenever *ns* starts, when requested by **qns reset**, or when *ns* notices that **ns.db** or one of the files named therein has changed. Rebuilding can take several minutes, especially on a busy machine. During a rebuild, the server appears active but does not answer requests; calls will block until the rebuild finishes.

*Ns* leaves remarks in file `There are more remarks` if the **-d** option was used.

**FILES**

`/usr/IPC/lib/ns.db`

**SEE ALSO**

[ipc\(3\)](#), [qns\(7\)](#)

**NAME**

postbgi – PostScript translator for BGI (Basic Graphical Instructions) files

**SYNOPSIS**

**postbgi** [ options ] [ files ]

**DESCRIPTION**

*Postbgi* translates BGI (Basic Graphical Instructions) *files* into PostScript and writes the results on the standard output. If no *files* are specified, or if – is one of the input *files*, the standard input is read. The following *options* are understood:

- cnum** Print *num* copies of each page. By default only one copy is printed.
- fname** Print text using font *name*. Any PostScript font can be used, although the best results will only be obtained with constant width fonts. The default font is Courier.
- mnum** Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, which by default is located at the center of each page. The default magnification is 1.0.
- nnum** Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default *num* is set to 1.
- olist** Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1–N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.
- pmode** Print *files* in either **portrait** or **landscape** *mode*. Only the first character of *mode* is significant. The default *mode* is portrait.
- wnum** Set the line width used for graphics to *num* points, where a point is approximately 1/72 of an inch. By default *num* is set to 0.0 points, which forces lines to be one pixel wide.
- xnum** Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is 0 inches.
- ynum** Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is 0 inches.
- Afile** Append a simple accounting record to *file* after all the input *files* have been successfully translated. By default no accounting data is produced.
- Lfile** Use *file* as the PostScript prologue, which by default is /usr/lib/postscript/postbgi.ps.

**DIAGNOSTICS**

0 exit status is returned if *files* were successfully processed.

**BUGS**

The default line width is too small for 'write to white' print engines, like the one used by the PS-2400.

**FILES**

/usr/lib/postscript/postbgi.ps

**SEE ALSO**

dpost(1), postprint(1), posttek(1), postdmd(1)

**NAME**

postio – serial interface for postscript printers

**SYNOPSIS**

**/usr/bin/postscript/postio** [ *option ...* ] [ *file ...* ]

**DESCRIPTION**

*Postio* sends *files* to a PostScript printer. It is usually called by the innards of *lp(1)*. If no files are named, the standard input is sent.

Mandatory argument **-l** names the printer. If the first character of *line* is /, it is assumed to be a local file-name like */dev/tty37*. Otherwise it is taken to be a network address, with default network *dk*, to which the printer is connected.

These options are probably the most useful:

- b***speed* Transmit data at baud rate *speed*, one of 1200, 2400, 4800, 9600 (default), and 19200.
- q** Disable status queries while *files* are being sent to the printer. When status queries are disabled a dummy message is appended to the log file before each block is transmitted.
- B***num* Set the internal buffer size for reading and writing *files* to *num* bytes, 2048 by default.
- D** Debug mode: copy everything read from the printer to the log file or standard error.
- L***file* Log data read from the printer in *file*. Standard error is the default. Normally only messages indicating a change in the printer's state are logged.
- P***string* Send *string* to the printer before any input *files*. The default is PostScript code that disables timeouts.
- R***num* If *num* is 1, run as a single process; if 2, use separate processes for reading and writing.

These options are not useful to spoolers like *lp*.

- i** Interactive mode: send the *files* to the printer, then copy standard input to the printer and printer output to standard error. Overrides many other options. To have a friendly chat with the printer, begin by typing *executive* on a line by itself.
- t** Copy printer output that doesn't look like status information to the standard output; intended for use with PostScript programs that write results.

This option should be used only as a last resort:

- S** Take special measures to send data slowly. Limits the internal buffer to 1024 bytes, implies **-R1** and disables **-q** and **-i**. Expensive in CPU time.

When *postio* starts, it attempts to force the printer into IDLE state by sending a sequence of control-**t** (status query), control-**c** (interrupt), and control-**d** (end of job) characters. When the printer is idle, the files are transmitted with an occasional control-**t** interspersed (except under **-q**). After all data have been sent, *postio* waits until the printer appears to have finished before exiting. Fatal error messages from the printer cause *postio* to exit prematurely.

**EXAMPLES**

postio -l/dev/tty01 file1 file2 Runing as a single process at 9600 baud, send file1 and file2 to printer */dev/tty01*.

postio -R2 -B4096 -l/dev/tty01 -Llog file1 file2 Similarly, but use two processes and a 4096-byte buffer, and copy printer messages to file **log**.

postio -t -l/dev/tty22 -Llog program >results Send the PostScript program to printer */dev/tty22*, place any data in **results**, put error messages in **log**.

postio -i -l/cs/dk!my/printer Connect interactively to the printer at network address */cs/dk!my/printer*.

**SEE ALSO**

*lp(1)*, *postscript(8)*

**DIAGNOSTICS**

Exit status 1 means a system error (e.g. can't open the printer), 2 means a PostScript error, 3 means both. Status 2 is usually caused by a syntax error in an input file.

**BUGS**

Multiple files with PostScript end-of-job marks are not guaranteed to work.

If a network is involved, **-b** may be ineffective and attempts by *postio* to flow-control data in both directions may not work. Option **-q** can help if the printer is connected to Radian Datakit.

**NAME**

postreverse – reverse the page order in a postscript file

**SYNOPSIS**

**postreverse** [ options ] [ file ]

**DESCRIPTION**

*Postreverse* reverses the page order in a minimally conforming PostScript *file* and writes the results on the standard output. If no *file* is specified, the standard input is read. The following *options* are understood:

- olist**       Select pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1–N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.
- r**            Don't reverse the pages in *file*.
- Tdir**        Use *dir* as the temporary file directory when reading from the standard input. By default *dir* is set to /tmp.

*Postreverse* can handle files that violate page independence, provided all global definitions are bracketed by `BeginGlobal` and `EndGlobal` comments. In addition files that mark the end of each page with `EndPage: label ordinal` comments will also reverse properly, provided the prologue and trailer sections can be located. If the end of the prologue isn't found, the entire *file* is copied, unmodified, to the standard output.

Since global definitions are pulled out of individual pages and put in the prologue, the output file can be minimally conforming, even if the input *file* wasn't.

**EXAMPLES**

Select pages 1 to 100 from **file** and reverse the pages,

```
postreverse -o1-100 file
```

Print 4 logical pages on each physical page and reverse all the pages,

```
postprint -n4 file | postreverse
```

Produce a minimally conforming file from output generated by `dpost` without reversing the pages,

```
dpost file | postreverse -r
```

**DIAGNOSTICS**

0 exit status is returned if *file* was successfully processed.

**BUGS**

No attempt has been made to deal with redefinitions of global variables or procedures. If standard input is used, the input *file* will be read three times before being reversed.

**SEE ALSO**

`dpost(1)`, `postprint(1)`, `posttek(1)`, `postbgi(1)`, `postdmd(1)`

**NAME**

dpost, postdaisy, postdmd, postprint – filters to produce postscript

**SYNOPSIS**

```
/usr/bin/postscript/dpost [ option ... ] [ file ... ]
/usr/bin/postscript/postdaisy [ option ... ] [ file ... ]
/usr/bin/postscript/postdmd [ option ... ] [ file ... ]
/usr/bin/postscript/postprint [ option ... ] [ file ... ]
/usr/bin/postscript/posttek [ option ... ] [ file ... ]
```

**DESCRIPTION**

These programs convert files of various formats into PostScript. The input formats are

```
dpost      troff(1) output
postdaisy  Diablo 1640 daisy-wheel
postdmd    bitfile(9) files, as produced by blitblt(9)
postprint  ASCII text
posttek    Tektronix 4014 graphics
```

Except as noted, the options are common to all the programs:

- cnum** Print *num* copies of each page. By default only one copy is printed.
- mnum** Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, located near the upper left corner of the page. The default magnification is 1.0.
- nnum** Print *num* logical pages on each piece of paper. The default is 1.
- olist** Print only pages specified in the comma-separated *list* of numbers and ranges. A range *N-M* means pages *N* through *M*; an initial *-N* means from the beginning to page *N*; and a final *N-* means from *N* to the end. Print only pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1-N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.
- pmode** Print in *mode* **p** (portrait) or **l** (landscape). The default is **p**.
- xnum** Translate the origin *num* inches along the positive x axis. By default, the origin is fixed near the upper left corner of the page, with positive x to the right and positive y down the page. Positive *num* moves everything right. The default offset is 0 inches.
- ynum** Translate the origin *num* inches along the positive y axis. Positive *num* moves text down the page. The default offset is 0.
- Afile** Append a simple accounting record to *file* after all input *files* have been successfully translated. By default no accounting data is produced.
- Lfile** Use *file* as the PostScript prologue.
- fname** Print *files* using font *name*. Any PostScript font can be used, but constant width fonts yield the best results. The default font is Courier. (*postdaisy*, *postprint*, and *posttek* only)
- f** Flip the sense of the bits in *files* before printing the bitmaps. (*postdmd* only)

In addition, three options allow the insertion of arbitrary PostScript at controlled points in the translation process:

- Cfile** Copy *file* to the output file. *File* follows the prologue but precedes any job initialization commands. *File* becomes part of the job's global environment and must contain legitimate PostScript commands.
- Pstring** Like **-C**, using a *string* instead of the contents of a file.
- Raction** Requests special *action* (e.g. manualfeed) on a per page or global basis. The *action* string has the general form *request:page:file*, from which *:page:file* or **:file** can be omitted. An omitted or 0 page number applies to all pages. If *file* is omitted the request lookup is done

in The collection of recognized requests can be modified or extended by changing this file. Multiple occurrences of the **-R** option behave as expected.

## FILES

/usr/lib/font/devpost/\*.out  
/usr/lib/font/devpost/charlib/\*  
/usr/lib/postscript/\*.ps  
    default prologues  
/usr/lib/tmac/tmac.pictures  
    *troff* macros for PostScript

## SEE ALSO

*lp(1)*, *postio(8)*

## DIAGNOSTICS

Exit status 2 usually means a syntax error in some input file.

## BUGS

Output files will often violate Adobe's file structuring conventions. Pipe the output of *dpost* through *postreverse* to produce a minimally conforming PostScript file.

Although *dpost* can handle files formatted for any *troff* device, emulation is expensive and can easily double the print time and the size of the output file.

No attempt has been made to implement the character sets or fonts available on all devices supported by *troff*. Missing characters are replaced by white space; unrecognized fonts are replaced by one of the Times fonts.

*Dpost* requires an **x res** command before the first **x init** and any file data.

**NAME**

pstat – print system facts

**SYNOPSIS**

/etc/pstat [ **-afipstuxT** ] [ *suboptions* ] [ *file* ] [ *namelist* ]

**DESCRIPTION**

*Pstat* interprets the contents of certain system tables. If *file* is given, the tables are sought there, otherwise in The required namelist is taken from *namelist*, default Options are

**-a** Under **-p**, describe all process slots rather than just active ones.

**-i** Print the inode table with the these headings:

**LOC** The core location of this table entry.

**FLAGS**

Miscellaneous state variables encoded thus:

**L** locked

**U** modified time (*filsys(5)*) must be corrected

**A** access time must be corrected

**O** file was opened

**W** wanted by another process (**L** flag is on)

**T** contains an active text

**CNT** Number of active references to this inode.

**FS** File system type, see *fmount(2)*.

**DEVICE**

Device number of file system in which this inode resides.

**INO** I-number within the file system.

**MODE**

Mode, see *stat(2)*.

**NLN** Number of links to this inode.

**UID** Userid of owner.

**SPTR** Core location of corresponding stream header, 0 if this is not a stream.

**SIZ/DEV**

Number of bytes in an ordinary file, or device number of a special file.

**MROOT**

Core location of root inode of file system mounted here, 0 if none.

**-x**

Print the text table with these headings:

**LOC** The core location of this table entry.

**FLAGS**

Miscellaneous state variables encoded thus:

**P** resulted from demand-page-from-inode exec format, see *exec(2)*

**T** traced through *proc(4)*

**W** text not yet written to swap device

**L** loading in progress

**K** locked

**w** wanted (**L** flag is on)

**DADDR**

Disk address in swap, in multiples of 512 bytes.

**CADDR**

Head of a linked list of loaded processes using this text segment.

**RSS** Size of physical memory occupied by text segment, in multiples of 512 bytes.

**SIZE** Size of text segment, in multiples of 512 bytes.

**IPTR** Core location of corresponding inode.

**CNT** Number of processes using this text segment.

**CCNT**

Number of processes in core using this text segment.



**-p**

Print process table for active processes with these headings:

<b>LOC</b>	The core location of this table entry.
<b>S</b>	Run state encoded thus:
<b>0</b>	no process
<b>1</b>	waiting for some event
<b>3</b>	runnable
<b>4</b>	being created
<b>5</b>	being terminated
<b>6</b>	stopped under trace
<b>F</b>	Miscellaneous state variables, or-ed together (hexadecimal):
<b>0000001</b>	loaded in memory
<b>0000002</b>	special system process (swapper or pager)
<b>0000004</b>	being swapped out
<b>0000008</b>	obscure swapout flag
<b>0000010</b>	traced
<b>0000020</b>	used in tracing
<b>0000040</b>	locked in core
<b>0000080</b>	waiting for pagein
<b>0000100</b>	prevented from swapping during <i>fork(2)</i>
<b>0000200</b>	gathering pages for raw i/o
<b>0000400</b>	exiting
<b>0008000</b>	associated text is demand paged from file
<b>0030000</b>	anomalous paging behaviour expected, see <i>vlimit</i> in <i>deprecated(2)</i>
<b>0040000</b>	in a sleep which will time out
<b>0400000</b>	in <i>select(2)</i>
<b>0800000</b>	traced via <i>proc(4)</i>
<b>1000000</b>	i/o via <i>proc</i> in progress
<b>2000000</b>	stop on exec
<b>4000000</b>	wanted by <i>proc</i> after pagein
<b>ADDR</b>	The core location of the page table entry for the first page of the 'u-area.'
<b>PRI</b>	Scheduling priority; smaller numbers run first.
<b>SIG</b>	Signals received; signals 1-32 coded in bits 0-31.
<b>UID</b>	Real userid.
<b>SLP</b>	Time blocked in seconds; times over 127 coded as 127.

**TIM** Time resident in seconds; times over 127 coded as 127.  
**CPU** Weighted integral of CPU time, for scheduler.  
**NI** Nice level, see [nice\(2\)](#).  
**PGRP**  
 Process group number.  
**PID** Process ID number.  
**PPID** Process ID of parent process.  
**RSS** Number of physical page frames allocated to this process.  
**SRSS** RSS at last swap, 0 if never swapped.  
**SIZE** Virtual size of process image (data+stack) in multiples of 512 bytes.  
**WCHAN**  
 Event address if waiting.  
**LINK** Pointer to next entry in list of runnable processes.  
**TEXTP**  
 If text is pure, pointer to location of text table entry.  
**CLKT**  
 Countdown for [alarm\(2\)](#) measured in seconds.

**-u**

Print information about a user process; the next argument is its address as given by ADDR under **-p** above. The process must be in main memory, or the file used can be a core image ([core\(5\)](#)) and the address 0.

**-f**

Print the open file table with these headings:

**LOC** The core location of this table entry.  
**FLG** Miscellaneous state variables encoded thus:  
**R** open for reading  
**W** open for writing  
**CNT** Number of processes that know this open file.  
**INO** The core location of the inode table entry for this file.  
**OFFS** The file offset, see [lseek\(2\)](#).

**-s**

Print information about swap space usage: the number of 1024 byte pages used and free, and the number of pages belonging to text images.

**-T**

Print the number of used and free slots in several system tables; useful to see if they are nearly full.

**FILES**

/unix  
 namelist  
 /dev/kmem  
 default source of tables

**SEE ALSO**

[ps\(1\)](#), [stat\(2\)](#), [filsys\(5\)](#)

M. J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, 1986

**BUGS**

This program is never up to date.

**NAME**

quot, findo – file system usage and hogs

**SYNOPSIS**

**/etc/quot** [ *option ...* ] [ *filesystem* ]

**findo** [ **-f** ] [ **-n** ] [ **-u** *userid* ] *device mount-directory*

**DESCRIPTION**

*Quot* prints the number of blocks in the named *filesystem* device currently owned by each user. If no *filesystem* is named, `/dev/USR` is assumed. The options are:

- n** Use as in the example below to list all files and owners.
- c** Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in files of that size or smaller.
- f** Print count of number of files as well as space owned by each user.
- b** Print space-time product in block-years in addition to space owned by each user.

*Findo* discovers files you might want to delete on the given block *device*, which must be mounted on the given directory. It lists, on the standard output, the sizes, ages in days, and names of files with any of the following characteristics:

- *Troff*(1) output files older than 24 hours. The names are marked `troff:` in the output.
- Week-old files named **core**, **a.out**, **mon.out**, **.pilog**, **junk\***, **temp\***, **ed.hup**, **qed.hup\***, **jim.recover**, **sam.save**, **sam.err**, **sam**, **[a-z]**, **dead.letter**, **foo[0-9]\***, **rst[0-9]+**, **.jx\***, **\*.dvi**, and files whose names resemble *apnews*(7) spool entries.
- Files over a month old named **\*.o**. The names are marked `old:`.
- Files owned by users selected with option **-u**; the names are marked `user:`.

The options are

- f** List files owned by users not in the password file.
- u** *userid*  
List files over 2 days old owned by the user with the given numeric *userid*.
- n** List files of any age owned by the specified users.

**EXAMPLES**

`ncheck filesystem | sort -n | quot -n filesystem` List all files and their owners.

**FILES**

`/etc/passwd` to get user names

**SEE ALSO**

*ls*(1), *du*(1), *icheck*(8), *fstab*(5)

**BUGS**

*Quot* counts holes in files as if they actually occupied space.

Patterns specifying the names and ages are compiled into *findo*.

*Findo*'s age distinction for files owned by a specific user is a historical dreg.

**NAME**

rarepl, rarct – replace bad blocks on MSCP disks

**SYNOPSIS**

*/etc/rarct* [ **-c** ] [ **-h** ] *special ...*

*/etc/rarepl special lbn ...*

**DESCRIPTION**

*Rarct* prints status information about MSCP disk drives like the RA60 and RA81. Normally the replacement table (RCT) is listed, as lines of the form

*rbn: flags: lbn*

where *rbn* is the replacement block number, *lbn* is the logical block number replaced by *rbn*, and *flags* are constructed from the following bits:

- 01** alternate (not primary) replacement block
- 02** normal, allocated replacement block
- 04** this replacement block is bad
- 010** this replacement block does not exist

Entries whose *flags* are zero, indicating a good, unused replacement block, are not listed.

The options suppress the RCT listing and perform other functions:

- h** Print some header data from the first block of the RCT. The system does not use this information.
- c** Print geometry information for the drive.

*Rarepl* causes logical block *lbn* on device *dev* to be marked as bad and replaced. The nearest available replacement block is used. The contents of *lbn* are copied into the replacement block if possible; if *lbn* is unreadable, the replacement block is initialized with zeros.

Both programs work only on the raw devices. *Rarepl* should be used only on a device which covers the entire drive (usually partition 7).

**SEE ALSO**

[ra\(4\)](#), [smash\(8\)](#)

**BUGS**

There are various controller- and drive-dependent anomalies. Some controllers, like the RQDX3, report an RCT but don't allow forwarding. On many controllers, the RCT exists only so programs in the host can look at it; the controller ignores its contents. There is no way to read the controller's 'real' forwarding data, only a way to set it for a particular block. Hence if the RCT is corrupted, the disk may still be used, but must be reformatted before additional bad blocks are remapped.

**NAME**

rc – boot script

**SYNOPSIS**

/etc/rc

**DESCRIPTION**

*Rc* is the command script invoked by *init(8)* to control reboots. During an automatic reboot, *rc* is invoked with the argument **autoboot**; typically this invokes */etc/fsck* to repair minor filesystem inconsistencies. If *rc* exits with a successful status, *init* proceeds to multi-user mode.

When the system enters multi-user mode, either during an auto-reboot or after the single-user shell terminates, *rc* is invoked without arguments. This usually causes it to mount filesystems, start daemons, clear and perform other housekeeping.

If any call to *rc* returns a nonzero status, *init* reverts to single-user mode.

**EXAMPLES**

A typical *rc* script:

```
date
case $1 in
autoboot)
    echo Autoboot:
    /etc/fsck -p || {echo "error in reboot"; exit 1}
esac
/etc/ldpcs /etc/pcs750.bin
>/etc/mtab
/etc/mount -a
/etc/savecore /tmp/dump /dev/rall
/etc/swapon -a
trap "" 1 2 3
/etc/update
/etc/cron .
rm -f /tmp/*
/usr/lib/asd/rmllocks
date >> /usr/adm/lastboot
/etc/accton /tmp/acct > /tmp/acct
/usr/ipc/mgrs/svcmgr
/etc/kdload
/usr/ipc/mgrs/dkhup; sleep 10
/usr/ipc/mgrs/dkmgr
/usr/netb/setup.go
/usr/net/face.go
wv -s
```

**SEE ALSO**

*init(8)*, *reboot(8)*

**NAME**

reboot – bootstrapping procedures

**DESCRIPTION**

Here are some recipes for booting and crashing the operating system on VAXes.

**Rebooting a running system**

The preferred way to reboot is to log in on the console as super-user, invoke **kill 1** to take the system to single user, unmount file systems with **/etc/umount -a** and halt and restart the system as described below under ‘Console boots.’

**Power fail and crash recovery**

The system will reboot itself at power-up or after crashes if auto-boot is enabled on the machine front panel or in the console software. If auto-restart is enabled, the system will first attempt to save a copy of physical memory on a reserved piece of disk. An automatic consistency check of the file systems is performed. Unless this fails the system will resume multi-user operations.

**Console boots**

Sync the disks if necessary and possible. To recover hardware control of the console, type a control-**P**. This will yield a >>> prompt from the VAX console subsystem (sic). The command

```
>>> H
```

will halt the CPU (except on the 11/750, where control-**P** halts the CPU right away).

On MicroVAXes, control-**P** doesn't work; hit the BREAK key instead.

To boot multi-user with an automatic file system check, give the console command

```
>>> B
```

Commands to boot single-user vary. On the VAX-11/750 and on MicroVAXes, use

```
>>> B/3
```

On the VAX-11/780 and VAX 8550 and 8700, use

```
>>> B MAN
```

This will prompt with \* for the name of the file to boot. The filename should be an executable image in the root directory of the filesystem at the beginning of the disk.

**System core images**

If the system crashes and auto-restart is enabled, a copy of physical memory is written to a reserved piece of disk. To save a core image of a hung system, type on the console (after control-**P** if necessary):

```
>>> S 80000010
```

The system will write the core image, then reboot automatically.

If the core image was written on /dev/ra11, the following incantation will print a stack traceback from the time of the crash:

```
adb /unix /dev/ra11
$<crash
$c
```

To save disk space, the core image is sometimes overlaid on part of the swap area, where normal system operation will soon overwrite it. *Savecore(8)* will copy the core image to an ordinary disk file.

**FILES**

/unix  
default system binary

**SEE ALSO**

[fsck\(8\)](#), [init\(8\)](#), [rc\(8\)](#), [savecore\(8\)](#)

**BUGS**

Older boot programs with different syntax are still around in a few places, especially on machines with Emulex UNIBUS disk controllers, for which silly boot ROMs are common.

There are commands **/etc/reboot** and **/etc/halt** which attempt to reboot and halt the system; their function is indeterminate and likely to change.

**NAME**

recnews – receive unprocessed articles via mail

**SYNOPSIS**

*/usr/lib/news/recnews* [ *newsgroup* [ *sender* ] ]

**DESCRIPTION**

*Recnews* reads a letter from the standard input; determines the article title, sender, and newsgroup; and gives the body to inews with the right arguments for insertion.

If *newsgroup* is omitted, the to line of the letter will be used. If *sender* is omitted, the sender will be determined from the from line of the letter. The title is determined from the subject line.

**SEE ALSO**

inews(8), uurec(8), sendnews(8), readnews(1), checknews(1)



**NAME**

restor – incremental file system restore

**SYNOPSIS**

*/etc/restor* key [ *argument ...* ]

**DESCRIPTION**

*Restor* is used to read magtapes dumped with the *dump* command. The *key* specifies what is to be done. *Key* is one of the characters **rRxt** optionally combined with **f**.

**f** Use the first *argument* as the name of the tape instead of the default.

**r or R**

The tape is read and loaded into the file system specified in *argument*. This should not be done lightly (see below). If the key is **R** *restor* asks which tape of a multi volume set to start on. This allows *restor* to be interrupted and then restarted (an *ichck* -s must be done before restart).

**x** Each file on the tape named by an *argument* is extracted. The file extracted is placed in a file with a numeric name supplied by *restor* (actually the inode number). In order to keep the amount of tape read to a minimum, the following procedure is recommended:

Mount volume 1 of the set of dump tapes.

Type the *restor* command.

*Restor* will announce whether or not it found the files, give the number it will name the file, and rewind the tape.

It then asks you to 'mount the desired tape volume'. Type the number of the volume you choose. On a multivolume dump the recommended procedure is to mount the last through the first volume in that order. *Restor* checks to see if any of the files requested are on the mounted tape (or a later tape, thus the reverse order) and doesn't read through the tape if no files are. If you are working with a single volume dump or the number of files being restored is large, respond to the query with '1' and *restor* will read the tapes in sequential order.

If you have a hierarchy to restore you can use [dumpdir\(8\)](#) to produce the list of names and a shell script to move the resulting files to their homes.

**t** Print the date the tape was written and the date the filesystem was dumped from.

The **r** option should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape onto this. Thus

```
/etc/mkfs /dev/rrp0g 145673
restor r /dev/rrp0g
```

is a typical sequence to restore a complete dump. Another *restor* can be done to get an incremental dump in on top of this.

A *dump* followed by a *mkfs* and a *restor* is used to change the size of a file system.

**FILES**

default tape unit varies with installation  
rst\*

**SEE ALSO**

[dump\(8\)](#), [mkfs\(8\)](#), [dumpdir\(8\)](#)

**DIAGNOSTICS**

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

**BUGS**

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, *restor* doesn't use it.

**NAME**

route, routed, remroutes – IP gateway routing

**SYNOPSIS**

```
/usr/ipc/mgrs/routed [ -v ] [ -t ] [ -q ] [ -hops ] [ addr ... ]
```

```
route add dest gateway
```

```
route delete dest
```

```
/etc/remroutes
```

**DESCRIPTION**

*Routed* runs the 4BSD RIP routing protocol on an IP network. It broadcasts routing information to the network at large, listens for routing messages from elsewhere, and informs the system of the routes it receives.

The options are

**-v** Log transmitted messages on the standard output.

**-t** Log received messages on the standard output.

**-q** Accept routing information but do not broadcast any.

**-hops** Add *hops* (a decimal number) to the hop count when broadcasting routes.

Information received for any named *addrs* is ignored.

*Routed* is usually run without options on gateway machines, and with the **-q** option on non-gateway machines.

*Route* sets up specific routes, to establish static routing or to adjust that set up by *routed*. The **add** command informs the system that internet address *dest* may be reached through internet address *gateway*; **delete** removes any routing for *dest*. The special destination \* represents the default routing: **route add** \* *gate* sets the default, **route delete** \* removes any default.

*Remroutes* removes all known routes.

**FILES**

```
/usr/ipc/log/routed
```

**SEE ALSO**

[ipconfig\(8\)](#)

**NAME**

sa, accton – system accounting

**SYNOPSIS**

`/etc/sa [ -abcdDfgijkKlnrstuv ] [ -e prefix ] [ file ]`

`/etc/accton [ file ]`

**DESCRIPTION**

With an argument naming an existing *file*, *accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

*Sa* reports on, cleans up, and generally maintains accounting files.

*Sa* is able to condense the information in `/usr/adm/acct` into a summary file `/usr/adm/savacct` which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system `/usr/adm/acct` can grow by 10000 blocks per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; `/usr/adm/acct` is the default.

Output fields are labeled: `cpu` for the sum of user and system times (in minutes), `re` for real time (also in minutes), `k` for cpu-time averaged core usage (in 1K units), `avio` for average number of IO operations per execution. With options fields labelled `tio` for total IO operations, `k*sec` for cpu storage integral (kilo-core seconds), `u` and `s` for user and system cpu time alone (both in minutes) will sometimes appear.

There are zillions of options:

- a** Place all command names containing unprintable characters and those used only once under the name `***other`.
- b** Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.
- c** Besides total user, system, and real time for each command print percentage of total time over all commands.
- d** Sort by average number of disk IO operations.
- D** Sort by total number of disk IO operations.
- e** Set the prefix for accounting file names to the next argument (`/usr/adm/` is the default).
- f** Assume answer `y` for option `-v`.
- g** Ignore `/usr/adm/acct`. Useful for processing only `savacct` and `usracct`.
- i** Don't read in summary file.
- j** Instead of total minutes time for each category, give seconds per call.
- k** Sort by cpu-time average memory usage.
- K** Print and sort by cpu-storage integral.
- l** Separate system and user time; normally they are combined.
- m** (money) Print number of processes and number of CPU minutes for each user.
- n** Sort by number of calls.
- r** Reverse order of sort.
- s** Merge accounting file into summary file `/usr/adm/savacct` when done.
- t** For each command report ratio of real time to the sum of user and system times.
- u** Superseding all other flags, print for each command in the accounting file the userid and command name.

- v Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with *y*, add the command to the category **\*\*junk\*\***. This is used to strip out garbage.

**FILES**

/usr/adm/acct  
raw accounting

/usr/adm/savacct  
summary

/usr/adm/usracct  
per-user summary

**SEE ALSO**

[ac\(8\)](#), [acct\(2\)](#)

**BUGS**

*Sa* needs more options.

**NAME**

savecore – save a core image of the operating system

**SYNOPSIS**

*/etc/savecore target dump*

**DESCRIPTION**

*Savecore* copies the core image saved after an operating system crash to an ordinary file. This is worth doing so that the crash image will not be overwritten immediately by another crash, or sometimes because the crash image was written in a place where normal system operation will overwrite it (e.g. in the swap area).

The crash image is taken from *dump* and written to *target*. If *target* exists and is a directory, the image is copied to a file in that directory with the first nonexistent name in the sequence **z.0 z.1 z.2 ...**; otherwise *target* is created or overwritten.

The crash image to be copied is checked for a magic number in a known location. If the magic number is correct, it is followed by the size of the image, and the time it was written; these numbers are printed before the dump is copied. If the magic number is wrong, the image is not copied. *Savecore* overwrites the magic number in *dump* after a successful copy.

The program runs faster if *dump* is the raw device.

For compatibility with an older program of the same name, the *dump* argument may be omitted; *savecore* will noisily examine each device specified for swapping in *fstab*(5) and each of several popular default swap devices for a valid magic number. The first device that looks right is taken to be the crash image.

*Savecore* is usually called when the system is booted, from *rc*(8).

**EXAMPLE**

```
/etc/savecore /tmp/dump /dev/rra11
```

**SEE ALSO**

*reboot*(8)

**BUGS**

The argument convention (the file to be written comes first) is unfortunate; it stems from compatibility.

**NAME**

scsish – SCSI shell

**SYNOPSIS**

**/usr/lib/worm/scsish**

**DESCRIPTION**

*Scsish* is a command interpreter for SCSI commands executed through **/dev/scsi** (see [scsi\(4\)](#)). Many commands are applicable to more or less all SCSI devices; some are specific to the SONY WDA-3000-10 optical disk jukebox. Any details not found here are in the manual for the jukebox.

Typically commands are sent to a particular drive (a number between 0 and 7 inclusive) on a particular device (normally a number between 0 and 5 inclusive). Most commands take a drive parameter (a number). The device number is set by the **id** command.

Occasionally, commands fail and will print the result of a **sense** command which is normally needed to clear the error status.

All input is in lower case and keywords and numbers are separated by white space. Commands are separated by a newline or semicolon.

**General SCSI Commands****capacity** *drive*

Report the capacity of *drive* as **nblocks x blocksize**.

**disk eject** *drive*

Eject the disk (or other removable medium) from *drive*.

**echo** *number*

Print *number* on standard output.

**help** Print a summary of the available commands.

**id** *n* Set the destination SCSI bus device number. By default, it is 2 which is the normal device number for the SONY jukebox.

**inquiry** *drive*

Print various bits of status about *drive*. For example,

```
drive 2,0: WORM device, '    SONY    WDA-3000-10 2.D'
        disk,write protect,,ready (0x9)
```

If *drive* is omitted, an inquiry is performed for drives 0 through 7.

**read** *drive block*

Print the contents of the 1024 byte block at *block* on *drive* in hexadecimal.

**read id** *drive*

Print the string starting at byte 42 in block 1 on *drive*. This corresponds to the initial *vol\_id* for [worm\(8\)](#) disks.

**reset** Attempt to reset the SCSI interface.

**sense** *drive*

Print the sense data for *drive*. Some of the interpretations of the sense bytes are idiosyncratic to SONY.

**ext sense** *drive*

Print the extended sense data for *drive*. Most of the interpretations of the extended sense bytes are idiosyncratic to SONY.

**sleep** *n*

Sleep for *n* seconds.

**start** *drive*

Start *drive* spinning.

**stop** *drive*

Stop *drive*.

**test drive**

Test unit ready for *drive*.

**SONY Commands****alternate drive**

Print the replacement block tables from the disk.

**media drive blkno nblks**

Print a summary of the media quality in *drive* for the *nblks* blocks starting at block number *blkno*. For example, a dirty disk can yield

```
drive 0: media check for 1000 blocks [0-999], upper drive
849 good, 1 unwritten, 147 <50 burst, 3 >96 burst,
```

Please report any instances of messages including **rare error** to the jukebox guru.

**ext media drive blkno nblks**

A verbose form of the **media** command.

**Jukebox Commands**

**config** Print the configuration data for the jukebox.

**rel drive shelf side**

Release the disk from *drive* to *shelf*. The value of *side* indicates whether it should be inverted on the way (**b**) or not (**a**). If *shelf* and *side* are absent, the disk is restored to its former shelf.

**set shelf side drive**

Put the disk from *shelf* into *drive*. The value of *side* indicates whether it should be inverted on the way (**b**) or not (**a**).

**internal n**

Execute various internal reports and diagnostics. **internal -1** with no argument will print a list of available diagnostics.

**status drive**

Print the status for *drive*. An absent *drive* is taken as 0. As the status is for the jukebox as a whole, the value of *drive* doesn't matter. A sample status output shows the jukebox hides the mapping of logical drive number and actual drive:

```
drive 0: ready,disk in LUN,power on,disk in drive 0, return shelf 2
drive 1: not ready,no disk in LUN,power on,disk in shelf 0
drive 2: not ready,no disk in LUN,power on,disk in shelf 0
drive 3: ready,disk in LUN,power on,disk in drive 1, return shelf 0
drive 4: not ready,no disk in LUN,power on,disk in shelf 0
drive 5: not ready,no disk in LUN,power on,disk in shelf 0
drive 6: not ready,no disk in LUN,power on,disk in shelf 0
drive 7: not ready,no disk in LUN,power on,disk in shelf 0
0: no disk
1: no disk
2: disk,
I/O shelf: no disk
carrier: disk shelf=0
upper drive: disk, LUN=0
lower drive: disk, LUN=3
```

**SEE ALSO**

[worm\(8\)](#), [scsi\(4\)](#)

**NAME**

sendcover – send cover sheet to the library

**SYNOPSIS**

**sendcover** *file* ...

**DESCRIPTION**

*Sendcover* sends a document cover sheet to the Bell Laboratories library for their document database. It is invoked automatically as a byproduct of running *troff -mcs*.

The cover sheet is translated from the form of *mcs(6)* to a form used in the library and certain other protocol information is added. The destination is not the same as that of *docsubmit(1)*.

**FILES**

/usr/lib/tmac/tmac.cs

/usr/lib/tmac/cstrans

**SEE ALSO**

*docsubmit(1)*, *mcs(6)*



**NAME**

sendnews – send news articles via mail

**SYNOPSIS**

sendnews [ **-o** ] [ **-a** ] [ **-b** ] [ **-n** newsgroups ] destination

**DESCRIPTION**

*sendnews* reads an article from its standard input, performs a set of changes to it, and gives it to the mail program to mail it to *destination*.

An 'N' is prepended to each line for decoding by *uurec(1)*.

The **-o** flag handles old format articles.

The **-a** flag is used for sending articles via the **ARPANET**. It maps the article's path from *uucphost!xxx* to *xxx@arpahost*.

The **-b** flag is used for sending articles via the **Berknet**. It maps the article's path from *uucphost!xxx* to *berkhost:xxx*.

The **-n** flag changes the article's newsgroup to the specified *newsgroup*.

**SEE ALSO**

*inews(8)*, *uurec(8)*, *recnews(8)*, *readnews(1)*, *checknews(1)*

**NAME**

ship, shipstat – automatic software distribution

**SYNOPSIS**

**ship** [ *option ...* ] [ *file ...* ]

**shipstat**

**DESCRIPTION**

*Ship* distributes the named files to other computers, where the files are installed under the same names. Shipping privileges are determined by the network manager on the receiving machine; see [svcmgr\(8\)](#).

Destinations are taken from environment variable **dest**, or from `/usr/lib/asd/dest/default` if **dest** is empty. If a destination is the name of a file in it is replaced by the contents of that file, each word of which is then examined in the same way. Otherwise the destination is a network address. The sending machine is omitted unless explicitly named in the environment variable, or unless option **-f** is present or environment variable **force** has a non-empty value.

*Ship* uses *inspkg* and [mkpkg\(8\)](#) to do its work. Links among the named files are imitated on the receiving computer, and named files that do not exist on the sending computer are deleted on the receiving computer. Other options are the same as those of *mkpkg*:

**-v** Emit running commentary on the standard error file.

**-D***path1* = *path2*

Pretend that any *file* name that begins with *path1* really begins with *path2*. Relative pathnames are extended to full pathnames before comparison.

**-x***command*

**-X***file* Include in the package instructions to execute the shell *command* or run the shell script *file* after all files have been installed. Only one of these options may occur. The *file* name in **-X** is affected by **-D**.

Shipments are generally acknowledged by mail after each destination has been tried at least once; see [asd\(8\)](#) for details.

*Shipstat* reports the status of all its caller's incomplete shipments, with the most recent first.

**FILES**

`/usr/lib/asd/dest/*`  
distribution lists

`/usr/lib/asd/dest/default` default distribution list

`/usr/spool/asd/logname`  
outgoing spool directories

**SEE ALSO**

[mkpkg\(8\)](#), [asd\(8\)](#), [svcmgr\(8\)](#)

**BUGS**

The **-f** option, if given, must be the first option and must not be combined with any other.

**NAME**

showq – status of stream input/output system

**SYNOPSIS**

`/etc/showq [ -v -V -s -m ] [ system ] [ mem ]`

**DESCRIPTION**

*Showq* reports connectivity and contents of I/O streams. By default, it lists the maximum number of stream blocks (of various sizes) ever used, then each stream and the queue modules in each stream, and then blocks that are unaccounted for (not on any queue or the free list).

By default, the system namelist is **/unix** and the place the streams are kept is **/dev/mem**.

The options are:

- v** Verbose. Show more, in particular the contents of data and control blocks on each queue.
- V** Very verbose. Show all blocks on every queue instead of giving up after a while.
- s** Silent. Examine queues for consistency, printing only a summary.
- m** Missing. Show the contents of missing blocks. (Perhaps this will give a clue about who lost them.)

**FILES**

`/unix`  
`/dev/mem`

**SEE ALSO**

[stream\(4\)](#), [netstat\(8\)](#)  
[mesgld\(4\)](#) for a list of message types

**NAME**

shutdown – take system down gracefully

**DESCRIPTION**

To be supplied.

**SEE ALSO**

reboot(8)

**NAME**

smash – rewrite bad disk sectors

**SYNOPSIS**

*/etc/smash device sector*

**DESCRIPTION**

*Smash* attempts to read the named (decimal, 512-byte) *sector* from the named *device*, and prints the error status from the read and the data read, in octal, regardless of the error status. It then prompts *write?*, to which there are three answers:

**y** Write the data back to the sector.

**c** Write zeros to the sector.

anything else

Quit.

After the sector is written, it is read again and the cycle repeats.

Writing the sector, even if its contents could be correctly read, will recompute the error correcting code. This may make soft ECC errors vanish, and will recover what can be recovered (sometimes not much) from hard ECC errors.

**SEE ALSO**

*rarepl(8)*

**NAME**

smstat – list smtp queues

**SYNOPSIS**

**smstat**

**DESCRIPTION**

*Smstat* prints a summary of pending mail messages queued by the programs in [smtp\(8\)](#). Each line contains the name of a spooling directory; the number of outbound messages, followed by C; and the number of inbound messages, followed by X.

**FILES**

/usr/spool/smtpq/\* spool directories

**SEE ALSO**

[smtp\(8\)](#)

**NAME**

smtp, smtpqer, smtpd, smtpsched – handle simple mail transfer protocol

**SYNOPSIS**

**/usr/lib/upas/smtp** [ *option ...* ] *replyaddr dest recipient ...*

**/usr/lib/upas/smtpqer** [ *option ...* ] *replyaddr dest recipient ...*

**/usr/lib/upas/smtpd** [ **-n** ] [ **-H** *host* ]

**/usr/lib/upas/smtpsched** [ *option ...* ] [ *queue ...* ]

**DESCRIPTION**

*Smtp* reads a mail message from the standard input, and sends it with the Internet SMTP protocol to the named *recipients* at network address *dest*. *Dest* has default network **tcp** and default service **tcp.25** (the conventional Internet SMTP port). Error reports are mailed to local address *replyaddr*.

*Smtp* operates in two modes, ‘Internet’ (default) and ‘Unix’. In Internet mode *recipient* addresses should be in full domain form. *From:* and *Date:* headers will be inserted as necessary to conform to Internet standards. In Unix mode addresses and message contents are not touched. The options are

**-u** Run in Unix mode.

**-H** *host*

Use *host* as the name of the sending system (taken from *whoami(5)* by default).

**-d** *domain*

Append the specified domain suffix to incomplete addresses.

*Smtpqer* reads a mail message from the standard input and stashes it away to be sent later by *smtpsched*. By default, *smtpsched* is started immediately; option **-n** prevents this. Other options and arguments are the same as for *smtp*.

*Smtpd* receives a message by speaking the server part of SMTP on the standard input and output. The message is stashed in a queue for later delivery as by *smtpqer*. Option **-n** prevents *smtpsched* from running immediately; option **-H** is as for *smtp*.

*Smtpsched* processes the queues assembled by *smtpqer* and *smtpd*, calling *mail(1)* for local messages and *smtp* for others. It should be run occasionally from *cron(8)*.

The *queue* arguments name particular queue directories to be processed; if no queue is named, all queues are processed. The options are

**-w** *days*

Send a warning about each message more than *days* old to the reply address.

**-r** *days*

Mail an error reply about each message more than *days* old, and discard the message.

**-s** *nproc*

Do not run more than *nproc* simultaneous copies of *smtpsched* started with this option.

**-c** Remove empty directories and inconsistent files.

**-t** Log actions without performing them.

**-C** Process ‘C’ command files (*smtp* calls) only.

**-X** Process ‘X’ command files (*rmail* calls) only.

**-v** Enable verbose logging.

The queues are kept in subdirectories of named by splitting the lower case remote system name into components separated by periods, concatenating the last two or fewer components, taking the last 14 characters, and stripping leading periods.

**FILES**

**/usr/spool/smtpq**

spooling directory

**/usr/spool/smtpq/smtpqsched.log**  
logging

**/usr/spool/smtpq/.consumers**  
list of process IDs running *smptqsched -s*

**SEE ALSO**

[mail\(1\)](#), [upas\(8\)](#), [smstat\(8\)](#)

DARPA standards RFC 822, RFC 976



**NAME**

sticky – executable files with persistent text

**DESCRIPTION**

While the 'sticky bit', mode 01000 (see *chmod(2)*), is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. (Directory entries can be removed so long as one link remains.)

Sharable files are made by the *-n* and *-z* options of *ld(1)*.

To replace a sticky file that has been used do: (1) Clear the sticky bit with *chmod(1)*. (2) Execute the old program to flush the swapped copy. This can be done safely even if others are using it. (3) Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with *chmod* and *chown(2)*(4) Set the sticky bit again.

Only the super-user can set the sticky bit.

**BUGS**

Are self-evident.

Is largely unnecessary on the VAX; matters only for large programs that will page heavily to start, since text pages are normally cached incore as long as possible after all instances of a text image exit.

**NAME**

su, setlog – substitute userid temporarily, become super-user

**SYNOPSIS**

*/etc/su* [ *user* ]

*/etc/setlog* *logname command ...*

**DESCRIPTION**

*Su* changes the userid to that of *user* (root by default) with groupid and login shell determined from the password file. If the current userid is not the super-user, the password for the new user is demanded. The userid stays in force until the new shell exits.

The current directory and environment are unchanged unless the new userid is super-user, in which case the environment variables **PS1** and **PATH**, if present, are set to standard values (white space and **/bin:/usr/bin:/etc**).

*Setlog* executes the specified *command* with login name *logname*. The environment is otherwise unchanged; in particular, the userid is not set. Only the super-user may use this command.

**SEE ALSO**

*sh(1)*, *getuid(2)*, *passwd(5)*

**NAME**

svcmgr – service remote computing requests

**SYNOPSIS**

`/usr/IPC/mgrs/svcmgr [ -d ]`

**DESCRIPTION**

*Svcmgr* performs services such as login and command execution, often in response to requests from network listeners like *dkmgr* and *tcpmgr*(8). It should be run once from *rc*(8).

*Svcmgr* is controlled by several files in directory services are defined in files **serv** and **serv.local**, authorization in **auth** and **auth.local**. The **.local** files are searched first. The idea is that **serv** and **auth** will be the same throughout an administrative cluster of machines, and anything peculiar to specific systems will be kept in **serv.local** and **auth.local**.

Each service is announced as a name in directory /CS using the routines in *ipc*(3). When a connection is requested to one of these services, *svcmgr* receives a file descriptor connected to the requester. A new process is created to perform the actions listed for that service in the *serv* files, usually resulting in a *login*(8) with standard input, output, and error files attached to the connection. Often there are flags to *login* specifying a local user name or a command to be executed. Environment variable CSOURCE is set to a string of the form

**source=remote-machine user=ruser line=lineinfo**

*Remote-machine* and *ruser* are supplied in the connection message; *lineinfo* network-dependent stuff of varying interest and meaning. If a particular command was specified (the **cmd** or **exec** action), *login* sets environment variable REXEC to 1.

The *auth* files are used to translate remote user names to local ones. They contain lines with four fields:

service name  
calling system name  
calling user name  
local user name

The service, calling system, and calling user names are regular expressions in the style of *regexp*(3). The calling system and calling user fields may be omitted; .\* is assumed. The local user name is a literal name, . to repeat the calling user name provided in the request, or : to explicitly reject a call. If the local user name is omitted, . is assumed.

Several service actions ‘look up the connection in the *auth* files.’ This means to find the first line in **auth.local** or **auth** for which the service, calling system, and calling user match the patterns, and return the local user name in that line (the same as the calling user if .). If no matching line is found, or if the first match has local user name :, the lookup fails.

The *serv* files contain lines with three fields:

service name  
a list of actions, separated by +  
the calling system name

The calling system name is a regular expression as in the *auth* file. The line matching an incoming call is the first whose service matches the requested service and whose regular expression matches the calling machine.

The possible actions are:

user(*x*)

Use local username *x*.

auth Look up the connection in the *auth* files. If a match is found, use the resulting local user. Otherwise reject the call.

v9auth

Look up the connection in the *auth* files; if a match is found, send OK to the caller, and use the result. If there is no match, send NO, and read a string of the form ‘login,passwd\n’. If the login and password describe a valid local user, send OK and use that user; otherwise send NO and try again (until the caller gives up). This is the authentication protocol used by *ipclogin* (see *ipc*(3)),

hence by *con(1)*, *push(1)*, and *pull*.

**inauth**

Read two null-terminated strings from the caller. If they aren't the same, reject the call. Otherwise look up the service, calling system, and the null-terminated string (as a user name) in the *auth* files, use the resulting local user if there's a match, reject the call otherwise. This is the authentication protocol used by *ipcrogin*, hence by *rsh* and *rlogin*; see *ipc(3)* and *con(1)*.

**tttyld**

Push the terminal line discipline *tttyld(4)* onto the connection.

**mesgld**

Push the reverse message line discipline (see *mesgld(4)*) onto the connection.

**term**

Read a null-terminated string from the caller, and set environment variable TERM to the result.

**args**

Read a null-terminated string from the caller, and save the result as arguments to a possible command.

**s5parms**

Extract arguments from the destination address in a way compatible with the DKHOST network software used by System V Datakit implementations, and save for later use.

**cmd(x)**

Execute shell command *x*, with any saved arguments, and with the connection as standard input, output, and error.

**login**

Provide a login session with the connection as standard input, output, and error.

**password**

Provide a login session, but ignore any local user name; always demand a login and password.

**exec**

Use any saved arguments as a shell command to be executed.

**gateway(gateway)**

Call network address *gateway* and send the connection info there. If all is well, pass the new connection's file descriptor to the original caller: the result is a connection through the gateway. *Gateway* should be a *svcmgr* service, perhaps on some other machine, with action **gateway**.

**gateway(localout)**

The intended target for **gateway**: read new connection info from the connection, and place a call to the new destination; if it succeeds, loop passing data between the new connection and the original one.

If the file */usr/ipc/log/svc* can be opened, *svcmgr* prints miscellaneous chatter there, including a record of each service request. The **-d** (debug) option increases the chatter.

## FILES

*/usr/ipc/lib/serv*  
*/usr/ipc/lib/serv.local*  
*/usr/ipc/lib/auth*  
*/usr/ipc/lib/auth.local*  
*/usr/ipc/log/svc*

## SEE ALSO

*con(1)*, *ipc(3)*, *dkmgr(8)*, *tcpmgr(8)*, *ipc(3)*

**NAME**

*swapon* – specify swapping device

**SYNOPSIS**

*/etc/swapon -a*

*/etc/swapon name ...*

**DESCRIPTION**

*Swapon* specifies additional devices on which paging and swapping are to take place. The system begins by using a single device; *swapon* must be used to enable others.

Usually there is a call to *swapon -a* in [rc\(8\)](#). Specific swap devices may be nominated with the second form.

**SEE ALSO**

[rc\(8\)](#), *vswapon* in [deprecated\(2\)](#)

**BUGS**

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

Possible swap devices must be listed in a table configured into the system; *swapon* can only enable devices in the table.

**NAME**

symorder – rearrange name list

**SYNOPSIS**

**symorder** orderlist symbolfile

**DESCRIPTION**

*Orderlist* is a file containing symbols to be found in symbolfile, 1 symbol per line.

*Symbolfile* is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from /vmunix.

**SEE ALSO**

nlist(3)

**NAME**

*sync*, *update* – update disk file systems

**SYNOPSIS**

*sync*

*/etc/update*

**DESCRIPTION**

*Sync* executes the [sync\(2\)](#) system primitive, to suggest that all disk writes be completed soon. It is wise to call *sync* before halting the system.

*Update* calls [sync\(2\)](#) every 30 seconds, so that file systems are fairly well up to date if the system crashes. It should be run once from [rc\(8\)](#).

**SEE ALSO**

[sync\(2\)](#), [reboot\(8\)](#)

**NAME**

tcpmgr – accept and place calls via the TCP protocol

**SYNOPSIS**

*/usr/ipc/mgrs/tcpmgr* [ **-m** *outnet* ]

**DESCRIPTION**

*Tcpmgr* receives and places TCP calls on an Internet TCP/IP network. Outbound TCP calls may be placed by calling *ipccopen* with network name **tcp** (see *ipc(3)*). Inbound calls to TCP port *n* are handed to the local service listed for that port in or to service **tcp.n** if there is no listing.

Option **-m** tells *tcpmgr* to claim to place outbound calls for network *outnet* rather than **tcp**.

The TCP protocol runs atop one or more IP networks. *Tcpmgr* arranges to receive all inbound TCP calls on all active IP networks, but other arrangements must be used to activate the IP networks themselves; see *ipconfig(8)*.

*Tcpmgr* records its activity in file *outnet* in directory **/usr/ipc/log**, default **/usr/ipc/log/tcp**.

This command is usually run once from *rc(8)*.

**FILES**

*/usr/ipc/log/tcp*

TCP network devices

*/dev/iptcp*

IP channel for the TCP protocol

*/usr/ipc/lib/in-services*

mapping between service name and port number

**SEE ALSO**

*con(1)*, *ipconfig(8)*, *svcmgr(8)*, *ipc(3)*



**NAME**

`tp` – manipulate tape archive

**SYNOPSIS**

`tp` [ *key* ] [ *name ...* ]

**DESCRIPTION**

*tp* saves and restores files on DECTape or magtape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the *key* is specified by one of the following letters:

- r** The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so `./abc` can never be the same as `/usr/dmr/abc` even if `/usr/dmr` is the current directory. If no file argument is given, `.` is the default.
- u** updates the tape. **u** is like **r**, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **u** is the default command if none is given.
- d** deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.
- x** extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.
- t** lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

- m** Specifies magtape as opposed to DECTape.
- 0,...,7** This modifier selects the drive on which the tape is mounted. For DECTape, **x** is default; for magtape `'0'` is the default.
- v** Normally *tp* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.
- c** means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u**. This option is assumed with magtape since it is impossible to selectively overwrite magtape.
- i** Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.
- f** Use the first named file, rather than a tape, as the archive. This option currently acts like **m**; *i.e.* **r** implies **c**, and neither **d** nor **u** are permitted.
- w** causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response **x** means 'exit'; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r**, **u**, and **d** no change has been made to the tape.

**FILES**

`/dev/tap?`  
`/dev/rmt?`

**SEE ALSO**

`ar(1)`, `tar(1)`

**DIAGNOSTICS**

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

**BUGS**

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar(1)* avoids the problem.

**NAME**

upas, rmail, translate – mail delivery system

**SYNOPSIS**

**rmail** *person ...*

**/usr/lib/upas/translate** *name*

**/usr/lib/upas/auth** *sender receiver*

**DESCRIPTION**

Users send mail by *mail(1)*. Remote machines use *rmail*. Both call on *upas* programs.

Mail addresses are interpreted according to rewrite rules as described below. When the addresses for a *mail* or *rmail* command have been interpreted, they are bundled by network and passed to network-specific handlers, such as *route.inet*.

*Translate* looks up a mail *name* in an alias list (see *mail(6)*) and places the result on standard output.

*Auth* is called by *upas* to authorize mail delivery for each *sender, receiver* pair. The mail is accepted if the previous hop was a trusted gateway machine in or all the machines in the source or destination path are in

**Rewrite rules**

Each line of the file */usr/lib/upas/rewrite* is a rule. Blank lines and lines beginning with *#* are ignored.

Each rewriting rule consists of (up to) 4 strings:

*pattern*

A regular expression in the style of *regexp(3)*. The *pattern* is applied to mail destination addresses. The pattern match is case-insensitive and must match the entire address.

*type* The type of rule; see below.

*arg1* An *ed(1)* style replacement string, with *\n* standing for the text matched by the *n*th parenthesized subpattern.

*arg2* Another *ed(1)* style replacement string.

In each of these fields the substring *\s* is replaced by the login id of the sender and the substring *\l* is replaced by the name of the local machine.

When delivering a message, *mail* starts with the first rule and continues down the list until a pattern matches the destination address. It then performs one of the following actions depending on rule type:

>> Append the mail to the file indicated by expanding *arg1*, provided that file appears to be a valid mailbox.

| Pipe the mail through the command formed from concatenating the expanded *arg1* and *arg2*.

**alias** Replace the address by the address(es) specified by expanding *arg1* and recur.

**translate**

Replace the address by the address(es) output by the command formed by expanding *arg1* and recur.

**auth** Call the program in expanded *arg1* and *authorize(reject)* the mail if it returns a zero(non-zero) return code.

*Mail* expands the addresses recursively until each address has matched a >> or | rule or until the recursion depth indicates a rewriting loop (currently 32).

An *auth* operator is only applied once per address. If no *auth* rule is encountered, the mail is accepted.

If several addresses match | rules and result in the same expanded *arg1*, the message is delivered to all those addresses by a single command, composed by concatenating the common expanded *arg1* and each expanded *arg2*. This is meant to make less work of a message to several recipients on the same machine. For example, the rule

```
([^\!]+)!(.+)| "uux - -a \s \l!rmail" \2
```

causes *mail* to generate the single delivery command `uux -a rob r70!rmail pjw ken`.

**EXAMPLES**

A sample rewrite file:

```
# local mail
[^\!@]+          translate "exec translate '.'"
local!([^\!@]+) >> /usr/spool/mail/\1
\1!(.+)         alias \1

# convert @ format to ! format
(_822_)!((.+)!)?([^\!]+)[@]([^\!@]+) \
                alias \1!\2\5!\4
([^\!]+)[@]([^\!@]+) alias _822_!\2!\1
_822_!(.+)      alias \1

# special domains
[^\!]+wisc\.edu!.+ alias xunet!.

# network gateways
(csnet|bitnet)!(.+) aliasinet!.
acsnet!.+

# real networks
inet!([^\!]+)!(.+) | "/usr/lib/upas/route.inet '\s' '\1' " '\2'"
([^\!]+)!(.+)      | "/usr/lib/upas/route '\s' '\1' " '\2'"
```

**FILES**

```
/usr/lib/upas/namefiles
    list of files to search

$HOME/lib/names
    private aliases

/usr/lib/upas/rewrite
    rewriting rules

/usr/lib/upas/attlist
    known AT.T machines

/usr/lib/upas/gateways
    machines that check mail authorization reliably

/usr/lib/upas/translate
    alias lookup

/usr/lib/upas/route.*
    mail interfaces to specific networks

/usr/lib/upas/route
    interface to local (i.e. AT.T) Datakit network

/bin/mail
    shell file that calls the mailer

/usr/lib/upas/send
    actually delivery program

/bin/rmail
    linked to /usr/lib/upas/send

/usr/spool/mail/*
    mailboxes

/usr/spool/mail/mail.log*
    delivery logs

/n/bowell/usr/lib/upas/mkfile
    updates various mail and uucp files
```

/etc/passwd  
authentication

/tmp/ma\*  
temp file

/tmp/ml\*  
lock file

\$HOME/dead.letter  
unmailable text

**SEE ALSO**

[uucp\(1\)](#), [mail\(1\)](#), [mail\(6\)](#), [smtp\(8\)](#)

D. L. Presotto and W. R. Cheswick, 'Upas—a simpler approach to network mail', this manual, Volume 2

**NAME**

uucico, uusched, uuxqt, kick, debug – uucp file transport and execution

**SYNOPSIS**

`/usr/lib/uucp/uucico` [ **-r1** **-s** *system* ] [ **-x** *debug* ] [ **-d** *spool\_directory* ]

`/usr/lib/uucp/uusched` [ **-x** *debug* ] [ **-u** *debug* ]

`/usr/lib/uucp/kick` *system*

`/usr/lib/uucp/debug` *system*

`/usr/lib/uucp/uuxqt` [ **-s** *system* ] [ **-x** *debug* ]

**DESCRIPTION**

*Uucico* transfers files between systems for *uucp*(1). It is normally invoked by *login* for an incoming connection (a slave), or by *uusched* to call out to another system (a master). The options are

**-r1** This is a master; option **-s** is required. In the absence of **-r1** the process is a slave and expects to speak to a master on standard input and standard output.

**-s** *system*  
Call the named *system*.

**-x** *debug*  
Turn on debugging output; *debug* is a single digit, larger for more output.

*Uusched* invokes *uucico* for each system with work pending. It is called by *uucp* and *uux*(1) after work is queued, and should be invoked regularly by *cron*(8). The options are

**-x** *debug*  
As for *uucico*.

**-u** *debug*  
Invoke *uucico* with option **-x** *debug*.

*Kick* invokes *uucico* in the background to call out to the specified *system*. *Debug* causes a call with a moderate amount of debugging output (**-x4**). Both attempt to remove existing system status information for *system*, so that a call will be attempted regardless of recent failures.

*Uuxqt* executes commands requested remotely by *uux*(1). It searches the *uucp* spool directories looking for filenames starting with **X.**, checks the *Permissions* file to see that all required data files are available and accessible and that the requested command is permitted for the requesting system, and invokes the command if all is well.

Before a command is invoked, the file creation mask (*umask*(2)) is set to 0 and these environment variables are set:

**UU\_MACHINE**

the name of the last sending machine

**UU\_USER**

the user that sent the job

**PATH**

set to `/bin:/usr/bin`

**USER**

set to `uucp`

**FILES**

`/usr/lib/uucp/Systems*`

`/usr/lib/uucp/Permissions`

`/usr/lib/uucp/Devices`

`/usr/lib/uucp/Maxuuscheds`

how many copies of *uusched* may be active at once

```
/usr/lib/uucp/Maxuuxqts  
ditto for uuxqt
```

```
/usr/spool/uucp/*
```

```
/usr/spool/uucppublic/*
```

```
/usr/spool/uucp/LCK*
```

**SEE ALSO**

[uucp\(1\)](#), [uux\(1\)](#), [uustat\(1\)](#)

D. A. Nowitz, 'UUCP Administration', this manual, Volume 2

**BUGS**

System and user names received by *uuxqt* should not be believed.

**NAME**

uucleanup – uucp spool directory clean-up

**SYNOPSIS**

**/usr/lib/uucp/uucleanup** [ *options* ]

**DESCRIPTION**

*Uucleanup* removes old files from the *uucp* spool directories. It is typically called by which may be run regularly by *cron*(8). The options are:

**-Cdays**

Remove **C.** (control) files that are at least *days* old, and send a message to the user who queued the job.

**-Ddays**

Remove **D.** (data) files that are at least *days* old. If the data file appears to contain a mail message, an attempt will be made to deliver it; if it contains a netnews article from another system, it will be handed to *mnews*.

**-Wdays**

Send a warning to the user who queued any jobs (**C.** files) at least *days* old. If one of the data files for the job appears to be a mail message, the message is included in the warning.

**-Xdays**

Remove any **X.** (remote execution) files at least *days* old.

**-mstring**

Include *string* in warning messages. The default is 'See your local administrator to locate the problem.'

**-odays**

Remove any other files that are at least *days* old.

**-ssystem**

Examine only files associated with *system*.

By default, **C.** files generate a warning when one day old, and are removed after 7 days; **D.** files are removed after 7 days; and **X.** and other files are removed after 2 days.

**FILES**

*/usr/lib/uucp*

directory with commands used by *uucleanup*

*/usr/spool/uucp*

spool directory

**SEE ALSO**

*uucp*(1)

D. A. Nowitz, 'UUCP Administration', this manual, Volume 2



**NAME**

uurec – receive processed news articles via mail

**SYNOPSIS**

uurec

**DESCRIPTION**

*uurec* reads news articles on the standard input sent by *sendnews(8)*, decodes them, and gives them to *inews(8)* for insertion.

**SEE ALSO**

*inews(8)*, *readnews(1)*, *recnews(8)*, *sendnews(8)*, *newscheck(1)*

**NAME**

*vipw* – edit the password file with vi

**SYNOPSIS**

***vipw***

**DESCRIPTION**

*Vipw* edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later

**SEE ALSO**

*chfn*(1), *chsh*(1), *passwd*(1), *passwd*(5), *adduser*(8)

**FILES**

*/etc/vipw.lock*

**BUGS**

*Vipw* does not remove the *vipw.lock* file; this is not a bug, but people tend to think it is.

No one deals with left-over */etc/ptmp* (the real lock) files after a system crash.

**NAME**

vmstat – report virtual memory statistics

**SYNOPSIS**

**vmstat** [ **-st** ] [ *interval* [ *count* ] ]

**DESCRIPTION**

*Vmstat* reports statistics about certain system activity. Option **-s** prints totals for miscellaneous events since the last boot. Option **-t** reports on paging events.

In the absence of other options, the optional *interval* argument causes *vmstat* to report once each *interval* seconds, repeated *count* times or forever.

The default format fields are:

**procs** information about numbers of processes in various states:

**r** in run queue  
**b** blocked for resources (I/O, paging, etc.)  
**w** runnable or short sleeper (< 20 secs) but swapped

**memory**

use of virtual and real memory:

**avm** number of pages belonging to processes that have run in the last 20 seconds  
**fre** size of memory free list

**page** paging activity, averaged each five seconds, in units per second:

**re** page reclaims (simulating reference bits)  
**at** text pages recovered from memory  
**pi** page-in events  
**po** page-out events  
**fr** pages freed per second  
**de** anticipated short term memory shortfall  
**sr** scan rate: pageout daemon rpm

**faults** trap rates, averaged each five seconds, in units per second:

**sy** system calls  
**cs** process context switches

**cpu** percentage use of CPU time:

**us** user time, both normal and low priority  
**sy** system time  
**id** cpu idle time  
**st** stream queue processing time

**FILES**

/dev/kmem  
 /unix

**BUGS**

This program is never up to date.

**NAME**

`vpac` – print raster printer/plotter accounting information

**SYNOPSIS**

`/etc/vpac [ -W ] [ -s ] [ -r ] [ -t ] [ name ... ]`

**DESCRIPTION**

*Vpac* reads the raster printer/plotter accounting files, accumulating the number of pages (for narrow fan-fold devices) or feet (for wide, roll paper devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars (billed at 2 cents / page or 8 cents / foot). If any *names* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

The `-W` flag causes accounting to be done for a wide roll paper device. The default is to do accounting for a narrow, fan-fold device. The `-t` flag causes the output to be sorted by feet of paper; usually the output is sorted alphabetically by name. The `-r` flag reverses the sorting order. The `-s` flag causes the accounting information to be summarized on the summary accounting file; this summarization is necessary since on a busy system, the accounting file can grow by several lines per day.

**FILES**

<code>/usr/adm/v?acct</code>	raw accounting files
<code>/usr/adm/v?_sum</code>	summary accounting files

**BUGS**

The relationship between the computed price and reality is as yet unknown.

**NAME**

wall – write to all users

**SYNOPSIS**

*/etc/wall*

**DESCRIPTION**

*Wall* reads its standard input until an end-of-file. It then sends this message, preceded by ‘Broadcast Message ...’, to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

**FILES**

*/etc/utmp*

**SEE ALSO**

[write\(1\)](#)

**DIAGNOSTICS**

‘Cannot send to ...’ when the open on a user’s tty file fails.

**NAME**

worm, jukebox – optical disk utilities

**SYNOPSIS**

**worm mkfs** [ **-fdevice** ] [ **-ccomments** ] [ **-bblksz** ] [ **-nblks** ] [ **-vnewvol\_id** ] *vol\_id*

**worm stat** [ **-fdevice** ] [ **-Fn** ] [ **-v** ] [ *vol\_id* ]

**worm ls** [ **-fdevice** ] [ **-l** ] [ *file ...* ]

**worm rm** [ **-fdevice** ] *vol\_id* [ *file ...* ]

**worm mv** [ **-fdevice** ] *vol\_id* *src* *dest*

**worm write** [ **-fdevice** ] *vol\_id* [ *file ...* ]

**worm read** [ **-fdevice** ] [ **-dm** ] *vol\_id* [ *file ...* ]

**worm cat** [ **-fdevice** ] *vol\_id* *file*

**worm copy** [ **-v** ] [ **-mmin\_free** ] [ **-fsrc\_dev** ] *src\_vol\_id* *dest\_dev* *dest\_vol\_id*

**worm offline** [ **-fdevice** ]

**worm btree** [ **-fdevice** ] *vol\_id*

**worm dir** [ **-fdevice** ] *vol\_id*

**worm tmpdir** [ **-fdevice** ] *vol\_id*

**worm mount** [ **-wsecs** ] [ *vol\_id* ]

**jukebox** [ **-aemprsuU** ] [ **-wsecs** ] [ *vol\_id* ]

**DESCRIPTION**

The *worm* programs manipulate arbitrary files. They are intended for use with the raw device associated with a Write-Once Read-Many (WORM) optical disk. The default device is Other devices are specified by **-fdevice** and a device name of a single digit *n* is taken as an abbreviation for Most of the commands implement a simple file system. Programs just wanting a raw device should still use **worm mkfs** so that the disk is properly labeled. The *vol\_id*, or label, should be unique and by convention, the *vol\_id*'s for the A and B sides of a disk should be the same string suffixed by a lowercase **a** and **b** respectively.

*Worm mkfs* labels an optical disk. The *comments* field is limited to 256 chars. It is purely descriptive and is printed by *worm stat -v*. The (default) blocksize is 1024 for our SONY disks. The number of blocks on a disk can be found by *ra(4)* or *scsish(8)*; the default size (1,600,000 for single density, 3,250,000 for double density) sets aside 30MB or so as a hedge against oversights. If the disk has already been initialised, its *vol\_id* must match *vol\_id*. A new *vol\_id* can be set with **-v**.

*Worm stat* prints out labeling information including the amount of free space left on the disk. Option *vol\_id* turns off all output except exit status: zero if *vol\_id* matches that of the disk, one otherwise. Option **-F** similarly exits with status zero if the disk has more than *n* free blocks, otherwise three. Option **-v** produces more output.

*Worm ls* simulates an emasculated *ls(1)*.

*Worm rm* makes the specified files unavailable to the rest of the *worm* commands.

*Worm mv* renames *src* to *dest*.

*Worm write* copies files onto the WORM. If no file arguments are given, filenames are read one per line from standard input. The total number of files and bytes is printed on standard output.

*Worm read* restores files from the WORM. If no file arguments are given, filenames are read one per line from standard input. Option **-d** causes directories to be created as needed. Option **-m** restores the original modification times.

*Worm cat* copies the named file from the WORM to the standard output.

*Worm copy* copies files directly from one disk to another. The names of the files to be copied are taken from standard input; groups (separated by blank lines) will be kept together. The names are typically generated by **worm ls**. The **-v** option prints out progress and summary information. The copy will terminate before copying a group that would leave the destination volume with less than *minfree* (default value is

40000) blocks free.

*Worm offline* makes the WORM go offline, ready for ejecting. This command is harmless; accessing an offline drive will cause it to spin up and go online without operator intervention. *Worm offline* only takes effect after the last close of the WORM and as a bonus, applies to any MSCP device such as an RA81.

*Worm tmpdir* saves a copy of the directory in `/usr/worm/tmp/vol_id` if the directory `/usr/worm/tmp` exists. This will speed up subsequent access substantially, although it will still be slower than *worm btree* below. On the other hand, *worm tmpdir* typically takes 5 minutes to run (on a VAX 11/750) whereas *worm btree* takes about 45 minutes.

*Worm btree* constructs a new directory for the whole disk (in the form of a *cbr(1)* database). The new superblock is at zero. All the worm commands go faster with such an index but it is intended to be done just once, after the disk is complete. The directory occupies of the order of 10MB but may be more. If you really have to add more files to the disk, you need to write zeros on the first 1K block of the WORM before using *worm write*.

*Worm dir* takes the btree directory from the disk and stores in Future uses of the disk will be much faster.

*Worm mount* returns the device on which the disk labelled *vol\_id* is mounted. If the drive(s) are busy and you have a jukebox, the `-ws` option tells how many seconds to wait before failing. The default is wait forever. If no *vol\_id* is given, print the drive status.

*Jukebox* manages the disks in the SONY jukebox. There are several options (default is `-s`):

- a**        Allocate a blank disk and label it *vol\_id*. Use *worm mkfs* to change any fields from their default value.
- e**        Eject the disk labeled *vol\_id*. To physically retrieve the disk, press the **OUT** button (the **OUT READY** light should be on). Repeat until the **IN READY** light goes on.
- m**        Mount the disk labelled *vol\_id* in some drive and print the drive number on standard output.
- p**        Print the list of disks in the jukebox.
- r**        Rebuild the list of disks by examining each disk in the jukebox. Do not do this unless you are sure you need to. If *vol\_id* is given, it should be one of the following letters and governs how disks are assigned shelf numbers. The default is to leave the shelf number unchanged. Other options (mainly useful for demos) are **c** (compresses the disks in the jukebox towards the bottom or lower numbered shelves), **r** (distributes the disks randomly), and **s** (sorts the disks by *vol\_id*).
- s**        Print the status of the jukebox.
- u**        Unload offline disks back onto their shelves.
- U**        Unload all disks (offline or not) back onto their shelves.
- wsecs**    This option only affects the behavior of **-m**. If all drives are busy, try again for *secs* seconds before failing.

To load a disk into the jukebox, press the **IN** button on the jukebox when the **IN READY** light is on. After the shutter opens, push the disk in firmly. The disk (blank or initialised) is not examined immediately but on demand.

### Etiquette

*Vol\_ids* should be unique as discussed above. The file `/n/wild/usr/worm/vol_ids` contains known *vol\_ids*. The commands for reading and writing require *vol\_id*'s to guard against accessing the wrong disk.

The recommended protocol for changing disks is if no one appears to be using the drive (by using *ps(1)*), execute *worm offline* and go to the drive. If, and only if, the drive has the DRIVE OFF (middle) light on, hit the EJECT button and change disks. If the light is not on, then someone is still using the disk and you should wait until they are done before hitting EJECT.

### Programming considerations

Programs should not depend on writing any block more than once; however, our SONY optical disks implement a small number of multiple writes via bad block replacement. A *read(2)* of an unwritten block

returns with an errno of **ENXIO**. On Vaxes, the WORM is an MSCP device; thus geometry information can be fetched as in [ra\(4\)](#).

For maximum speed, read and write in large blocks (preferably 63K) and avoid seeks. A seek across the whole disk takes about 1 second.

The device `/dev/worm?` is simply an appropriate raw [ra\(4\)](#) device, partition 7 (the whole disk).

## EXAMPLES

```
worm mkfs -c"512x512x24 movies" tdmoviesa
worm write tdmoviesa < filenames
worm read -d tdmoviesa bumblebee/act2/frame1
```

## FILES

```
/dev/worm?
/n/wild/usr/worm/vol_ids
/n/wild/usr/worm/jukedir
```

## SEE ALSO

[backup\(8\)](#), [scsish\(8\)](#), [backup\(1\)](#)

## BUGS

The output of `worm ls` is not necessarily sorted.



**NAME**

`xstr` – preprocessor for sharing strings in C programs

**SYNOPSIS**

```
xstr [-c] [-] [file]
```

**DESCRIPTION**

*Xstr* maintains a file `strings` into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

```
xstr -c name.c
```

will extract the strings from the named C source, replacing string references by expressions of the form `(xstr[number])` for some *number*. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file `x.c`. The strings from this file are placed in the `strings` data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file `xs.c` declaring the common *xstr* space can be created by running `xstr` with no arguments. This `xs.c` file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Without option `-c`, creates files `x.c` and `xs.c` as before, but does not use or affect any `strings` file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument `-` is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

**FILES**

```
strings      Data base of strings
x.c          Massaged C source
xs.c         C source for definition of array xstr
/tmp/xs*     Temp file
```

**BUGS**

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

**NAME**

intro – introduction to 5620-related software

**SYNOPSIS**

**PATH=stuff:/usr/jerq/bin**

**DESCRIPTION**

Section 9 of this manual lists software for running or supporting Teletype DMD-5620 terminals. Subsections 9.1-9.7 mirror the purposes of the preceding sections 1-7, with 9.1 being commands, 9.6 being games, etc. Command synopses assume that the shell search path includes

Few commands deal with a 5620 in native mode. [32ld\(9\)](#) loads programs into the terminal and [mux\(9\)](#) starts the characteristic ‘layer’ or window system. Almost all other commands in section 9 either run on Unix or within *mux* layers.

A layer is technically a virtual terminal, but is almost indistinguishable in software from a real terminal; in particular, the interface described in [ttyld\(4\)](#) applies to layers, except for the additional editing capabilities discussed in [mux\(9\)](#)

The commands in sections 9.1 and 9.6 run on Unix, but most implicitly call *32ld* to down-load a program that replaces the default terminal process running in the layer. To Unix the interface is still that of a terminal; in particular */dev/tty* (see [fd\(4\)](#)) is always connected to the layer. The default *mux* terminal program implements the teletype driver function itself. When a program is down-loaded, there is no teletype driver; programs that desire one must push the teletype line discipline on the stream, and arrange to pop the line discipline on exit; see [stream\(4\)](#) and [ttyld\(4\)](#). Some commands may simply emulate other terminals by down-loading a terminal program (see [term\(9\)](#)) others, such as the text editor [sam\(9\)](#) are really two programs — one on Unix and one in the layer — communicating using standard input/output on Unix and [sendchar\(\)/rcvchar\(\)](#) in the terminal; see [request\(9\)](#)

There is an identity between bitmaps and layers in the graphics software. Graphic objects are bitmaps. The [newlayer\(9\)](#) primitives that operate on layers are aliased to bitmap primitives of (9.3), and the data structures are isomorphic. When running under *mux*, a programmer need not consider layers as graphical objects at all; the operating system checks the arguments to the graphics primitives and dispatches the appropriate operator depending on the type of the argument. Except in stand-alone software, layers are an invisible implementation detail.

**Teletype 630**

*Mux* and various programs that run in *mux* layers have been ported to Teletype DMD-630 terminals. The ported software is not available on all machines and is not specifically documented. Look in */usr/630/bin* to see what's there.

**FILES**

*/usr/jerq/bin*

jerq-related Unix object programs

*/usr/jerq/sbin*

terminal programs, usually down-loaded automatically by programs in */usr/jerq/bin*

*/usr/630/bin*

630-related Unix object programs

*/usr/630/lib*

terminal programs

**SEE ALSO**

[32ld\(9\)](#) [mux\(9\)](#) [stream\(4\)](#)

**BUGS**

There are two mechanical-drawing programs, *cip* and *ped*, two ‘artistic’ drawing programs, *paint* and *twid*, one ‘graphic’ drawing program, *brush*, two pixel-level drawing programs, *icon* and *jf*, all for working on binary images. None dominates.

**NAME**

32ld – bootstrap loader for 5620

**SYNOPSIS**

**32ld** [ *option ...* ] *file* [ *argument ...* ]

**DESCRIPTION**

*32ld* loads the MAC-32 object *file* for execution in a 5620 connected to the standard output. When loading into a [mux\(9\)](#) layer, the *arguments* are passed to the program as in Unix. The options are:

- d** Print on the standard error file the sizes of the text, data and bss segments of *file*. The standard error must be separated from the standard output to avoid corrupting the down-load.
- p** Print down-loading protocol statistics on the diagnostic output (for stand-alone loading only).
- z** Load the process but don't run it. It may be started using *3pi*; see [pi\(9\)](#) This option works only under *mux*.

The environment variable **JPATH** is the analog of the shell's **PATH** variable to define a set of directories in which to search for *file*.

**SEE ALSO**

[jx\(9\)](#) [mux\(9\)](#)

**NAME**

3cc, 3as, 3ar, 3ld, 3nm, 3size, 3strip, cprs – MAC-32 C compiler

**SYNOPSIS**

**3cc** [ *option ...* ] *file ...*  
**cprs** *infile outfile*

**DESCRIPTION**

3cc is the C compiler for the MAC-32 microprocessor in the Teletype DMD-5620 terminal. Its default action is to compile programs to run under the *mux(9)* environment.

The behavior of 3cc is similar to *cc(1)*. Here are listed only options with special behavior for 5620s.

- J** Compile the named programs, and link them for running stand-alone on a 5620 terminal.
- O** Invoke an object-code improver (not recommended).
- m** Compile the named programs for ordinary (non-jerq) environments.
- Dname=def**
- Dname** Define the *name* to the preprocessor, as if by #define. If no definition is given, the name is defined as 1. The symbol **MUX** is predefined unless **-J** or **-m** is set.
- Idir** #include files whose names do not begin with / are always sought first in the directory of the *file* argument, then in directories named in **-I** options, then in directories on a standard list, which includes

Associated object-code manipulating programs exist. Their behavior is similar to the programs cited below. The loader, assembler and archive program are System V derivatives, and are slightly different in behavior; see the System V manuals. For typical uses, these differences are irrelevant. The support programs include:

- 3as* assembler, see *as(1)*
- 3ar* archive, see *ar(1)* (there is no *3ranlib*)
- 3ld* link editor, see *ld(1)*
- 3nm* name list, see *nm(1)*, doesn't work on archives
- 3size* object code size, see *size(1)*
- 3strip* symbol table; see *strip(1)*. (**-r** is mandatory for *mux*-runnable binaries.)

3strip has no **-g** flag; but *cprs* removes redundant symbol table entries while copying *infile* to *outfile*.

**FILES**

- a.out loaded output
- /tmp/ctm\* temporary
- /lib/cpp preprocessor
- /usr/jerq/lib/m32/comp compiler
- /usr/jerq/lib/m32/optim optimizer
- /usr/jerq/lib/\*.o runtime startoff, etc.
- /usr/jerq/lib/libc.a standard library
- /usr/jerq/lib/libj.a stand-alone graphics library /usr/jerq/lib/libmj.a mux-runnable graphics library (default)

/usr/jerq/lib/muxmap  
loader I-file

/usr/jerq/include  
standard directory for #include files

**SEE ALSO**

System V manuals for *3ar*, *3ld*, *3as* and *cprs* documentation.

**NAME**

add, sub, mul, div, eqpt, eqrect, inset, muldiv, ptinrect, raddp, rsubp, rectXrect, rectclip – arithmetic on points and rectangles

**SYNOPSIS**

```
#include <jerq.h>

Point add(p, q) Point p, q;
Point sub(p, q) Point p, q;
Point mul(p, a) Point p; int a;
Point div(p, a) Point p; int a;
int eqpt(p, q) Point p, q;
int eqrect(r, s) Rectangle r, s;
Rectangle inset(r, n) Rectangle r; int n;
int muldiv(a, b, c) int a, b, c;
int ptinrect(p, r) Point p; Rectangle r;
Rectangle raddp(r, p) Rectangle r; Point p;
Rectangle rsubp(r, p) Rectangle r; Point p;
int rectXrect(r, s) Rectangle r, s;
int rectclip(rp, s) Rectangle *rp, s;
```

**DESCRIPTION**

*Add* returns the Point sum of its arguments: **Pt**( $p.x+q.x$ ,  $p.y+q.y$ ). *Sub* returns the Point difference of its arguments: **Pt**( $p.x-q.x$ ,  $p.y-q.y$ ). *Mul* returns the Point **Pt**( $p.x*a$ ,  $p.y*a$ ). *Div* returns the Point **Pt**( $p.x/a$ ,  $p.y/a$ ).

*Eqpt* and *eqrect* compare their arguments and return 0 if unequal, 1 if equal.

*Inset* returns the Rectangle **Rect**( $r.origin.x+n$ ,  $r.origin.y+n$ ,  $r.corner.x-n$ ,  $r.corner.y-n$ ). The following code creates a clear rectangle **r** with a 2-pixel wide border inside **r**:

```
rectf(.display, r, F_OR);
rectf(.display, inset(r, 2), F_CLR);
```

*Muldiv* is a macro that returns the 16-bit result  $(a*b)/c$ , with  $(a*b)$  calculated to 32 bits, so no precision is lost.

*Ptinrect* returns 1 if *p* is a point within *r*, and 0 otherwise.

*Raddp* returns the Rectangle **Rect**(**add**( $r.origin$ , *p*), **add**( $r.corner$ , *p*)); *rsubp* returns the Rectangle **Rect**(**sub**( $r.origin$ , *p*), **sub**( $r.corner$ , *p*)).

*RectXrect* returns 1 if *r* and *s* share any point; 0 otherwise.

*Rectclip* clips in place the Rectangle pointed to by *rp* so that it is completely contained within *s*. The return value is 1 if any part of *\*rp* is within *s*. Otherwise, the return value is 0 and *\*rp* is unchanged.

**SEE ALSO**

[types\(9\)](#)

**NAME**

alloc, free, balloc, bfree, gcalloc, gcfree – allocate memory

**SYNOPSIS**

```
#include <jerq.h>
```

```
char *alloc(nbytes) unsigned nbytes;
```

```
void free(s) char *s;
```

```
Bitmap *balloc(r) Rectangle r;
```

```
void bfree(b) Bitmap *b;
```

```
char *gcalloc(nbytes, where) unsigned long nbytes; char **where;
```

```
void gcfree(s) char *s;
```

**DESCRIPTION**

*Alloc* corresponds to the standard C function *calloc*; see [malloc\(3\)](#). It returns a pointer to a block of *nbytes* contiguous bytes of storage, or 0 if unavailable. The storage is aligned on 4-byte boundaries and is cleared to zeros. *Free* frees storage allocated by *alloc*.

*Balloc* returns a pointer to a Bitmap large enough to contain the Rectangle *r*, or 0 for failure. The coordinate system inside the Bitmap is set by *r*: the **origin** and **corner** of the Bitmap are those of *r*. *Bfree* frees the storage associated with a Bitmap allocated by *balloc*.

*Gcalloc* provides a simple garbage-compacting allocator. It returns a pointer to a block of *nbytes* contiguous bytes of storage, or if unavailable. The storage is initialized to zeros. *Where* is a pointer to the user's data where the location of the block is to be saved. The return value of *gcalloc* is stored in *\*where* and will be updated after each compaction. Therefore, a program using *gcalloc* should never store the location of memory obtained from *gcalloc* anywhere other than *where*. Typically, this location is contained in a structure, such as a Bitmap (*balloc* uses *gcalloc*). *Gcfree* frees the storage block at *p*.

**SEE ALSO**

[types\(9\)](#) [malloc\(3\)](#)

**NAME**

Code, addr, bitblt, point, rectf, screenswap, segment, texture – graphics functions

**SYNOPSIS**

```
#include <jerq.h>

typedef int Code;
Code F_STORE, F_XOR, F_OR, F_CLR;

Word *addr(b, p) Bitmap *b; Point p;

void bitblt(sb, r, db, p, f) Bitmap *sb, *db; Rectangle r; Point p; Code f;

void point(b, p, f) Bitmap *b; Point p; Code f;

void rectf(b, r, f) Bitmap *b; Rectangle r; Code f;

void screenswap(b, r, s) Bitmap *b; Rectangle r, s;

void segment(b, p, q, f) Bitmap *b; Point p, q; Code f;

void texture(b, r, t, f) Bitmap *b; Rectangle r; Texture *t; Code f;
```

**DESCRIPTION**

The type **Code** tells the graphics primitives what operation perform. The possible values are:

```
F_STORE  target = source
F_OR     target |= source
F_XOR   target ^= source
F_CLR   target .= source
```

In other words, if a Rectangle is copied to another place with Code **F\_OR**, the result will be the bitwise OR of the contents of the source Rectangle and the target area. For operations with no explicit source, such as line drawing, the source is taken to be an infinite bitmap with zeros everywhere except on the object (e.g. line) generated by the operator, with coordinates aligned with the destination bitmap. **F\_STORE** is the same as **F\_OR** for non-rectangular operations.

*Addr* returns the address of the Word containing the bit at Point *p* in the Bitmap *b*.

*Bitblt* (bit-block transfer) copies the data in Rectangle *r* in Bitmap *sb* to the congruent Rectangle with *origin* *p* in Bitmap *db*. The nature of the copy is specified by the Code *f*.

*Point* draws the pixel at location *p* in the Bitmap *b* according to Code *f*.

*Screenswap* does an in-place exchange of the on-screen Rectangle *s* and the Rectangle *r* within the Bitmap *b*. Its action is undefined if *r* and *s* are not congruent. The Rectangle *s* is not clipped to the Bitmap *b*, only to the screen.

*Segment* draws a line segment in Bitmap *b* from Point *p* to *q*, with Code *f*. The segment is half-open: *p* is the first point of the segment and *q* is the first point beyond the segment, so adjacent segments sharing endpoints abut. Like all the other graphics operations, *segment* clips the line so that only the portion of the line intersecting the bitmap is displayed.

*Texture* draws, with function *f* in the Rectangle *r* in Bitmap *b*, the Texture specified by *t*. The texture is replicated to cover *r*. *Rectf* is equivalent to *texture* with *\*t* set to all one's.

In the above definitions, the type Bitmap may be replaced with Layer anywhere; see [newlayer\(9\)](#)

**SEE ALSO**

[types\(9\)](#)



**NAME**

bitfile – format of bitmap file

**DESCRIPTION**

Binary files produced by [blitblt\(9\)](#) and other bitmap-generating programs are formatted as follows:

Byte no.	Description
0, 1:	Zero.
2, 3:	<i>x</i> -coordinate of the rectangle origin (low-order byte, high-order byte).
4, 5:	<i>Y</i> -coordinate of the rectangle origin (low-order byte, high-order byte).
6, 7:	<i>x</i> -coordinate of the rectangle corner (low-order byte, high-order byte).
8, 9:	<i>Y</i> -coordinate of the rectangle corner (low-order byte, high-order byte).
remainder:	Compressed raster data. Each raster is exclusive-or'd with the previous one, and zero-extended (if necessary) to a 16-bit boundary. It is then encoded into byte sequences, each of which consists of a control byte followed by two or more data bytes:
Control	Data
$n$ (< 127)	$2 \times n$ bytes of raster data, running from left to right.
<b>0x80</b> + $n$	2 bytes of raster data, to be replicated from left to right $n$ times.

There are also two ASCII formats in current use. Textures and 16×16 icons, as created by [icon\(9\)](#) are encoded as a **Texture** declaration with initializer, to be copied unchanged into C program source; see [types\(9\)](#) Faces and other large icons are without any surrounding C syntax. In either case, each scan line of the bitmap is a comma-separated list of C-style short hexadecimal constants; scan lines are separated by newlines.

**SEE ALSO**

[blitblt\(9\)](#) [icon\(9\)](#) [types\(9\)](#) [vismon\(9\)](#)

**NAME**

blitblt, menudrop – save or print a screen image

**SYNOPSIS**

**blitblt** [ **-p** *command* ]

**menudrop**

**DESCRIPTION**

*Blitblt* copies a selected area of a *mux*(9) screen into a file or to a program. It is menu-driven off button 3 to select a rectangular area and to treat it by flipping the border from wide to narrow and back, inverting video, saving the selected area in a file, or sending it to a program, usually for printing. Details of certain menu items:

**choose layer**

**layer rectangle**

One gets the bits of a layer, obscured or not; the other gets screen bits including superposed layers.

**run/halt**

Restart or stop the terminal process in the selected layer.

**write file**

Write the selected area into a file or pipe in *bitfile*(9) format. The filename is typed at the bottom of the *blitblt* layer. A bare newline repeats the previous name. If the first character is |, the remainder of the line is taken as a shell command to pipe into. (A likely command is |lp for hard copy.)

| *command*

Pipe the selected area to the *command* specified by the **-p** option.

*Menudrop* may be used with *blitblt* to make images containing ‘menus’ as fraudulent overlaid layers. The program is menu-driven off button 3:

**drop menu**

A non-*mux* menu selected in another window will be drawn and will remain on screen after the button selecting the menu has been released. Subsequent menu selections will delete the previous menu layer and create a new one. Once such a menu-bearing layer is present, the *menudrop* menu changes to allow cursor placement, highlighting of menu items, lifting of the displayed menu, etc. The functionality of the program using the menu is not affected.

**mux menus**

The next click of button 1 or 2 will drop the corresponding (non-functional) *mux* menu at the mouse position.

**exit**

*Menudrop* will exit in a clean manner.

**EXAMPLES**

blitblt

blitblt -p "lp -p bpost" Arrange for piping output to a laser printer: a good way, and a surefire way.

**SEE ALSO**

*mbits*(6), *bitfile*(9)

**BUGS**

Animated layers result in broken images. Use the **halt** function.

If a pipe request fails, the *blitblt* layer becomes unusable.

The default *command* for *write file* is obsolete.

Deleting a *menudrop* layer, rather than exiting through the menu, can crash the terminal.

Programs that use private menu packages are unaffected by *menudrop*; using a debugger to stop a program in midmenu may get the same effect.

**NAME**

blitmap – road maps and path finding

**SYNOPSIS**

**blitmap** [ *option ...* ]

**DESCRIPTION**

*Blitmap* displays road maps. It relies on the mouse to select regions, functions, and to give formats for typed commands. The metropolitan N.Y.-N.J. area is the default map. *Blitmap's* screen consists of two frames, a large frame for plotting maps and printing messages to the user, and a one-line command frame at the bottom. *Blitmap* recognizes two commands from the keyboard, to designate a region and to scale or plot a route from one point to another. The commands, which may be typed at any time, follow. Here *option* is as in the command line.

[ *option ...* ] **radius** *address* [ , *town or zip* ] Plot an area with the given radius in miles around the *address*.

**path** [ *option ...* ] **from** *address* **to** *address* [ , *town or zip* ] Trace a route on a map and print traveling directions from point to point.

*Address* may be a number and street or an intersection such as, main or 600 Mountain av, new providence.

Button 3 Menu

**Regions**

Select which region to plot. Available regions are San Francisco, New York City and North Jersey, Washington, Los Angeles and Ann Arbor.

**Zoom-in**

Using button 3 and the box icon, enclose the area desired and *blitmap* will plot a map of that area centered at the center of the drawn rectangle.

**Zoom-out**

Enclose an area with a rectangle and the map shown will be reduced to the rectangle size and the rest of the map filled in. The center will be at the center of the drawn rectangle.

**Center**

With button 3 point to new center. The radius will remain the same.

**Prev. map**

*Blitmap* plots the previous frame.

**To draw map****To find path**

Tell about the keyboard commands

**Quit** Confirm with button 3.

Button 2 controls map editing functions. No editing is actually done, but by using the **-f** option, a file of changes will be written, which may be added to the actual database.

The options specify the algorithm of the path search and plotting choices:

- 2** Two ended search (default).
- 1** One way search.
- b** Breadth search.
- C** Cyclists – ignore costs for turns.
- F** Stop at first route connect with breadth search.
- H** Hierarchical search. (Give priority to major roads.)
- G** In breadth search, ignore ones whose cost + dist > 4/3 total airline distance.
- J** Use precomputed routes. (Available from 600 mountain av, New Providence.)
- V** Verbose directions (all intersections given).
- W** Walkers – no cost for turns and ignore one-way streets.

- A** Print every possible label.
- B** Print business names.
- Mlx** Forces a detailed street plot for maps whose radius is greater than 10,000 ft.
- b** Don't print boundaries.
- ix** Plot only streets with importance >x; x=0 is default.
- j** Do sketch map only.
- l** Don't print labels.
- r** Don't print railroads.
- s** Don't print streets.
- w** Don't print waterways.

## FILES

/n/seki/usr/rje/BLIT/term/term  
terminal support program

/n/seki/m?/map/\*  
map files

## BUGS

Since the data bases have not been checked and many streets are not connected, some paths may be circuitous. There are no connecting roads from N.J into N.Y or from Middlesex county into Union. The routing programs will churn, trying to find a through street and will not give up.

There are no one-way tags on the streets.

*Blitmap* does not know if it has been reshaped.

**NAME**

brush – painting program

**SYNOPSIS**

**brush** [ **-f** *fontdir* ] [ **-p** *picdir* ] [ **-t** *texdir* ]

**DESCRIPTION**

*Brush* paints images under mouse control. Options are

- f** font directory by default)
- p** the directory in which to keep pictures (current directory by default)
- t** texture directory, where brushes and shades live (current directory by default)

In general, button 1 draws, button 2 erases; the cursor assumes the shape of the current brush. Button 3 is used to select options, sweep out areas, or cancel operations in progress.

The borders on either side of the drawing area contain menus of available shades and brushes. The current brush and shade are outlined by boxes. To choose another, click button 3 at it.

The top border contains a help area, drawing options, and certain commands. Selections are made by pointing with button 3. Some cycle through options; others bring up menus. The items are:

**help** Icons in three boxes indicate what buttons 1, 2, and 3 will do at any given time:

- |                   |                               |
|-------------------|-------------------------------|
| paintbrush        | draw with this button         |
| pencil eraser     | erase with this button        |
| menu with cursor  | menu on this button           |
| thumbs down       | cancel or finish an operation |
| pointing hand     | indicate a point              |
| square with arrow | sweep a rectangle             |
| circle with arrow | sweep a circle                |
| skull             | exit the program              |

**smooth**

Smooth the contours of magnified images.

**align** Force circles, discs, text, and other images to align with texture cell boundaries.

**image** Manipulate the ‘current image’, (box, ellipse, etc.) selected from the drawing menu described below. Button 3 makes the image disappear, reserved for future use. The image menu contains:

**same** Bring back the current image.

**magnify**

Sweep a rectangle indicating the size of the magnified image. The numbers that appear are horizontal and vertical magnification factors.

**shrink**

Shrink to 1/4 size. Indicate whether image is shaded or black . white.

**flip** Reflect left-right or top-bottom.

**rotate** Rotate counterclockwise or clockwise 90 degrees.

**slant** Drag the current image rectangle into a parallelogram.

**outline**

Replace the current image with its outline.

**shadow**

Draw a ‘shadow’ behind the current image.

**shadow**

Draw a ‘shadow’ behind the current image.

**new** Make a new image by copying a rectangular portion of the screen.

To move an image on the screen, select **new** from the **image** menu. Sweep the area to be moved, click button 2 to erase it, move it, and click button 1 to draw it.

- drawing style**  
Select continuous curves, dotted lines, disconnected dots, or an 'airbrush' effect when painting.
- constrain**  
Select freehand (wavy-line icon) or horizontal-vertical drawing (angular icon).
- reflect** Draw symmetric figures. The icon shows the symmetries: **x=0** (left-right), **y=0** (top-bottom), or **both**, relative to the center of the screen.
- draw mode**  
Set the drawing mode to one of **or**, **xor**, **store**, **and**, **copy** (preserves interior whitespace of images).
- text style**  
Set the text style to one of **normal**, **outline**, **bold**, **shadow**, **italic**.
- font name**  
Set the text font. Menu selection **new** prompts for a font name from the font directory.
- i/o** Interact with host machine. Menu items are:
- save** Copy screen, brushes, or shades to a file. Prompts for a file name, starting with the default picture directory (if any). Backspace past this if you wish to save elsewhere; hit return to quit. Next sweep a rectangle to be saved. Bitmaps are saved in *bitfile(9)* format.
  - recall** Prompts for a file name. The recalled picture becomes the current image.
  - exit** Leave the program. Confirm by two clicks on button 3.

The menu on button 3 in the drawing area contains these selections:

- lines** Indicate first point, then position cursor with rubber band line for subsequent lines. Button 1 draws, button 2 erases.
- curves**  
Indicate first control point, then position cursor with rubber band line for subsequent control points. A curve (spline) will be drawn (erased) using these control points, depending on whether the last button hit is button 1 (draw), or button 2 (erase).
- box**
- ellipse**
- disc** (A disc is a filled ellipse). Sweep a rectangle; numbers show the dimensions. A single dot marks the center of an ellipse. The image becomes the current image; use buttons 1 and 2 to draw or erase with it.
- string** Type in text. The string becomes the current image.
- texture**  
Sweep a rectangle. The current image becomes a rectangle of this size textured with current shade.
- fill** Sweep a rectangle, then indicate interior seed points using button 1, or button 3 to quit. Enclosed regions will be filled with current shade. Any button cancels the fill.
- clear**
- invert** Sweep a rectangle to be cleared or color-inverted.
- fade** Sweep a rectangle. Holding button 2 will fade this area, as if erasing in spray paint mode with a random pattern instead of a shade.
- new brush**
- new shade**  
Menu select whether to edit or snarf from screen (with button 3). If editing, the current brush (shade) will appear magnified in upper left corner. Edit with buttons 1 and 2, quit with 3. Several 'spare' brushes appear at the bottom of the brush menu.
- details**  
Select an area with box cursor to be magnified for detailed editing.

## SEE ALSO

*paint(9)* *mbits(6)*, *bitfile(9)*

**BUGS**

The smoothing operation fully smooths only for magnification factors that are powers of two.

Bitmaps moved off the top or bottom of the physical screen can pick up noise.

**Copy** mode generates a mask the first time a given image is moved. This can take a while for large images. Be patient.

**NAME**

button123, mouse, cursallow, cursinhibit, cursset, cursswitch, getrect123 – mouse control

**SYNOPSIS**

```
#include <jerq.h>

extern struct Mouse {
    Point xy;
    short buttons;
} mouse;

int button(n) int n;
int button1(), button2(), button3();
int button12(), button23(), button123();

void cursinhibit();
void cursallow();

void cursset(p); Point p;

Texture *cursswitch(t); Texture *t;

Rectangle getrect(n) int n;
Rectangle getrect1(), getrect2(), getrect3();
Rectangle getrect12(), getrect23(), getrect123();
```

**DESCRIPTION**

When the mouse is requested (see [request\(9\)](#)) the mouse state is updated asynchronously in the structure **mouse**. The coordinates of the mouse are held in **mouse.xy**, and the state of the buttons in **mouse.buttons**. Each process's **mouse** structure is independent of the others, so that (except for *cursset*) actions such as changing the tracking cursor do not affect the mouse in other processes.

The macro *button* and its counterparts return the state of the associated mouse button: non-zero if the button is depressed, 0 otherwise. The buttons are numbered 1 to 3 from left to right. *Button12* and the other multi-button functions return the OR of their states: true if either button 1 or button 2 is depressed.

*Cursinhibit* turns off interrupt-time cursor tracking (the drawing of the cursor on the screen), although the mouse coordinates are still kept current and available. *Cursallow* enables interrupt-time cursor tracking. *Cursallow* and *cursinhibit* stack: to enable cursor tracking after two calls to *cursinhibit*, two calls to *cursallow* are required.

*Cursset* moves the mouse cursor to the Point *p*.

*Cursswitch* changes the mouse cursor (a 16×16 pixel image) to that specified by the Texture *t*. If the argument is **(Texture\*)0**, the cursor is restored to the default arrow. *Cursswitch* returns the previous value of the cursor: the argument of the previous call to *cursswitch*.

*Getrect* prompts the user with a box cursor and waits for a rectangle to be swept out with the named button, identified as with the *button* primitives. It returns the screen coordinates of the box swept. The box may be partly or wholly outside the process's layer.



**NAME**

cip – draw pictures for typesetting

**SYNOPSIS**

**cip**

**DESCRIPTION**

*Cip* prepares or modifies [pic\(1\)](#) descriptions, which may subsequently be typeset. It provides a palette of shapes: box, circle, ellipse, line, arc, spline, and text. Button 1 selects shapes from the palette or the screen. Button 2 places or redraws move) shapes. Button 3 controls menus.

File names and text strings are entered from the keyboard. Keyboard input always ends with a newline. A current file name is remembered and offered for file operations; backspace over it to substitute a new name, or type newline to accept it.

The `define` menu item allows a box to be swept, collecting all contained shapes into a group. Groups are selected as whole. When a group is selected, a special menu appears. Item `separate` dissolves the group; `reflect` reflects about a horizontal midline; after `copy` button 2 places copies at the cursor. Item `edit` confines activity to the group. Changes are reflected in all copies of the group. To leave the group, click button 1 at `edit depth`.

**SEE ALSO**

[pic\(1\)](#), [ped\(9\)](#)

Sally A. Browning, 'Cip User's Manual: One Picture is Worth a Thousand Words', this manual, Volume 2

**BUGS**

*Cip* cannot handle arbitrary *pic* programs, just programs in the style that it produces.

**NAME**

circle, disc, arc, ellipse, eldisc, elarc – circle-drawing functions

**SYNOPSIS**

```
#include <jerq.h>
```

```
void circle(bp, p, r, f) Bitmap *bp; Point p; int r; Code f;
```

```
void disc(bp, p, r, f) Bitmap *bp; Point p; int r; Code f;
```

```
void arc(bp, p0, p1, p2, f) Bitmap *bp; Point p0, p1, p2; Code f;
```

```
void ellipse(bp, p, a, b, f) Bitmap *bp; Point p; int a, b; Code f;
```

```
void eldisc(bp, p, a, b, f) Bitmap *bp; Point p; int a, b; Code f;
```

```
void elarc(bp, p0, a, b, p1, p2, f) Bitmap *bp; Point p0, p1, p2; int a, b; Code f;
```

**DESCRIPTION**

*Circle* draws the best approximate circle of radius  $r$  centered at Point  $p$  in the Bitmap  $bp$  with Code  $f$ . The circle is guaranteed to be symmetrical about the horizontal, vertical and diagonal axes. *Disc* draws the corresponding disc.

*Arc* draws a circular arc centered on  $p0$ , traveling counter-clockwise from  $p1$  to the point on the circle closest to  $p2$ .

*Ellipse* draws an ellipse centered at  $p$  with horizontal semi-axis  $a$  and vertical semi-axis  $b$  in Bitmap  $bp$  with Code  $f$ . *Eldisc* draws the corresponding elliptical disc. *Elarc* draws the corresponding elliptical arc, traveling counter-clockwise from the ellipse point closest to  $p1$  to the point closest to  $p2$ . (Beware the regrettable difference between the calling conventions for *arc* and *elarc*.)

**BUGS**

When an endpoint of an arc lies near a tail of an ellipse so thin that its ends degenerate into straight lines, *elarc* does not try to distinguish which side of the tail the point belongs on.

**NAME**

cos, sin, atan2, sqrt, norm – integer math functions

**SYNOPSIS**

**int cos(d) int d;**

**int sin(d) int d;**

**int atan2(x, y) int x, y;**

**int norm(x, y, z) int x, y, z;**

**int sqrt(x) long x;**

**DESCRIPTION**

*Cos* and *sin* return scaled integer approximations to the trigonometric functions. The argument values are in degrees. The return values are scaled so that **cos(0)==1024**. Thus, to calculate the mathematical expression  $x = a \cos(d)$ , the multiplication must be scaled:

$$x = \text{muldiv}(x0, \cos(d), 1024)$$

*Atan2* returns the approximate arc-tangent of  $y/x$ . The return value is in integral degrees.

*Sqrt* returns the 16-bit signed integer closest to the square root of its 32-bit signed argument.

*Norm* returns the Euclidean length of the three-vector  $(x, y, z)$ .

**DIAGNOSTICS**

*Sqrt* returns 0 for negative arguments; and **atan2(0,0)==0**. *Norm* does not protect against overflow.

**BUGS**

*Atan2* may be off by as much as two degrees.

**NAME**

crabs – graphical marine adventure game

**SYNOPSIS**

**crabs** [ **-i** ] [ **-s** *duration* ] [ **-v** *velocity* ] [ *number* ]

**DESCRIPTION**

In *crabs*, difficult situations are encountered in trying to kill or capture crustaceans swarming in a murky sea. You will have to work very rapidly to keep your territory free of seabed intruders. At first, you may even find it hard to keep a clear view of your surroundings, but later discoveries about the spirit of the game will suggest a solution.

There are several options.

- i** causes the intruders to play intelligently, allowing them to avoid detection.
- s** simplifies the game for the first *duration* time intervals. Default is 0. 5-10 is recommended for beginners, although you may want to forgo this option the first time, just to see how interesting it can get.
- v** adjusts the velocity of the crabs, 1 being fastest. Default is 5.

*Number* specifies the number of intruders. Default is 30.

**NAME**

demo, swar, pacman – graphic demonstrations and games

**SYNOPSIS**

**demo** [ *name* ]

**DESCRIPTION**

If a demo is named, *demo* runs it, otherwise *demo* produces a list of what's available.

Games that permit interaction are often controlled by the mouse; experiment to find out what it does. Some less obvious interactions are listed below.

*Swar* is a two-player game. One player uses the asdwx keys, the other 12350 keys on the keypad.

*Pacman* is controlled by the hjkl keys or the mouse.

**SEE ALSO**

[crabs\(9\)](#)

**BUGS**

Some of the programs don't play fair.

**NAME**

face, mugs – show faces, make face icons from pictures

**SYNOPSIS**

**face** *machine!user file ...*

**mugs** [ **-a** ]

**DESCRIPTION**

*Face* displays the 48×48 bit icons specified by its arguments. If an argument contains an exclamation mark, it is assumed to be a machine and user pair and is looked up in the face file system, *faced(9)* otherwise it is taken to be a file name. If the file does not exist and contains no slashes, it is looked up in

When *face*'s layer is full, it waits for a character to be typed before continuing.

*Mugs* interactively converts grey-scale images in the form of *picfile(5)* into 48×48 icons for display by *face* and *vismon(9)*. It prompts for the name of a picture file, displaying a large approximation to the original picture and a matrix of 48×48 icons of varying contrast and brightness. Button 1 selects one of the 48×48's. Button 3 presents a menu with entries:

**window**

Select a square window in the large picture using button 3. Touch down at the top and center of the square and slide around to adjust its size. Appropriately cropped 48×48's will be displayed.

**in** Zoom in on a smaller part of contrast-brightness space, displaying an array of 48×48's that look more-or-less like the selected one. Repeated **ins** will zoom in farther.

**out** Opposite of **in**.

**save** Type in the name of a file in which to save the currently selected 48×48.

**read** Type in the name of a picture file containing the next face to process.

**exit** Confirm with button 3.

Option **-a** indicates that picture files have non-square pixels with aspect ratio 1.25, as produced by the ITI frame-grabber attached to kwee. Normally pixels are assumed to be square.

**EXAMPLES**

face /n/face/coma/\*/48x48x1 All the users of coma.

**SEE ALSO**

*faced(9)* *icon(9)* *vismon(9)* *imscan(1)*, *picfile(5)*

**NAME**

faced – network face server

**SYNOPSIS**

**/usr/net/face.go**

**DESCRIPTION**

The network face server provides a database of 48×48 bit icons and other facial representations. It is implemented as a network file system similar to [netfs\(8\)](#).

The file system, conventionally mounted on **/n/face**, has a fixed three-level hierarchy. The first level is a machine name, the second level a user name, and the third level a resolution. Thus the file `/n/face/kwee/pjw/48x48x1` is the standard face icon (for user pjw) on machine kwee:

Many local users also have 512×512 byte high-resolution faces, named **512x512x8**. Other resolutions may also be present for a particular face. One-bit images are stored in the format used by [icon\(9\)](#) eight-bit images are arrays of bytes. The directories for machines sharing a user community, such as those on a Datakit node, are linked together and given a name appropriate to the community. For example, **/n/face/kwee** is a link to **/n/face/astro**.

To access the face for a mail name *machine!uid* take the result of the first successful open from the following list of files:

```
/n/face/machine/uid/48x48x1
/n/face/misc./uid/48x48x1
/n/face/machine/unknown/48x48x1
/n/face/misc./unknown/48x48x1
```

The directory **misc.** holds faces for generic users such as root and uucp. The face server is made available on a machine by running **/usr/net/face.go** from [rc\(8\)](#).

The face server data is administered by a pair of ASCII files that associate related machines and faces. The machine table `machine.tab` attaches machines to communities; in it the line

```
kwee=astro
```

puts machine kwee in community astro. The people table `people.tab` associates a machine/user pair in the face server with a file in one of the source directories `/n/kwee/usr/jerq/icon/face48` or `face512`. Thus

```
astro/pjw=pjweinberger
```

causes the images stored in source files named `pjweinberger` to be available in the face server in directory **/n/face/astro/pjw**. As well, each disk file used by the face server is linked (by its original name) into the directory **/n/face/48x48x1** or **/n/face/512x512x8** for easy access to all the images.

**FILES**

```
/n/kwee/usr/jerq/icon/face48
    directory of low resolution faces

/n/kwee/t0/face/512x512x8
    directory of high resolution faces

/n/kwee/usr/net/face/people.tab
    people/file equivalences

/n/kwee/usr/net/face/machine.tab
    machine/community equivalences
```

**SEE ALSO**

[netfs\(8\)](#), [face\(9\)](#) [icon\(9\)](#) [vismon\(9\)](#)

**BUGS**

After updating the tables, an indeterminate time may pass before the new faces are available. All face server files are unwritable.

**NAME**

movies – graphics movie file formats

**DESCRIPTION**

Movie files are generated by *preflicks* and used by *fflicks*; see [flicks\(9\)](#) The format of a movie files is:

```
struct Header {
    unsigned char version;
    short header_length;
    short nr_frames;
    unsigned char nr_tables;
    struct LOOKUP_TABLE {
        short number_of_entries;
        struct {
            short count;
            unsigned char value;
        } table[256];
    } Table[nr_tables];
};
struct Frame {
    short width, height;
    short compacted_length;
    unsigned char which_table;
    unsigned char data[compacted_length];
} Frame[nr_frames];
```

Each **short** in the above structure is present as a two-byte number in the file, most significant byte first. Each **unsigned char** is a single byte.

**version**

software version number, to ensure compatibility between producer and consumer of the file.

**header\_length**

total length in bytes of the lookup table(s) used to encode the file plus three bytes (the next three that follow).

**nr\_frames**

total number of movie frames in the file.

**nr\_tables**

number of lookup tables.

**nr\_entries**

number of entries in the lookup table (maximum 256).

**count value** pixel value and a count of how many times that value is to be repeated.

Immediately following the lookup tables begin the frames encoded in an indirect run-length code. Each frame is described by **width**, **height**, and the **compacted\_length** of the frame in bytes. The frame is coded in raster-scan order as a sequence of indexes into the table numbered **which\_table** (counting from 0).

**FILES**

\_movie

**SEE ALSO**

[flicks\(9\)](#) [pico\(1\)](#), [rebecca\(9\)](#)



**NAME**

flicks, fflicks, preflicks, 2mux – movie graphics for 5620

**SYNOPSIS**

**flicks** [ **-fmt** ] *file* ...

**preflicks** [ **-fmtvloics** ] *file* ...

**fflicks** [ *flickfile* ]

**DESCRIPTION**

*Flicks* interprets each of the *files* as a grey-scale frame in the form of *picfile(5)* (or a square raster of unsigned bytes), dithers them, and displays them on the terminal. Once the frames have been downloaded the frames can be played as a movie, controlled by a menu on button 3. Most menu selections are self-explanatory. *Step* shows individual frames, stepping forward with button 1, or backward with button 2. Button 3 brings back the main menu.

The size of a frame is an option:

- f** full size: same size as the input (typically 512×512)
- m** medium size: half the input size (typically 256×256) default
- t** tiny size: quarter of the input size (typically 128×128)

If only one image is processed, full size is the default. For more than 11 pictures, tiny size is default. Anything in between is medium size by default.

The frames are rendered with dithering by default, and with error propagation if **-e** is specified.

*Fflicks* downloads frames that have been preprocessed by *preflicks* into a *flickfile*. *Fflicks* downloads much faster than *flicks*.

The options for *preflicks* include **-f**, **-m**, and **-t** as for *flicks*, plus

- l** Use logarithmic dither.
- v** Chatter on standard error.
- o** Write the flickfile onto standard output; by default output goes into file **\_movie**.
- i** Print a summary of the contents of the flickfiles.
- c** Catenate named flickfiles onto the standard output.
- sX,Y** the (one only) input file is assumed to be a sequence of X×Y-byte frames. If X and Y may be omitted, 512×512 is assumed.

*Fflicks* display is controlled by a menu on button 3. The selection ‘movie rate’ tries to run the display at 24 frames/sec.

Frames prepared with *preflicks* are compacted. Thus *fflicks* can play a longer sequence than *flicks*: up to roughly 120 medium sized or 480 tiny frames (20 seconds of movie). Still longer sequences (about twice as long) can be downloaded if *fflicks* is run within *2mux* instead of *mux*. The price of compaction is speed. Menu selections are available for uncompacting some (even- or odd-numbered) frames.

**FILES**

**\_movie**  
**/usr/jerq/lib/2term**

**SEE ALSO**

*pic(1)*, *picfile(5)*, *rebecca(9)* *flickfile(9)* *movie(9)*

**NAME**

font – jerq font layouts

**SYNOPSIS**

```
#include <jerq.h>
#include <font.h>
```

```
typedef struct Fontchar Fontchar;
```

```
typedef struct Font Font;
```

```
extern Font defont;
```

**DESCRIPTION**

A *Font* is a character set, stored as a single Bitmap with the characters placed side-by-side. It is described by the following data structures.

```
typedef struct Fontchar {
    short x;                /* left edge of bits */
    unsigned char top;      /* first non-zero scan-line */
    unsigned char bottom;  /* last non-zero scan-line */
    char left;             /* offset of baseline */
    unsigned char width;   /* width of baseline */
} Fontchar;
typedef struct Font {
    short n;                /* number of chars in font */
    char height;           /* height of bitmap */
    char ascent;           /* top of bitmap to baseline */
    long unused;
    Bitmap *bits;          /* where the characters are */
    Fontchar info[n+1];    /* n+1 character descriptors */
} Font;
```

Characters in `bits` abut exactly, so the displayed width of the character `c` is `Font.info[c+1].x - Font.info[c].x`. The first left columns of pixels in a character overlap the previous character. The upper left corner of the nonempty columns appears at `(x,0)` in the bitmap. Width is the distance to move horizontally after drawing a character. The font bitmap has a fixed height; parameters `top` and `bottom` may speed up the copying of a character.

Characters are positioned by their upper left corners.

Fonts are stored on disk in binary with byte order that of the terminal. First in the file is the Font structure with `bits` elided. The data for the bitmap follows. The header for the bitmap must be inferred from `Font.height` and `Font.info[Font.n].x`.

**EXAMPLES**

```
Fontchar *i = f->info + c;
bitblt(f->bits, Rect(i->x, i->top, (i+1)->x, i->bottom),
        .display, Pt(p.x+i->left, p.y+i->top), fc);
p.x += i->width;
    Copy character c from font f to point p with Code F_XOR or F_OR.
```

For Code `F_STORE`, use `Rect(i->x, 0, (i+1)->x, f->height)`.

**SEE ALSO**

[jf\(9\)](#) [string\(9\)](#) [getfont\(9\)](#)

**BUGS**

The unused field is used, by [getfont\(9\)](#)

**NAME**

gebaca, gebam – get back at corporate america

**SYNOPSIS**

**/usr/games/gebaca**

**demo gebam**

**DESCRIPTION**

*Gebaca* is an arcade-type shoot-em-down with familiar characters. It runs on Teletype 5620 terminals in native mode only. Use the mouse to dodge and shoot.

*Gebam* is a cheap ripoff that runs under [mux\(9\)](#)

**NAME**

getfont – replace terminal's default font

**SYNOPSIS**

**getfont** [ *option ...* ] [ *font* ]

**DESCRIPTION**

*Getfont* reads font data from file *font*. The current layer and subsequently created layers use this font as *defont*; see [string\(9\)](#) If *font* does not directly name a file, it is looked for in directory

The options are:

- m** The font change applies to the basic [mux\(9\)](#) menu as well as to layers.
- l** The font change applies to the current layer only.
- d** Print debugging information about fonts before and after.

*Getfont* discards inaccessible fonts. To reclaim store without loading a font, call it with no *font* argument.

**EXAMPLES**

getfont pelm.10 Larger type for demos and eyesight problems.

getfont defont Restore the original font.

**FILES**

/usr/jerq/font

**SEE ALSO**

[string\(9\)](#) [font\(9\)](#) [font\(6\)](#)

**NAME**

graphdraw graphic – edit (combinatoric) graphs, convert to pic files

**SYNOPSIS**

**graphdraw** [*file* ]

**graphic** [ *option ...* ] *file*

**DESCRIPTION**

*Graphdraw* interactively edits and displays undirected graphs, and can also be used to display real-time animation of algorithms. If a *file* is mentioned, the graph stored in that file is edited.

Click button 1 in command line (at bottom of window) to type in commands:

**r** *file*            Read file and display graph.

**w** *file*            Write current graph to file.

**cd** *directory*    Change directory.

**!** *program file*

Execute animation *program* with *file* as input.

**q**                Quit.

Button 3 gets a menu of actions, which are usually accomplished by pointing with button 1. The parenthesized equivalents in the following list are explained under ‘Algorithm animation’.

**create vertex**

Vertex is placed where button 1 is clicked. (**vc** *x y*)

**delete**

Delete selected vertex and associated edges. (**vd** *i*)

**move**

Selected vertex moves with mouse until button 1 is released. (**vm** *i*)

**copy**

Copy of selected vertex and associated edges moves with mouse. (**vc** *i x y*)

**create/delete edge**

Point to first endpoint and click button 1. Point to second endpoint and click button 1. Continue selecting second endpoints with button 1. To unselect first endpoint, click button 2. (**ec** *ij* / **ed** *ij*)

**restart**

Click button 1 to clear screen and discard current graph. (**pr**)

**standard window**

Restart and reshape window to standard size, in which the drawing area is square and as large as possible.

**small/large/no grid**

Impose/remove visible grid to which all new coordinates will be rounded.

**exit**

Click button 1 to confirm.

**labels menu****label vertex**

Select vertex with button 1. Current label appears on command line. To accept it, click button 1.

Otherwise, type in new label and hit return. (**vl** *i w*)

**number vertices**

Vertex labels are set to the consecutive integers 1,2,...; this is the default. (**vn**)

**label edge**

Default is 1. (**el** *ij w*)

**show/hide vertex labels** (**vs**, **vh**)**show/hide edge labels** (**es**, **eh**)**turn Euclidean edge labels on/off**

Distances are measured in pixels. (**ee**)

**show/hide sum of edges****graphics menu****light/heavy/empty/full/invisible vertex**

Select style from menu with button 3; select vertices to change with button 1. The default is light. (**vg** *i c*)

**light/heavy edge (eg  $i j c$ )****macros menu**

Arrange for sets of vertices to act together. Actions on any vertex in the set apply to the whole set. Copying duplicates edges internal to the set. Creating an edge between vertexes in two different sets creates edges from every vertex in one set to every vertex in the other (bipartite subgraph).

**select set**

Sweep a rectangle around the set with button 1. Dissociate conflicting sets.

**unselect set**

Dissociate set containing selected vertex.

**shrink/expand set**

Selected set is shrunk/expanded about its center.

**reshape set**

Selected set is redrawn in swept rectangle.

**complete/disconnect subgraph**

Create/delete edges between every pair of vertices in a set.

*Graphic* is a filter which, when applied to a file in graphdraw format, outputs *pic* code for the graph.

The options are:

- v Print vertex labels.
- e Print edge labels.
- i Optimize for imagen printer (default is d202).

**File format**

Graphs are stored as adjacency lists.

First line:  $n m t$ , where  $n$  is the number of vertices,  $m$  is the number of edges, and  $t$  is an optional graph type. The only legal type is the default type **u** (undirected).

For each vertex, an initial line:  $d w x y c$ , where  $d$  is the degree of the vertex,  $w$  is its label,  $x$  and  $y$  are its coordinates in the window, and  $c$  is an optional graphics code, **L**=light (default), **H**=heavy (circled dot), **F**=full (large bullet), **E**=empty (empty circle), **I**=invisible. Window coordinates will be scaled to fit when graph is read in.

After the initial line follow  $d$  lines for the vertex's edges:  $i w c$ , where  $i$  is the index (1 to  $n$ ) of the other endpoint,  $w$  is the edge label, and  $c$  is an optional graphics code, **L** or **H**.

**Algorithm animation**

The typed command `!program file` causes the standard output of *program* to be captured by the host and interpreted as commands to *graphdraw*. The resulting movie can be killed or temporarily halted from the terminal by clicking button 2 and choosing the desired option from the resulting menu.

Animation codes (defined parenthetically with menu items above) appear one per line. Their arguments are:  $i$ , index of a vertex (normally the  $i$ th to be created);  $x, y$ , integer coordinates in the range 0 to **maxcoord**;  $w$  a label; or  $c$ , a graphic code.

Other animation codes are

- pw**  $n$  Change the value of **maxcoord** to  $n$ . Default is 10,000.
- vl**  $i w$  Give vertex  $i$  the label  $w$ .
- pd**  $t$  Delay program for  $t$  clicks of the 60Hz clock.
- ps** Halt program until user clicks button 2 to continue.
- pm** *message* Print *message* on command line.

**SEE ALSO**

[dag\(1\)](#), [pic\(1\)](#)

**BUGS**

It is impossible to move or reshape a *graphdraw* layer, except via standard window.

**NAME**

icon – icon editor

**SYNOPSIS**

**icon**

**DESCRIPTION**

*Icon* is a pixel-level editor for textures and small bitmaps. *Icon* presents a magnified pixel grid and a true-size image. Editing is done on the magnified grid. Pixels can be turned black by pressing the button 1, and white by pressing button 2.

Button 3 provides an iconic menu of editing commands. Some commands require a rectangle to be swept; this is done either by the middle button (which supplies a fixed 16×16 rectangle) or by the right button (for rectangles of any size).

arrow    Move region (sweep rectangle and click at destination).

overlapping regions

    Copy region (sweep rectangle and click at destination).

cross    Invert region (sweep rectangle).

eraser   Erase region (sweep rectangle).

horizontal (vertical) folded arrow

    Reflect region horizontally (vertically) (sweep rectangle).

clockwise (counterclockwise) arrow

    Rotate region deasil (withershins) (sweep rectangle).

horizontal (vertical) sheared lines

    Shear a region horizontally (vertically) (sweep rectangle and point at destination of nearest corner of rectangle).

scaled square

    Scale a region (sweep rectangle and sweep destination rectangle).

tweed pattern

    Texture a region (sweep source rectangle and a (bigger) destination rectangle to be tiled with copies of the source).

glasses

    Read file (type file name and position the icon by clicking). The subdirectories of /usr/jerq/icon/ are searched automatically after the current directory.

grid     Switch on or off the background grids.

extend region

    Change the size of the drawing area.

pen      Write file (sweep rectangle and type file name). See [bitfile\(9\)](#) for the format.

overlapping rectangles

    Bitblt region (driven by submenus on the right button).

mouse   Pick up a 16×16 rectangle and make it the current cursor (click a button to pick up a 16×16 region, and click again to revert to normal).

help     Display help information (click a button to revert to normal).

band-aid

    Undo last drawing operation.

**FILES**

/usr/jerq/icon/\*/\*

**SEE ALSO**

[bitfile\(9\)](#)

**NAME**

*jf* – font editor

**SYNOPSIS**

**jf** [*file ...* ]

**DESCRIPTION**

*Jf* edits jerq font files. If *file* does not begin with a slash and is not a font file, it is looked up in a standard font directory.

*Jf* is mostly mouse- and menu-driven, except when prompting for file names. *Jf* divides its layer into two types of areas: Font displays show all characters in a given font in actual size. When characters are opened for editing, they appear magnified in edit displays.

Button 1 is the ‘do it’ button. Clicking button 1 inside a font display opens a character for editing; inside an edit display it sets a pixel. It may have other functions selected via menus, in which case the function is indicated by a special cursor.

Button 2 is the ‘undo it’ button. Clicking button 2 closes a character or clears a pixel, unless conditioned otherwise via menu selection.

Button 3 is the ‘menu’ button. Clicking button 3 selects a menu, pops control back to the top level, or (when the gunsight cursor shows) picks a font or character to be affected. Sometimes menu selection is the only (non-trivial) option available, as indicated by a ‘menu’ cursor.

A font is described by several parameters; these are either read from the font file, or set by default by the **make new font** function: **max width** (default 16 pixels), **height** (16)–measured from the top, **ascent** (16)–the distance of the printing baseline from the top, and **range** (1)–the highest-numbered character in the font. (The first character is numbered 0.) All may be changed under the **set sizes** menu. **Squeeze font**, in the **open/close font** menu, reduces max width as much as possible.

Each character has a width, which is shown by the length of the baseline in the edit display. The **char width** may be set under the **set sizes** menu; button 1 sets it to 0, button 2 sets it to a specified pixel within the max width. The quantity **char left** may be used for kerning. If positive, it shifts a character right and causes **max width** to increase if necessary; if negative, the character will be shifted left. Otherwise *char left* is irrelevant to font editing.

The **bit function** menu controls copying among characters in any of the *bitblt(9)* Codes: **F\_STORE**, **F\_CLR**, **F\_OR**, **F\_XOR**. Press button 3 on the source character; hold it down while moving and release it on the destination.

Several fonts may be open at once. When editing a font, it is often convenient to open a second copy for recovering botched characters.

**FILES**

/usr/jerq/font/\*  
jerq fonts

/usr/jerq/include/font.h  
jerq font header file

**SEE ALSO**

*font(9)*

**DIAGNOSTICS**

When out of memory or screen area, *jf* ignores the offending operation.



**NAME**

jim, jim.recover – text editor

**SYNOPSIS**

**jim** [ *file ...* ]

**vim.recover** [ **-f** ] [ **-t** ] [ *file ...* ]

**DESCRIPTION**

*Jim* is an old text editor for the jerq terminal. It relies on the mouse to select text and commands. It runs only under *mux(9)*. *Jim*'s screen consists of a number of *frames*, a one-line command and diagnostic frame at the bottom and zero or more larger file frames above it. Except where indicated, these frames behave identically. One of the frames is always the current frame, to which typing and editing commands refer, and one of the file frames is the working frame, to which file commands such as pattern searching and IO refer.

A frame has at any time a selected region of text, indicated by reverse video highlighting. The selected region may be a null string between two characters, indicated by a narrow vertical bar between the characters. The editor has a single 'save buffer' containing an arbitrary string. The editing commands invoke transformers between the selected region and the save buffer.

The mouse buttons are used for the most common operations. Button 1 (left) is used for selection. Clicking button 1 in a frame which is not the current frame makes the indicated frame current. Clicking button 1 in the current frame selects the null string closest to the mouse cursor. Making the same null selection twice ('double clicking') selects (in decreasing precedence) the bracketed or quoted string, word or line enclosing the selection. By pushing and holding button 1, an arbitrary contiguous visible string may be selected. Button 2 provides a small menu of text manipulation functions, described below. Button 3 provides control for inter-frame operations.

The button 2 menu entries are:

- cut** Copy the selected text to the save buffer and delete it from the frame. If the selected text is null, the save buffer is unaffected.
- paste** Replace the selected text by the contents of the save buffer.
- snarf** Copy the selected text to the save buffer. If the selected text is null, the save buffer is unaffected.
- look** Search forward for the next occurrence of the selected text or, if the selection is null, to the next occurrence of the text in the save buffer.
- <mux>** Exchange save buffers with *mux*.

Also stored on the button 2 menu are the last Unix command and last search string typed (see below); these may be selected to repeat the action.

Typing replaces the selected text with the typed text. If the selected text is not null, the first character typed forces an implicit **cut**. Control characters are discarded, but BS (control-**H**), ETB (control-**W**) and ESC have special meanings. BS is the usual backspace character, which erases the character before the selected text (which is a null string when it takes effect). ETB erases back to the word boundary preceding the selected text. There is no line kill character. ESC selects the text typed since the last button hit or ESC. If an ESC is typed immediately after a button hit or ESC, it is identical to a **cut**. ESC and **paste** provide the functionality for a simple undo feature.

The button 3 menu entries are:

- new** Create a new frame, much as in *mux*.
- reshape** Change the shape of the indicated frame, as in *mux*. The frame is indicated by a button 3 hit after the selection.
- close** Close the indicated frame and its associated file.
- write** Write the indicated frame's contents to its associated file.

The rest of the menu is a list of file names available for editing. To work in a different file, select the file from the menu. If the file is not open on the screen, the cursor will switch to an outline box to prompt for

a rectangle to be swept out with button 3, as in the New operator of *mux*. (Unlike *mux*, there is a shorthand: sweeping the empty rectangle creates the largest possible rectangle.) The file is not read until its frame is first opened. If the file is already open, it will simply be made the workframe and current frame (for typing). The format of the lines in the menu is

- possibly an apostrophe, indicating that the file has been modified since last written,
- possibly a period or asterisk, indicating the file is open (asterisk) or the workframe (period),
- a blank,
- and the file name. The file name may be abbreviated by compacting path components to keep the menu manageable, but the last component will always be complete.

The work frame has a 'scroll bar'—a black vertical bar down the left edge. A small tick in the bar indicates the relative position of the frame within the file. Pointing to the scroll bar and clicking a button controls scrolling operations in the file:

- button 1            Move the line at the top of the screen to the *y* position of the mouse.
- button 2            Move to the absolute position in the file indicated by the *y* position of the mouse.
- button 3            Move the line at the *y* position of the mouse to the top of the screen.

The bottom line frame is used for a few typed commands, modeled on *ed(1)*, which operate on the work frame. When a carriage return is typed in the bottom line, the line is interpreted as a command. The bottom line scrolls, but only when the first character of the next line is typed. Thus, typically, after some message appears in the bottom line, a command need only be typed; the previous contents of the line will be automatically cleared. The commands available are:

- e** *file*    Edit the named *file*, or use the current file name if none specified. Note that each file frame has an associated file name.
- f** *file*    Set the name of the file associated with the work frame, if one is specified, and display the result.
- g** *file ...*  
Enter the named *files* into the filename menu, without duplication, and set the work frame to one of the named files. If the new work frame's file is not open, the user is prompted to create its frame. The arguments to **g** are passed through *echo(1)* for shell metacharacter interpretation.
- w** *file*    Write the named *file*, or use the current file name if none specified. The special command *w* writes all modified files with file names.
- q**        Quit the editor.
- =**        Print the line number of the beginning of the selected text.
- /**        Search forward for the string matching the regular expression after the slash. If found, the matching text is selected. The regular expressions are exactly as in *egrep(1)*, with two additions: the character **@** matches any character, including newline, and the sequence **\n** specifies a newline, even in character classes. The negation of a character class does not match a newline. An empty regular expression (slash-newline) repeats the last regular expression.
- ?**        Search backwards for the expression after the query.
- 94**      Select the text of line 94, as in *ed*.
- cd**      Set the working directory, as in the shell. There is no **CDPATH** search.
- >command**  
Send the selected text to the standard input of the Unix *command*.
- < command**  
Replace the selected text by the standard output of the Unix *command*.
- | command**  
Replace the selected text by the standard output of the Unix *command*, given the original selected text as standard input.

If any of **< > |** is preceded by an asterisk **\***, the command is applied to the entire file, instead of just the selected text. If the command for **<** or **|** exits with non-zero status, the original text is not deleted;

otherwise, the new text is selected. Finally, the standard error output of the command, which is merged with the standard output for `>`, is saved in the file. If the file is non-empty when the command completes, the first line is displayed in the diagnostic frame. Therefore the command `>pwd` will report *jim*'s current directory.

Attempts to quit with modified files, or edit a new file in a modified frame, are rejected. A second `q` or `e` command will succeed. The `Q` or `E` commands ignore modifications and work immediately. Some consistency checks are performed for the `w` command. *Jim* will reject write requests which it considers dangerous (such as writes which would change a file modified since *jim* read it into its memory). A second `w` will always write the file.

If *jim* receives a hangup signal, it writes a file which is a shell command file that, when executed, will retrieve the files that were modified when *jim* exited. The `-t` option prints a table of contents, but does not unpack the files. By default, *jim.recover* is interactive; the `-f` option suppresses the interaction. If no files are named to it will recover all the saved files.

## FILES

`$HOME/jim.err`

`$HOME/jim.recover`

## BUGS

The regular expression matcher is non-deterministic, and may be slow for spectacular expressions.

When reshaped, the open frames must be re-opened manually.

The `<` and `|` operators should snarf the original text.

**NAME**

jiocctl – mux iocctl requests

**SYNOPSIS**

```
#include "/usr/jerq/include/jiocctl.h"
```

```
iocctl(fd, request, 0)
```

```
iocctl(fd, JWINSIZE, win)
```

```
struct winsize *win;
```

**DESCRIPTION**

*Mux(9)* supports several *iocctl(2)* requests for Unix programs attached to layers. The requests are:

**JMUNIX**

returns 0 if file descriptor *fd* is connected to a *mux* layer, -1 otherwise.

**JTERM**

resets the layer connected to *fd* to the default terminal program.

**JBOOT**

initiates the down-load protocol to replace the layer's terminal program. Usually called by *32ld(9)*

**JZOMBOOT**

is the same as *JBOOT*, but disables execution of the program when the download is complete (see the **-z** flag of *32ld*).

**JWINSIZE**

returns, in the location pointed to by the third argument, a structure describing the size of the layer connected to *fd*, with character 0 being the unit of size. The structure is:

```
struct winsize {
    char  bytesx, bytesy; /* size in characters */
    short bitsx, bitsy; /* size in pixels */
};
```

**JEXIT**

causes *mux* to exit.

**SEE ALSO**

*32ld(9)* *mux(9)* *iocctl(2)*

**NAME**

`jx` – 5620 execution and stdio interpreter

**SYNOPSIS**

`jx file [ argument ... ]`

**DESCRIPTION**

*Jx* downloads the program in *file* to the terminal or layer on its controlling tty and runs it there, simulating standard I/O functions of [stdio\(3\)](#). *Jx* works either stand-alone or in a layer.

The **stdout** and **stderr** streams, if directed to the controlling terminal, will be squirreled away during execution and copied to the terminal after the down-loaded program exits.

Programs to be run by *jx* should include `<jerqio.h>` and call **exit()** upon termination in order to restart the default terminal program. Programs to be run stand-alone should be compiled with the **-J** option of [3cc\(9\)](#). No special options are required for running in a layer.

*Stdio(3)* functions available under *jx* are

<code>getc</code>	<code>putc</code>	<code>fopen</code>	<code>popen</code>	<code>printf</code>	<code>fread</code>
<code>getchar</code>	<code>putchar</code>	<code>freopen</code>	<code>pclose</code>	<code>sprintf</code>	<code>fwrite</code>
<code>fgets</code>	<code>puts</code>	<code>fclose</code>		<code>fprintf</code>	
	<code>fputs</code>	<code>access</code>			
	<code>fflush</code>				

Unlike in [stdio\(3\)](#), *getc* and *putc* are functions, not macros. *Printf* has only **d**, **s**, **c**, **o**, and **x**. **u** prints an unsigned decimal number. **D** prints an unsigned long decimal number.

Since *jx* uses *sendchar*, *sendnchars*, and *revchar*, *jx* programs should avoid these, and use only the standard I/O routines.

**FILES**

`/usr/jerq/include/jerqio.h`

`/usr/jerq/lib/sysint`  
standard I/O interpreter

`$HOME/.jxout`  
saved standard output

`$HOME/.jxerr`  
saved standard diagnostic output


**SEE ALSO**

[request\(9\)](#) [stdio\(3\)](#)

**BUGS**

Keyboard standard input doesn't work; use *kbdchar*; see [request\(9\)](#). Stand-alone programs do not receive arguments.

**NAME**

lens – bitmap 

**SYNOPSIS**

**lens**

**DESCRIPTION**

*Lens* is an interactive screen bitmap magnifier. When it starts, it displays an enlarged image of a magnifying glass in its layer, which becomes a setting sun when *lens* wants to confirm a command to exit.

The first item in the button 2 menu, which rotates among **go**, **peek**, and **stop**, determines the activity of the magnifier. Clicking button 1 serves as an abbreviation for selecting **go** or **peek**. When the magnifier is going, a crawling-bordered rectangle is drawn around the source, and the *lens* window contains the magnified image. The mouse controls the position of the source rectangle.

During peeking, the rectangle last selected while going is re-examined periodically, and the contents are magnified, whether or not the *lens* window is currently selected.

When stopped, the *lens* window is inactive.

The button 2 menu also allows changing the magnification factor. The magnification factors are chosen from the Fibonacci numbers, and menu items for the next size smaller and larger are presented as, e.g., 3x or 8x. The current magnification factor is not displayed in the menu, only the next factors larger and smaller. The initial magnification factor is two.

Button 2 may also be used to select the intervals at which peeking updates occur. These intervals are selected, in ticks, from among the powers of two, where a tick is one-sixtieth of a second. These choices are presented as, e.g., 32 ticks or 128 ticks. The initial interval between peeks is 64 ticks, approximately one second.

The image window may be controlled by the button 2 menu item which toggles between **inset** and **full size**. In inset mode, the image is displayed inside the image window of the magnified lens icon. In full size mode, the image is displayed in the entire *lens* window.

The final button 2 menu entry is *exit*. A setting sun is displayed, and button 3 must be clicked to confirm.

**BUGS**

While going, the display is only refreshed when the mouse is moved.

While peeking, it is assumed that the *lens* window contains an accurate magnification of what was on the screen at the time of the last magnification. If *lens* is used to examine its own image, strange things may occur.

Due to the bitmap reshaping techniques employed by the magnification algorithms, high magnification factors will not work with large image windows. Precisely, if the product of the vertical magnification factor and the width of the destination rectangle overflows a signed short integer, predictable but undesirable results will occur.

**NAME**

libc – standard library functions

**DESCRIPTION**

Various standard functions from Section 3 are available in 5620 programs:

abs atoi atol chrtab qsort rand srand streak strchr strchr stremp strepy strncat strncmp strncpy strlen

In addition, certain [stdio\(3\)](#) programs are available under the [jx\(9\)](#) emulator.

**SEE ALSO**

[arith\(3\)](#), [atof\(3\)](#), [chrtab\(3\)](#), [libc\(9\)](#) [qsort\(3\)](#), [rand\(3\)](#), [string\(3\)](#)

**NAME**

lsh – create layers and run shell commands

**SYNOPSIS**

**lsh** [<file] [>file]

**DESCRIPTION**

*Lsh* runs under *mpx(1)* and reproduces a specified setup of layers. Each line of the standard input is of the form:

x0 y0 x1 y1 shell-command

For each line *Lsh* creates a layer whose diagonal spans the points  $(x_0, y_0)$  and  $(x_1, y_1)$ , where  $(0, 0)$  is the upper left corner of the screen and  $(800, 1024)$  is the lower right. If a shell-command is given, it is executed in that layer.

The standard output gives the coordinates of each layer that already exists and its downloaded object file, if any. This provides coordinates for an input script to duplicate a handmade setup.

**BUGS**

Standard input cannot be the keyboard.



**NAME**

`mcc` – MC68000 C compiler

**SYNOPSIS**

**mcc** [ *option* ] ( .. ).SH DESCRIPTION *Mcc* is the C compiler for the Motorola 68000. Its default action is to compile programs to run under the *mpx(1)* environment on a Blit terminal.

*Mcc* accepts several types of arguments:

Arguments whose names end with `.c` are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with `.o` substituted for `.c`. The `.o` file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with `.s` are taken to be assembly source programs and are assembled, producing a `.o` file.

Programs using floating-point must be compiled with the `-lf` load-time option to load the floating-point support package.

The following options are interpreted by *mcc*. Load time options, described under *mld(1)*, are passed to *mld*.

- `-c` Suppress the loading phase of the compilation; force an object file to be produced even if only one program is compiled.
- `-j` Compile the named programs, and load and link them for running stand-alone on a Blit terminal.
- `-m` Compile the named programs for ordinary (non-Blit) environments.
- `-w` Suppress warning diagnostics.
- `-O` Invoke an object-code improver.
- `-S` Compile the named C programs, and leave the assembler-language output on corresponding files suffixed `.s`.
- `-E` Run only the macro preprocessor on the named C programs, and send the result to the standard output.
- `-C` prevent the macro preprocessor from eliding comments.
- `-o output`  
Name the final output file *output*. If this option is used the file `a.out` will be left undisturbed.
- `-Dname=def`
- `-D$name`  
Define the *name* to the preprocessor, as if by `#define`. If no definition is given, the name is defined as `"1"`. The symbol *mc68000* is predefined.
- `-U$name`  
Remove any initial definition of *name*.
- `-ISdir` `#include` files whose names do not begin with `/` are always sought first in the directory of the *file* argument, then in directories named in `-I` options, then in directories on a standard list.
- `-B$string`  
Find substitute compiler passes in the files named *string* with the suffixes `cpp`, `ccom` and `c2`. If *string* is empty, use a standard backup version.
- `-t[p012]`  
Find only the designated compiler passes in the files whose names are constructed by a `-B` option. In the absence of a `-B` option, the *string* is taken to be `/usr/c/`.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier *mcc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

**FILES**

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/usr/jerq/lib/ccom	compiler
/usr/jerq/lib/ocom	backup compiler
/usr/jerq/lib/mc2	optimizer
/usr/jerq/lib/l.o	runtime startoff for <b>-j</b>
/usr/jerq/lib/notsolow.o	runtime startoff for <b>-m</b>
/usr/jerq/lib/libc.a	standard library
/usr/jerq/lib/libf.a	floating-point library
/usr/jerq/lib/libj.a	graphics library (used in <b>-lj</b> ).
/usr/jerq/lib/libsys.a	system and I/O library (used in <b>-lj</b> ).
/usr/jerq/include	standard directory for '#include' files

**OTHER PROGRAMS**

The usual array of associated object-code manipulating programs exists, with specifications identical to the usual Unix programs, and with names prefixed with an 'm.' These programs include:

mas	assembler, see <a href="#">as(1)</a>
mlorder	order library, <a href="#">lorder(1)</a> (there is no mranlib)
mnm	name list, see <a href="#">nm(1)</a>
msize	object code size, <a href="#">size(1)</a>
mstrip	strip symbol table, <a href="#">strip(1)</a>

**SEE ALSO**

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978  
 B. W. Kernighan, *Programming in C—a tutorial*  
 D. M. Ritchie, *C Reference Manual*  
[mld\(1\)](#), [cc\(1\)](#)

**DIAGNOSTICS**

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

**NAME**

menuhit, hmenuhit – present user with menu and get selection

**SYNOPSIS**

```
#include <jerq.h>
```

```
int menuhit(m, b)
```

```
Menu *m;
```

```
#include <menu.h>
```

```
NMitem *hmenuhit(m, b)
```

```
NMenu *m;
```

**DESCRIPTION**

*Menuhit* presents the user with a menu specified by the Menu pointer *m* and returns an integer indicating the selection made, or  $-1$  for no selection. The integer *b* specifies which button to use for the interaction: 1, 2 or 3. *Menuhit* assumes that the button is already depressed when it is called. The user makes a selection by lifting the button when the cursor points at the desired selection; lifting the button outside the menu indicates no selection.

Menus can be built in two ways, either as an array of strings or with a generator function:

```
typedef struct {
    char **item;           /* string array, ending with 0 */
    char *(*generator)(); /* used if item == 0 */
    short prevhit;        /* offset from top of last select */
    short prevtop;        /* topmost item displayed */
} Menu;
char *menutext[]={"Item 0", "Item 1", "Item 2", 0};
Menu stringsmenu={ menutext };
```

or

```
char *menugen();
Menu genmenu={ (char **)0, menugen };
```

The generator function is passed an integer parameter *n*, and must return the string for the *n*th menu entry, or 0 if *n* is beyond the number of entries in the menu. The *n*'s may come in any order but the result is only needed until the next call.

Regardless of the method of generation, characters with the **0200** bit set are regarded as fill characters. For example, the string "\240X" will appear in the menu as a right-justified X (the ASCII space character). Menu strings without fill characters are drawn centered in the menu.

The fields *prevhit* and *prevtop* are used to guide which items are displayed and which item the mouse points to initially. They should be nonnegative. Both *menuhit* and *hmenuhit* may choose to ignore these fields.

*Hmenuhit* supports hierarchical menus. Submenus are denoted graphically by a right-pointing arrow. Moving the cursor onto the arrow causes the submenu to appear. Hierarchical menus are built of **NMItems** defined as

```
typedef struct NMenu {
    char *text;
    char *help;
    struct NMenu *next;
    void (*dfn)(), (*bfn)(), (*hfn)();
    long data;
} NMitem;
```

The **text** field is shown to the user; characters with the **0200** bit set behave as above. The contents of the **help** field are shown whenever the user holds down button 1 at the same time as the button specified by the parameter *b*. If *b* is 1, you get help all the time. The **next** field is the address of a submenu or (**NMenu \***)0 if there is none. The two functions **(\*dfn)()** and **(\*bfn)()** support dynamic submenus. *Dfn* is called just before the submenu is invoked. Its argument is the current menu item. Similarly, *bfn* is called

with the current menu item just after the submenu has finished. *Hfn* is called only when a menu item is selected; its argument is the current menu item. The menu has been undrawn before *hfn* is called. The return value from *hmenuhit* is the menu item selected or (**NMenu \***)**0** if none was selected. To permit communication between menu functions and the calling program, the *data* field is available for the user; it is ignored by *hmenuhit*.

An **NMenu**, like a **Menu**, may be built by list or by generator. An **NMenu** generator takes an integer parameter *n* and returns a pointer to an **NMitem**. In either case, the list of menu items is terminated by an item with a **0 text** field.

## EXAMPLES

Simple code to use **stringsmenu** declared above:

```
switch(menuhit(.stringsmenu, 3)){
case 0:  item_0();
        break;
case 1:  item_1();
        break;
case 2:  item_2();
        break;
case -1: noselection();
        break;
}
```

To provide a submenu for item 1:

```
NMitem *gen();
NMenu illist = { 0, gen };
void item_2(), item_3();
NMitem imenu = {
    { "item 1", "item 1 help", .illist },
    { "item 2", "item 2 help", 0, 0, 0, item_2 },
    { "item 3", 0, 0, 0, 0, item_3 },
    { 0 }
};
NMenu b3 = { imenu };
(void)hmenuhit(.b3, 3);
```

**NAME**

mld – MC68000 link editor (loader)

**SYNOPSIS**

**mld** [ option ] ... file ...

**DESCRIPTION**

*Mld* combines several Motorola 68000 object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *mld* combines them, producing an object module which can be either executed or become the input for a further *mld* run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of *mld* is left on **a.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

A library is a collection of object modules gathered into a file by *ar* (1). If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important.

The symbols 'etext', 'edata' and 'end' are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

*Mld* understands several options. Except for **-l**, they should appear before the file names.

- b**      relocate the program so its first instruction is at the absolute position indicated by the decimal *address* after the **-b** option.
- B**      Similar to **-b**, but only set the base address for the BSS segment. This option is usually used in conjunction with **-b** when loading programs to run from ROM.
- d**      Force definition of common storage even if the **-r** flag is present.
- lx**      This option is an abbreviation for the library name '/usr/lib/libx.a', where *x* is a string. If that does not exist, *mld* tries '/usr/jerq/lib/libx.a' A library is searched when its name is encountered, so the placement of a **-l** is significant.
- o**      The *name* argument after **-o** is used as the name of the *mld* output file, instead of **a.out**.
- r**      Generate relocation bits in the output file so that it can be the subject of another *mld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.
- s**      Remove the symbol table and relocation symbols to save space in the resulting binary.
- R**      Similar to **-r**, but flag an error if there are undefined symbols.
- M**      Set the resulting a.out's magic number to 0406, to signify a binary runnable under *mpx*(1)
- v**      Generate copious debugging information on standard output.

**FILES**

/usr/jerq/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	output file

**SEE ALSO**

ld(1), mcc(1), ar(1)

**NAME**

mouse – jerq mouse user interface

**DESCRIPTION**

Most jerq programs use the mouse for control, either by pointing at things on the screen or by making selections from a menu. The mouse buttons are different from keys on a keyboard in that events are reported when a button is released (let ‘up’) as well as depressed (pressed ‘down’). It therefore matters not only *where* and *when* a button is pressed, but for how long. For example, menus are drawn when a button is depressed, and remain displayed as long as the button is held down. While the button is down, moving the cursor over the menu highlights entries in the menu; the entry (possibly none) under the cursor when the button is *released* is the selection returned to the program. Large menus also present a ‘scroll bar’ on the left side of the menu. Moving the mouse inside the scroll bar chooses which subset of the available entries are displayed and therefore selectable.

There is a convention about how the buttons are used. The left button (button 1) is used to point: selecting which layer to work in, which file inside the editor, some text in the file, etc. The middle button (button 2) produces a menu of actions related to the selection: remove the selected text, replace it, etc. The right button (button 3) presents a menu of global, program-wide actions: pick up a new file, rearrange the files on the screen, etc. Programs follow this convention well enough that an unfamiliar program can often be learned simply by trying it. The main violators of the convention are drawing programs, which use button 1 to draw things and button 2 to undraw them, but this is also a consistent convention.

The mouse cursor is usually an arrow pointing at a pixel, but programs often change the cursor to an iconic representation of the program’s state. The most common cursors are:

arrow    standard cursor

coffee cup

Program will be busy for a while.

rectangle and arrow

Program expects a rectangle to be ‘swept out’ by pressing a button (usually 3) at one corner and releasing at the diagonally opposite corner.

gunsight

Program expects an object to be selected by pointing at it and pressing a button (usually 3).

upside-down mouse

Program is thinking; the mouse is inoperative.

**NAME**

movie, stills – algorithm animation

**SYNOPSIS**

**movie** [ **-t** *termprog* ] [ **-m** *memory* ] [ *file* ]

**stills** [ *files ...* ]

**DESCRIPTION**

*Movie* converts a ‘movie script’ into an internal representation, then displays it in a window on a Teletype 5620, AT:T630, or X-11 system (depending on which version has been compiled). If the filename is of the form *file.s*, *movie* creates the intermediate form in *file.i*, which will be used in subsequent calls if it is more recent than *file.s*. The options are:

**-t** *termprog*

Load *termprog* instead of the default terminal program.

**-m***mem* Use *mem* bytes of terminal memory instead of the default.

In the terminal, button 1 stops and starts the movie; button 2 adjusts view sizes and selects clicks; button 3 sets various parameters.

**Movie scripts**

A movie consists of multiple independent views, each presented as a rectangular sub-window. If no **view** statements appear, there is a single implicit view **def.view**. Any text or geometrical object may be labeled with a name and colon. Labels and coordinates are local to views. A recurring label erases the previous object with that label.

Comments follow #; blank lines are ignored.

**text** *options x y string*

Text is centered and medium size by default; options: one of **center ljust rjust above below**, and one of **small medium big bigbig**. A leading quote is stripped from *string*, as is a trailing quote if a leading one is present.

**line** *options x1 y1 x2 y2*

Lines are solid by default; options: one of **fat fatfat dotted dashed** and one of **-> <- <->**.

**box** *options xmin ymin xmax ymax*

A box may be **filled**.

**circle** *options x1 y1 radius*

Radius is measured in the *x* dimension. A circle may be **filled**.

**erase** *label*

Erase an object explicitly.

**clear** Erase all objects currently in the current view.

**click** *optional-name*

Place a mark in the intermediate with this name; clicks are used to control stepping in a movie or to define frames for a set of stills.

**view** *name*

Associate subsequent objects with this view, until changed again.

*Stills* converts selected frames of a movie into commands for *pic(1)*. Commands for *stills* begin with **.begin stills** and end with **.end stills**.

**FILES**

All files are in **/usr/lib/movie**.

develop

Shell script to control conversion from script language to internal form.

fdevelop

C program that does the work.

stills.awk

Awk program to process stills language into *pic(1)*.

anim Host end of the animation system.

animterm  
terminal end.

newer  
Test whether one file is newer than another.

**SEE ALSO**

[flicks\(9\)](#) [pic\(1\)](#)

J. L. Bentley and B. W. Kernighan, 'A System for Algorithm Animation', this manual, Volume 2

**BUGS**

The 630 can only handle 65000 bytes of memory.



**NAME**

*mux*, *ismux*, *invert* – layer multiplexer for 5620

**SYNOPSIS**

**mux** [ **-l** *command* ... ]

**mux** **exit**

**mux** **cd** *directory*

**ismux** [ - ]

**invert**

**DESCRIPTION**

*Mux* manages asynchronous windows, or layers. Upon invocation, it loads the terminal with a program (default settable by the environment variable **MUXTERM**) that is the primary user interface. Option **-l** also creates a layer and invokes the shell to run *commands* in it. (See [windows\(9\)](#))

The command *mux* leaves *mux*, destroying all layers; *mux* changes the directory of *mux*, and hence of layers later created, but not of the current layers.

Each layer is essentially a separate terminal. Characters typed into the layer are sent to the standard input of a Unix process bound to the layer, and characters written on the standard output of that process appear in the layer. When a layer is created, a separate shell (the value of the **SHELL** environment variable, or *sh* by default) is established, and bound to the layer.

Layers are created, deleted, and rearranged using the mouse. Depressing mouse button 3 activates a menu of layer operations. Releasing button 3 then selects an operation. At this point, a gunsight or box cursor indicates that an operation is pending. Hitting button 3 again activates the operation on the layer pointed to by the cursor.

The **New** operation, to create a layer, requires a rectangle to be swept out, across any diagonal, while button 3 is depressed. A box outline cursor indicates that a rectangle is to be created. The **Reshape** operation, to change the size and location of a layer on the screen, requires first that a layer be indicated (gunsight cursor) and a new rectangle be swept out (box cursor). The other operations are self-explanatory.

In a non-current layer, button 1 is a shorthand for **Top** and **Current**, which pulls a layer to the front of the screen and makes it the active layer for keyboard and mouse input. The current layer is indicated by a heavy border.

There is a point in each layer, called the ‘Unix point’, where the next character from the host Unix system will be inserted. The Unix point advances whenever characters are received from the host, but not when echoing typed characters. When a newline is typed after the Unix point, characters between the Unix point and the newline, inclusive, are sent to the host and the Unix point advanced to after the newline. This means that shell prompts and other output will be inserted before characters that have been typed ahead. No other characters are sent to the host (but see the discussion of raw mode below). Therefore partially typed lines or text anywhere before the Unix point may be edited.

The default terminal program allows any text on the screen to be edited, much as in [sam\(9\)](#) Text may be selected by sweeping it with button 1 depressed. Typed characters replace selected text.

All layers share a common ‘snarf buffer’ (distinct from *sam*’s). The **cut** operation on button 2 deletes selected text and puts it in the buffer; **snarf** copies selected text to the buffer; **paste** replaces selected text (which may be null) from the buffer; and **send** copies the snarf buffer to after the Unix point.

Normally the terminal doesn’t scroll as text is received, but a button 2 menu item selects scrolling.

A scroll bar indicates what portion of all the text stored for a layer is on the screen. (It measures characters, not lines.) Releasing button 1 in the scroll bar brings the line at the top of the screen to the cursor; releasing button 3 takes the line at the cursor to the top of the screen. Button 2, treating the scroll bar as a ruler, brings the indicated point in the whole stored text to the top of the screen. Slide the cursor off either end of the scroll bar with button 2 depressed to get right to an end of the file.

The NUM LOCK key advances a half page.

*Ismux* reports on its standard error whether its standard output is a *mux* layer, and also generates the appropriate exit status. With option **-**, no message is produced.

*Invert* reverses the sense of video, from black on white to white on black, or *vice versa*.

Independent user-level programs can be loaded into layers, see [32ld\(9\)](#) SHIFT-SETUP freezes *mux* and complements the video of the layer of the running user-level terminal process. Hitting button 2 in this state will attempt to kill the process; 1 or 3 will leave it running.

In raw mode or no-echo mode (see [ttyld\(4\)](#)) the Unix point advances with each character typed after it. In 8bit mode, characters with octal codes 0200 and greater print according to the ISO Latin1 alphabet; see [ascii\(6\)](#).

## FILES

/tmp/.mux\* temporary file used by **-l** option

## SEE ALSO

[32ld\(9\)](#) [sam\(9\)](#) [jx\(9\)](#) [term\(9\)](#) [windows\(9\)](#)

R. Pike, 'Blit Download Protocols', this manual, Vol. 2

## DIAGNOSTICS

*Mux* refuses to create a layer when there is not enough memory. Space can be recovered by deleting a layer.

Error messages from *mux* are written directly to the layer which caused them. They are usually meaningful only to system administrators, and indicate system difficulties.

## BUGS

Reshape only works properly for processes that arrange to see if they have been reshaped, although most programs make this arrangement.

The behavior of raw mode prohibits editing partially typed lines when running [cu\(1\)](#).

**NAME**

strinsure, strinsert, strdelete, strzero, setmuxbuf, getmuxbuf, movstring – dynamic strings in mux

**SYNOPSIS**

```
#include <jerq.h>

strinsure(s, n); String *s;
strinsert(d, i, s); String *d, *s;
strdelete(d, i, j); String *d;
strzero(d); String *d;
setmuxbuf(s); String *s;
getmuxbuf(d); String *d;
movstring(n, sp, dp); char *sp, *dp;
```

**DESCRIPTION**

These functions manipulate strings represented in the following form.

```
struct String {
    char *s;
    short n;
    short size;
};
```

The string proper occupies the first **n** characters of a data block of **size** characters pointed to by **s**. Initially both **size** and **s** should be 0. Strings are always counted, not terminated by **\0**. The functions obtain space as needed from *galloc*; see [alloc\(9\)](#). Thus a **String** structure should never be copied.

*Strinsure* arranges that  $s->size \geq n$ . It must be called before any operation that could overflow the current size.

*Strinsert* inserts a copy of source *s* into destination *d* beginning at character *i* (counted from 0), adding  $s->n$  to  $d->n$ .

*Strdelete* removes characters *i* through *j-1* from string *d*, subtracting  $j-i$ , which must be nonnegative, from  $d->n$ .

*Strzero* frees the memory associated with *d* and sets both  $d->n$  and  $d->size$  to zero.

*Setmuxbuf* copies string *s* into the snarf buffer maintained by [mux\(9\)](#). *getmuxbuf* copies from the snarf buffer into *d*.

*Movstr* copies a block of *n* characters beginning at *sp* to a block beginning at *dp*. If *n* is negative it copies  $-n$  characters ending at  $sp-1$  to a block ending at  $dp-1$ . Notice that *movstring* does not operate on **Strings**.

**SEE ALSO**

[libc\(9\)](#)

**NAME**

`newlayer`, `dellayer`, `downback`, `lbitblt`, `lpoint`, `lrectf`, `lsegment`, `ltexture`, `upfront` – layer control and graphics

**SYNOPSIS**

```
#include <jerq.h>
```

```
Layer *newlayer(r); Rectangle r;
```

```
void dellayer(l) Layer *l;
```

```
void lbitblt(sl, r, dl, p, f) Layer *sl, *dl; Rectangle r; Point p; Code f;
```

```
void lpoint(l, p, f) Layer *l; Point p; Code f;
```

```
void lrectf(l, r, f) Layer *l; Rectangle r; Code f;
```

```
void lsegment(l, p, q, f) Layer *l; Point p, q; Code f;
```

```
void ltexture(l, r, t, f) Layer *l; Rectangle r; Texture *t; Code f;
```

```
void upfront(l) Layer *l;
```

```
void downback(l) Layer *l;
```

**DESCRIPTION**

*Newlayer* creates a layer in Rectangle *r* in the physical display bitmap, and returns its address, or 0 on failure. *Newproc*(9) explains how to attach a process to a layer.

*Dellayer* de-allocates a layer; the associated process must also be freed (see *newproc*(9))

*Upfront* and *downback* are the subroutines corresponding to the *mux*(9) menu items **Top** and **Bottom**.

The routines *lbitblt*, *lpoint*, *lsegment* and *ltexture* are equivalent to their *bitblt*(9) counterparts except that they never inhibit the mouse cursor, so they are mainly useful only for implementation of efficient composite graphics operations such as circle-drawing. Because of the duality of Bitmaps and Layers, arguments of either type may be passed freely to any of the graphics primitives.

**SEE ALSO**

[bitblt\(9\)](#) [newproc\(9\)](#)

Rob Pike, *Graphics in Overlapping Bitmap Layers*, ACM Trans. on Graphics, April 1983.

**NAME**

*P*, *newproc*, *muxnewwind*, *newwindow*, *tolayer*, *debug*, *getproc*, *getproctab*, *putname*, *getname* – *jerq* process control

**SYNOPSIS**

```
#include <jerq.h>

extern struct Proc *P;

struct Proc *newproc(f) void (*f)();

struct Proc *newwindow(f); void (*f)();

void tolayer(l) Layer *l;

void debug();

struct Proc *getproc();

struct Proc *getproctab();

int putname(string, data) char *string; long data;

struct Nqueue *getname(string) char *string;

#include <msgs.h>

void muxnewwind(p, c) struct Proc *p; int c;
```

**DESCRIPTION**

Processes in the *jerq* consist of a coroutine-style process structure and an associated layer (see [newlayer\(9\)](#)) allocated independently. This section describes the process allocation and control primitives. They are direct links to the system's own control structures, so given *mux*'s open addressing, they should be used with care.

Each process has a global variable *P* that points to its process structure. The only regular use of *P* is to check that the process has been moved or reshaped:

```
if(P->state . RESHAPED){
    do_reshape();
    P->state .= RESHAPED;
}
```

The definition of **struct Proc** is in the include file `<jerqproc.h>`, which is included automatically by `<jerq.h>`.

*Newproc* allocates a new process, returning a pointer to it, or 0 if one cannot be allocated. Argument *f* points to the program text to be executed. The special case *f*=0 creates a process running the default terminal program, and is almost always how *newproc* should be called; use [32ld\(9\)](#) to run non-standard programs. A process is disabled by setting *p*->*state* to zero. After calling *newproc*, the process must be bound to a layer and Unix told of its presence, typically as:

```
struct Proc *p;
Rectangle r;
p = newproc((struct Proc *)0);
if(p == 0)
    error();
p->layer = newlayer(r);
if(p->layer == 0){
    p->state = 0;
    error();
}
p->rect = r;
muxnewwind(p, C_NEW);
```

The second argument to *muxnewwind* should be **C\_RESHAPE** if an existing process is being given a new layer. If the process is *not* running the default terminal program, its variables *display* and *Drect* must be set:

```

struct udata *u=((struct udata *)p->data);
u->Direct=p->rect;
u->Jdisplay=p->layer;

```

This procedure works regardless of whether the process being manipulated is itself.

*Newwindow* creates a process by the above procedure, going through the standard user interface to select the rectangle for the process's layer.

*Tolayer* takes an argument *layer* pointer and makes the process in that layer the receiver of mouse and keyboard events.

*Getproc* presents the user with a gunsight cursor and returns the address of the process whose layer is indicated with the mouse. *Getproctab* simply returns the address of the base of the process table array. This is an array of **NPROC** process structures. **NPROC** is stored in the word immediately lower in address than the process table.

*Debug* announces to the system that the calling process is prepared to handle exceptions by other processes.

*Putname* and *getname* manage a bulletin board for interprocess communication. Further communication may be arranged through shared memory. *Putname* associates *data* with *string*, returning nonzero normally, or 0 if the data could not be stored. *Getname* returns a pointer to a structure which contains

**struct Proc \*proc**

pointer to the process structure of the layer that most recently announced the string

**long data**

the corresponding data

*Getname* returns 0 if no such string has been announced. A pointer returned by *getname* remains valid: a client may rendezvous with a server by calling *getname* once and repeatedly testing the associated **proc** pointer thereafter.

## BUGS

These primitives are awkward at best, and are subject to change. Creating a process without a layer or *vice versa* is dangerous.

**NAME**

pads – user interface package

**DESCRIPTION**

*Pads* is a mouse-based interface for browsing a network of windows.

Button 1 points. Pointing at a window makes it current, with a heavy border; pointing at a line of text makes it current, inverts its video, and scrolls it to the middle of the window. A scroll bar at the left of each window shows how much of the text of a window is visible; pointing into the scroll region controls what text is displayed.

Button 2 has a menu of operations that apply to the current line. Operations above the separator are specific to each line; operations below the separator are generic line operations:

**cut** Remove the line.

**sever** Remove the line and all lines above it.

**fold** If lines pass the right margin, continue them on following lines.

**truncate**

Truncate lines at the right margin.

Button 3 has a menu of window-level operations, and is in three parts. Below the lower separator is a list of windows; selecting one makes it current. They appear in front-to-back screen order, current at the top. Operations above the upper separator are specific to each window; operations between the separators are generic window operations:

**reshape****move**

**close** Like **reshape**, **move**, and **delete** in *mux(9)*

**fold****truncate**

apply to all lines in the window.

Keyboard characters accumulate at the bottom of the layer. If the current line accepts input, it flashes with each keystroke; otherwise, if the current window accepts input, its border flashes. Carriage return is ignored until a line or window accepts the text, whereupon the input line is sent to the line or window. The ESC key substitutes the *mux(9)* global snarf buffer.

If the first character of a line from the keyboard is < or > the remainder of the line is interpreted as a shell command. For <, each line of the command's standard output is sent to the line or window, as though it had come from the keyboard. For >, the line or lines of the window become the command's standard input. Each line or window that accepts keyboard input produces some help in response to ?. Special cursor icons occasionally appear:

arrow-dot-dot-dot

The host is completing an operation; the terminal is ready asynchronously.

exclamation mark

Confirm a dangerous menu selection by pressing that menu's button again.

**SEE ALSO**

T. A. Cargill, *Pads Programming Guide*

**NAME**

paint – draw pictures in a layer

**SYNOPSIS**

**paint**

**DESCRIPTION**

*Paint* is a program for artistic interactive drawing. Buttons 1 and 2 draw in different ways, e.g. depositing and erasing paint. Button 3 gets a menu. Certain menu items contain arrows, which if touched call sub-menus. Moving off the right of a submenu causes it to disappear. Some items toggle a state on and off; a \* appears in the abnormal state. Pressing button 1 while holding button 3 gets a short help message for the menu item. The top-level menu contains:

Style → Different kinds of brush strokes  
 Operation → Ways of putting paint on canvas  
 Texture → Things to do to the texture pattern  
 Brush → Things to do to the paintbrush  
 Canvas → Things to do to the whole picture  
 State → Change things saved in *.paintstate*  
 Fill Fill an area of the picture  
 Green Erase the entire picture  
 Mask Display mask instead of image  
 Exit

The **Style** submenu:

Paint Multiple brush spots while holding button 1 or 2  
 Circles Circles; press at center and release at circumference  
 Lines Rubber-band brush lines  
 Curves Continuous strokes while holding button 1 or 2  
 Line Style → Solid, dotted, dashed, etc. lines

Entries in the **Line Style** sub-submenu are strings of As, Bs and dot that describe dotted and dashed lines. A stands for the brush on the button pushed, B stands for the brush on the other button; . for no brush at all. The string is cycled through at successive points when drawing Lines, Curves, or Circles. Thus A means a solid line, A. . . means a 1 in 4 dotted line, and AAAA. . . . means 4-pixel dashes.

The **Operation** submenu assigns a pair of operations for buttons 1 and 2. A hidden ‘mask’ plane describes the shape that has been painted; black pixels in the mask are inside, green outside. Likewise, the brush consists of a pair of rectangular image and mask planes. There are 11 effective operations to combine the part of the brush inside its mask with the part of the picture it sits on (see the Porter/Duff paper for details); selected pairs can be assigned to the buttons:

Above/Erase Button 1 paints on top, Button 2 erases  
 Below/Erase Button 1 paints behind, Button 2 erases  
 Above/Below Button 1 paints on top, Button 2 behind  
 Inside/Erase Button 1 paints inside, Button 2 erases  
 Brush/Clear Special effects  
 AoutB/AinB Special effects  
 BinA/BatopASpecial effects  
 Xor/Above Special effects

**Above** paints on top of the picture, as in ‘normal’ paint programs.

**Below** paints underneath—only in places that were not previously covered.

**Inside** paints on top, but only inside the already-painted part.

The other 7 operations are best described as ‘special effects’. Try them out to see what they do, or look at the Porter/Duff paper.

Texture facilities paint with a repeating 16×16 pattern instead of copies of a brush. The **Texture** submenu contains:

Texture Turn texturing on or off



Make	Pick a texture from the picture
Negate	Reverse the texture's green and black
Save	Name a texture and copy it into a file
Library	→ List and retrieve textures in library
Get	Type a name and get a texture from a file

**Make** gives a 16×16 square cursor with which to pick a texture.

The **Brush** submenu has the same items for brushes. **Make** allows you to sweep out a region to use as a brush.

The **Canvas** submenu contains **Negate**, **Save**, **Library**, and **Get**, in this case pertaining to entire pictures. A library picture is saved in a file containing the image plane then the mask plane in *bitfile(9)* format.

The file **.paintstate** in the current directory remembers the names of the current brush, texture, and libraries between sessions. The **State** submenu displays the library names at the bottom of the layer, where they can be edited:

Brushes	Name the brush directory
Pictures	Name the picture directory
Textures	Name the texture directory

The **Fill** menu item gives an arrowhead cursor. If you touch down with button 3 at a point not painted, the rookwise-connected region containing it will fill with black. On completion, the black will be replaced by the current texture. While the region is filling, any button click aborts the operation.

The current selections from the **Brush Library**, **Style**, **Operation**, **Texture**, and **Line Style** menus are marked with a \*, and are displayed in the information box at the bottom of the layer.

## FILES

/usr/jerq/lib/paint/brush  
the default brush library

/usr/jerq/lib/paint/tex  
the default texture library

.paintstate  
state of terminated program

## SEE ALSO

*mbits(6)*, *bitfile(9)* *brush(9)* *cip(9)* *ped(9)*

Thomas Porter and Tom Duff, 'Compositing Digital Images,' Siggraph '84 Proceedings

**NAME**

ped, tped – picture editor

**SYNOPSIS**

**ped** [ -f ] [ *file* ... ]

**tped** [ *option* ... ] [ *file* ... ]

**DESCRIPTION**

*Ped* is an interactive drawing program for 5620 terminals. A *file* argument is equivalent to an *e* command as described below. Most features of *ped* are menu-controlled and self-explanatory; further details are in the reference.

Button 1 selects actions from a permanent menu and to draw or pick up an object. Button 3 terminates drawing actions or changes the permanent menu. Button 2 causes the permanent menu to revert to *basic*.

The operation of *ped* is split between host and terminal. When a file is first read, it is kept on the host; *bring* gets it to the 5620.

Option **-f** causes *ped* to display all text in one size to save time and space.

Some of the actions on permanent menus (switched by button 3) are described below. The last action is usually remembered and may be executed repeatedly until another is selected. Thus, for example, one can fill many polygons with one button click per polygon. Actions marked ( *t* ) in the menu toggle on and off.

**basic menu****blitblt**

Copy part of the screen to file see [blitblt\(9\)](#)

**exit**

Leave *ped*, requires a confirming push of button 3.

**markers**

Make visible the defining points of objects; these are the only points sensitive to selection by button 1.

**type comm**

Take input from the keyboard.

**e file** Begin editing *file*, remember its name, as in [ed\(1\)](#). Commands **f** (file name), **r** (read), **w** (write), behave similarly.

**qq**

Same as **exit** in **basic** menu; altered files will be saved in

**cd**

Change working directory.

**pwd**

Print working directory.

## newline

Reactivate mouse.

**u string**

Remember *string* as a shell command for the selection user in menu *refine*.

**ch size**

**rotate** displays a vector from the center (of the bounding box) of an object to the selected point. The object is rotated and scaled to bring that point to a second selected position.

**h-elong**

Change aspect ratio. The inverse is **v-elong**.

**move** Button 3 cancels a move or copy. To help untangle overlapping objects, the cancellation does not take place until returning to the basic menu.

**attach**

Move an open polygon (a broken line) and hook it to the end of another.

**join**

Connect the ends of two polygons with a new line.

**link**

Cause multiple polygons to move and be filled as one (useful for making holes). Linked polygons must all be open or all be closed.

**match**

Move objects to bring selected points together.

**center**

Move objects to bring their centers together.

**family/pt**

Select objects to be moved or deleted together.

**draw** Button 1 fixes a point; button 3 terminates an object.

**text** Type one or more lines terminated by an empty line.

**grid** Snap points to locations on a grid, which indexes through settings FMC (fine, medium, coarse, none).

**fix sz** Set option **-f**.

**family/bx**

Sweep a box around objects to be moved or deleted together.

**reshape****formal**

Adjust nearly rectangular lines to be perfectly so.

**spline** A piecewise parabolic fit tangent to the midpoints of a broken line.

**corner**

Make a guiding point of a spline to be multiple – a corner in an otherwise smooth curve.

**refine**

**adj t** Left-justify, right-justify, or center text.

**edit text**

Display text at the top, where button 1 selects a position for inserting by typing or deleting by backspacing. Button 3 concludes the editing.

**shade** Assign textures for filling polygons, circles, or spline-bounded regions. Curves are filled schematically on the 5620, but accurately on the host.

**color** Assign colors for display on other devices.

**remote**

Perform all editing on the host using the terminal as a display device only.

*Tped* converts files of graphic information produced by *ped* into typesetting requests for *troff(1)*. The options are:

**-Tdev** Prepare output for particular devices known to *troff*: **-Taps** or **-T202**.

**-b** Place a box around each picture.

The input may be straight *ped* output or may be arbitrary text files with *ped* output embedded between pairs of delimiting lines:

**.GS** [ *size* ] *ped file* . . . **.GE**

or in another file:

**.GS** [ *size* ] *pedfilename*

The optional size gives width or height: **w=inches** or **h=inches**.

**FILES**

*.pederr*

*ped.save*

**SEE ALSO**

*cip(9)* *paint(9)* *brush(9)* *graphdraw(9)* *pic(1)*, *ideal(1)*, *blitblt(9)*

T. Pavlidis, 'PED Users Manual', this manual, Volume 2

**DIAGNOSTICS**

Error messages from the host are placed in file **.pederr**.

**BUGS**

Pictures may spill into the menu or message areas.

Some experimentation with *tped* printout parameters may be needed to obtain satisfactory results.

**NAME**

pengo – squash the sno-bees

**SYNOPSIS**

**demo pengo**

**DESCRIPTION**

*Pengo* plays the video game. Any button replaces the penguin picture by the game.

The mouse controls the movement of the penguin. (The usual hjkl keys also move the penguin, with the space bar stopping movement.)

Button 1: Stop the penguin at the next block boundary.

Button 2: Push (or break) a block, or splash the water boundary.

Button 3: Display a menu to control aspects of the game.

The penguin moves in one direction at constant speed unless acted upon by an outside force: moving the mouse or encountering a wall or border. If button 2 is pressed when a block is encountered then the block is pushed. If another block (or a wall) is behind the first then the block will shatter, scoring 30 points. Similarly breaking an egg scores 500. An unobstructed block will slide until it hits an obstacle, sweeping along any sno-bees in its path and crushing them. Getting one sno-bee with a block scores 400 points, two 1600, three 3200, four 6400.

Lining up the three blocks that bear crosses is worth 5000 points if they are lined against a wall, 10000 otherwise. Bonus penguins are given out every so often.

Pushing on the border causes ripples to propagate along it, stunning any sno-bees that are touching it. A penguin may crush a stunned sno-bee underfoot for 100 points.

Button 3 gets a menu with entries **Pause, Stats, New Game, Quit**. All require another click of button 3 to complete. **Stats** presents three sliders controlled by button 1:

**C** Change ( of time that the sno-bees change direction)

**R** Random ( of time that a random direction is chosen when changing)

**B** Break of time that a sno-bee will break a block that is blocking its way).

**NAME**

pi, 3pi – process inspector

**SYNOPSIS**

**pi** [ **-t** *corefile objectfile* ]

**3pi** [ **-p** *person* ]

**DESCRIPTION**

*Pi* is a C debugger that is bound dynamically to multiple subject processes or core dumps. It works better for programs compiled *cc -g*. *Pi* uses the *Pads(9)* multi-window user interface. There are three types of windows: debugger control windows, which access the global state of the debugger; process control windows (exactly one per process), which start and stop processes and connect to process-specific functions; and process inspection windows, which include viewers for source text and memory, formatted various ways.

The most important debugger control window is the *pi* window itself. Each line within the *pi* window refers to a specific process. These lines may be introduced to the window by running *ps(1)* from the button 3 menu; by typing a file name, either a *proc(4)* name, or the name of a core image followed by the name of the binary that created the core; or by typing a command, prefixed by an exclamation !, to be executed as a child of *pi*. There are several ways to access a process (using the button 2 menu), each of which generates a process control window:

**open process**

Attach to a running process, often one started with *hang(1)*.

**open core**

Attach to a core image.

**open child**

Attach to a process forked by a process being debugged by the current *pi*.

**take over**

Rebind an existing process window hierarchy (pointed to with the mouse) to the named process, which must be an instance of the identical program.

**hang . open proc**

Execute a command afresh, beginning it in the stopped state, and redirecting IO to

**hang . take over**

Same, also binding to an existing process window.

The process window indicates the process's state, shows the call stack traceback and connects to windows that access source text, local variables within a stack frame, raw memory, and so on. These windows are cross-connected, so, for example, an instruction in a process's assembly language window can be inspected in hexadecimal in the raw memory window. Closing the process control window closes all the windows under it.

The following menu functions are provided by the various window types in *pi*. Initially there are these windows available:

**Help** Reminder of user interface mechanics.

**Pi** Overall control of processes, core dumps and programs. A process is identified by its pathname or command line. Process symbols are found in the executable file from which the process was loaded, but may be overridden. Symbols for core dumps must be supplied explicitly, after the core filename. **Synopsis:** Identify and open process or core dump; run a program as *Pi's* child; take over a process with the debugging environment of a different one.

**Pwd/cd**

change the working directory of the debugger.

**Process Window**

Selecting and opening a process from the Pi window creates a new window with overall control of that process. It shows the process state, and a traceback if the process is halted or dead. States are:

**ACTIVE**

running normally

**HALTED**

halted asynchronously by a debugger

**BREAKPOINT**

halted on reaching breakpoint

**STMT STEPPED**

halted after executing C source statement(s)

**INSTR STEPPED**

halted after executing machine instruction(s)

**EVENT PENDING**

halted about to receive a signal being traced

**ERROR STATE**

the process has probably exited

The menu operations on the process are:

**go** let the process run

**stop** stop the process

**kill** send **SIGKILL** to the process; see [signal\(2\)](#)

**src text**

open source text window(s)

**Signals**

open window for sending and trapping signals

**Globals**

open window for evaluating expression in global scope

**RawMemory**

open window for editing uninterpreted memory

**Assembler**

open window for disassembler

Each line of the call stack traceback describes one function. Each function in the traceback can open an expression evaluator window or display its current source line.

**Globals and Stack Frame Windows**

These windows evaluate expressions with respect to global scope, and scope in a function, respectively. A stack frame window is opened from a line in the call stack traceback or from a line of source text. A stack frame can find its active source line in a source window or the stack frame window of its caller.

C expressions can be entered by the keyboard or mouse. The unary operators *fabs* and *sizeof* are supported; the only assignment operator is =. Functions from the user program may be called. New expressions can be derived from old ones by the keyboard or mouse. In menus and the keyboard, \$ means the expression in the current line. The VAX registers are called **\$r0** to **\$r15**; the address of a register is the location at which it was saved. The format in which values are displayed can be changed. The raw memory editor may be entered using an expression's value as address.

An expression may be made a *spy*. The value of a spy expression is evaluated and displayed each time the debugger looks at the process. If the value of a spy changes the process is halted at the next instruction, statement or breakpoint.

The comma operator is useful in conditional breakpoints because the values of its subexpressions are displayed. E.g. `x, y, x==y` traces the values of `x` and `y` when the condition fails; `x, y, 0` just traces.

To cross scope boundaries, the environment (a function identifier) in which an expression is to be evaluated may be specified as: `{ expr } function`. From the source directory window, file static variables can be promoted to appear in the menu of global variables.

**Source Text Windows**

The source file directory window lists all the source files, if there are two or more. A textual prefix, entered from the keyboard, points to a source directory, without changing the working directory. Each source file is in a separate window, opened when needed. The source file's pathname as given to *cc* can be overridden from the keyboard. If things go wrong, use **reopen** to open the file afresh. **Synopsis:**

set/clear (conditional) breakpoint; single-step source statements; step into (rather than over) a function; go the process; show the statement for the current PC; open a stack frame window for a source line's function; evaluate expression; disassemble first instruction of source statement; context search for string.

### Breakpoints Window

Lists all the active source and assembler breakpoints and related errors. **Synopsis:** show source or assembler for a breakpoint; clear breakpoint; clear all breakpoints.

### Signals Window

Lists all signal types, showing which ones are traced. **Synopsis:** Change which signals are traced; send a signal to the subject process; clear pending signal; stop process on *exec*.

### Raw Memory Window

In this window memory is viewed as a sequence of 1-, 2-, 4- or 8-byte cells. **Synopsis:** set cell address; change cell size; change display format; display cells above and below; indirect to cell; change cell value; spy on memory cell; disassemble instruction at cell.

### (Dis)assembler Window

In this window memory is viewed as a sequence of instructions. **Synopsis:** set instruction address; display more instructions; change display format; display instruction as cell in raw memory window; set/clear breakpoint on instruction; open stack frame window for instruction's function; display instruction at current PC; single step instruction(s); step over a function call instead of into the function.

### Exec and Fork

If a process controlled by *pi* does an *exec()* and an *exec* break is set in the Signals window, the process is suspended as if started by *hang(1)*. To debug the process after the *exec*, close the original process window and re-open it. When re-opened it will get the new symbol tables.

To debug a child process: (i) set a breakpoint in code that will be executed in the child after the fork; (ii) execute the fork *at full speed* (the child inherits the parent's breakpoints, which aren't there if the parent is stepped); (iii) *before altering any breakpoints*, get a fresh *ps* in the Pi window and apply **open child** to the child. The child should be stopped on the inherited breakpoint, but it and all other breakpoints should have been cleared.

### Kernel

The state of kernel variables associated with a process may be examined by giving their name or virtual address. The **UNIX** environment variable specifies the file from which the system was loaded; the default is */unix*. Kernel dumps may be examined by opening the 'kernel pi' window.

### Just A Traceback

With the **-t** option *pi* writes a traceback on its standard output and quits.

### 3pi

*3pi* is a variant of *pi* for debugging 5620 programs running under *mux(9)* It creates two terminal processes: one for its access to terminal memory and graphics and a second for its *Pads(9)* interface.

### Remote Debugging

With the **-p** option *3pi* loads its first process, but not *Pads*. Instead, it mails a *3pi* command to *person*, to be executed on any host in the local network. That *3pi* command loads *Pads* on *person's* terminal, and connects to the originator's terminal. If separate hosts are involved and the versions of critical files differ, be careful with pathnames.

### 3pi Graphics

Points, rectangles, textures and bitmaps can be displayed graphically.

### 3pi - pi

Most differences come from obvious differences in the hardware and software architectures. Also, in *3pi* function calls are executed by a debugger process on its own call stack.

## SEE ALSO

T. A. Cargill, 'The Feel of Pi', this manual, Volume 2  
*hang(1)*, *proc(4)*, *adb(1)*, *cin(1)*, *nm(1)*, *pads(9)*



## **BUGS**

In switch statements there is no boundary between the last case and the branch code; the program *appears* to jump to the last case (but is really in the branch) and then to the real case.

A changed spy only stops the process at a breakpoint or while stepping. An expression can be cast only by menu.

Functions may only be called when the process is stopped and not in a system call.

**NAME**

proof – troff output interpreter for 5620

**SYNOPSIS**

**proof** [ **-f** *fonts* ] [ *file* ]

**DESCRIPTION**

*Proof* reads [troff\(1\)](#) intermediate language from *file* or standard input and simulates the resulting pages on the screen. If no file name is given and standard input is a terminal, proof terminates immediately leaving a ‘proof layer’. By invoking *proof* in a proof layer you can avoid download time.

Fonts are loaded as required. The usual [mux\(9\)](#) font, **defont**, is used for unknown fonts. Option **-f** preloads fonts. Names are given relative to `/usr/jerq/font` and are separated by commas. The most-used fonts are `-fR.10,I.10,B.10,S.10`.

After a layer’s worth of text is displayed, *proof* pauses for a command from keyboard or mouse button 3. The typed versions of commands are:

newline

Go on to next portion of text. (Button 3 equivalent: more.)

**q** Quit, leaving a proof layer.

**x** Exit and restart the regular terminal program. (Equivalent to `q` followed by `term mux`; see [term\(9\)](#))

**pn** Print page *n*. An out-of-bounds page number means the end nearer to that number; a missing number means page 0; a signed number means an offset to the current page.

Button 1 gets a scroll box, which represents a full page of text. An interior rectangle shows what part of the page is now visible. The interior rectangle moves with the mouse, causing the layer to scroll both vertically and horizontally. Button 2 gets a speedometer. The bar of the speedometer moves with the mouse to control the rate at which new information is displayed.

**EXAMPLES**

`troff -ms memo | proof` Format a memo and display it.

`(eqn memo | troff -ms) 2>diags | proof` Display a memo with equations. Avoid sending diagnostics to the screen; see [BUGS](#).

**FILES**

`/usr/jerq/font/*`  
fonts

`/usr/jerq/font/.missing`  
list of referenced but unconverted fonts

**SEE ALSO**

[lp\(1\)](#), [font\(6\)](#), [reader\(9\)](#) [psi\(9\)](#)

Brian W. Kernighan, *A Typesetter-independent Troff*

**BUGS**

*Proof* breaks if other messages are directed to its layer. In particular, unredirected *troff* diagnostics will break the pipeline `troff | proof`.

Windowing can get confused if the *troff* output is not approximately sorted in ascending y-order.

A proof layer imitates `term 33`, not `mux`. Among other difficulties, it will not be reusable if downloaded across the network.

**NAME**

psi – postscript interpreter

**SYNOPSIS**

psi [ *option ...* ] [ *file* ]

**DESCRIPTION**

*Psi* reads Postscript input from *file* or from standard input and simulates the resulting pages in a *mux(9)* layer. The program remains in the layer at exit; further invocations of *psi* in that layer avoid download time.

The options are

- pn** Display page *n*, where *n* is determined from the **Page** comments in the file. If these are not present, page selection will not work.
- R** Pages in the file are in reverse order. This flag must be used on such files for the *-p* option to work.
- r** Display the image at full scale, with the bottom left corner positioned at the bottom left corner of the window. (By default, the image is scaled to fit the window, maintaining the aspect ratio of a printer.)
- a x y** Display the image at full scale with position *x,y* of the image placed at the bottom left corner of the window.

*Psi* works on either a Teletype 5620, 630 or 730 terminal as determined by the environment variable **TERM**.

Fonts are implemented with size-24 bitmap fonts. Those available are Symbol, Courier, Times-Roman, Times-Italic, Times-Bold, Times-BoldItalic, Helvetica, Helvetica-Oblique, Helvetica-Bold, Helvetica-BoldOblique. Fonts Courier-Bold, Courier-Oblique, and Courier-BoldOblique are mapped to Courier. Other postscript fonts, including type1, may be used if they are supplied before they're referenced.

When the 'cherries' icon is displayed, use mouse button 3 to move forward (**more**), to a particular page (**page**), or quit (**done**). Button 2 exits the program completely.

**EXAMPLES**

```
troff -ms memo | lp -dstout -H | psi
troff -ms memo | dpost | psi
```

Two equivalent ways to format a memo, convert it to PostScript, and display it.

For best results with TeX documents, use **dvips** with the **-Tjrq**, **-Tgnot**, or **-D 100** option to get fonts of the proper resolution and run *psi* with the *-r* or *-a* flag to prevent *psi* from scaling.

**FILES**

psi.err  
error messages

**SEE ALSO**

*lp(1)*, *dvips(1)*, *postscript(8)*, *proof(9)* *psifile(1)*, *psix(1)*

**DIAGNOSTICS**

A 'dead mouse' icon signals an error; error comments are placed on file

Symbols that lack bitmaps are replaced by '?' and an error is reported.

**BUGS**

A *psi* layer imitates `term 33`, not *mux*. Among other difficulties, it will not be reusable if downloaded across the network.

Unimplemented PostScript features are rotated images and half tone screens. Imagemasks may only be rotated by multiples of 90 degrees, not by arbitrary angles.

Skipping pages may cause operators to be undefined.

**NAME**

reader – electronic retrieval of typeset documents

**SYNOPSIS**

**reader** *name*

**DESCRIPTION**

*Reader* presents the named paper on a 5620 terminal in a form designed for readability, not for similarity to the printed version. The *name* is a pathname for a manuscript in the *papers(7)* database with any final *.d* elided) or the name of a *troff(1)* input file. Mouse button 1 selects subheads; button 3 moves forward ('more') or backward ('less'). The program exits completely on button 2, or tentatively (to avoid downloading upon reexecution) on button 3 ('done').

When the text in a screen overlaps text in a previous screen, a tick mark in the bar (not a scroll bar) at the left of the screen shows where new material begins.

Fully installed papers in the database, which appear as directories suffixed *.d*, have been preprocessed so that *reader* can present figures and complex equations. In *troff* input, it understands straightforward text and *eqn(1)*, the macro packages *ms(6)*, *mm*, and *me*, but cannot handle arbitrary motions such as appear in figures and complex equations.

**FILES**

*/n/bowell/pap/Titles*  
titles, authors and installation dates

*/n/bowell/pap/\*org*  
membership list

*/n/bowell/pap/center/department/author/papername[d]*

**SEE ALSO**

*troff(1)*, *proof(9)* *docsubmit(1)*, *papers(7)*

**BUGS**

Button 1 knows only already-read subheads unless the paper has been preprocessed.

*Reader* can only handle papers written in *troff* with standard (**-ms**, **-mm**, **-me**) macro packages.

**NAME**

rebecca – graphics touch-up editor

**SYNOPSIS**

*rebecca file*

**DESCRIPTION**

*Rebecca* is an interactive retouching tool for digitized grey-scale images. The *file* must be a headerless 512×512 black-and-white digitized image. Example (read only) files are in directory

‘Floating instruments’ for editing can be dragged with button 2 to different locations.

*Resolution.* The tick mark on the long bar can be moved up or down with button 1. Printed to the right of the bar is the current resolution— a power of 2 representing the number of file pixels across the screen image.

*Grid.* Click button 1 at the circular button to toggle the grid. Turning on the grid is useful sometimes to see how fast a screen update is proceeding: it eats away the grid.

*Write.* Write the file on the host by clicking button 1 at the box labeled *write*. The write box has a \* if a change was made to the file since it was last written.

*Runlength encoding.* Clicking button 1 at this box toggles the mode of data transmission between host and terminal.

*Reopen.* This instrument cancels any changes made to the file since the last time it was written.

*Move/Pan.* Click button 1 at one of the 5 areas of the diamond. The middle resets the display to a full size picture. Left, right, up, or down will move (pan) 1/4 screen in the corresponding direction (useful only on zoomed pictures).

*Zoom/Unzoom.* Click button 1 at Z (zoom) or U (unzoom). Z prompts with a square box to be positioned on the area of the picture to be inspected at full resolution. If you click button 1 before you confirm, the sides of the box are halved. Clicking button 2 doubles them. Any combination of two buttons cancels the zoom; button 3 confirms it.

*Paint.* Click button 1 at the box labeled + = -. Painting with + adds grey values to pixels; = assigns values; - subtracts values. Click button 1 at a pixel location to apply the paint. Click button 3 to sweep a rectangle to paint all pixels within it. Pick a paint value (default is white) by clicking button 2 at the grey scale at the bottom or at any pixel in the image. Click button 2 at the paint box to cancel the paint mode.

*Smear.* Pointing at a pixel with both buttons 1 and 2 down averages it with its 8 neighbors (most useful when zoomed in to pixel level). Typical usage: apply some white or black paint with the paint box, then smear it.

*Probe.* Click button 1 at the probe box P:. Point at a pixel in the image. The *x-y* coordinates and the greyscale value of the pixel will be printed.

*Contrast.* Move the ends of the line under the grey scale bar to expand or compress the grey scale.

*Rubber Sheet.* The box named *sheet* prompts for a rectangle. Sweep out the rectangle over an area you want to manipulate, then reposition the corners by dragging them to new locations with button 1. Confirm the selection with button 3. Other instruments are usable while the update proceeds.

**SEE ALSO**

[pico\(1\)](#), [flicks\(9\)](#) [picfile\(5\)](#), [flickfile\(9\)](#)

**NAME**

request, own, wait, alarm, sleep, nap, kbdchar, rcvchar, realtime, sendchar, sendnchars, kill, exit – 5620  
input/output requests

**SYNOPSIS**

```
#include <jerq.h>

void request(r) int r;

int own(r) int r;

int wait(r) int r;

void alarm(t) unsigned t;

void sleep(t) unsigned t;

void nap(t) unsigned t;

long realtime();

int kbdchar();

int rcvchar();

void sendchar(c) int c;

void sendnchars(n, cp) int n; char *cp;

void kill(s) int s;

void exit();
```

**DESCRIPTION**

*Request* announces a program's intent to use I/O devices and resources, and is usually called once early in a program. The bit vector *r* indicates which resources are to be used by OR'ing together one or more of the elements **KBD** (keyboard), **MOUSE**, **RCV** (characters received by terminal from Unix), **SEND** (characters sent from terminal to Unix) and **ALARM**. For example, **request(MOUSE|KBD)** indicates that the process wants to use the mouse and keyboard. If the keyboard is not requested, characters typed will be sent to the standard input of the Unix process. If the mouse is not requested, mouse events in the process's layer will be interpreted by the system rather than passed to the process. **SEND** and **CPU** (see **wait** below) are always implicitly requested. *Request* sleeps for one clock tick to synchronize mouse control with the kernel.

*Own* returns a bit vector of which I/O resources have data available. For example, **own().KBD** indicates whether a character is available to be read by *kbdchar* (see below), **own().MOUSE** tells if the process's **mouse** structure (see [button\(9\)](#)) is current, and **own().ALARM** indicates whether the alarm timer has fired.

*Wait*'s argument *r* is a bit vector composed as for *request*. *Wait* suspends the process, enabling others, until at least one of the requested resources is available. The return value is a bit vector indicating which of the requested resources are available — the same as **own().r**.

Processes wishing to give up the processor to enable other processes to run may call **wait(CPU)**; it will return as soon as all other active processes have had a chance to run. **CPU** is a fake resource which is always requested. The **SEND** pseudo-resource is unused; **wait(SEND)** always succeeds.

*Alarm* starts a timer which will fire *t* ticks (60ths of a second) into the future. A pseudo-resource **ALARM** can be used to check the status of the timer with *own* or *wait*. Calling *alarm* implicitly requests the **ALARM** pseudo-resource.

*Nap* busy loops for *t* ticks of the 60Hz internal clock. To avoid beating with the display, programs drawing rapidly changing scenes should *nap* for two ticks between updates, to synchronize the display and memory. *Nap* busy loops until the time is up; *sleep* is identical except that it gives up the processor for the interval. Except when unwilling to give up the mouse, a program should call *sleep* in preference to *nap*. *Sleep* does not interfere with *alarm*, and vice versa.

*Realtime* returns the number of 60Hz clock ticks since *mux* started.

*Kbdchar* returns the next keyboard character typed to the process. If no characters have been typed, or **KBD** has not been *requested*, *kbdchar* returns **-1**.

*Rcvchar* returns the next character received from the host, typically written on the standard output of a Unix process. If there are no characters available, or **RCV** has not been *requested*, *rcvchar* returns `-1`.

*Sendchar* sends a single byte to the host, which will normally be read on the standard input of the Unix process. *Sendnchars* sends to the host *n* characters pointed to by *p*.

*Kill* sends the associated Unix process the signal *s*; see [signal\(2\)](#).

*Exit* terminates the process. Unlike on Unix, *exit* does not return an exit status to a parent. Calling *exit* replaces the running process by the default terminal program. Any associated Unix process must arrange for its own demise; *exit* is a purely local function. When a process calls *exit*, all local resources: keyboard, mouse, storage, etc., are deallocated automatically.

*Realtime* returns the number of sixtieths of a second elapsed since [mux\(9\)](#) was started.

## EXAMPLES

```
request(KBD|RCV);
for(;;){
    r=wait(KBD|RCV);
    if(r.KBD)
        keyboard(kbdchar());
    if(r.RCV)
        receive(rcvchar());
}
```

Take input from either the keyboard or the host.

## SEE ALSO

[button\(9\)](#)

## BUGS

**own().MOUSE** does not guarantee that you own the mouse. The correct test is  
`(own().MOUSE) .. ptirect(mouse.xy, Drect)`

**NAME**

ruler – measure things on the screen

**SYNOPSIS**

**ruler**

**DESCRIPTION**

*Ruler* measures things on a *mux(9)* screen. Press button 1 to sweep out a rectangle anywhere on the screen. For each rectangle swept, *ruler* displays the coordinates of the rectangle's corners (labeled **down** and **up**), the size of the rectangle and length of its diagonal.

There is a menu on button 3. The **pixels** and **chars** items control whether the size and diagonal are measured in units of pixels or characters; **stop** deactivates *ruler* without exiting; **measure** reactivates *ruler*.

**BUGS**

Character units are arbitrarily defined as the width and height of a 0 in the *ruler* layer. This may have nothing to do with character sizes in other layers.

Ruler's menu must pop up in its own layer, perhaps far away from the cursor.



**NAME**

sam – screen editor with structural regular expressions

**SYNOPSIS**

**sam** [ *option ...* ] [ *files* ]

*sam -r machine*

**sam.save**

**DESCRIPTION**

*Sam* is a multi-file editor. It modifies a local copy of a Unix file. The copy is here called a *file*; a Unix file is distinguished by the trademarked adjective. The files are listed in a menu available through mouse button 3 or the **n** command. Each file has an associated name, usually the name of the Unix file from which it was read, and a ‘modified’ bit that indicates whether the editor’s file agrees with the Unix file. The Unix file is not read into the editor’s file until it first becomes the current file—that to which editing commands apply—whereupon its menu entry is printed. The options are

**-d** Do not download the terminal part of *sam*. Editing will be done with the command language only, as in [ed\(1\)](#).

**-r machine**

Run the host part remotely on the specified machine, the terminal part locally. This extends graphic editing to files on machines that don’t ordinarily support it or across non-*nfs*(8) connections.

**Regular expressions**

Regular expressions are as in *egrep* (see [gre\(1\)](#)), with the addition of **@** and **\n**. A regular expression may never contain a literal newline character. The elements of regular expressions are:

**.** Match any character except newline.

**\n** Match newline.

**\x** For any character except **n** match the character (here **x**).

**@** Match any character.

**[abc]** Match any character in the square brackets. **\n** may be mentioned.

**[^abc]** Match any character not in the square brackets, but never a newline. Both these forms accept a range of ASCII characters indicated by a dash, as in **a-z**.

**^** Match the null string immediately after a newline.

**\$** Match the null string immediately before a newline.

Any other character except newline matches itself.

In the following, *r1* and *r2* are regular expressions.

**(r1)** Match what *r1* matches.

**r1|r2** Match what *r1* or what *r2* matches.

**r1\*** Match zero or more adjacent matches of *r1*.

**r1+** Match one or more adjacent matches of *r1*.

**r1?** Match zero or one matches of *r1*.

The operators **\***, **+** and **?** are highest precedence, then catenation, then **|** is lowest. The empty regular expression stands for the last complete expression encountered. A regular expression in *sam* matches the longest leftmost substring formally matched by the expression. Searching in the reverse direction is equivalent to searching backwards with the catenation operations reversed in the expression.

**Addresses**

An address identifies a substring in a file. In the following, ‘character *n*’ means the null string after the *n*-th character in the file, with 1 the first character in the file. ‘Line *n*’ means the *n*-th match, starting at the beginning of the file, of the regular expression **.\*\n?** (The peculiar properties of a last line without a newline are temporarily undefined.) All files always have a current substring, called dot, that is the

default address.

### Simple Addresses

**#*n*** The empty string after character *n*; **#0** is the beginning of the file.

***n*** Line *n*.

**/*regex***

**?*regex*?**

The substring that matches the regular expression, found by looking toward the end (*/*) or beginning (*?*) of the file, and if necessary continuing the search from the other end to the starting point of the search. The matched substring may straddle the starting point.

**0** The string before the first full line. This is not necessarily the null string; see + and - below.

**\$** The null string at the end of the file.

**.** Dot.

**'** The mark in the file (see the **k** command below).

**"*regex*"**

Preceding a simple address (default **.**), refers to the address evaluated in the unique file whose menu line matches the regular expression.

### Compound Addresses

In the following, *a1* and *a2* are addresses.

***a1+a2***

The address *a2* evaluated starting at the end of *a1*.

***a1-a2*** The address *a2* evaluated looking in the reverse direction starting at the beginning of *a1*.

***a1,a2*** The substring from the beginning of *a1* to the end of *a2*. If *a1* is missing, **0** is substituted. If *a2* is missing, **\$** is substituted.

***a1;a2*** Like *a1,a2*, but with *a2* evaluated at the end of, and dot set to, *a1*.

The operators + and - are high precedence, while , and ; are low precedence.

In both + and - forms, if *a2* is a line or character address with a missing number, the number defaults to 1. If *a1* is missing, **.** is substituted. If both *a1* and *a2* are present and distinguishable, + may be elided. *a2* may be a regular expression; if it is delimited by *?*'s, the effect of the + or - is reversed.

It is an error for a compound address to represent a malformed substring. Some useful idioms: *a1+* (*a1-+*) selects the line containing the end (beginning) of *a1*. **0/*regex*** locates the first match of the expression in the file. (The form **0//** sets dot unnecessarily.) ***Jregex*///** finds the second following occurrence of the expression, and ***./regex*** extends dot.

### Commands

In the following, text demarcated by slashes represents text delimited by any printable ASCII character except alphanumerics. Any number of trailing delimiters may be elided, with multiple elisions then representing null strings, but the first delimiter must always be present. In any delimited text, newline may not appear literally; **\n** may be typed for newline; and **\** quotes the delimiter, here **/**. Backslash is otherwise interpreted literally, except in **s** commands.

Most commands may be prefixed by an address to indicate their range of operation. Those that may not are marked with a \* below. If a command takes an address and none is supplied, dot is used. The sole exception is the **w** command, which defaults to **0,\$**. In the description, 'range' is used to represent whatever address is supplied. Many commands set the value of dot as a side effect. If so, it is always set to the 'result' of the change: the empty string for a deletion, the new text for an insertion, etc. (but see the **s** and **e** commands).

**Text commands****a***text/*

or

**a***lines of text***.** Insert the text into the file after the range. Set dot.**c****i** Same as **a**, but **c** replaces the text, while **i** inserts *before* the range.**d** Delete the text in the range. Set dot.**s***regex/text/*

Substitute *text* for the first match to the regular expression in the range. Set dot to the modified range. In *text* the character **.** stands for the string that matched the expression. Backslash behaves as usual unless followed by a digit: **\d** stands for the string that matched the subexpression begun by the *d*-th left parenthesis. If *s* is followed immediately by a number *n*, as in **s2/x/y/**, the *n*-th match in the range is substituted. If the command is followed by a **g**, as in **s/x/y/g**, all matches in the range are substituted.

**m** *al***t** *al* Move the range to after *al* (**m**), or copy it (**t**). Set dot.**Display commands****p** Print the text in the range. Set dot.**=** Print the line address and character address of the range.**=#** Print just the character address of the range.**File commands**

In these commands a *file-list* may be expressed *<Unix-command* in which case the file names are taken as words (in the shell sense) generated by the Unix command.

**\* b** *file-list*Set the current file to the first file named in the list that *sam* also has in its menu.**\* B** *file-list*Same as **b**, except that file names not in the menu are entered there, and all file names in the list are examined.**\* n** Print a menu of files. The format is:

' or blank indicating the file is modified or clean,

indicating the the file is unread or has been read (in the terminal, \* means more than one window is open),

indicating the current file,

a blank,

and the file name.

**\* D** *file-list*Delete the named files from the menu. If no files are named, the current file is deleted. It is an error to **D** a modified file, but a subsequent **D** will delete such a file.**I/O Commands****\* e** *filename*

Replace the file by the contents of the named Unix file. Set dot to the beginning of the file.

**r** *filename*

Replace the text in the range by the contents of the named Unix file. Set dot.

**w** *filename*Write the range (default **0,\$**) to the named Unix file.**\* f** *filename*

Set the file name and print the resulting menu entry.

If the file name is absent from any of these, the current file name is used. **e** always sets the file name, **r** and **w** do so if the file has no name.

**<** *Unix-command*

Replace the range by the standard output of the Unix command.

**>** *Unix-command*

Sends the range to the standard input of the Unix command.

| *Unix-command*

Send the range to the standard input, and replace it by the standard output, of the Unix command.

\* ! *Unix-command*

Run the Unix command.

\* **cd** *directory*

Change working directory. If no directory is specified, **\$HOME** is used.

In any of <, >, | or !, if the *Unix command* is omitted the last *Unix command* (of any type) is substituted. If *sam* is downloaded, ! sets standard input to and otherwise unassigned output (**stdout** for ! and >, **stderr** for all) is placed in **\$HOME/sam.err** and the first few lines are printed.

**Loops and Conditionals***x/regexpl command*

For each match of the regular expression in the range, run the command with dot set to the match. Set dot to the last match. If the regular expression and its slashes are omitted, /. \* $\backslash$ n/ is assumed. Null string matches potentially occur before every character of the range and at the end of the range.

*y/regexpl command*

Like **x**, but run the command for each substring that lies before, between, or after the matches that would be generated by **x**. There is no default behavior. Null substrings potentially occur before every character in the range.

\* **X/regexpl command**

For each file whose menu entry matches the regular expression, run the command. If the expression is omitted, the command is run in every file.

\* **Y/regexpl command**

Same as **X**, but for files that do not match the regular expression, and the expression is required.

*g/regexpl command**v/regexpl command*

If the range contains (**g**) or does not contain (**v**) a match for the expression, set dot to the range and run the command.

These may be nested arbitrarily deeply, but only one instance of either **X** or **Y** may appear in a single command. An empty command in an **x** or **y** defaults to **p**; an empty command in **X** or **Y** defaults to **f**. **g** and **v** do not have defaults.

**Miscellany**

**k** Set the current file's mark to the range. Does not set dot.

\* **q** Quit. It is an error to quit with modified files, but a second **q** will succeed.

\* **u n** Undo the last *n* (default 1) top-level commands that changed the contents or name of the current file, and any other file whose most recent change was simultaneous with the current file's change. Successive **u**'s move further back in time. The only commands for which **u** is ineffective are **cd**, **u**, **q**, **w** and **D**.

(empty)

If the range is explicit, set dot to the range. If *sam* is downloaded, the resulting dot is selected on the screen; otherwise it is printed. If no address is specified (the command is a newline) dot is extended in either direction to line boundaries and printed. If dot is thereby unchanged, it is set to **+.1** and printed.

**Grouping and multiple changes**

Commands may be grouped by enclosing them in braces **{}**. Commands within the braces must appear on separate lines (no backslashes are required between commands). Semantically, an opening brace is like a command: it takes an (optional) address and sets dot for each sub-command. Commands within the braces are executed sequentially, but changes made by one command are not visible to other commands (see the next section of this manual). Braces may be nested arbitrarily.

When a command makes a number of changes to a file, as in **x/re/c/text/**, the addresses of all changes to the file are computed in the original file. If the changes are in sequence, they are applied to the file. Successive insertions at the same address are catenated into a single insertion composed of the several insertions in the order applied.

**The terminal**

What follows refers to behavior of *sam* when downloaded, that is, when operating as a display editor on a bitmap display. This is the default behavior; invoking *sam* with the **-d** (no download) option provides

access to the command language only.

Each file may have zero or more windows open. Each window is equivalent and is updated simultaneously with changes in other windows on the same file. Each window has an independent value of dot, indicated by a highlighted substring on the display. Dot may be in a region not within the window. There is usually a 'current window', marked with a dark border, to which typed text and editing commands apply. Text may be typed and edited as in *mux*(9) also the escape key (ESC) selects (sets dot to) text typed since the last mouse button hit.

The button 3 menu controls window operations. The top of the menu provides the following operators, each of which prompts with one or more *mux*-like cursors to prompt for selection of a window or sweeping of a rectangle. 'Sweeping' a null rectangle gets a large window, disjoint from the command window or the whole screen, depending on where the null rectangle is.

**new** Create a new, empty file.

**xerox** Create a copy of an existing window.

**reshape**

As in *mux*.

**close** Delete the window. In the last window of a file, **close** is equivalent to a **D** for the file.

**write** Equivalent to a **w** for the file.

Below these operators is a list of available files, starting with **sam**, the command window. Selecting a file from the list makes the most recently used window on that file current, unless it is already current, in which case selections cycle through the open windows. If no windows are open on the file, the user is prompted to open one. Files other than **sam** are marked with one of the characters *-+\** according as zero, one, or more windows are open on the file. A further mark *.* appears on the file in the current window and a single quote, *'*, on a file modified since last write.

Nothing can be done without a command window, for which *sam* prompts initially. The command window is an ordinary window except that text typed to it is interpreted as commands for the editor rather than passive text, and text printed by editor commands appears in it. The behavior is like *mux*, with a 'command point' that separates commands being typed from previous output. Commands typed in the command window apply to the current open file—the file in the most recently current window.

### Manipulating text

Button 1 changes selection, much like *mux*. Pointing to a non-current window with button 1 makes it current; within the current window, button 1 selects text, thus setting dot. Double-clicking selects text to the boundaries of words, lines, quoted strings or bracketed strings, depending on the text at the click.

Button 2 provides a menu of editing commands:

**cut** Delete dot and save the deleted text in the snarf buffer.

**paste** Replace the text in dot by the contents of the snarf buffer.

**snarf** Save the text in dot in the snarf buffer.

**look** Search forward for the next occurrence of the literal text in dot. If dot is the null string, the text in the snarf buffer is used. The snarf buffer is unaffected.

**<mux>**

Exchange snarf buffers with *mux*.

*/regex*

Search forward for the next match of the last regular expression typed in a command. (Not in command window.)

**send** Send the text in dot, or the snarf buffer if dot is the null string, as if it were typed to the command window. Saves the sent text in the snarf buffer. (Command window only.)

**scroll**

**noscroll**

Select whether to reveal automatically text that appears off the end of the command window. (Command window only.)

### Abnormal termination

If *sam* terminates other than by a **q** command (by hangup, deleting its layer, etc.), modified files are saved in an executable file. This program, when executed, asks whether to write each file back to a Unix file.

The answer *y* causes writing; anything else skips the file.

**FILES**

`$HOME/sam.save`

`$HOME/sam.err`

**SEE ALSO**

[ed\(1\)](#), [sed\(1\)](#), [vi\(1\)](#), [gre\(1\)](#)

**BUGS**

The **u** command undoes characters—and backspaces—typed directly into a file window in unpredictable increments.

**NAME**

samuel – text editor and C browser

**SYNOPSIS**

**samuel** [ *options* ] [ files ]

**DESCRIPTION**

*Samuel* is the editor *sam(9)* with additional features, including a browser for C and C++ programs. Most new features are available from the button 3 menu or commands typed in the command window. The new menu entries are **unopen**, **smudge**, **advisor**, **browser**, and **interpreter**.

**Unopen**

**Unopen** closes a window or file without removing the file name from the menu.

**Smudge**

**Smudge** associates a descriptive tag with a window and places the tag in the **smudge** submenu. The tag may be hit like a file name to switch to the window.

**Advisor**

**Advisor** gives information about the selected library function name or C keyword.

**Browser**

When **browser** is first hit, the browser's data base is initialized for the currently active files. A submenu then shows browsing functions.

**reference**

Find all references to the selected C symbol. 'Selected' means either highlighted with button 1 or contained in the snarf buffer.

**definition**

Find the definition of the selected function name, #define symbol, structure, union, class or typedef name.

**called by**

Find all functions called by the selected function name.

**calls to**

Find all calls to the selected function name.

**find** Find all instances of the selected pattern.

**egrep** Find all instances of the selected pattern, interpreted as in *egrep(1)*

**all defs**

Find definitions of all functions.

**files** List files currently in browser data base.

**rebuild**

Rebuild the data base with the current list of files.

**exit** Exit the browser.

**samuel**

Replace the contents of dot with the results of the last search.

Search results are placed in a **browser** submenu labeled with the search string. Hitting an item in a search submenu closes the currently active window (unless that would lose data) and opens a window of the same size for the file containing the item, with the window positioned at the item.

**Interpreter**

When **interpreter** is first hit, the interpreter is initialized for interactive use, and a submenu then shows interpreting functions.

**cin** Toggle the use of the command window. The first hit allows the user to send information to the interpreter from the command window. The second hit returns the command window to the editor. This interface will change in the near future.

- doit** Send the selected text to the interpreter. ‘Selected’ means either highlighted with button 1 or contained in the snarf buffer.
- load** Load a file into the interpreter. The user selects the window to load when the ‘bullseye’ prompt is presented. The **load** submenu provides functions to **load** a single file, **loadall** files in the editor, or load the **function** that contains dot (the edit point).
- view** Sets the current view. The **view** submenu provides functions to set the current **view**, a list of all **views**, describe **whatis** the selected identifier, and **where** the execution stopped in the interpreter.
- return**  
Returns from a breakpoint. The **return** submenu provides functions to **return** from a breakpoint, set a breakpoint and clear a breakpoint
- interrupt**  
Interrupt the interpreter.
- eof** Sends an EOF to the interpreter. Useful when the user program expects to see a **<control-d>**.
- exit** Exit the interpreter.

### Other features

*Help.* Press button 1 simultaneously with button 2 or 3 to see a short description of the button 2 or 3 item. In a search submenu, the information includes file name, line number and, where appropriate, function name; for a smudge submenu, the file name associated with the tag.

*File menu.* When too many files appear in the button 3 menu, they are moved to a submenu.

*Font.* On the 630 MTG Terminal, the button 2 menu includes a **font** item with a submenu that lists fonts in the terminal’s cache. The font may be set independently in each window. New windows and menus use the last font selected.

### Commands

- z** Make *samuel* menu items visible; see **-v** below. Start the browser unless it is already running.
- z-** Make *samuel* menu items invisible.
- zF dbfile**  
If *dbfile* is specified, start, or restart, the browser with *dbfile* as a read-only data base file; see options **-f** and **-F** below. Otherwise display the current database file.
- zA advisordb**  
If *advisordb* is specified, set the **ADVISOR** environment variable. Otherwise display the value of **ADVISOR**.
- za keyword**  
Search for *keyword* in the advisor database.
- zu file-list**  
Unopen the named files. If no files are named, the current file is unopened. It is an error to **zu** a modified file, but a subsequent **zu** will unopen such a file.
- zc** Delete dot and save the deleted text in the snarf buffer.
- zp** Replace the text in dot by the contents of the snarf buffer.
- zs** Save the text in dot in the snarf buffer. *keyword* in the advisor database.

### Options

- f file.db**  
Create the data base in the named file. If the file already exists and any files have been modified since the last build, update the data base.
- F file.db**  
The data base already exists in the named file. The file is read-only; rebuilds are not allowed.
- i filenames**  
Use the named files in creating the data base.
- I includedir**  
Search directory *includedir* for included files. This option may appear more than once.
- s sourcedir**  
Search directory *sourcedir* for referenced function definitions. This option may appear more than once.



- Dname=def**
- Dname**  
Define the *name* to *cin*, as if by **#define**. If no definition is given, the name is defined as 1.
- Uname**  
Remove any initial definition of *name*.
- lx**  
This option is an abbreviation for the library name **/lib/lib.x.a**, where *x* is a string. If that does not exist, *cin* tries **/usr/lib/lib.x.a**. A library is searched when its name is encountered, so the placement of a **-l** is significant.
- uname**  
Enters *name* as undefined into *cin*'s symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.
- V func:n**  
Declare function *func* to have a variable number of arguments, the first *n* of which are to be type checked.
- c**  
If the terminal is a 630 MTG, cache the terminal portion of *samuel*; later invocations will be executed from the cache without downloading.
- v**  
Make *samuel* behave like *sam*; use the **z** command to restore *samuel*.

### Environment Variables

#### INCLUDEDIRS

Colon-separated list of directories to search for **#include** files.

#### SOURCEDIRS

Colon-separated list of directories to search for additional source files.

#### SAMUEL

Directory containing samuel utilities. Overrides the default locations listed below.

#### TMPDIR

Directory used to create temporary files, by default.

#### ADVISOR

Colon-separated list of advisor data base files. These are searched in specified order followed by the standard samuel data base file.

**DMD** Directory for standard dmd software, **/usr/jerq/lib** by default.

### FILES

**\$HOME/sam.err**  
saved diagnostic output from Unix commands

**\$HOME/sam.save**  
bundled files on unexpected exit

**\$DMD/samuel/samuel.m**  
terminal support program for samuel

**\$DMD/samuel/samuel.cs**  
C browser support program for samuel

**\$DMD/samuel/samuel.ca**  
C advisor support program

**\$DMD/samuel/samuel.ca.dat**  
C advisor data base

**\$DMD/samuel/samuel.st**  
samuel statistics gathering program

**\$TMPDIR/cscope\*.0**  
default data base file

\$TMPDIR/cscope\*.1  
results of last search

\$TMPDIR/cscope\*.2  
temporary

## SEE ALSO

sam(9.1)

J. J. Puttress, *The C Browser* (11229-861017-19TMS).

J. J. Puttress, *The C Browser: Examples* (11229-861014-18TMS).

T. J. Kowalski, H. H. Goguen, J. J. Puttress, *The C Interpreter: A Tutorial for Cin Version 0.18* (11229-880606-07TMS).

R. Pike, *The Text Editor Sam* (11271-870423-06TMS).

R. Pike, *A Tutorial for the SAM Command Language* (11271-860924-07TMS).

J. L. Steffen, Interactive Examination of a C Program with Cscope. *USENIX Winter Conference Proceedings Dallas 1985*, 170-175.

## BUGS

*Samuel* will not correctly browse C source with syntax errors.

**NAME**

string, defont, strwidth, infont, outfont, getfont – text and font operations

**SYNOPSIS**

```
#include <jerq.h>
```

```
#include <font.h>
```

```
Point string(ft, s, b, p, f) Font *ft; char *s; Bitmap *b; Point p; Code f;
```

```
extern Font defont;
```

```
int strwidth(ft, s) Font *ft; char *s;
```

```
Font *infont(inch) int (*inch)();
```

```
int outfont(ft, ouch) Font *ft; int (*ouch)();
```

```
void ffree(ft) Font *ft;
```

```
#include <jerqio.h>
```

```
Font *getfont(file) char *file;
```

**DESCRIPTION**

*String* draws the null-terminated string *s* using characters from font *ft* in Bitmap *b* at Point *p*, with Code *f*. The return value is the location of the first character *after s*; passed to another call to *string*, the two strings will be concatenated. The characters are drawn such that the **origin** point of the bounding rectangle of a maximum height character lies at *p*. Therefore, a character drawn on the screen at (0,0) will occupy the upper-leftmost character position on the screen. *String* draws characters as they are in the font. No special action is taken for control characters such as tabs or newlines.

The global *defont* is the name of the standard font (not a pointer to it).

*Strwidth* returns the width in pixels of the null-terminated string *s*, interpreted in the Font *\*ft*. The height of a character string is simply **ft->height**.

*Infont* creates a font by reading the byte-wise binary representation returned by successive calls to *inch*. It returns 0 on error. *Inch* must return successive bytes of the Unix file representation of the font, and -1 at end-of-file. *Outfont* calls the routine *ouch* to write successive bytes of the binary representation of font *ft*. It returns -1 on error, as must *ouch*. For programs running under *jx*, *getfont* returns a pointer to a font read from the named *file*, essentially by calling *infont* with argument routine *getc*. It returns 0 on error. *Ffree* frees a font allocated by *infont* or *getfont*.

**NAME**

term – terminal emulators for mux

**SYNOPSIS**

**exec term** *termtype*

**DESCRIPTION**

*Term* replaces the program in the layer on its standard output with an emulator for the terminal type specified by *termtype*. In the resulting layer, environment variable TERM is set appropriately. Known types are

**2621** Hewlett-Packard 2621

**2621c** Same, with data compression between host and terminal; useful at line speeds of 2400 baud and lower.

**4014** Tektronix 4014

**5620** Teletype DMD 5620 stand-alone terminal.

**5620c** Same, with data compression.

**33** Teletype Model 33 (actually closer to 35).

Also, *termtype* **mux** restores and initializes a standard [mux\(9\)](#) terminal program.

**BUGS**

Nonstandard terminal emulators do not work across *dcon*, but usually do across *ndcon* connections; see [dcon\(1\)](#).

Unexported shell parameters and functions are lost.

**NAME**

thinkblt, think – print on thinkjet

**SYNOPSIS**

**thinkblt** [ *stream* ]

**think** [ **-o** *stream* ] [ *file ...* ]

**nroff -Tthink ... | think** [ **-o** *stream* ]

**DESCRIPTION**

*Thinkblt* downloads an interrupt driver for the HP ThinkJet printer, provides a menu of operations for printing various data residing in the terminal, and sets up a *stream* by default) on which *think* can print data from the host. It is intended to be down-loaded once per terminal session. Most of the menu items are identical to those of *blitblt(9)*. The remaining ones are:

**print bitmap**      Print whatever bitmap is currently selected, in analogy to *blitblt(9)*. The widest printable bitmap is 640 pixels across.

**print mux buffer**  
                    Print the *mux* ‘snarf’ buffer.

**reset printer**      Sends ESC-E.

While the printer is operating, a different menu allows one to abort or pause the print operation. The printer has a fairly large internal buffer, so response may be slow.

Files on the host may be printed by giving them as arguments or standard input to *think*. When used with *nroff*, names like `\('e` may be used to access the special characters provided by the hardware; the *nroff* terminal driving file has a complete list; see *troff(1)*. Both *nroff* and *pr(1)* will paginate properly if top-of-form is set so that the paper tear is aligned flush with the top of the metal clip which holds the absorber.

**FILES**

\$HOME/.THINK

/usr/lib/term/tab.think nroff descriptor file

**SEE ALSO**

*troff(1)*, *pr(1)*, *blitblt(9)*

**BUGS**

The 5620 ROM program is unable to cope with interrupts from the printer; it is therefore necessary to download *mux(9)* before turning on the printer.

*Thinkblt* substitutes its own interrupt routine for the (trivial) one provided by *mux(9)*. The latter is restored upon exit, but havoc may result if the *thinkblt* layer is simply deleted.

The special *nroff* character names are not currently supported by any other device.

**NAME**

thinkchar, thinknchars, thinkflush, thinkmap, thinkabort – ThinkJet routines

**SYNOPSIS**

```
#include <jerq.h>
#include <thinkclient.h>

int thinkchar(c) int c;

int thinknchars(n, p) int n; char *p;

int thinkflush()

int thinkmap(b, r) Bitmap *b; Rectangle r;

int thinkabort()
```

**DESCRIPTION**

These macros provide access to the routines used internally by *thinkblt(9)*. *Thinkchar* and *thinknchars* send characters to the printer; characters are buffered so that *thinkflush* must be called after the last transmission. *Thinkmap* sends all or part of a bitmap (it calls *thinkflush* automatically). *Thinkabort* stops transmission as quickly as possible, throwing away any characters that may be queued up.

**SEE ALSO**

*thinkblt(9)* *newproc(9)* *types(9)*

**DIAGNOSTICS**

The routines return zero on success, a positive value on failure, and a negative value if *thinkblt(9)* is not loaded.

**NAME**

twid, pen – doodle on the screen

**SYNOPSIS**

**twid**

**pen**

**DESCRIPTION**

*Twid* is a beginner's 'oil paint' program; serious artists will use *paint(9)* Button 3 gets a palette (menus of paints are unappetizing), and buttons 1 and 2 apply paint.

The palette has a list of names of subpalettes. After making a selection, depress button 3 again to display the subpalette. The palette names are:

**style** Choose drawing style: **ink** (Rembrandt), **point** (Seurat), **line** (Mondrian), **curve** (Matisse) and **disk** (Disney).

**texture**

selects a texture (paint) to be applied with the brush. The default set of textures is sufficient for Lichtenstein. Use the <new> button to create new ones: use button 1 (2) to select the area under the cursor (its bitwise complement), and type a name for *twid* to call it.

**brush** selects the brush size and shape. Predefined brushes are square, for effects ranging from Dali to Van Gogh; to be more modern use <new> (again, you must name the new brush).

**buttons**

By default, button 1 puts paint down and button 2 picks it up again. This palette lets you change that behavior.

**copy** provides commands for moving and rotating sections of the picture.

**unix** offers commands for reading and writing files, and exiting.

The current style, texture and brush are indicated in their palettes by an asterisk \*.

*Pen* writes on the screen with smooth strokes. It can scribble on layers or on the background, even while other programs are running. It can be used to make drawings, annotations, highlights, or graffiti.

To write, hold button 1 while moving the mouse. Button 3 gets a menu to stop drawing and return to normal fettered activity, resume drawing, clean up, or exit the program.

**BUGS**

If the pen layer where the ink is kept is too small, furious writing can cause the pen to run dry. When this happens, release button 1 and press it again.

**NAME**

Word, Point, Rectangle, Bitmap, Texture, Pt, Rect, Rpt, display, Direct, Jrect – graphics data types

**SYNOPSIS**

```
#include <jerq.h>

typedef int Word;
typedef struct Point Point;
typedef struct Rectangle Rectangle;
typedef struct Bitmap Bitmap;
typedef struct Texture Texture;

extern Bitmap display;
extern Rectangle Direct, Jrect;

Point Pt(x, y) int x, y;

Rectangle Rect(x0, y0, x1, y1) int x0, y0, x1, y1;

Rectangle Rpt() Point p0, p1;
```

**DESCRIPTION**

A **Word** is a 32-bit integer, and is the unit of storage used in the graphics software.

A **Point** is a location in a Bitmap (see below), such as the display, and is defined as:

```
typedef struct Point {
    short x;
    short y;
} Point;
```

The coordinate system has *x* increasing to the right and *y* increasing down. All objects and operators in the graphics world live in the same coordinate space—that of the display bitmap.

A **Rectangle** is a rectangular area in a Bitmap.

```
typedef struct Rectangle {
    Point origin;      /* upper left */
    Point corner;     /* lower right */
} Rectangle;
```

By definition, **origin.x** <= **corner.x** and **origin.y** <= **corner.y**. By convention, the right (maximum *x*) and bottom (maximum *y*) edges are excluded from the represented rectangle, so abutting rectangles have no points in common. Thus, **corner** contains the coordinates of the first point beyond the rectangle. The image on the display is contained in the Rectangle **{0, 0, XMAX, YMAX}**, where **XMAX=800** and **YMAX=1024**.

A **Bitmap** holds a rectangular image, stored in contiguous memory starting at *base*.

```
typedef struct Bitmap {
    Word *base;        /* pointer to start of data */
    unsigned width;    /* width in Words of total data area */
    Rectangle rect;    /* rectangle in data area, screen coords */
} Bitmap;
```

Each **width** Words of memory form a scan-line of the image, and **rect** defines the coordinate system inside the **Bitmap**: **rect.origin** is the location in the Bitmap of the upper-leftmost point in the image. The coordinate system is arranged so *x* positions equal to 0 mod 32 are in the leftmost bit of a Word.

A **Texture** is a 16×16 dot bit pattern.

```
typedef struct {
    Word bits[16];
} Texture;
```

Textures are aligned to absolute display positions, so adjacent areas colored with the same Texture mesh smoothly.



The functions *Pt*, *Rect* and *Rpt* construct geometrical data types from their components. Since they are implemented as macros, they only work in function argument lists.

The global *display* is a Bitmap describing the display area of the process. *Drect* is a Rectangle defining, in screen coordinates, the display area available to the program (inside the layer's border). *Jrect* is the Rectangle **{0, 0, XMAX, YMAX}**.

**NAME**

vismon, sysmon, vwhois – system statistics and mail notification

**SYNOPSIS**

**vismon** [ *-n* ] [ *-m* ] [ *system ...* ]

**sysmon** [ *-n* ] [ *-m* ] [ *system ...* ]

**vwhois** *person*

**DESCRIPTION**

*Vismon* monitors use of one or more Unix *systems*. It displays time of day, announcements, and CPU usage statistics.

CPU usage is reported as a numerical load average (average number of runnable processes) and its change in the last minute, and a bar graph showing, left-to-right, the proportion of CPU time spent in: default-priority user processes, low priority (*nice*) processes, system kernel, stream I/O, and idle time.

Arrival of mail or communications via *wall(8)* or *write(1)* is announced. Mail announcements include an icon of the sender. Communications appear in a shell (*sh(1)*) layer superimposed on *vismon*'s layer. This layer may be used for reply.

The options are:

**-n**      Update the bar graph every *n* seconds. (*n*=5 by default.)

**-m**      Do not monitor CPU usage on other systems.

Button 2 selectively toggles the monitoring of other systems. The list of systems is obtained from one of the following: a file named in the VISMUN environment variable, or

*Sysmon* is the same as *vismon* without icons.

*Vwhois* causes a dummy mail announcement from *person* to appear in *vismon* layers.

**FILES**

/usr/jerq/sbin/sysmon.m  
terminal program

/usr/jerq/lib/sysdaemon  
remote monitoring program

/usr/jerq/lib/sysdaemon  
responder for remote monitoring

/usr/spool/mail

/usr/spool/mail/mail.log

/n/face/\*  
vismon pictures

/usr/jerq/lib/vismon

\$HOME/lib/vismon  
menu of machines

**SEE ALSO**

*face(9)* *faced(9)*

**DIAGNOSTICS**

'Can't open comm window' means a shell layer cannot be created. To receive any further communications, delete some layer.

**BUGS**

There's more to system performance than meets the eye.

**NAME**

`windows`, `jps`, `reshape` – create and initialize windows

**SYNOPSIS**

**windows** [ *ox oy cx cy command ...* ]

**jps**

**reshape** [ **-r** ] *x y*

**DESCRIPTION**

For each set of arguments, *windows* makes a [mux\(9\)](#) layer with rectangle `Rect(ox, oy, cx, cy)` (see [types\(9\)](#)) then executes the *command* therein. The *command* may be null (""). Any number of layers may be specified; each *command* and its arguments must be given as a single argument to *windows*.

In windows that are not expected to be reused and do not need a shell, it is good practice to invoke the *command* with **exec**; see [sh\(1\)](#).

*Jps* prints the rectangle coordinates of each window and the arguments (if any) with which it was downloaded, to help set up the *windows* command.

*Reshape* adjusts its layer so that the display rectangle inside the border is *x* by *y* pixels. Under option **-r** it adjusts the width/height ratio to *x/y*, with the new shape as large as will fit inside the old.

**SEE ALSO**

[mux\(9\)](#) [ruler\(9\)](#)

**DIAGNOSTICS**

*Windows* may adjust rectangles to a minimum size or to stay within the usual layer bounds (8 pixels inside the screen edge).

Layer creation can fail if there are no process slots or memory left in the terminal.

*Reshape* clips a layer that is too big and does nothing if the layer is too small or if there is not enough memory.

**BUGS**

*Jps* reports what has been downloaded to the 5620; usually this is not the same as the command that must be used in *windows* to cause the download.

*Reshape* destroys the contents of the layer; it should work elsewhere.

**NAME**

cdl – circuit description language

**DESCRIPTION**

The circuit descriptions used by the various design aid programs are expressed in dialects of the circuit design language described below. A complete description consists of two parts; an electrical circuit with chips, pins and connecting signals, and a physical layout with pins and chip positions. The commands described below are recommended; others exist and may work but are regarded as obsolete.

**LOGICAL DESIGN**

A circuit consists of *chips* connected by *signals*. The point of connection is denoted by a *pin*. Each chip has a *type* which describes its logical and electrical characteristics. (For example, **74S181** is a chip type.)

Types, signals and chips are identified by name. Pins are identified by name and number. A *name* is a string of letters, digits or any of the characters `+-. $ / : < = > [ ] _`. Sometimes, the first character may not be a digit. A name may not be longer than 16 characters.

In the following description, literals appear as **bold**, whereas names are in *italic*. [ ] enclose an optional item and a list of items is written

{*item*}

Commands are separated by either newline or semi-colon. A comment starts with a `#` and ends with a newline and may appear on any line. All white space serves only to separate tokens.

**General**

**.p** *number*

Specifies the page number for subsequent input.

**.f** [*file*]

Subsequent input originated in *file*. If *file* is not present, the previous file name is restored.

**.q**

End of file.

**Signal Description**

*signal* [ *pin-number* ] [ [ , ] *pin-name* ]

*name* = *signal*

Lines that do not start with a period are signal definition lines. Signal definitions refer to the most recent **.c** command, the pin name and number refers to the chip.

**Circuit Description**

**.c** *name* [ [ , ] *type* ]

**.o** *name* [ [ , ] *type* ]

Instantiates a chip *name*, of type *type*. This is typically used for I/O connectors. The command may occur more than once. The type of a chip need only be specified once in a circuit description. Signal descriptions that follow a **.c** or **.o** command refer to pins on the chip.

**.c** *name* = *chip*

*chip* must be previously defined and *name* is a synonym for *chip*.

**.m** *name1* *name2*

Macro parameter definition. The signal *name1* is to be associated with macro parameter *name2*.

**.h** *signal*

Hand wired signal. The argument is the *name* of a signal that will be ignored by an automatic wiring program.

### Chip Type Description

**.t** *name package [pin] ...*

Define a chip type *name*. The name of the *package* in which it is installed, and pin numbers, *pin*, for the special signal connections are specified. The special voltage pin numbers, if present, must be in the same sequence with which the special signals are numbered. This usage is discouraged; use the **.t[TT]** commands described below. (See **.v** command.) All commands of the form ".t?" are meant to follow a **.t** line.

**.t** *name = type*

*name* is a synonym for *type*.

**.tt** *sequence\_of\_single\_character\_pin\_descriptors*

The number of characters must equal the numbers of pins on this *type*. The meaning of the descriptors is given in *wcheck*.

**.tT** *sequence\_of\_single\_character\_pin\_descriptors*

This means the same as the equivalent **.tt** command except that every **[gvwxyz]** pin must have a corresponding **.vb** pin.

**.ta** *pin1 ... pin2 ...*

*pin1 ...* is the set of address pins, in order, such that the most significant address bit appears first in the list. *pin2 ...* is the set of output pins.

**.td** *delay pin1 ... - pin2 ...*

The propagation delay (conventionally in nanoseconds) from inputs *pin1 ...* to outputs *pin2 ...* is given.

**.ti** *hi lo pin ...*

The input (or output) current range for the set of pins is given by *hi* and *lo*. Current is conventionally expressed in milliamperes.

**.tp** *name number ...*

The given pin *name* is associated with the pin *number*. *Name* may contain generators such as **Q[0-7]** which cause pin names **Q0 ... Q7** to be assigned to the pin numbers given. Multiple bracket constructs may be used. In any case, the resulting list is lexicographically sorted before assigning to pin numbers.

**.ts** *setup pin ...*

Specifies the setup time required by the device at the pins given.

**.tw** *c1 c2*

*c1* is the average current drawn by the device in milliamperes and *c2* is the maximum. Both are specified as floating point numbers.

### PHYSICAL DESIGN

The physical design consists of a *board* containing *pin-holes*. The description details the positions of the pin-holes and the position and orientation of the chips. No special case is made of I/O connectors; they are best considered as unmoveable packages. The description is divided into two files; details can be found in *board(7)*

The coordinate system for the board is with *x* increasing to the right and *y* increasing upwards. The origin is at the lower left corner; thus, no coordinate should ever be negative. The circuit board and components mounted on it are described as rectangles. They are positioned so that their sides are parallel to one or other of the axes used to describe circuit board geometry. Measurements are expressed in units of **1/100** of an inch. All are integers and have no explicit decimal point. Coordinates are expressed as pairs of integers separated by '/' with the *x* coordinate appearing first. All rectangular regions are half open; the upper and right edges are outside the rectangle.

It is sometimes necessary to provide a list of coordinates. Invariably each coordinate is associated with a numbered item (say, a pin number). A one item list consists of the item number followed by its coordinates as in

28 170/250

A series of equally spaced and consecutively numbered items can be described by giving the first and last item descriptions and separating the two with '-' as in

28 170/250 - 30 190/200

(item number 29 appears at position 180/225). If the item numbers are equally spaced but not consecutive a step size can follow the ‘-’ as in

12 200/700 -9 147 200/100

(which describes the positions of items numbered 12, 21, 30 etc.).

## Board Description

### **.B** *string*

The board name is set to *string*.

### **.A** *coord coord coord coord*

The points used in board alignment are *coord*, *coord*, *coord*, *coord*.

### **.K** *name pmin pmax ox oy cx cy*

Define a package *name* with a bounding rectangle with lower left corner (*ox,oy*) and upper right corner (*cx,cy*) as values relative to pin *pmin* of the package. The package has pins numbered from *pmin* to *pmax* inclusive; expect trouble if *pmin* is not zero or one. Placement of a package involves both its pins and rectangle. The rectangle must not intersect any other placed package, and there must be a pin-hole for each of the pins.

### **.ka** *anything*

After skipping white space the rest of the line is stored as an artwork reference.

### **.kd** *letter*

Specifies the drill type for following **.kp** commands. There can be multiple **.kd** commands per package. Currently recognized drill types are found in `/usr/jhc/pins/drills`.

### **.kp** { *pin coord* }

One or more **.kp** commands following a **.k** command gives the list of pins and their coordinates relative to pin *pmin*.

### **.ku**

Guarantees this package will not be moved by any automatic process.

### **.v** *number name*

Define Voltage and Ground special signals. The special signals are numbered consecutively from zero to five. The arguments are the special signal *number* and the signal *name* to which it corresponds.

### **.vb** { *pin coord* }

Special signal pin positions. One or more **.vb** commands following a **.v** command gives the list of pins and their positions on the circuit board. The pins should be numbered consecutively from one.

### **.vd** *number*

Specifies the drill type for following **.vb** commands. There can be multiple **.vd** commands. The types are as described for **.kd**.

### **.C** *name coord orientation flags*

Specifies the position and orientation for the chip *name*. The orientation is the number of right angles clockwise to rotate the package. The meaning of *flags* can be found in `/usr/include/cdl.h`; it should be initialised to zero.

### **.P** *coord lx ly spacing diam*

Define a rectangular array of pin-holes with diameter of *diam*. The lower left corner of the rectangle is *coord*, and the width and height are *lx,ly* respectively. The pins are placed *spacing* apart. If *spacing* is of the form *sx/sy*, the spacings in the *x* and *y* directions are set independently.

### **.R** *coord lx ly type*

Define a special rectangular region. Type **.A** defines a region that will not be used by the automatic placement algorithm.

### **.W** *chip1 pin1 chip2 pin2 net*

Define a wire link between *pin1* of *chip1* and *pin2* of *chip2*. The net name is *net*.

A line with any undefined key causes most programs to halt.

*CDL(10.5)*

*CDL(10.5)*

**SEE ALSO**  
*cdm(10)*

**NAME**

fizz – physical layout input language

**DESCRIPTION**

*Fizz* is a set of tools to build circuit boards from a circuit description. This section describes the input format for the various *fizz* commands. Most of the UCDS tools produce files in *cdl(10)* format; these need to be converted into *fizz* format by *fizz cvt*.

**Concepts**

Types, signals and chips are identified by name. Pins are identified by name and number. A *name* is a string of letters, digits or any of the characters `+-.$/:<=>[]_`. Sometimes, the first character may not be a digit. A name may not be longer than 137 characters.

The physical design consists of a *board* containing *pin-holes*. The description details the positions of the pin-holes and the position and orientation of the chips. I/O connectors may be considered as chips with unmoveable packages.

The coordinate system for the board has *x* increasing to the right and *y* increasing upwards. The origin is at the lower left corner; no coordinate should ever be negative. The circuit board and components mounted on it are described as rectangles. They are positioned so that their sides are parallel to one or other of the axes. Measurements are integers measuring 0.001 inch. Coordinates are expressed as pairs of integers separated by / with the *x* coordinate appearing first. All rectangular regions are half open; the upper and right edges are outside the rectangle.

**Syntax**

The input is a sequence of items. An item consists of a item-type followed by a number of fields. Multiple fields are indicated by a trailing { on the keyword line and terminated by a line containing a single } . Fields are a keyword followed by the value for that field. Certain values are spread over multiple lines between {} as described above.

It is sometimes necessary to provide a list of coordinates. Invariably each coordinate is associated with a numbered object (say, a pin number). A one coordinate list consists of the index number followed by its coordinates as in

**28 1700/2500**

A series of equally spaced and consecutively numbered coordinates can be described by giving the first and last coordinates and separating the two with - as in

**28 1700/2500 - 30 1900/2000**

Coordinate 29 is 1800/2250. If the index numbers are equally spaced but not consecutive a step size can follow the ‘-’ as in

**12 2000/7000 -9 147 2000/1000**

This describes coordinates numbered 12, 21, 30, and so on. If a letter follows the coordinate specifications, it specifies the drill to be used for the pinholes. The known drill types are

<b>A</b>	33
<b>B</b>	34
<b>C</b>	39
<b>D</b>	42
<b>E</b>	50
<b>F</b>	62
<b>G</b>	106
<b>H</b>	107
<b>I</b>	108
<b>J</b>	20
<b>K</b>	110
<b>L</b>	111
<b>M</b>	112
<b>N</b>	113
<b>O</b>	114



```

P      115
Q      116
R      117
S      118
T      119
U      100
V      20
W      122
X      123
Y      124
Z      125

```

### Items

In the following descriptions, each item has a sample input defining all possible fields. Some fields are optional; mandatory fields are marked by \*\* which is *not* part of the actual input.

```

Board{
    name board_name
    align 1600/2000 9600/1700 1400/7100 9600/6600
    layer signal side 1
    plane 1 + VCC 2000 2000 8000 8000
    datums 100/100 135 100/8000 45 10000/100 45
}

```

The board name is set to *board\_name*. The alignment points are used by **wrap -s** to align the board in Joe's semi-automatic wire wrapping machine. All four alignment points must be given. The *layer* field associates a layer number with a name to be used in XY artwork output. The layer numbers **0** and **1** are the two outside layers. The **plane** fields represent signal planes for circuit boards. The format is *layer sense signame minx miny maxx maxy*. *Sense* is a character meaning add (+) or subtract (-) the rectangle for the signal *signame*. The planes can be viewed with *place(10)* Note that multiple signals can be present in one layer. The *datums* field sets the positions and orientations of the three datums (alignment marks for artwork). The orientation is the angle formed by the two squares in the datum.

```

Package{
**    name DIP20
**    br -600 0 9600 3000
**    pins 1 20{
        1 0/0 - 10 9000/0 v
        11 9000/3000 - 20 0/3000 v
    }
    drills 1 2{
        1 500/1500 - 2 8500/1500 v
    }
    keepout 0 - VCC -1000 -4000 10000 3400
    plane 0 - VCC -1000 -4000 10000 3400
    plane 0 + VDD -500 -3500 9500 2900
    xymask clump {
        arbitrary XY mask stuff
    }
}

```

Each package definition may have an arbitrary origin. The bounding rectangle **br** is used for placement; the values are ll.x, ll.y, ur.x, ur.y. The **drills** field is for mounting bolts etc; it does not affect placement. Both the **pins** and **drills** fields take a minimum and maximum pin number. Placement of a package involves both its pins and rectangle. The rectangle must not intersect any other placed package, and there must be a pin-hole for each of the pins. The **keepout** field looks like a plane definition (the sense is always set to -). Multiwire wiring will not enter the specified plane. The **plane** fields are similar to those in **Board** but are instantiated for every chip using this package. The **xymask** field denotes the clump name (*clump*) for this package and some optional XY mask input (used by *artwork (10.1)*). The XY mask input has leading tabs deleted, not white space, as blanks are significant to XY mask.

```

Chip{
**   name miscinv
**   type 74F240
}

```

This simply specifies the chip type.

```

Type{
**   name 74F240
**   pkg DIP20
**   tt ii3i3i3i3gi3i3i3i3iv
}

```

The **tt** field must have a letter for every pin of the package. Any pin whose letter is one of **gvwxyz** or **GVWXYZ** will be automatically attached to special signal 0,1,2,3,4,5 respectively. Other letters are ignored (they are used by other tools).

```

Net port 4{
    select 8
    miscinv 14
    syncff 13
    ackff 1
}

```

Signal nets have the net name and number of points on the item line. All other lines are simple *chipname, pinnumber* pairs. Net descriptions are normally produced by *fizz cvt* from the output of *cdm* or *cd-mglob*.

```

Route{
**   name port
**   alg hand
    route{
        ackff 1
        miscinv 14
        select 8
        syncff 13
    }
}

```

This describes the routing for net *name*. The algorithm must be one of **tsp** (normal travelling salesman), **tspe** (travelling salesman specifying one end), **mst** (minimal spanning tree), **mst3** (minimal spanning tree of degree three), **default** (whatever is specified in the *wrap* command) and **hand** (the exact order is given). The routing is a list of *chipname, pinnumber* pairs.

```

Positions{
    select 3200/2300 0 0
    miscinv 4900/1700 0 0
    syncff 2400/2700 0 0
}

```

Specify the position data for each chip. Each line has the form *chipname coord orientation flags*. The orientation is the number of right angles clockwise to rotate the package. The following bits in *flags* have a defined meaning:

- 4** this chip is unplaced
  - 8** the bounding rectangle is ignored in placement
  - 16** the pinholes are ignored in placement. **32** the names are ignored in the silk screen output.
- Flags* should be initialised to zero.

```
Pinholes{
    1400/6900 3200 300 10 V
    6650/6900 3200 300 10 V
    1600/1700 8100 1000 10/30 V
    1600/2700 8100 1000 10/30 V
}
```

Each pinhole specification has the form *coord lx ly spacing diam* which defines a rectangular array of pinholes with diameter of *diam*. The lower left corner of the rectangle is *coord*, and the width and height are *lx,ly* respectively. The pins are placed *spacing* apart. If *spacing* is of the form *sx/sy*, the spacings in the *x* and *y* directions are set independently.

```
Vsig 0{
    name GND
    pins 96{
        1 1800/2100 - 16 9300/2100 A
        17 1800/3100 - 32 9300/3100 A
        33 1800/4100 - 48 9300/4100 A
        49 1800/5100 - 64 9300/5100 A
        65 1800/6100 - 80 9300/6100 A
        81 1800/6700 - 96 9300/6700 A
    }
}
```

This defines the special signals. The special signal number follows **Vsig**. Pins are numbered from 1; the number of pins is given in the **pins** field line. A warning is given if any pins are not specified.

## SEE ALSO

[fizz\(10\)](#)

**NAME**

fsm – finite state machine language format

**DESCRIPTION**

**Fsm** is designed to write finite state machines. It assumes that there are some number of input and output pins. These must be declared first. The input clock speed can also be declared so that the compiler will calculate the length of loops given in the time format. The input programs resemble C. There must be a procedure named **main** for the compiler to proceed. Procedures declared "inline" are called directly by the compiler to generate inline code. Otherwise the syntax is very familiar. Note that all procedures **must** be declared void. Therefore, there are no expressions on the return statement.

The [yacc\(1\)](#) syntax for *fsm* is given below:

```

program          : declarations procedures
declarations    : declarations declaration ;
                 | empty
declaration     : input
                 | outputDecl
input           : INPUT inputDetails
inputDetails    : BIT ID
                 | FIELD ID < NUMBER : NUMBER >
                 | CLOCK clockFrequency frequency
clockFrequency : NUMBER
                 | NUMBER . NUMBER
frequency      : MHZ
                 | KHZ
outputDecl     : OUTPUT outputDetails
outputDetails  : BIT ID
                 | FIELD ID < NUMBER : NUMBER >
procedures     : procedures procedure
                 | empty
procedure      : inline VOID ID ( id_list ) statement
inline        : INLINE
                 | empty
statements     : statements statement
                 | empty
statement      : output
                 | loop
                 | do
                 | enabled
                 | ifprefix statement
                 | ifelseprefix statement
                 | while
                 | repeat
                 | goto
                 | break
                 | continue
                 | call
                 | label statement
                 | { statements }
                 | ;
call          : ID ( expression_list ) ;
loop         : LOOP statement
enabled      : ENABLED statement
ifprefix     : IF boolean
ifelseprefix : ifprefix statement ELSE
while       : WHILE whileHead boolean whileTail statement
do         : DO statement dopart ;

```

```

dopart      : UNTIL boolean
            | WHILE boolean
repeat     : REPEAT NUMBER DO statement
output     : OUTPUT ( field_list ) outputSuffix ;
outputSuffix : FOR timesOrCycles
timesOrCycles : NUMBER times
            | NUMBER CYCLES
times      : NS
            | US
            | MS
goto       : GOTO ID
break     : BREAK
continue  : CONTINUE
label     : ID :
boolean   : ( expression )
id_list   : ID
            | id_list , ID
            | empty
expression_list : expression
            | expression_list , expression
            | empty
field_list : field
            | field_list , field
field     : ID = expression
expression : ( expression )
            | expression + expression
            | expression - expression
            | expression . expression
            | expression | expression
            | expression ^ expression
            | expression >> expression
            | expression << expression
            | expression
            | ! expression
            | INPUT ( ID )
            | ID
            | NUMBER

```

**NAME**

graw – graw file format

**DESCRIPTION**

**Graw** files are very simple. There is one primitive per line, each primitive indicated by a single character identifier. All strings are enclosed in quotes. Definition need not precede use, though in practice graw outputs *ref* (aka include) primitives first and master definitions are seldom found outside libraries.

**Graw** file interpreters should look up *ref* files according to some search path.

Syntax:

```
body:  prim | body prim
prim:  line | box | string | dots | macro | inst | ref | master
line:  l point point
box:   b rect
string: s chars disp point
dots:  d rect
macro: z rect
inst:  i chars point
ref:   r filename
master: mstart body mend
mstart: m chars
mend:  e
rect:  point point
point: INT INT
disp:  INT
chars: " STRING "
```

**Graw** *string* displacements are specified by five bit codes defined below:

```
/* string placement displacements */
#define HALFX 1
#define FULLX 2
#define HALFY 4
#define FULLY 8
#define INVIS 16
```

Invisible *strings* are typically defined for masters with connection points. Though the text is usually not displayed or printed, the remaining four bits should nonetheless specify a proper displacement for the sake of back-annotation.

**FILES**

/n/ross/lib/graw/gates.g the standard gate file

**SEE ALSO**

graw(10)

**NAME**

*lde* – logic design expression language format

**DESCRIPTION**

*Lde* format contains six declaration areas that must appear in the following order:

- .x** an optional chip declaration area,
- .tt** an optional line for use by *wcheck* and/or *smoke*,
- .i** an input declaration area,
- .o** an output declaration area,
- .f** an optional field declaration area,
- .e** and an expression area.

The *lde* language is much like C. Identifiers may include +-.*lde* does not use ';' to end a statement. Symbols must be declared before used. Declaration is by appearance in the **.i** or **.o** areas or appearance on the left of an = in the **.f** or **.e** areas. Since *lde* is an expression language, no flow control (such as **if** or **switch**) is allowed. An expression selector is available; *expr*{[[*exprb*]:]*exprc*.[[*exprd*]:]*expr*...} has the value of *exprc* if *expr* equals *exprb*. If there is no *exprb* and there is a colon then *exprc* is the default case. If there is no *exprb* and no colon the the pre-incremented value of the prior value of *exprb* is used, the prior value of *exprb* is initialized to -1.

The chip declaration area may contain two strings, *nameandtype*.

The **.i** and **.o** areas contain *identifier* [ '=' or ':' *numeric\_pinnumber* ] (The ':' is used by *lde -w* as a mark for externals.) Some programs use the order of appearance of the identifiers.

The field declaration area contains constructions of the form *n\_id* = *o\_id* *o\_id* ... where *n\_id* is a new identifier for a multibit value the least significant bit of which is the first old identifier, *o\_id*.

The expression area contains assignments of expressions to identifiers. Identifiers may be modified by a postpended single quote, "'", in which case a value of one has the meaning "don't\_care" for the unmodified identifier.

Numeric values may be passed from the command line, they appear as **\$m**. The (zero based) *m*th occurrence of -*n* one the command line substitutes the value *n* for the symbol **\$m**.

**EXAMPLES**

```

/*
 * bkrom
 * classifies the location of the
 * black king.
 * 0-6 manhattan distance to center
 * 7 orig square
 * 8-11 k-side
 * 12-15 q-side
 */
.x
    bkrom 74S287

.i
    wkx0 wkx1 wkx2
    wky0 wky1 wky2
    GND1 GND2 GND3

.o
    kb0 kb1 kb2 kb3

.f
    kx = wkx0 wkx1 wkx2
    ky = wky0 wky1 wky2
    kb = kb0 kb1 kb2 kb3

.e
    xd = (kx) { 3, 2, 1, 0, 0, 1, 2, 3 }
    yd = (ky) { 3, 2, 1, 0, 0, 1, 2, 3 }
    d = xd + yd
    kb =
        (ky == 6)?
            (kx) { 12, 13, d, d, d, d, 8, 9 }:
```

```

                (ky == 7)?
                (kx) { 14, 15, d, d, 7, d, 10, 11 } :
                d
/*
* By convention the output enable term for
* PAL's is 100 + the corresponding pin number.
* this example includes a .tt line for use by wcheck.
*/
.x      Bpal      PAL16L8A
.tt      iiiiiiiign22222n2v
.i
        A0 :      1      A1 :      2      A2 :      3      A3 :      4
        A4 :      5      A5 :      6      A6 :      7      A7 :      8
        A8 :      9

.o
        SE+ :     12      RNE+ :     13      TD+ :     14      TU+ :     15
        SFSE :    16      Y5 :      17
        e12 = 112      e13 = 113      e14 = 114      e15 = 115
        e16 = 116      e17 = 117      e19 = 119

.f
        cnt = A0 A1 A2 A3 A4
        ardy = A5
        crdy = A6
        flushb- = A8
        flusha- = A7

.e
        tmp = ((cnt == 0) ? ardy ? 1 : 0 :
              (cnt == 6) ? (crdy || !flushb-) ? 1 : 0 : 1 )

        /* shift enable + for major data path, also count enable */
        SE+ = !tmp

        /* random number clock enable - */
        RNE+ = !(flusha- ? 0 : tmp )

        /* transfer down - for ireg */
        TD+ = !((cnt == 0) .. ardy)

        /* transfer up + (invert outside) for oreg<0:3> */
        TU+ = !((cnt == 6) .. crdy .. flushb-)

        /* shift flush status enable */
        SFSE = !(cnt == 3)

        /* ack- back to ardy */
        Y5 = !((cnt == 0) .. ardy)

        /* ready to A */
        BRDY = !( (cnt == 0)? 1 : 0)
        e12 = 1      e13 = 1      e14 = 1      e15 = 1
        e16 = 1      e17 = 1      e19 = 1
/*
* An example using parameter passing and Don't_care
*/
.x      dram      PAL16R6
.tt      iiiiiinnngin22222nv
.i
        CK:1 OE-:11
        dreq:2 stall:3 cerr:4 read:5 qword:6
        rasefb=18 casxfb=17 casyfb=16 wefb=15
        dsfb0=14 dsfb1=13

.o
        rase:18 casx:17 casy:16 we:15
        ds0:14
        ds1:13

.f

```



```

DS = rase casx casy we ds1 ds0
DSfb = rasefb casxfb casyfb wefb dsfb1 dsfb0
.e
X.NCAS = 0100          /* don't care bits      */

DC = 0200          /* don't care state */
S.RAS = 040
S.CAS = 020
S.NCAS = 010
S.WE = 004

/* low order 2 bits of state vector */
A = $0 B = $1 C = $2 D = $3

I0 = C          /* state assignment */
D10 = S.RAS + A
D11 = S.RAS + S.CAS + X.NCAS + B
D12 = S.RAS + S.CAS + X.NCAS + A
D13 = S.RAS + S.NCAS + B
D23 = S.RAS + B
D14 = S.RAS + S.NCAS + D
D24 = S.RAS + D
D15 = S.RAS + S.NCAS + C
D25 = B
D16 = A
D26 = D
D31 = S.RAS + S.CAS + X.NCAS + D
D32 = S.RAS + S.CAS + X.NCAS + S.WE + A
D33 = S.RAS + S.NCAS + S.WE + B
D43 = S.RAS + S.CAS + X.NCAS + S.WE + B
D34 = S.RAS + S.NCAS + S.WE + D
D44 = S.RAS + S.CAS + X.NCAS + S.WE + D
D35 = S.RAS + S.NCAS + S.WE + C
D36 = S.WE

DS- = DSfb {
    I0: dreq ? D10 : I0,          /* idle state */
    D10: read ? D11 : D31,
    D31: stall ? D31 : D32,
    D32: qword ? (stall ? D32 : D33) : D36,
    D33: stall ? D43 : D34,
    D43: stall ? D43 : D34,
    D34: stall ? D44 : D35,
    D44: stall ? D44 : D35,
    D35: D36,
    D36: I0,
    D11: stall ? D11 : D12,
    D12: qword ? D13 : D16,
    D13: cerr ? D23 : D14,
    D23: D14,
    D14: cerr ? D24 : D15,
    D24: D15,
    D15: cerr ? D25 : D16,
    D25: D16,
    D16: cerr ? D26 : I0,
    D26: dreq ? D10 : I0,
    : DC
}

DS = 077 ^ DS-
DS' = (DS- == DC) ? 0 :
      ((DS- . X.NCAS) ? S.NCAS : 0)

```

**NAME**

mds – kollmorgen symbolic data format

**DESCRIPTION**

MDS is the symbolic format for the Kollmorgen Pck Division channel router. This is an abbreviated description of the format.

*Introduction*

All MDS data is made up of integers, symbols and keywords. All coordinates are expressed in mils, i.e., 1/1000 of an inch (just like fizz!). Coordinates can be negative or positive and are denoted by matching parentheses of the form (X Y). Symbols require quotes around them if not made up of numbers, letters, \$ or a `_`.

*Data types*

There are 15 data types identified by their reserved word. The following is a list of valid types:

Border	Board edges and keepouts
Check	Design rules to check (not required)
Design	Name of the design
F2	From-to (before routing)
Fail	Failed from-to (after routing)
File	pointer to another file
Fix	Hand routed wire
Hole	Drill hole
Level	Wiring surface (typically only two: COMP and PBSN)
Net	Net
Panel	Board of some type (optional)
Route	Routed from-to with intermediate points at each bend
Term	Termination site for a net (optional)
Wire	Wire path
Wire_region	Routing zone for layers

**Border id coordinates**

Borders are closed loops composed by the coordinates and named by a identifier.

**Check [rule] [-MIN: n] [-ON|-OFF]**

Specifies what design rules should be checked by the *repair* program.

**Design [name]**

Names a design. Strictly optional.

**F2 netname coordinates [switches]**

From-tos are the principal data format of the routing system. They are created from nets by F2gen. Intermediate points are specified by following the coordinate by a -I or -W. Valid switches include:

- LEVEL: level  
level name (or number)
- ORDER: cost\_function  
Sets cost function (see section below)
- IFL: n  
number of inflection points; default 9
- MIN\_FS\_END:  
sets minimum first and last segment length. Default = 0.
- MIN\_WIRE\_END:  
sets only wire end segment length.
- MDR: n

manhattan distance ratio (in tenths, default 1.5 = 15)  
 -AXIAL | -ESCAPE | -EITHER | -DIAGONAL  
 Route path direction; Axial is along axes, escape indicates  
 diagonals on ends only.  
 -AWD: n  
 Adjacent Wire Distance  
 -CLW: window  
 -XDW: window  
 CLW checks for coupled length violations within window {n,l}  
 where n is the center to center distance and l is the length.  
 -XOVER: limit  
 Crossover limit

#### **Fail netname coordinates [switches]**

Fails are identical to from-tos except for be called fails. See above list of switches.

#### **File [filename] [switches]**

This include the filename. The switch specifies what kind of file it is.

#### **Fix sequence\_no [switches] coordinates**

Specifies a hand routed net. It is identical to route records (see below). The endpoints must agree with the fail it fixes.

#### **Hole code [switches] coordinates**

Specifies a drilled or LASER'ed hole. The switches are:

-SIZE: n  
 size of n mils  
 -WIRED | -NOTWIRED  
 dictates if hole can be wired  
 -TOLERANCE: n  
 -LASED -LEVEL: n | -DRILLED  
 if not drilled, only one layer can be specified

#### **Level level [-F2\_DENSITY: n]**

Specifies wiring level directly; designs are assumed to be on one level unless otherwise told. The optional switch specifies density of routes on the layer.

#### **Net [pre\_switches] or Net name [pre\_switches] coordinates [post\_switches]**

If the net name is omitted, then the switches are global. The pre\_switches include all of the from-to switches plus the following:

-LINK: n  
 Limit of from-tos using this node; default: 2  
 -FIX | -DECOMP  
 Fixes order in from-to list or decomposes it  
 -FIX\_START | -NOFIX\_START  
 -FIX\_END | -NOFIX\_END  
 Treat first or last node as if LINK: 1; default: NOFIX  
 -TERM\_TO: pool  
 -TERM\_END: pool  
 -TERM\_WIRE\_END: pool  
 -TERM\_LIMIT: n  
 -NOTERM  
 Reduces link by 1. Assigns terminator from pool. Default is -NOTERM  
 -ORDER: cost\_function  
 Possible cost functions are:

AIR\_SL - Airline, shortest to longest  
 AIR\_LS - Airline, longest to shortest  
 MAN\_SL - Manhattan, shortest to longest  
 MAN\_LS - Manhattan, longest to shortest  
 X\_SL - X, shortest to longest  
 X\_LS - X, longest to shortest  
 Y\_SL - Y, shortest to longest  
 Y\_LS - Y, longest to shortest  
 -BALANCE: cost\_function  
 Possible cost functions are:  
 COUNT - by from-to count  
 AIR - by "airline distance"  
 MAN - by "Manhattan metric"  
 X - by X coordinate  
 Y - by Y coordinate  
 -SUPPLY | -SIGNAL  
 -SUPPLY nets are ignored; default is -SIGNAL.

The post\_switches are:

-LINK: n  
 -FIX | -DECOMP  
 -TERM\_TO: pool

#### **Panel [name] coordinates [-DESIGN: name]**

Defines a coordinate system for translated output data. Strictly optional.

#### **Route [sequence\_no] [switches] coordinates**

If the sequence number and coordinates are omitted, then the switches are globally applied. Valid switches include:

-NET: name  
 -LEVEL: level  
 -W\_DIA: n  
 Used by wire clearance checks  
 -CLW: window  
 -XDW: window  
 -AWD: n  
 Adjacent Wire Distance (default 0)  
 -PASS: n  
 Set pass number (starts at 1)  
 -XOVER: limit  
 Sets limit to wire crossovers; can be NONE, ONE or TWO.

#### **Term pool coordinates**

Pool together a set of coordinates of terminals given by XY coordinates. Typically used by the TER-MGEN program to assign terminators automatically. Used by ECL freaks.

#### **Wire [sequence\_no] [switches] coordinates**

Just like routes except ...

#### **Wire\_region [zone] [switches]**

Specifies an XY plane where wiring can be done by the router. If the zone is omitted then the specification is global. The switches are:

-LEVEL: level  
 -AWE\_WE: n  
 Axial Wire Edge to Wire Edge distance  
 -AWE\_HE: n

Axial Wire Edge to Hole Edge distance  
-DWE\_WE: n  
Diagonal Wire Edge to Wire Edge distance  
-DWE\_HE: n  
Diagonal Wire Edge to Hole Edge distance  
-W\_DIA: n  
Wire diameter  
-NSID: n  
Normal/Segment Intercept Distance.  
-MAX\_HTURN: n  
Maximum turn angle (in degrees)  
-DIR: name | -DBECTORY: name  
Prepend this name to map file output name.

The following is a typical map file for input to Mapgen:

```
Level COMP
Level PBSN -F2_density: 50
Net -balance: air
Wire_region A -Level: COMP -Max_Hturn: 135 -NSID: 15 -W_dia: 8
Wire_region A -AWE_WE: 8 -AWE_HE: 15 -DWE_WE: 27 -DWE_HE: 8
Wire_region B -Level: PBSN -Max_Hturn: 135 -NSID: 15 -W_dia: 8
Wire_region B -AWE_WE: 8 -AWE_HE: 15 -DWE_WE: 27 -DWE_HE: 8
```

**NAME**

minterm – minterm file format

**DESCRIPTION**

The *minterm* file format consists of at least one binary valued function definition. A function definition begins `.o n ...` followed by line(s) that have the form `term:mask ...`. The first *n* following `.o` is a numeric symbol of the function (usually an output pin number of a rom or pal integrated circuit). Any other *n*'s are numeric symbols of input binary variables. *Term* and *mask* are decimal numbers.

There is a correspondence between the bits of the numbers in binary representation and the input symbols, the first input symbol is associated with the least significant bit. The meaning of a bit with value 1 in *mask* is 'do care', and the meaning of a bit with value 1 in *term* is 'input must be 1'. Thus the *term:mask* is a implicant, and a set of them when *or*'ed together describes the input conditions for which the output symbol will have a value of 1.

For example:

```
.o 3 1 2
3:3
.o 4 1 2
1:3 2:3 3:3
.o 5 2 3
1:3 2:3
.o 11
.o 9
0:0
```

Output 3 is the *and* function of inputs 1 and 2; output 4 is the *or* function of inputs 1 and 2 (*quine*(10.1) would change this to 1:1 2:2); output 5 is the *exclusive-or* function of inputs 2 and 3; output 11 is a constant 0 and output 9 is a constant 1.

**SEE ALSO**

*lde*(10) *quine*(10) *cover*(10) *hazard*(10) *pal*(10)

**NAME**

paddle – pal description language

**DESCRIPTION**

*paddle* is a description language for detailing the fuse format of programmable devices. *paddle* is used by *xpal*(10) to create the fuse map that *urom*(1) and friends want. *paddle* permits multiple fuse arrays provided they are given unique names. Each definition begins by defining the name of the part along with possible synonyms. This is followed by (1) an array declaration (2) a fuse block definition (3) a type declaration (the .tt line) and lastly, a (4) pins declaration. The array declaration permits declaration of input and output pins to the array. The use of the *complement* keyword create 2 input lines for a given pin. The general form of a pin declaration is pin:#terms=fuse, where #terms is the *maximum* number of terms for the pin and fuse is the optional fuse number. Here is part of the declaration of a 20L10:

```
20L10=NS20L10=AM20L10 {
  array and/or {
    inputs {
      complement+ external {
        2, 1,
        .
        .
        11, 13
      }
    }
    outputs {
      external {
        123:1,
        23:3,
        .
        .
        .
      }
    }
  }
  type "iiiiiiiiiiiig i345555555v"
}
```

**SEE ALSO**

*xpal*(10)

**NAME**

saf – spider a format

**DESCRIPTION***Non data format*

This is a non-data record which is required if transmission is to be made to the SPIDER DEC 10 machine at the Northern Illinois Works.

Columns 1-3: (3) Character string "HO1".

Columns 4-5: (2) Number of non-data records. Always "01".

Columns 6-6: (1) Type of records, Fixed or Variable. Set to Fixed "F".

Columns 7-9: (3) Record Length - Set to "132".

Columns 10-13: (4) Data format type - Set to "ASCII".

Columns 14-17: (4) Job type descriptor - Set to "USER".

Columns 18-31: (14) Product Identifier. Set to Circuit Pack Code e.g., TN33Text.

Columns 32-35: (4) Product Issue. Set to Artmaster Issue number. Text.

Columns 36-43: (8) Warrenville/IH Run date (00/00/00).

Columns 44-51: (8) Warrenville/IH Run time (00:00:00).

Columns 52-57: (6) Number of data records in the file. Must be padded with leading zero's.

*Common file format*

Columns 1-2: (2) The character string "CF".

Columns 3-35: (33) Circuit Pack Code, e.g., TN334, MC4C001A1, etc. Text.

Columns 36-50: (15) Circuit Pack Series, e.g., 1, 3-5, etc. Text.

Columns 51-54: (5) Artmaster issue number, e.g., 1, 2a, etc. Text.

Columns 55-63: (9) Completed Circuit Pack Comcode number. Integer.

Columns 64-78: (15) Completed Circuit Pack CLEI code. Text.

Columns 79-83: (5) Completed Circuit Pack CLEI code Series number. Text.

Columns 84-88: (5) Circuit Pack Keying Code. Text.

Columns 89-93: (5) Circuit Pack Bar Label Code. Text.

Columns 94-132: (39) Blanks.

*Component reference data*

There is to be one record for each part including hardware. Coordinates are to be the Equipment location (Default Datum - Pin 1).

Columns 1-2: (2) The text string "CR".

Columns 3-7: (5) Component Type, e.g., DIO, R, etc. Text.

Columns 8-27: (20) Component Code, e.g., KS16645L1, etc. Text.

Columns 28-37: (10) Component Value, e.g., 10K, 20UF, etc. Text.

Columns 38-46: (9) Component Comcode Number. Integer.

Columns 47-58: (12) Component Generic Code. Text.

Columns 59-68: (10) Microcode Stamping Code. Text.

Columns 69-73: (5) Component Length in mils. Integer.

Columns 74-78: (5) Component Width (mils). Integer.

Columns 79-83: (5) Component Height (mils). Integer.

Columns 84-88: (5) Component Lead Wire Size (mils) - diameter of circular or rectangular smaller side. Integer.

Columns 89-93: (5) Component Lead Wire Size (mils) - rectangular larger side. Integer.

Columns 94-96: (3) Number of Component leads. Integer.

Columns 97-108: (12) Component reference designation. Text.

Columns 109-115: (7) X-Coordinate (tenths of mils). Real.

Columns 116-122: (7) Y-Coordinate (tenths of mils). Real.

Columns 123-127: (5) Component Lead Span (mils). Integer.

Columns 128-130: (3) Component Rotation in degrees. Integer.



Column 131: (1) Surface Mount Indicator field. Text.

Column 132: (1) Mounting Option field. Text.

Legal Values Are:

"E" - On End - Front Side Mounting

"O" - On End - Back Side Mounting

"F" - Flat - Front Side Mounting

"B" - Flat - Back Side Mounting

"U" - Up - Front Side Mounting

"P" - Up - Back Side Mounting

#### *Drill data*

Columns 1-2: (2) The text string "DD".

Columns 3-9: (7) X-Coordinate (tenths of mils). Real.

Columns 10-16: (7) Y-Coordinate (tenths of mils). Real.

Columns 17-21: (5) Hole Size (mils). Integer.

Columns 22-22: (1) Plated thru indicator. Text.

Columns 23-132: (110) Blanks.

#### *Connectivity data*

This data should contain all unused component pins and theates are actual location of the pin involved.

Columns 1-2: (2) The text string "CD".

Columns 3-8: (6) Record type. Must be one of the strings: "VIA", "USED", or "UNUSED".

Columns 9-20: (12) Net Name. Text.

Columns 21-32: (12) Reference Designation. Text.

Columns 33-40: (8) Terminal Name. Text.

Columns 41-47: (7) X Coordinate (tenths of mils). Real.

Columns 48-54: (7) Y Coordinate (tenths of mils). Real.

Columns 55-56: (2) Net type attribute. Text.

Columns 57-57: (1) Text.

Columns 58-132: (75) Blanks.

**NAME**

stock – stock list

**DESCRIPTION**

The stock file is a plain text file. The first column is the part name, the second column is the bin (bins have the form <bin number><section><drawer>), the third column is the quantity and the remaining string is the chip description. The latest entries include the manufacturer at the end of the line in the form "[manufacturer]".

**SEE ALSO**

*findparts*(10) *ics*(10)

**FILES**

/usr/ucds/lib/stock

**BUGS**

The quantity is seldom up to date.