NAME

ksh, ksh93 - KornShell, a command and programming language

SYNOPSIS

ksh [±**abcefhikmnoprstuvxBCDP**] [**-R** file] [±**o** option] ... [**-**] [arg ...]
rksh [±**abcefhikmnoprstuvxBCD**] [**-R** file] [±**o** option] ... [**-**] [arg ...]

DESCRIPTION

Ksh is a command and programming language that executes commands read from a terminal or a file. *Rksh* is a restricted version of the command interpreter *ksh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. *Rpfksh* is a profile shell version of the command interpreter *ksh*; it is used to to execute commands with the attributes specified by the user's profiles (see *pfexec(1)*). 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 components of *variable* names. A *vname* is a sequence of one or more identifiers separated by a **.** and optionally preceded by a **.**. Vnames are used as function and variable names. A *word* is a sequence of *characters* from the character set defined by the current locale, excluding non-quoted *metacharacters*.

A *command* is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A built-in command is a command that is carried out by the shell itself without creating a separate process. Some commands are built-in purely for convenience and are not documented here. Built-ins that cause side effects in the shell environment and built-ins that are found before performing a path search (see *Execution* below) are documented here. For historical reasons, some of these built-ins behave differently than other built-ins and are called *special built-ins*.

Commands.

A *simple-command* is a list of variable assignments (see *Variable Assignments* below) or a sequence of *blank* separated words which may be preceded by a list of variable assignments (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; 0-255 if it terminates normally; 256+*signum* if it terminates abnormally (the name of the signal corresponding to the exit status can be obtained via the **-l** option of the **kill** built-in utility).

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, except possibly the last, 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 unless the **pipefail** option is enabled. Each pipeline can be preceded by the *reserved word* **!** which causes the exit status of the pipeline to become 0 if the exit status of the last command is non-zero, and 1 if the exit status of the last command is 0.

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 pipeline with a two-way pipe established to the parent shell; the standard input and output of the spawned pipeline can be written to and read from by the parent shell by applying the redirection operators **<&** and **>&**

with arg **p** to commands and by using **-p** option of the built-in commands **read** and **print** described later. The symbol **&&** (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. One or more new-lines may appear in a *list* instead of a semicolon, to delimit a command. The first *item* of the first *pipeline* of a *list* that is a simple command not beginning with a redirection, and not occurring within a **while**, **until**, or **if** *list*, can be preceded by a semicolon. This semicolon is ignored unless the **showme** option is enabled as described with the **set** built-in below.

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 *vname* [**in** *word* ...] **;do** *list* **;done**
Each time a **for** command is executed, *vname* 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 starting from **1** (see *Parameter Expansion* below). Execution ends when there are no more words in the list.

for (([*expr1*] **;** [*expr2*] **;** [*expr3*] **)) ;do** *list* **;done**
The arithmetic expression *expr1* is evaluated first (see *Arithmetic evaluation* below). The arithmetic expression *expr2* is repeatedly evaluated until it evaluates to zero and when non-zero, *list* is executed and the arithmetic expression *expr3* evaluated. If any expression is omitted, then it behaves as if it evaluated to 1.

select *vname* [**in** *word* ...] **;do** *list* **;done**
A **select** command prints on standard error (file descriptor 2) the set of *word*s, each preceded by a number. If**in** *wor d* ... is omitted, then the positional parameters starting from **1** are used instead (see *Parameter Expansion* 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 *word*s, then the value of the variable *vname* 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 variable *vname* is set to *null*. The contents of the line read from standard input is saved in the variable **REPLY**. The*list* is executed for eac h selection until a **break** or *end-of-file* is encountered. If the **REPLY** variable is set to *null* by the execution of *list*, then the selection list is printed before displaying the **PS3** prompt for the next selection.

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). The**;;** operator causes execution of **case** to terminate. If **;&** is used in place of **;;** the next subsequent list, if any, is executed.

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 each successive **elif** *list*, the **else** *list* is executed. If the **if** *list* has non-zero exit status and there is no **else** *list*, 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.

((*expression***))**
The *expression* is evaluated using the rules for arithmetic evaluation described below. If the value of the arithmetic expression is non-zero, the exit status is 0, otherwise the exit status is 1.

(*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 evaluation as an arithmetic command as described above.

{ *list*;}
list is simply executed. Note that unlike the metacharacters (and), { and } are *reserved words* and must occur at the beginning of a line or after a ; in order to be recognized.

[[*expression*]]
Evaluates *expression* and returns a zero exit status when *expression* is true. See *Conditional Expressions* below, for a description of *expression*.

function *varname* { *list* ;}
varname () { *list* ;}
Define a function which is referenced by *varname*. A function whose*varname* con tains a . is called a discipline function and the portion of the*varname* preceding the last . must refer to an existing variable. The body of the function is the*list* of commands between { and }. A function defined with the**function** *varname* syn tax can also be used as an argument to the . special built-in command to get the equivalent behavior as if the *varname*() syntax were used to define it. (See *Functions* below.)

namespace *identifier* { *list* ;}
Defines or uses the name space *identifier* and runs the commands in *list* in this name space. (See*Name Sp aces* below.)

& [*name* [*arg...*]]
Causes subsequent *list* commands terminated by & to be placed in the background job pool *name*. If*name* is omitted a default unnamed pool is used. Commands in a named background pool may be executed remotely.

time [*pipeline*]
If *pipeline* is omitted the user and system time for the current shell and completed child processes is printed on standard error. Otherwise, *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error. The **TIMEFORMAT** variable may be set to a format string that specifies how the timing information should be displayed. See**Shell V ariables** below for a description of the **TIMEFORMAT** variable.

The following reserved words are recognized as reserved only when they are the first word of a command and are not quoted:

if then else elif fi case esac for while until do done { } **function select time** [[ ]] **!**

Variable Assignments.
One or more variable assignments can start a simple command or can be arguments to the **typeset**, **enum**, **export**, or **readonly** special built-in commands as well as to other declaration commands created as types. The syntax for an *assignment* is of the form:

varname=*word*
varname[*word*]=*word*
No space is permitted between *varname* and the = or between = and *word*.

varname=(*assign_list*)
No space is permitted between *varname* and the =. The variable *varname* is unset before the assignment. An*assign_list* can be one of the follo wing:

word ...
Indexed array assignment.

[*word*]=*word* ...
: Associative array assignment. If preceded by **typeset -a** this will create an indexed array instead.

assignment ...
: Compound variable assignment. This creates a compound variable *varname* with sub-variables of the form *varname*.*name*, where *name* is the name portion of *assignment*. The value of *varname* will contain all the assignment elements. Additional assignments made to sub-variables of *varname* will also be displayed as part of the value of *varname*. If no *assignment*s are specified, *varname* will be a compound variable allowing subsequence child elements to be defined.

typeset [*options*] *assignment* ...
: Nested variable assignment. Multiple assignments can be specified by separating each of them with a **;**. The previous value is unset before the assignment. Other declaration commands such as **readonly, enum**, and other declaration commands can be used in place of **typeset**.

. *filename*
: Include the assignment commands contained in *filename*.

In addition, a += can be used in place of the = to signify adding to or appending to the previous value. When+= is applied to an arithmetic t ype, *word* is evaluated as an arithmetic expression and added to the current value. When applied to a string variable, the value defined by *word* is appended to the value. For compound assignments, the previous value is not unset and the new values are appended to the current ones provided that the types are compatible.

The right hand side of a variable assignment undergoes all the expansion listed below except word splitting, brace expansion, and file name generation. When the left hand side is an assignment is a compound variable and the right hand is the name of a compound variable, the compound variable on the right will be copied or appended to the compound variable on the left.

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. An **alias** name consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter expansion and command substitution characters, the characters / and =. The replacement string can contain any valid shell script including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for 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 built-in commands but cannot be used to redefine the reserved words listed above. Aliases can be created and listed with the**alias** command and can be removed with the **unalias** command.

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect, the **alias** definition command has to be executed before the command which references the alias is read.

The following aliases are compiled into the shell but can be unset or redefined:

autoload='typeset -fu'
command='command '
compound='typeset -C'
fc=hist
float='typeset -lE'
functions='typeset -f'

hash=′alias -t --′
history=′hist -l′
integer=′typeset -li′
nameref=′typeset -n′
nohup=′nohup ′
r=′hist -s′
redirect=′command exec′
source=′command .′
stop=′kill -s STOP′
suspend=′kill -s STOP $$′
times=′{ { time;} 2>&1;}′
type=′whence -v′

Tilde Substitution.

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. For tilde substitution, *word* also refers to the *word* portion of parameter expansion (see *Parameter Expansion* below). If it does, then the word up to a / is checked to see if it matches a user name in the password database (See *getpwname(3).)* If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. If no match is found, the original text is left unchanged. A~ b y itself, or in front of a /, is replaced by **$HOME**. A~ follo wed by a + or **-** is replaced by the value of **$PWD** and **$OLDPWD** respectively.

In addition, when expanding a *variable assignment*, *tilde* substitution is attempted when the value of the assignment begins with a ~, and when a ~ appears after a **:**. The**:** also terminates a ~ login name.

Command Substitution.

The standard output from a command list enclosed in parentheses preceded by a dollar sign (**$(***list***)**), or in a brace group preceded by a dollar sign (**${** *list***;}**), or in a pair of grave accents (**`** **`**) may be used as part or all of a word; trailing new-lines are removed. In the second case, the { and } are treated as a reserved words so that { must be followed by a *blank* and } must appear at the beginning of the line or follow a **;**. In the third (obsolete) form, the string between the quotes is processed for special quoting characters before the command is executed (see *Quoting* below). The command substitution**$(cat file)** can be replaced b y the equivalent but faster **$(<file)**. The command substitution**$(***n***<#)** will expand to the curren t byte offset for file descriptor *n*. Except for the second form, the command list is run in a subshell so that no side effects are possible. For the second form, the final } will be recognized as a reserved word after any token.

Arithmetic Substitution.

An arithmetic expression enclosed in double parentheses preceded by a dollar sign (**$(())**) is replaced by the value of the arithmetic expression within the double parentheses.

Process Substitution.

Each command argument of the form **<(***list***)** or **>(***list***)** will run process *list* asynchronously connected to some file in **/dev/fd** if this directory exists, or else a fifo a temporary directory. The name of this file will become the argument to the command. If the form with > is selected then writing on this file will provide input for *list*. If< is used, then the file passed as an argumen t will contain the output of the *list* process. For example,

paste <(cut -f1*file1***) <(cut -f3***file2***) | tee >(***pr ocess1***) >(***process2***)**

cuts fields 1 and 3 from the files *file1* and *file2* respectively, *pastes* the results together, and sends it to the processes *process1* and *process2*, as well as putting it onto the standard output. Note that the file, which is passed as an argument to the command, is a UNIX pipe(2) so programs that expect to lseek(2) on the file will not work.

Process substitution of the form **<(***list***)** can also be used with the < redirection operator which causes the output of *list* to be standard input or the input for whatever file descriptor is specified.

Parameter Expansion.

A *parameter* is a *variable*, one or more digits, or any of the characters *, **@**, #, **?**, **-**, **$**, and **!**. A *variable* is denoted by a *vname*. To create a variable whose *vname* contains a **.**, a variable whose *vname* consists of everything before the last **.** must already exist. A *variable* has a *value* and zero or more *attributes*. *Variables* can be assigned *values* and *attributes* by using the **typeset** special built-in command. The attributes supported by the shell are described later with the **typeset** special built-in command. Exported variables pass values and attributes to the environment.

The shell supports both indexed and associative arrays. An element of an array variable is referenced by a *subscript*. A*subscript* for an indexed arra y is denoted by an *arithmetic expression* (see *Arithmetic evaluation* below) between a [and a]. To assign values to an indexed array, use *vname*=(*value* ...) or **set -A** *vname value* ... . The value of all non-negative subscripts must be in the range of 0 through 4,194,303. A negative subscript is treated as an offset from the maximum current index +1 so that -1 refers to the last element. Indexed arrays can be declared with the **-a** option to **typeset.** Indexed arrays need not be declared. Any reference to a variable with a valid subscript is legal and an array will be created if necessary.

An associative array is created with the **-A** option to **typeset.** A *subscript* for an associative array is denoted by a string enclosed between [and].

Referencing any array without a subscript is equivalent to referencing the array with subscript 0.

The *value* of a *variable* may be assigned by writing:

vname=*value* [*vname*=*value*] ...

or

vname[*subscript*]=*value* [*vname*[*subscript*]=*value*] ...

Note that no space is allowed before or after the =.

Attributes assigned by the *typeset* special built-in command apply to all elements of the array. An array element can be a simple variable, a compound variable or an array variable. An element of an indexed array can be either an indexed array or an associative array. An element of an associative array can also be either. To refer to an array element that is part of an array element, concatenate the subscript in brackets. For example, to refer to the *foobar* element of an associative array that is defined as the third element of the indexed array, use **${***vname*[*3*][*foobar*]**}**

A *nameref* is a variable that is a reference to another variable. A nameref is created with the**-n** attribute of **typeset**. The value of the variable at the time of the **typeset** command becomes the variable that will be referenced whenever the nameref variable is used. The name of a nameref cannot contain a **.**. When a variable or function name contains a **.**, and the portion of the name up to the first **.** matches the name of a nameref, the variable referred to is obtained by replacing the nameref portion with the name of the variable referenced by the nameref. If a nameref is used as the index of a **for** loop, a name reference is established for each item in the list. A nameref provides a convenient way to refer to the variable inside a function whose name is passed as an argument to a function. For example, if the name of a variable is passed as the first argument to a function, the command

typeset -n var=$1

inside the function causes references and assignments to **var** to be references and assignments to the variable whose name has been passed to the function.

If any of the floating point attributes, **-E**, **-F**, or **-X**, or the integer attribute, **-i**, is set for *vname*, then 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 built-in command. Parameter **$0** is set from argument zero when the shell is invoked.

The character **$** is used to introduce substitutable *parameters*.

${*parameter***}**

The shell reads all the characters from **${** to the matching **}** as part of the same word even if it contains braces or metacharacters. The value, if any, of the parameter is substituted. The braces are required when*p arameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name, when the variable name contains a **.**. The braces are also required when a variable is subscripted unless it is part of an Arithmetic Expression or a Conditional Expression. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is * or **@**, then all the positional parameters, starting with **$1**, are substituted (separated by a field separator character). If an array *vname* with last subscript * **@**, or for index arrays of the form *sub1* **..** *sub2.* is used, then the value for each of the elements between *sub1* and *sub2* inclusive (or all elements for * and **@**) is substituted, separated by the first character of the value of **IFS**.

${#*parameter***}**

If *parameter* is * or **@**, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

${#*vname***[*]}**

${#*vname***[@]}**

The number of elements in the array *vname* is substituted.

${@*vname***}**

Expands to the type name (See *Type Variables* below) or attributes of the variable referred to by *vname.*

${!*vname***}**

Expands to the name of the variable referred to by *vname.* This will be*vname* except when *vname* is a name reference.

${!*vname***[***subscript***]}**

Expands to name of the subscript unless *subscript* is *****, **@**. or of the form*sub1* **..** *sub2.* When *subscript* is *****, the list of array subscripts for *vname* is generated. For a variable that is not an array, the value is 0 if the variable is set. Otherwise it is null. When*subscript* is **@**, same as above, except that when used in double quotes, each array subscript yields a separate argument. When*subscript* is of the form *sub1* **..** *sub2* it expands to the list of subscripts between *sub1* and *sub2* inclusive using the same quoting rules as **@**.

${!*prefix****}**

Expands to the names of the variables whose names begin with *prefix.*

${*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 not interactive). If*wor d* is omitted then a standard message 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:

print ${d:-$(pwd)}

If the colon (**:**) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

${*parameter***:***offset***:***length***}**

${*parameter***:***offset***}**

Expands to the portion of the value of *parameter* starting at the character (counting from **0**) determined by expanding *offset* as an arithmetic expression and consisting of the

number of characters determined by the arithmetic expression defined by *length.* In the second form, the remainder of the value is used. If A negative *offset* counts backwards from the end of *parameter.* Note that one or more*blank*s is required in fron t of a minus sign to prevent the shell from interpreting the operator as **:-**. If*p arameter* is ∗ or **@**, or is an array name indexed by ∗ or **@**, then *offset* and *length* refer to the array index and number of elements respectively. A negative *offset* is taken relative to one greater than the highest subscript for indexed arrays. The order for associate arrays is unspecified.

${*parameter***#***pattern***}**
${*parameter***##***pattern***}**

If the shell *pattern* matches the beginning of the value of *parameter*, then the value of this expansion 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 second form the largest matching pattern is deleted. When *parameter* is **@**, *****, or an array variable with subscript **@** or *****, the substring operation is applied to each element in turn.

${*parameter***%***pattern***}**
${*parameter***%%***pattern***}**

If the shell *pattern* matches the end of the value of *parameter*, then the value of this expansion is the value of the *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 second form the largest matching pattern is deleted. When *parameter* is **@**, *****, or an array variable with subscript **@** or *****, the substring operation is applied to each element in turn.

${*parameter***/***pattern***/***string***}**
${*parameter***//***pattern***/***string***}**
${*parameter***/#***pattern***/***string***}**
${*parameter***/%***pattern***/***string***}**

Expands *parameter* and replaces the longest match of *pattern* with the given *string.* Each occurrence of *n* in *string* is replaced by the portion of *parameter* that matches the *n*-th sub-pattern. In the first form, only the first occurrence of *pattern* is replaced. In the second form, each match for *pattern* is replaced by the given *string.* The third form restricts the pattern match to the beginning of the string while the fourth form restricts the pattern match to the end of the string. When *string* is null, the *pattern* will be deleted and the **/** in front of *string* may be omitted. When *parameter* is **@**, *****, or an array variable with subscript **@** or *****, the substitution operation is applied to each element in turn. In this case, the *string* portion of *word* will be re-evaluated for each element.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Options 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.

_ Initially, the value of _ is an absolute pathname of the shell or script being executed as passed in the *environment.* Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail. While defining a compound variable or a type, _ is initialized as a reference to the compound variable or type. When a discipline function is invoked, _ is initialized as a reference to the variable associated with the call to this function. Finally when _ is used as the name of the first variable of a type definition, the new type is derived from the type of the first variable (See *Type Variables* below.).

! The process id or the pool name and job number of the last background command invoked or the most recent job put in the background with the **bg** built-in command. Background jobs started in a named pool will be in the form *pool***.***number* where *pool* is the pool name and *number* is the job number within that pool.

.sh.command
When processing a **DEBUG** trap, this variable contains the current command line that is about to run.

.sh.edchar
This variable contains the value of the keyboard character (or sequence of characters if the first character is an ESC, ascii **033**) that has been entered when processing a **KEYBD** trap (see*Key Bindings* belo w). If the value is changed as part of the trap action, then the new value replaces the key (or key sequence) that caused the trap.

.sh.edcol
The character position of the cursor at the time of the most recent **KEYBD** trap.

.sh.edmode
The value is set to ESC when processing a **KEYBD** trap while in**vi** insert mode. (See *Vi Editing Mode* below.) Otherwise,**.sh.edmode** is n ull when processing a **KEYBD** trap.

.sh.edtext
The characters in the input buffer at the time of the most recent **KEYBD** trap. The value is null when not processing a **KEYBD** trap.

.sh.file
The pathname of the file than contains the current command.

.sh.fun
The name of the current function that is being executed.

.sh.level
Set to the current function depth. This can be changed inside a DEBUG trap and will set the context to the specified level.

.sh.lineno
Set during a DEBUG trap to the line number for the caller of each function.

.sh.match
An indexed array which stores the most recent match and sub-pattern matches after conditional pattern matches that match and after variables expansions using the operators #, %, or /. The**0**-th elemen t stores the complete match and the *i*-th. element stores the *i*-th submatch. The**.sh.matc h** variable becomes unset when the variable that has expanded is assigned a new value.

.sh.math
Used for defining arithmetic functions (see *Arithmetic evaluation* below). and stores the list of user defined arithmetic functions.

.sh.name
Set to the name of the variable at the time that a discipline function is invoked.

.sh.subscript
Set to the name subscript of the variable at the time that a discipline function is invoked.

.sh.subshell
The current depth for subshells and command substitution.

.sh.value
Set to the value of the variable at the time that the **set** or **append** discipline function is invoked. When a user defined arithmetic function is invoked, the value of **.sh.value** is saved and **.sh.value** is set to long double precision floating point. **.sh.value** is restored when the function returns.

.sh.version
Set to a value that identifies the version of this shell.

KSH_VERSION
A name reference to **.sh.version**.

LINENO
The current line number within the script or function being executed.

OLDPWD
The previous working directory set by the **cd** command.

OPTARG
The value of the last option argument processed by the **getopts** built-in command.

OPTIND
The index of the last option argument processed by the **getopts** built-in command.

PPID The process number of the parent of the shell.

PWD The present working directory set by the **cd** command.

RANDOM
Each time this variable is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.

REPLY
This variable is set by the **select** statement and by the **read** built-in command when no arguments are supplied.

SECONDS
Each time this variable is referenced, the number of seconds since shell invocation is returned. If this variable is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

SHLVL
An integer variable the is incremented each time the shell is invoked and is exported. If **SHLVL** is not in the environment when the shell is invoked, it is set to 1.

The following variables 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 **VISUAL** variable is not set, the value of this variable will be checked for the patterns as described with **VISUAL** below and the corresponding editing option (see Special Command **set** below) will be turned on.

ENV If this variable is set, then parameter expansion, command substitution, and arithmetic substitution are performed on the value to generate the pathname of the script that will be executed when the shell is invoked interactively (see *Invocation* below). This file is typically used for **alias** and **function** definitions. The default value is **$HOME/.kshrc**. On systems that support a system wide **/etc/ksh.kshrc** initialization file, if the filename generated by the expansion of **ENV** begins with**/./** or **././** the system wide initialization file will not be executed.

FCEDIT
Obsolete name for the default editor name for the **hist** command. **FCEDIT** is not used when **HISTEDIT** is set.

FIGNORE
A pattern that defines the set of filenames that will be ignored when performing filename matching.

FPATH
The search path for function definitions. The directories in this path are searched for a file with the same name as the function or command when a function with the **-u** attribute is referenced and when a command is not found. If an executable file with the name of that command is found, then it is read and executed in the current environment. Unlike **PATH**, the current directory must be represented explicitly by **.** rather than by adjacent **:** characters or a beginning or ending **:**.

HISTCMD
Number of the current command in the history file.

HISTEDIT
Name for the default editor name for the **hist** command.

HISTFILE
If this variable 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 variable 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 512.

HOME
The default argument (home directory) for the **cd** command.

IFS Internal field separators, normally **space**, **tab**, and **new-line** that are used to separate the results of command substitution or parameter expansion and to separate fields with the built-in command **read**. The first character of the **IFS** variable is used to separate arguments for the **$*** substitution (see *Quoting* below). Each single occurrence of an **IFS** character in the string to be split, that is not in the *isspace* character class, and any adjacent characters in **IFS** that are in the*isspace* character class, delimit a field. One or more characters in **IFS** that belong to the *isspace* character class, delimit a field. In addition, if the same *isspace* character appears consecutively inside **IFS**, this character is treated as if it were not in the *isspace* class, so that if **IFS** consists of two **tab** characters, then two adjacent **tab** characters delimit a null field.

JOBMAX
This variable defines the maximum number running background jobs that can run at a time. When this limit is reached, the shell will wait for a job to complete before staring a new job.

LANG This variable determines the locale category for any category not specifically selected with a variable starting with **LC_** or **LANG**.

LC_ALL
This variable overrides the value of the **LANG** variable and any other **LC_** variable.

LC_COLLATE
This variable determines the locale category for character collation information.

LC_CTYPE
This variable determines the locale category for character handling functions. It determines the character classes for pattern matching (see *File Name Generation* below).

LC_NUMERIC
This variable determines the locale category for the decimal point character.

LINES If this variable is set, the value is used to determine the column length for printing **select** lists. Select lists will print vertically until about two-thirds of **LINES** lines are filled.

MAIL If this variable is set to the name of a mail file *and* the **MAILPATH** variable 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** variables. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.

MAILPATH
A colon (**:**) separated list of file names. If this variable 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 expansion, command substitution, and arithmetic substitution with the variable **$_** 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**rksh** (except in **.profile).**

PS1 The value of this variable is expanded for parameter expansion, command substitution, and arithmetic 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). Two successive occurrences of **!** will produce a single **!** when the prompt string is printed.

PS2 Secondary prompt string, by default "> ".

PS3 Selection prompt string used within a **select** loop, by default "**#?** ".

PS4 The value of this variable is expanded for parameter evaluation, command substitution, and arithmetic substitution and precedes each line of an execution trace. By default, **PS4** is "+ ". In addition when **PS4** is unset, the execution trace prompt is also "+ ".

SHELL
The pathname of the *shell* is kept in the environment. At invocation, if the basename of this variable is **rsh**, **rksh**, or **krsh**, then the shell becomes restricted. If it is **pfsh** or **pfksh**, then the shell becomes a profile shell (see *pfexec(1)).*

TIMEFORMAT
The value of this parameter is used as a format string specifying how the timing information for pipelines prefixed with the **time** reserved word should be displayed. The% c haracter introduces a format sequence that is expanded to a time value or other information. The format sequences and their meanings are as follows.

%%	A literal %.
%[*p*][**l**]**R**	The elapsed time in seconds.
%[*p*][**l**]**U**	The number of CPU seconds spent in user mode.
%[*p*][**l**]**S**	The number of CPU seconds spent in system mode.
%P	The CPU percentage, computed as (U + S) / R.

The brackets denote optional portions. The optional *p* is a digit specifying the *precision*, the number of fractional digits after a decimal point. A value of 0 causes no decimal point or fraction to be output. At most three places after the decimal point can be displayed; values of *p* greater than 3 are treated as 3. If *p* is not specified, the value 3 is used.

The optional **l** specifies a longer format, including hours if greater than zero, minutes, and seconds of the form *HH*h*MM*m*SS.FF*s. The value of *p* determines

whether or not the fraction is included.

All other characters are output without change and a trailing newline is added. If unset, the default value, **$'nrealt%2lRnusert%2lUnsys%2lS'**, is used. If the value is null, no timing information is displayed.

TMOUT
If set to a value greater than zero, **TMOUT** will be the default timeout value for the **read** built-in command. The **select** compound command terminates after **TMOUT** seconds when input is from a terminal. Otherwise, the shell will terminate if a line is not entered within the prescribed number of seconds while reading from a terminal. (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 matches the pattern **[Vv][Ii]**, then the **vi** option (see Special Command **set** below) is turned on. If the value matches the pattern **gmacs** , the **gmacs** option is turned on. If the value matches the pattern **macs**, then the **emacs** option will be turned on. The value of **VISUAL** overrides the value of **EDITOR.**

The shell gives default values to **PATH**, **PS1**, **PS2**, **PS3**, **PS4**, **MAILCHECK**, **FCEDIT**, **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).

Field Splitting.
After parameter expansion and command substitution, the results of substitutions are scanned for the field separator characters (those found in **IFS**) and split into distinct fields where such characters are found. Explicit null fields (or ′′) are retained. Implicit null fields (those resulting from *parameters* that have no values or command substitutions with no output) are removed.

If the **braceexpand** (**-B**) option is set then each of the fields resulting from **IFS** are checked to see if they contain one or more of the brace patterns {*****,*****}, {*l1*..*l2*} , {*n1*..*n2*} , {*n1*..*n2*% *fmt*} , {*n1*..*n2* ..*n3*} , or {*n1*..*n2* ..*n3*%*fmt*} , where ***** represents any character, *l1*,*l2* are letters and *n1*,*n2*,*n3* are signed numbers and *fmt* is a format specified as used by **printf**. In each case, fields are created by prepending the characters before the { and appending the characters after the } to each of the strings generated by the characters between the { and }. The resulting fields are checked to see if they have any brace patterns.

In the first form, a field is created for each string between { and **,**, between **,** and **,**, and between **,** and }. The string represented by ***** can contain embedded matching { and } without quoting. Otherwise, each { and } with ***** must be quoted.

In the seconds form, *l1* and *l2* must both be either upper case or both be lower case characters in the C locale. In this case a field is created for each character from *l1* thru *l2*.

In the remaining forms, a field is created for each number starting at *n1* and continuing until it reaches *n2* incrementing *n1* by *n3*. The cases where*n3* is not specified beha ve as if *n3* where **1** if *n1*<=*n2* and **-1** otherwise. If forms which specify **%***fmt* any format flags, widths and precisions can be specified and *fmt* can end in any of the specifiers **cdiouxX**. For example, **{a,z}{1..5..3%02d}{b..c}x** expands to the 8 fields, **a01bx**, **a01cx**, **a04bx**, **a04cx**, **z01bx**, **z01cx**, **z04bx** and **z4cx**.

File Name Generation.
Following splitting, each field 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*. Each file name component that contains any pattern character is replaced with a lexicographically sorted set of names that matches the pattern from that directory. If no file name is found that matches the pattern, then that component of the filename is left unchanged unless the pattern is prefixed with **~(N)** in which case it is removed as described below. If **FIGNORE** is set, then each file

name component that matches the pattern defined by the value of **FIGNORE** is ignored when generating the matching filenames. The names **.** and **..** are also ignored. If **FIGNORE** is not set, the character **.** at the start of each file name component will be ignored unless the first character of the pattern corresponding to this component is the character **.** itself. Note, that for other uses of pattern matching the **/** and **.** are not treated specially.

- ***** Matches any string, including the null string. When used for filename expansion, if the **globstar** option is on, two adjacent *****'s by itself will match all files and zero or more directories and subdirectories. If followed by a **/** then only directories and subdirectories will match.
- **?** 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 **!** or **^** then any character not enclosed is matched. A **-** can be included in the character set by putting it as the first or last character.

  Within **[** and **]**, character classes can be specified with the syntax **[:***class***:]** where class is one of the following classes defined in the ANSI-C standard: (Note that **word** is equivalent to **alnum** plus the character _.)

  alnum alpha blank cntrl digit graph lower print punct space upper word xdigit

  Within **[** and **]**, an equivalence class can be specified with the syntax **[=***c***=]** which matches all characters with the same primary collation weight (as defined by the current locale) as the character *c*. Within**[** and **]**, **[.***symb ol***.]** matches the collating symbol *symbol*.

A *pattern-list* is a list of one or more patterns separated from each other with a **&** or **|**. A**&** signifies that all patterns must be matched whereas **|** requires that only one pattern be matched. Composite patterns can be formed with one or more of the following sub-patterns:

- **?(***pattern-list***)** Optionally matches any one of the given patterns.
- ***(***pattern-list***)** Matches zero or more occurrences of the given patterns.
- **+(***pattern-list***)** Matches one or more occurrences of the given patterns.
- **{***n***}(***pattern-list***)** Matches *n* occurrences of the given patterns.
- **{***m***,***n***}(***pattern-list***)** Matches from *m* to *n* occurrences of the given patterns. If *m* is omitted, **0** will be used. If *n* is omitted at least *m* occurrences will be matched.
- **@(***pattern-list***)** Matches exactly one of the given patterns.
- **!(***pattern-list***)** Matches anything except one of the given patterns.

By default, each pattern, or sub-pattern will match the longest string possible consistent with generating the longest overall match. If more than one match is possible, the one starting closest to the beginning of the string will be chosen. However, for each of the above compound patterns a **-** can be inserted in front of the **(** to cause the shortest match to the specified *pattern-list* to be used.

When *pattern-list* is contained within parentheses, the backslash character is treated specially even when inside a character class. All ANSI-C character escapes are recognized and match the specified character. In addition the following escape sequences are recognized:

- **d** Matches any character in the **digit** class.
- **D** Matches any character not in the **digit** class.
- **s** Matches any character in the **space** class.

S Matches any character not in the **space** class.
w Matches any character in the **word** class.
W Matches any character not in the **word** class.

A pattern of the form **%(***pattern-pair*(s)**)** is a sub-pattern that can be used to match nested character expressions. Each *pattern-pair* is a two character sequence which cannot contain **&** or **|**. The first *pattern-pair* specifies the starting and ending characters for the match. Each subsequent *pattern-pair* represents the beginning and ending characters of a nested group that will be skipped over when counting starting and ending character matches. The behavior is unspecified when the first character of a *pattern-pair* is alpha-numeric except for the following:

D Causes the ending character to terminate the search for this pattern without finding a match.
E Causes the ending character to be interpreted as an escape character.
L Causes the ending character to be interpreted as a quote character causing all characters to be ignored when looking for a match.
Q Causes the ending character to be interpreted as a quote character causing all characters other than any escape character to be ignored when looking for a match.

Thus, **%({}QE)**, matches characters starting at { until the matching } is found not counting any { or } that is inside a double quoted string or preceded by the escape character . Without the {} this pattern matches any C language string.

Each sub-pattern in a composite pattern is numbered, starting at 1, by the location of the **(** within the pattern. The sequence *n*, where *n* is a single digit and *n* comes after the *n*-th. sub-pattern, matches the same string as the sub-pattern itself.

Finally a pattern can contain sub-patterns of the form **~(***options***:***pattern-list***)**, where either *options* or **:***pattern-list* can be omitted. Unlike the other compound patterns, these sub-patterns are not counted in the numbered sub-patterns. **:***p attern-list* must be omitted for options **F**, **G**, **N** , and **V** below. If*options* is presen t, it can consist of one or more of the following:

+ Enable the following options. This is the default.
- Disable the following options.
E The remainder of the pattern uses extended regular expression syntax like the egrep(1) command.
F The remainder of the pattern uses fgrep(1) expression syntax.
G The remainder of the pattern uses basic regular expression syntax like the grep(1) command.
K The remainder of the pattern uses shell pattern syntax. This is the default.
N This is ignored. However, when it is the first letter and is used with file name generation, and no matches occur, the file pattern expands to the empty string.
X The remainder of the pattern uses augmented regular expression syntax like the *xgrep(1)* command.
P The remainder of the pattern uses perl(1) regular expression syntax. Not all perl regular expression syntax is currently implemented.
V The remainder of the pattern uses System V regular expression syntax.
i Treat the match as case insensitive.
g File the longest match (greedy). This is the default.
l Left anchor the pattern. This is the default for **K** style patterns.
r Right anchor the pattern. This is the default for **K** style patterns.

If both *options* and **:***pattern-list* are specified, then the options apply only to *pattern-list*. Otherwise, these options remain in effect until they are disabled by a subsequent **~(***...***)** or at the end of the sub-pattern containing **~(***...***)**.

Quoting.

Each of the *metacharacters* listed earlier (see *Definitions* above) has a special meaning to the shell and causes 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 removed. All characters enclosed

between a pair of single quote marks (′′) that is not preceded by a **$** are quoted. A single quote cannot appear within the single quotes. A single quoted string preceded by an unquoted **$** is processed as an ANSI-C string except for the following:

0 Causes the remainder of the string to be ignored.
E Equivalent to the escape character (ascii **033**),
e Equivalent to the escape character (ascii **033**),
c*x* Expands to the character control-*x*.
C[**.***name***.**]
Expands to the collating element *name*.

Inside double quote marks (), parameter and command substitution occur and quotes the characters , ` , , and **$**. A**$** in fron t of a double quoted string will be ignored in the C or POSIX locale, and may cause the string to be replaced by a locale specific string otherwise. The meaning of **$***∗* and **$@** is identical when not quoted or when used as a variable assignment value or as a file name. However, when used as a command argument, **$***∗* is equivalent to **$1***d***$2***d*..., where *d* is the first character of the **IFS** variable, whereas **$@** is equivalent to **$1 $2** .... Inside grave quote marks (` `), quotes the characters , `, and **$**. If the grave quotes occur within double quotes, then also quotes the character .

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or built-in command names listed below cannot be altered by quoting them.

Arithmetic Evaluation.

The shell performs arithmetic evaluation for arithmetic substitution, to evaluate an arithmetic command, to evaluate an indexed array subscript, and to evaluate arguments to the built-in commands **shift** and **let**. Evaluations are performed using double precision floating point arithmetic or long double precision floating point for systems that provide this data type. Floating point constants follow the ANSI-C programming language floating point conventions. The floating point constants **Nan** and **Inf** can be use to represent not a number and infinity respectively. Integer constants follow the ANSI-C programming language integer constant conventions although only single byte character constants are recognized and character casts are not recognized. In addition constants can be of the form [*base*#]*n* where *base* is a decimal number between two and sixty-four representing the arithmetic base and *n* is a number in that base. The digits above 9 are represented by the lower case letters, the upper case letters, @, and _ respectively. For bases less than or equal to 36, upper and lower case characters can be used interchangeably.

An arithmetic expression uses the same syntax, precedence, and associativity of expression as the C language. All the C language operators that apply to floating point quantities can be used. In addition, the operator ****** can be used for exponentiation. It has higher precedence than multiplication and is left associative. In addition, when the value of an arithmetic variable or sub-expression can be represented as a long integer, all C language integer arithmetic operations can be performed. Variables can be referenced by name within an arithmetic expression without using the parameter expansion syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

Any of the following math library functions that are in the C math library can be used within an arithmetic expression:

abs acos acosh asin asinh atan atan2 atanh cbrt ceil copysign cos cosh erf erfc exp exp2 expm1 fabs fpclassify fdim finite floor fma fmax fmod j0 j1 jn hypot ilogb int isfinite isinf isnan isnormal issubnormal issubordered iszero lgamma log log10 log2 logb nearbyint nextafter nexttoward pow rint round scalb signbit sin sinh sqrt tan tanh tgamma trunc y0 y1 yn

In addition, arithmetic functions can be define as shell functions with a variant of the **function** *name* syntax,

function .sh.math.*name ident ...* { *list* **;}**

where *name* is the function name used in the arithmetic expression and each identifier, *ident* is a name reference to the long double precision floating point argument. The value of **.sh.value** when the function returns is the value of this function. User defined functions can take up to 3 arguments and override C math library functions.

An internal representation of a *variable* as a double precision floating point can be specified with the **-E** [*n*], **-F** [*n*], or **-X** [*n*] option of the **typeset** special built-in command. The **-E** option causes the expansion of the value to be represented using scientific notation when it is expanded. The optional option argument *n* defines the number of significant figures. The **-F** option causes the expansion to be represented as a floating decimal number when it is expanded. The **-X** option cause the expansion to be represented using the **%a** format defined by ISO C-99. The optional option argument *n* defines the number of places after the decimal (or radix) point in this case.

An internal integer representation of a *variable* can be specified with the **-i** [*n*] option of the **typeset** special built-in command. The optional option argument *n* specifies an arithmetic base to be used when expanding the variable. If you do not specify an arithmetic base, base 10 will be used.

Arithmetic evaluation is performed on the value of each assignment to a variable with the **-E**, **-F**, **-X**, or **-i** attribute. Assigning a floating point number to a variable whose type is an integer causes the fractional part to be truncated.

Prompting.

When used interactively, the shell prompts with the value of **PS1** after expanding it for parameter expansion, command substitution, and arithmetic substitution, before reading a command. In addition, each single **!** in the prompt is replaced by the command number. A**!!** is required to place **!** in the prompt. 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.

Conditional Expressions.

A *conditional expression* is used with the [[compound command to test attributes of files and to compare strings. Field splitting and file name generation are not performed on the words between [[and]]. Each expression can be constructed from one or more of the following unary or binary expressions:

string True, if *string* is not null.
-a *file* Same as **-e** below. This is obsolete.
-b *file* True, if *file* exists and is a block special file.
-c *file* True, if *file* exists and is a character special file.
-d *file* True, if *file* exists and is a directory.
-e *file* True, if *file* exists.
-f *file* True, if *file* exists and is an ordinary file.
-g *file* True, if *file* exists and it has its setgid bit set.
-k *file* True, if *file* exists and it has its sticky bit set.
-n *string*
True, if length of *string* is non-zero.
-o ?*option*
True, if option named *option* is a valid option name.
-o *option*
True, if option named *option* is on.
-p *file* True, if *file* exists and is a fifo special file or a pipe.
-r *file* True, if *file* exists and is readable by current process.
-s *file* True, if *file* exists and has size greater than zero.
-t *fildes*
True, if file descriptor number *fildes* is open and associated with a terminal device.

-u *file* True, if *file* exists and it has its setuid bit set.

-v *name*
True, if variable *name* is a valid variable name and is set.

-w *file* True, if *file* exists and is writable by current process.

-x *file* True, if *file* exists and is executable by current process. If *file* exists and is a directory, then true if the current process has permission to search in the directory.

-z *string*
True, if length of *string* is zero.

-L *file* True, if *file* exists and is a symbolic link.

-h *file* True, if *file* exists and is a symbolic link.

-N *file* True, if *file* exists and the modification time is greater than the last access time.

-O *file* True, if *file* exists and is owned by the effective user id of this process.

-G *file* True, if *file* exists and its group matches the effective group id of this process.

-R *name*
True if variable *name* is a name reference.

-S *file* True, if *file* exists and is a socket.

file1 **-nt** *file2*
True, if *file1* exists and *file2* does not, or *file1* is newer than *file2*.

file1 **-ot** *file2*
True, if *file2* exists and *file1* does not, or *file1* is older than *file2*.

file1 **-ef** *file2*
True, if *file1* and *file2* exist and refer to the same file.

string == *pattern*
True, if *string* matches *pattern*. Any part of *pattern* can be quoted to cause it to be matched as a string. With a successful match to a pattern, the **.sh.match** array variable will contain the match and sub-pattern matches.

string = *pattern*
Same as == above, but is obsolete.

string **!=** *pattern*
True, if *string* does not match *pattern*. When the*string* matc hes the *pattern* the **.sh.match** array variable will contain the match and sub-pattern matches.

string =~ *ere*
True if *string* matches the pattern ~**(E)***ere* where *ere* is an extended regular expression.

string1 < *string2*
True, if *string1* comes before *string2* based on ASCII value of their characters.

string1 > *string2*
True, if *string1* comes after *string2* based on ASCII value of their characters.

The following obsolete arithmetic comparisons are also permitted:

exp1 **-eq** *exp2*
True, if *exp1* is equal to *exp2*.

exp1 **-ne** *exp2*
True, if *exp1* is not equal to *exp2*.

exp1 **-lt** *exp2*
True, if *exp1* is less than *exp2*.

exp1 **-gt** *exp2*
True, if *exp1* is greater than *exp2*.

exp1 **-le** *exp2*
True, if *exp1* is less than or equal to *exp2*.

exp1 **-ge** *exp2*
True, if *exp1* is greater than or equal to *exp2*.

In each of the above expressions, if *file* is of the form **/dev/fd/***n*, where *n* is an integer, then the test is applied to the open file whose descriptor number is *n*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(*expression*)
: True, if *expression* is true. Used to group expressions.

! *expression*
: True if *expression* is false.

expression1 **&&** *expression2*
: True, if *expression1* and *expression2* are both true.

expression1 **||** *expression2*
: True, if either *expression1* or *expression2* is true.

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 substitution, parameter expansion, and arithmetic substitution occur before *word* or *digit* is used except as noted below. File name generation occurs only if the shell is interactive and the pattern matches a single file. Field splitting is not performed.

In each of the following redirections, if *file* is of the form **/dev/sctp/***host*/*port*, **/dev/tcp/***host*/*port*, or **/dev/udp/***host*/*port*, where *host* is a hostname or host address, and *port* is a service given by name or an integer port number, then the redirection attempts to make a **tcp**, **sctp** or **udp** connection to the corresponding socket.

No intervening space is allowed between the characters of redirection operators.

<*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. If the file exists, and the **noclobber** option is on, this causes an error; otherwise, it is truncated to zero length.

>|*word*
: Same as >, except that it overrides the **noclobber** option.

>**;***word*
: Write output to a temporary file. If the command completes successfully rename it to *word*, otherwise, delete the temporary file. >**;***wor d* cannot be used with the *exec(2).* built-in.

>>*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*
: Open file *word* for reading and writing as standard output.

<>**;***word*
: The same as <>*word* except that if the command completes successfully, *word* is truncated to the offset at command completion. <>**;***wor d* cannot be used with the *exec(2).* built-in.

<<[**-**]*word*
: The shell input is read up to a line that is the same as *word* after any quoting has been removed, or to an end-of-file. No parameter substitution, command substitution, arithmetic 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 expansion, command substitution, and arithmetic substitution occur, **new-line** is ignored, and must be used to quote the characters , **$**, **`** . If**-** is appended to <<, then all leading tabs are stripped from *word* and from the document. If**#** is appended to <<, then leading spaces and tabs will be stripped off the first line of the document and up to an equivalent indentation will be stripped from the remaining lines and from *word*. A tab stop is assumed to occur at every 8 columns for the purposes of determining the indentation.

<<<*word*
: A short form of here document in which *word* becomes the contents of the here-document after any parameter expansion, command substitution, and arithmetic substitution occur.

<&*digit*	The standard input is duplicated from file descriptor *digit* (see dup(2)). Similarly for the standard output using **>&** *digit*.
<&*digit***-**	The file descriptor given by *digit* is moved to standard input. Similarly for the standard output using **>&** *digit***-**.
<&-	The standard input is closed. Similarly for the standard output using **>&-**.
<&p	The input from the co-process is moved to standard input.
>&p	The output to the co-process is moved to standard output.
<# ((*expr*))	Evaluate arithmetic expression *expr* and position file descriptor 0 to the resulting value bytes from the start of the file. The variables **CUR** and **EOF** evaluate to the current offset and end-of-file offset respectively when evaluating *expr*.
># ((*offset*))	The same as **<#** except applies to file descriptor 1.
<#*pattern*	Seeks forward to the beginning of the next line containing *pattern*.
<##*pattern*	The same as **<#** except that the portion of the file that is skipped is copied to standard output.

If one of the above is preceded by a digit, with no intervening space, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). If one of the above, other than **>&-** and the **>#** and **<#** forms, is preceded by {*varname*} with no intervening space, then a file descriptor number > 10 will be selected by the shell and stored in the variable *varname*. If**>&-** or the an y of the **>#** and **<#** forms is preceded by {*varname*} the value of *varname* defines the file descriptor to close or position. For example:

... 2>&1

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1 and

exec {n}<file

means open file named **file** for reading and store the file descriptor number in variable **n**.

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 w ere 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 variable for each name found, giving it the corresponding value and attributes and marking it *export*. Executed commands inherit the environment. If the user modifies the values of these variables 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 variable assignments. A variable assignment argument is a word of the form *identifier=value*. Thus:

TERM=450 cmd argsand
(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned except for special built-in commands listed below - those that are preceded with a dagger).

If the obsolete **-k** option is set, *all* variable 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

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions.

For historical reasons, there are two ways to define functions, the *name*() syntax and the **function** *name* syntax, described in the *Commands* section above. 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 defined by the **function** *name* syntax and called by name execute in the same process as the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on **EXIT** set inside a function is executed in the environment of the caller after the function completes. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special built-in command used within a function defines local variables whose scope includes the current function. They can be passed to functions that they call in the variable assignment list that precedes the call or as arguments passed as name references. Errors within functions return control to the caller.

Functions defined with the *name*() syntax and functions defined with the **function** *name* syntax that are invoked with the **.** special built-in are executed in the caller's environment and share all variables and traps with the caller. Errors within these function executions cause the script that contains them to abort.

The special built-in command **return** is used to return from function calls.

Function names can be listed with the **-f** or **+f** option of the **typeset** special built-in command. The text of functions, when available, will also be listed with **-f**. Functions can be undefined with the **-f** option of the **unset** special built-in command.

Ordinarily, functions are unset when the shell executes a shell script. Functions that need to be defined across separate invocations of the shell should be placed in a directory and the **FPATH** variable should contain the name of this directory. They may also be specified in the **ENV** file.

Discipline Functions.

Each variable can have zero or more discipline functions associated with it. The shell initially understands the discipline names **get**, **set**, **append**, and **unset** but can be added when defining new types. On most systems others can be added at run time via the C programming interface extension provided by the **builtin** built-in utility. If the**get** discipline is defined for a v ariable, it is invoked whenever the given variable is referenced. If the variable **.sh.value** is assigned a value inside the discipline function, the referenced variable will evaluate to this value instead. If the **set** discipline is defined for a variable, it is invoked whenever the given variable is assigned a value. If the **append** discipline is defined for a variable, it is invoked whenever a value is appended to the given variable. The variable **.sh.value** is given the value of the variable before invoking the discipline, and the variable will be assigned the value of **.sh.value** after the discipline completes. If

.sh.value is unset inside the discipline, then that value is unchanged. If the**unset** discipline is defined for a variable, it is invoked whenever the given variable is unset. The variable will not be unset unless it is unset explicitly from within this discipline function.

The variable **.sh.name** contains the name of the variable for which the discipline function is called, **.sh.subscript** is the subscript of the variable, and **.sh.value** will contain the value being assigned inside the **set** discipline function. The variable _ is a reference to the variable including the subscript if any. For the **set** discipline, changing **.sh.value** will change the value that gets assigned. Finally, the expansion **${***var***.***name***}**, when *name* is the name of a discipline, and there is no variable of this name, is equivalent to the command substitution **${** *var***.***name***;}**.

Name Spaces.

Commands and functions that are executed as part of the *list* of a **namespace** command that modify variables or create new ones, create a new variable whose name is the name of the name space as given by *identifier* preceded by **.**. When a variable whose name is *name* is referenced, it is first searched for using **.***identifier***.***name*. Similarly, a function defined by a command in the **namespace** *list* is created using the name space name preceded by a **.**.

When the *list* of a **namespace** command contains a **namespace** command, the names of variables and functions that are created consist of the variable or function name preceded by the list of *identifier*s each preceded by **.**.

Outside of a name space, a variable or function created inside a name space can be referenced by preceding it with the name space name.

By default, variables staring with **.sh** are in the **sh** name space.

Type Variables.

Typed variables provide a way to create data structure and objects. A type can be defined either by a shared library, by the **enum** built-in command described below, or by using the new **-T** option of the **typeset** built-in command. With the **-T** option of **typeset**, the type name, specified as an option argument to **-T**, is set with a compound variable assignment that defines the type. Function definitions can appear inside the compound variable assignment and these become discipline functions for this type and can be invoked or redefined by each instance of the type. The function name **create** is treated specially. It is invoked for each instance of the type that is created but is not inherited and cannot be redefined for each instance.

When a type is defined a special built-in command of that name is added. These built-ins are declaration commands and follow the same expansion rules as all the special built-in commands defined below that are preceded by . These commands can subsequently be used inside further type definitions. The man page for these commands can be generated by using the **--man** option or any of the other **--** options described with **getopts**. The**-r**, **-a**, **-A**, **-h**, and **-S** options of **typeset** are permitted with each of these new built-ins.

An instance of a type is created by invoking the type name followed by one or more instance names. Each instance of the type is initialized with a copy of the sub-variables except for sub-variables that are defined with the **-S** option. Variables defined with the **-S** are shared by all instances of the type. Each instance can change the value of any sub-variable and can also define new discipline functions of the same names as those defined by the type definition as well as any standard discipline names. No additional sub-variables can be defined for any instance.

When defining a type, if the value of a sub-variable is not set and the **-r** attribute is specified, it causes the sub-variable to be a required sub-variable. Whenever an instance of a type is created, all required sub-variables must be specified. These sub-variables become readonly in each instance.

When **unset** is invoked on a sub-variable within a type, and the **-r** attribute has not been specified for this field, the value is reset to the default value associative with the type. Invoking **unset** on a type instance not contained within another type deletes all sub-variables and the variable

itself.

A type definition can be derived from another type definition by defining the first sub-variable name as _ and defining its type as the base type. Any remaining definitions will be additions and modifications that apply to the new type. If the new type name is the same is that of the base type, the type will be replaced and the original type will no longer be accessible.

The **typeset** command with the **-T** and no option argument or operands will write all the type definitions to standard output in a form that that can be read in to create all they types.

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** tak es 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.

A job pool is a collection of jobs started with *list* **&** associated with a name.

There are several ways to refer to jobs in the shell. A job can be referred to by the process id of any process of the job or by one of the following:

%*number*
: The job with the given number.

pool All the jobs in the job pool named by *pool*.

pool.*number*
: The job number *number* in the job pool named by *pool*.

%*string*
: Any job whose command line begins with *string*.

%?*string*
: Any job whose command line contains *string*.

%% Current job.

%+ Equivalent to %%.

%- Previous job. In addition, unless noted otherwise, wherever a job can be specified, the name of a background job pool can be used to represent all the jobs in that pool.

The 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. The**notify** option of the **set** command causes the shell to print these job change messages as soon as they occur.

When the **monitor** option is on, each background job that completes triggers any trap set for **CHLD**.

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 immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated. When a login shell receives a HUP signal, it sends a HUP signal to each job that has not been disowned with the **disown** built-in command described below.

Signals.
The INT and QUIT signals for an invoked command are ignored if the command is followed by **&** and the **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the **trap** built-in command below).

Execution.
Each time a command is read, the above substitutions are carried out. If the command name matches one of the *Special Built-in Commands* listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches a user defined function. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. A function is also executed in the current shell process. When the *function* completes or issues a **return**, the positional parameter list is restored. For functions defined with the **function** *name* syntax, any trap set on **EXIT** within the function is executed. The exit value of a *function* is the value of the last command executed. If a command name is not a *special built-in command* or a user defined *function*, but it is one of the built-in commands listed below, it is executed in the current shell process.

The shell variables **PATH** followed by the variable **FPATH** defines the list of directories to search for the command name. Alternative directory names are separated by a colon (**:**). The default path is **/bin:/usr/bin:** (specifying **/bin**, **/usr/bin**, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a /, then the search path is not used. Otherwise, each directory in the list of directories defined by **PATH** and **FPATH** is checked in order. If the directory being searched is contained in **FPATH** and contains a file whose name matches the command being searched, then this file is loaded into the current shell environment as if it were the argument to the **.** command except that only preset aliases are expanded, and a function of the given name is executed as described above.

If this directory is not in **FPATH** the shell first determines whether there is a built-in version of a command corresponding to a given pathname and if so it is invoked in the current process. If no built-in is found, the shell checks for a file named **.paths** in this directory. If found and there is a line of the form **FPATH=***path* where *path* names an existing directory then that directory is searched after immediately after the current directory as if it were found in the **FPATH** variable. If *path* does not begin with /, it is checked for relative to the directory being searched.

The **.paths** file is then checked for a line of the form **PLUGIN_LIB=***libname* [**:** *libname*] ... . Each library named by *libname* will be searched for as if it were an option argument to **builtin -f**, and if it contains a built-in of the specified name this will be executed instead of a command by this name. Any built-in loaded from a library found this way will be associated with the directory containing the **.paths** file so it will only execute if not found in an earlier directory.

Finally, the directory will be checked for a file of the given name. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A separate shell is spawned to read it. All non-exported variables are removed in this case. If the shell command file doesn't have read permission, or if the *setuid* and/or *setgid* bits are set on the file, then the shell executes an agent whose job it is to set up the permissions and execute the shell with the shell command file passed down as an open file. If the **.paths** contains a line of the form *name=value* in the first or second line, then the environment variable *name* is modified by prepending the directory specified by *value* to the directory list. If *value* is not an absolute directory, then it specifies a directory relative to the directory that the executable was found. If the environment variable *name* does not already exist it will be added to the environment list for the specified command. A parenthesized command is executed in a sub-shell without removing non-exported variables.

Command Re-entry.

The text of the last **HISTSIZE** (default 512) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or if the file it names is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The built-in command**hist** is used to list or edit a portion of 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 **hist** then the value of the variable **HISTEDIT** is used. If **HISTEDIT** is unset, the obsolete variable **FCEDIT** is used. If **FCEDIT** is not defined, then**/bin/ed** is used. The edited command(s) is prin ted and re-executed upon leaving the editor unless you quit without writing. The **-s** option (and in obsolete versions, 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, with the preset alias **r**, which is aliased to **′hist -s′**, typing '**r bad=good c**' will re-execute the most recent command which starts with the letter **c**, replacing the first occurrence of 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 **newline** ('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.

Unless the **multiline** option is on, 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 window width is too small to display the prompt and leave at least 8 columns to enter input, the prompt is truncated from the left. 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.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

Each of the edit modes has an operation to list the files or commands that match a partially entered word. When applied to the first word on the line, or the first word after a **;**, **|**, **&**, or **(**, and the word does not begin with ~ or contain a /, the list of aliases, functions, and executable commands defined by the **PATH** variable that could match the partial word is displayed. Otherwise, the list of files that match the given word is displayed. If the partially entered word does not contain any file expansion characters, a ***** is appended before generating these lists. After displaying the generated list, the input line is redrawn. These operations are called command name listing and file name listing, respectively. There are additional operations, referred to as command name completion and file name completion, which compute the list of matching commands or files, but instead of printing the list, replace the current word with a complete or partial match. For file name completion, if the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory. Otherwise, the longest common prefix for all the matching files replaces the word. For command name completion, only the portion of the file names after the last / are used to find the longest command prefix. If only a single name matches this prefix, then the word is replaced with the command name followed by a space. When using a *tab* for completion that does not yield a unique match, a subsequent *tab* will provide a numbered list of matching alternatives. A specific selection can be made by entering the selection number followed by a *tab*.

Key Bindings.

The **KEYBD** trap can be used to intercept keys as they are typed and change the characters that are actually seen by the shell. This trap is executed after each character (or sequence of characters when the first character is ESC) is entered while reading from a terminal. The variable **.sh.edchar** contains the character or character sequence which generated the trap. Changing the value of **.sh.edchar** in the trap action causes the shell to behave as if the new value were entered from the keyboard rather than the original value.

The variable **.sh.edcol** is set to the input column number of the cursor at the time of the input. The variable **.sh.edmode** is set to ESC when in **vi** insert mode (see below) and is null otherwise. By prepending **${.sh.editmode}** to a value assigned to **.sh.edchar** it will cause the shell to change to control mode if it is not already in this mode.

This trap is not invoked for characters entered as arguments to editing directives, or while reading input for a character search.

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) k ey.)

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-[C	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The**emacs** editor's idea of a w ord is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-[D	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
M-[H	Move cursor to start of line.
^E	Move cursor to end of line.
M-[Y	Move cursor to end of line.
^]*char*	Move cursor forward to character *char* on current line.
M-^]*char*	Move cursor backward to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the stty(1) command, usually **^H** or #.) Delete previous character.
lnext	(User defined literal next character as defined by the stty(1) command, or **^V** if not defined.) Removes the next character's editing features (if any).
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is **^?** (DEL, the default) then this command will not work).
^T	Transpose current character with previous character and advance the cursor in *emacs* mode. Transpose two previous characters in *gmacs* mode.

^C Capitalize current character.
M-c Capitalize current word.
M-l Change the current word to lower case.
^K Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor position, then delete from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from cursor up to given cursor position.
^W Kill from the cursor to the mark.
M-p Push the region from the cursor to the mark on the stack.
kill (User defined kill character as defined by the stty command, usually **^G** or **@**.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y Restore last item removed from line. (Yank item back to the line.)
^L Line feed and print current line.
M-^L Clear the screen.
^@ (Null character) Set mark.
M-*space* (Meta space) Set mark.
^J (New line) Execute the current line.
^M (Return) Execute the current line.
eof End-of-file character, normally **^D**, is processed as an End-of-file only if the current line is null.
^P Fetch previous command. Each time **^P** is entered the previous command back in time is accessed. Moves back one line when not on the first line of a multi-line command.
M-[A If the cursor is at the end of the line, it is equivalent to **^R** with *string* set to the contents of the current line. Otherwise, it is equivalent to **^P.**
M-< Fetch the least recent (oldest) history line.
M-> Fetch the most recent (youngest) history line.
^N Fetch next command line. Each time **^N** is entered the next command line forward in time is accessed.
M-[B Equivalent to **^N.**
^R*string* Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a 'RETURN' or 'NEW LINE'. If string is preceded by a ^, the matched line must begin with *string*. If *string* is omitted, then the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
^O Operate - Execute the current line and fetch the next line relative to current line from the history file.
M-*digits* (Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are**^F**, **^B**, *er ase*, **^C**, **^D**, **^K**, **^R**, **^P**, **^N**, **^]**, **M-.**, **M-^]**, **M-_**, **M-=**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h**, **M-l** and **M-^H**.
M-*letter* Soft-key - Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the above meta-functions.
M-[*letter* Soft-key - Your alias list is searched for an alias by the name *__letter* and if an alias of this name is defined, its value will be inserted on the input queue. This can be used to program function keys on many terminals.
M-. The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_ Same as **M-.**.
M-* Attempt file name generation on the current word. An asterisk is appended if the word doesn't match any file or contain any special pattern characters.

M-ESC Command or file name completion as described above.

^I *tab* Attempts command or file name completion as described above. If a partial completion occurs, repeating this will behave as if **M-=** were entered. If no match is found or entered after *space*, a *tab* is inserted.

M-= If not preceded by a numeric parameter, it generates the list of matching commands or file names as described above. Otherwise, the word under the cursor is replaced by the item corresponding to the value of the numeric parameter from the most recently generated command or file list. If the cursor is not on a word, it is inserted instead.

^U Multiply parameter of next command by 4.

Escape next character. Editing characters, the user's erase, kill and interrupt (normally **^?**) 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).

M-^V Display version of the shell.

M-# If the line does not begin with a #, a # is inserted at the beginning of the line and after each new-line, and the line is entered. This causes a comment to be inserted in the history file. If the line begins with a #, the # is deleted and one # after each new-line is also deleted.

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.

erase (User defined erase character as defined by the stty command, usually **^H** or #.) Delete previous character.

^W Delete the previous blank separated word. On some systems the**vira w** option may be required for this to work.

eof As the first character of the line causes the shell to terminate unless the **ignoreeof** option is set. Otherwise this character is ignored.

lnext (User defined literal next character as defined by the stty(1) or **^V** if not defined.) Removes the next character's editing features (if any). On some systems the **viraw** option may be required for this to work.

Escape the next *erase* or *kill* character.

^I *tab* Attempts command or file name completion as described above and returns to input mode. If a partial completion occurs, repeating this will behave as if = were entered from control mode. If no match is found or entered after *space*, a *tab* is inserted.

Motion Edit Commands

These commands will move the cursor.

[*count*]**l** Cursor forward (right) one character.

[*count*]**[C** Cursor forward (right) one character.

[*count*]**w** Cursor forward one alpha-numeric word.

[*count*]**W** Cursor to the beginning of the next word that follows a blank.
[*count*]**e** Cursor to end of word.
[*count*]**E** Cursor to end of the current blank delimited word.
[*count*]**h** Cursor backward (left) one character.
[*count*][**D** Cursor backward (left) one character.
[*count*]**b** Cursor backward one word.
[*count*]**B** Cursor to preceding blank separated word.
[*count*]**|** Cursor to column *count*.
[*count*]**f***c* Find the next character *c* in the current line.
[*count*]**F***c* Find the previous character *c* in the current line.
[*count*]**t***c* Equivalent to **f** followed by **h**.
[*count*]**T***c* Equivalent to **F** followed by **l**.
[*count*]**;** Repeats *count* times, the last single character find command, **f**, **F**, **t**, or **T**.
[*count*]**,** Reverses the last single character find command *count* times.
0 Cursor to start of line.
^ Cursor to start of line.
[H Cursor to first non-blank character in line.
$ Cursor to end of line.
[Y Cursor to end of line.
% Moves to balancing **(**, **)**, {, }, [, or]. If cursor is not on one of the above characters, the remainder of the line is searched for the first occurrence of one of the above characters first.

Search Edit Commands

These commands access your command history.

[*count*]**k** Fetch previous command. Each time **k** is entered the previous command back in time is accessed.
[*count*]**-** Equivalent to **k**.
[*count*][**A** If cursor is at the end of the line it is equivalent to / with *string^set to the contents of the current line.* Otherwise, it is equivalent to **k**.
[*count*]**j** Fetch next command. Each time **j** is entered the next command forward in time is accessed.
[*count*]**+** Equivalent to **j**.
[*count*][**B** Equivalent to **j**.
[*count*]**G** The command number *count* is fetched. The default is the least recent history command.
/*string* Search backward through history for a previous command containing *string*. *String* is terminated by a 'RETURN' or 'NEW LINE'. If *string* is preceded by a **^**, the matched line must begin with *string*. If*string* is n ull, the previous string will be used.
?*string* Same as / except that search will be in the forward direction.
n Search for next match of the last pattern to / or **?** commands.
N Search for next match of the last pattern to / or **?**, but in reverse direction.

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 **$a**.
[*count*]**c***motion*
c[*count*]*motion*
Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is **c**, the entire line will be deleted and input mode entered.
C Delete the current character through the end of line and enter input mode. Equivalent to **c$**.

S Equivalent to **cc**.

[*count*]**s** Replace characters under the cursor in input mode.

D Delete the current character through the end of line. Equivalent to **d$**.

[*count*]**d***motion*

d[*count*]*motion*

Delete current character through the character that *motion* would move to. 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 **0i**.

[*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.

[*count*]**r***c* Replace the *count* character(s) starting at the current cursor position with *c*, and advance the cursor.

[*count*]**x** Delete current character.

[*count*]**X** Delete preceding character.

[*count*]**.** Repeat the previous text modification command.

[*count*]~ Invert the case of the *count* character(s) starting at the current cursor position 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.
Command or file name completion as described above.

Other Edit Commands

Miscellaneous commands.

[*count*]**y***motion*

y[*count*]*motion*

Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.

yy Yanks the entire line.

Y Yanks from current position to end of line. Equivalent to **y$**.

u Undo the last text modifying command.

U Undo all the text modifying commands performed on the line.

[*count*]**v** Returns the command **hist -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.

If the first character of the command is a #, then this command deletes this # and each # that follows a newline. Otherwise, sends the line after inserting a # in front of each line in the command. Useful for causing the current line to be inserted in the history as a comment and uncommenting previously commented commands in the history file.

[*count*]**=** If *count* is not specified, it generates the list of matching commands or file names as described above. Otherwise, the word under the the cursor is replaced by the *count* item from the most recently generated command or file list. If the cursor is not on a word, it is inserted instead.

@*letter* Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

^V Display version of the shell.

Built-in Commands.

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is zero. Except for **:**, **true**, **false**, **echo**, **newgrp**, and **login**, all built-in commands accept **--** to indicate end of options. They also interpret the option **--man** as a request to display the man page onto standard error and **-?** as a help request which prints a *usage* message on standard error. Commands that are preceded by one or two symbols are special built-in commands and are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. They are not valid function names.
5. Words following a command preceded by that are in the format of a variable assignment are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the = sign and field splitting and file name generation are not performed. These are called*de claration* built-ins.

: [*arg* ...]
The command only expands parameters.

. *name* [*arg* **...**]
If *name* is a function defined with the **function** *name* reserved word syntax, the function is executed in the current environment (as if it had been defined with the *name*() syntax.) Otherwise if*name* refers to a file, the file is read in its en tirety and the commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing the file. If any arguments *arg* are given, they become the positional parameters while processing the **.** command and the original positional parameters are restored upon completion. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed.

alias [**-ptx**] [*name*[=*value*]] ...
alias with no arguments prints the list of aliases in the form *name=value* on standard output. The**-p** option causes the w ord **alias** to be inserted before each one. When one or more arguments are given, 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 obsolete **-t** option 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 alias remains tracked. Without the**-t** option, for eac h *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The obsolete**-x** option has no effect. The exit status is non-zero if a *name* is given, but no value, and no alias has been defined for the *name*.

bg [*job*...]
This command is only on systems that support job control. Puts each specified *job* into the background. The current job is put in the background if *job* is not specified. See *Jobs* for a description of the format of *job*.

break [*n*]
Exit from the enclosing **for**, **while**, **until**, or **select** loop, if any. If*n* is specified, then break *n* levels.

builtin [**-ds**] [**-f** *file*] [*name* ...]
If *name* is not specified, and no **-f** option is specified, the built-ins are printed on standard output. The **-s** option prints only the special built-ins. Otherwise, each *name* represents the pathname whose basename is the name of the built-in. The entry point function name is determined by prepending **b_** to the built-in name. A built-in specified by a

pathname will only be executed when that pathname would be found during the path search. Built-ins found in libraries loaded via the **.paths** file will be associate with the pathname of the directory containing the **.paths** file.

The ISO C/C++ prototype is **b_***mycommand***(int** *argc*, **char ***argv*[], **void ***context***)** for the builtin command *mycommand* where *argv* is array an of *argc* elements and context is an optional pointer to a **Shell_t** structure as described in **<ast/shell.h>**.

Special built-ins cannot be bound to a pathname or deleted. The **-d** option deletes each of the given built-ins. On systems that support dynamic loading, the **-f** option names a shared library containing the code for built-ins. The shared library prefix and/or suffix, which depend on the system, can be omitted. Once a library is loaded, its symbols become available for subsequent invocations of **builtin**. Multiple libraries can be specified with separate invocations of the **builtin** command. Libraries are searched in the reverse order in which they are specified. When a library is loaded, it looks for a function in the library whose name is **lib_init()** and invokes this function with an argument of **0**.

cd [**-LP**] [*arg*]
cd [**-LP**] *old new*

This command can be in either of two forms. In the first form it changes the current directory to *arg*. If*ar g* is **-** the directory is changed to the previous directory. The shell variable **HOME** is the default*ar g*. The variable **PWD** is set to the current directory. The shell variable **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is**<n ull>** (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.

By default, symbolic link names are treated literally when finding the directory name. This is equivalent to the **-L** option. The **-P** option causes symbolic links to be resolved when determining the directory. The last instance of**-L** or **-P** on the command line determines which method is used.

The **cd** command may not be executed by **rksh. rksh93.**

command [**-pvxV**] *name* [*arg* ...]

Without the **-v** or **-V** options, **command** executes *name* with the arguments given by *arg*. The**-p** option causes a default path to be searc hed rather than the one defined by the value of **PATH**. Functions will not be searched for when finding *name*. In addition, if *name* refers to a special built-in, none of the special properties associated with the leading daggers will be honored. (For example, the predefined alias **redirect='command exec'** prevents a script from terminating when an invalid redirection is given.) With the **-x** option, if command execution would result in a failure because there are too many arguments, errno **E2BIG**, the shell will invoke command *name* multiple times with a subset of the arguments on each invocation. Arguments that occur prior to the first word that expands to multiple arguments and after the last word that expands to multiple arguments will be passed on each invocation. The exit status will be the maximum invocation exit status. With the **-v** option, **command** is equivalent to the built-in **whence** command described below. The**-V** option causes **command** to act lik e **whence -v**.

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.

disown [*job*...]
Causes the shell not to send a HUP signal to each given *job*, or all active jobs if *job* is omitted, when a login shell terminates.

echo [*arg* ...]
When the first *arg* does not begin with a -, and none of the arguments contain a , then **echo** prints each of its arguments separated by a space and terminated by a new-line. Otherwise, the behavior of **echo** is system dependent and **print** or **printf** described below should be used. See echo(1) for usage and description.

enum [**-i**] *type*[=(*value* ...)]
Creates a declaration command named *type* that is an integer type that allows one of the specified *value*s as enumeration names. If =(*value* ...) is omitted, then *type* must be an indexed array variable with at least two elements and the values are taken from this array variable. If**-i** is specified the v alues are case insensitive.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [**-c**] [**-a** *name*] [*arg* ...]
If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. The **-c** option causes the environment to be cleared before applying variable assignments associated with the **exec** invocation. The**-a** option causes *name* rather than the first *arg*, to become **argv[0]** for the new process. Input/output arguments may appear and affect the current process. If *arg* is not 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*. The value will be the least significant 8 bits of the specified status. 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 [**-p**] [*name*[=*value*]] ...
If *name* is not given, the names and values of each variable with the export attribute are printed with the values quoted in a manner that allows them to be re-input. The **export** command is the same as **typeset -x** except that if you use **export** within a function, no local variable is created. The **-p** option causes the word **export** to be inserted before each one. Otherwise, the given *name*s are marked for automatic export to the *environment* of subsequently-executed commands.

false Does nothing, and exits 1. Used with **until** for infinite loops.

fg [*job*...]
This command is only on systems that support job control. Each *job* specified is brought to the foreground and waited for in the specified order. Otherwise, the current job is brought into the foreground. See *Jobs* for a description of the format of *job*.

getconf [*name* [*pathname*]]
Prints the current value of the configuration parameter given by *name*. The configuration parameters are defined by the IEEE POSIX 1003.1 and IEEE POSIX 1003.2 standards. (See *pathconf(2)* and *sysconf(2).)* The *pathname* argument is required for parameters whose value depends on the location in the file system. If no arguments are given, **getconf** prints the names and values of the current configuration parameters. The pathname / is used for each of the parameters that requires *pathname*.

getopts [**-a** *name*] *optstring vname* [*arg* ...]

Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a + or a **-**. An option not beginning with+ or **-** or the argument **--** ends the options. Options beginning with + are only recognized when *optstring* begins with a +. *optstring* contains the letters that **getopts** recognizes. If a letter is followed by a **:**, that option is expected to have an argument. The options can be separated from the argument by blanks. The option **-?** causes **getopts** to generate a usage message on standard error. The **-a** argument can be used to specify the name to use for the usage message, which defaults to **$0**.

getopts places the next option letter it finds inside variable *vname* each time it is invoked. The option letter will be prepended with a+ when *ar g* begins with a +. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.

A leading **:** in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *vname* to **?** for an unknown option and to **:** when a required option argument is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options.

There is no way to specify any of the options **:**, +, **-**, **?**, [, and]. The option# can only be specified as the first option.

hist [**-e** *ename*] [**-nlr**] [*first* [*last*]]
hist -s [*old*=*new*] [*c ommand*]

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**-l** option 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 v alue of the variable **HISTEDIT** is used. If **HISTEDIT** is not set, then **FCEDIT** (default/**bin/ed**) is used as the editor. When editing is complete, the edited command(s) is executed if the changes have been saved. If *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 option **-r** reverses the order of the commands and the option **-n** suppresses command numbers when listing. In the second form, *command* is interpreted as *first* described above and defaults to the last command executed. The resulting command is executed after the optional substitution*old*=*new* is performed.

jobs [**-lnp**] [*job* ...]

Lists information about each given job; or all active jobs if *job* is omitted. The **-l** option lists process ids in addition to the normal information. The **-n** option only displays jobs that have stopped or exited since last notified. The **-p** option causes only the process group to be listed. See *Jobs* for a description of the format of *job*.

kill [**-s** *signame*] *job* ...
kill [**-n** *signum*] *job* ...
kill -Ll [*sig* ...]

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number with the **-n** option or by name with the **-s** option (as given in **<signal.h>**, stripped of the prefix "SIG" with the exception that SIGCLD is named CHLD). For backward compatibility, the **n** and **s** can be omitted and the number or name placed immediately after the **-**. 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 *job* can be the process id of a process that is not a member of one of the active jobs. See *Jobs* for a description of the format of *job*. In the third

form, **kill -l**, or **kill -L**, if *sig* is not specified, the signal names are listed. The **-l** option list only the signal names. **-L** options lists eac h signal name and corresponding number. Otherwise, for each *sig* that is a name, the corresponding signal number is listed. For each *sig* that is a number, the signal name corresponding to the least significant 8 bits of *sig* is listed.

let *arg* ...
Each *arg* is a separate *arithmetic expression* to be evaluated. **let** only recognizes octal constants starting with **0** when the **set** option **letoctal** is on. See *Arithmetic Evaluation* above, for a description of arithmetic expression evaluation.

The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

newgrp [*arg* ...]
Equivalent to **exec /bin/newgrp** *arg* ....

print [**-CRenprsv**] [**-u** *unit*] [**-f** *format*] [*arg* ...]
With no options or with option **-** or **--**, each *arg* is printed on standard output. The **-f** option causes the arguments to be printed as described by **printf**. In this case, any **e**, **n**, **r**, **R** options are ignored. Otherwise, unless the **-C**, **-R**, **-r**, or **-v** are specified, the following escape conventions will be applied:

a The alert character (ascii **07**).
b The backspace character (ascii **010**).
c Causes **print** to end without processing more arguments and not adding a new-line.
f The formfeed character (ascii **014**).
n The new-line character (ascii **012**).
r The carriage return character (ascii **015**).
t The tab character (ascii **011**).
v The vertical tab character (ascii **013**).
E The escape character (ascii **033**).
The backslash character .
0*x* The character defined by the 1, 2, or 3-digit octal string given by *x*.

The **-R** option will print all subsequent arguments and options other than **-n**. The**-e** causes the above escape conventions to be applied. This is the default behavior. It reverses the effect of an earlier **-r**. The**-p** option causes the argumen ts to be written onto the pipe of the process spawned with **|&** instead of standard output. The **-v** option treats each *arg* as a variable name and writes the value in the **printf %B** format. The **-C** option treats each *arg* as a variable name and writes the value in the **printf %#B** format. The**-s** option causes the argumen ts to be written onto the history file instead of standard output. The **-u** option can be used to specify a one digit file descriptor unit number *unit* on which the output will be placed. The default is 1. If the option **-n** is used, no **new-line** is added to the output.

printf *format* [*arg* ...]
The arguments *arg* are printed on standard output in accordance with the ANSI-C formatting rules associated with the format string *format*. If the number of arguments exceeds the number of format specifications, the **format** string is reused to format remaining arguments. The following extensions can also be used:

%b A **%b** format can be used instead of **%s** to cause escape sequences in the corresponding *arg* to be expanded as described in **print.**
%B A **%B** option causes each of the arguments to be treated as variable names and the binary value of variable will be printed. The alternate flag# causes a compound variable to be output on a single line. This is most useful for compound variables and variables whose attribute is **-b**.
%H A **%H** format can be used instead of **%s** to cause characters in *arg* that are special in HTML and XML to be output as their entity name. The alternate flag #

formats the output for use as a URI.

%P A **%P** format can be used instead of **%s** to cause *arg* to be interpreted as an extended regular expression and be printed as a shell pattern.

%R A **%R** format can be used instead of **%s** to cause *arg* to be interpreted as a shell pattern and to be printed as an extended regular expression.

%q A **%q** format can be used instead of **%s** to cause the resulting string to be quoted in a manner than can be reinput to the shell. When **q** is preceded by the alternative format specifier, #, the string is quoted in manner suitable as a field in a **.csv** format file.

%(*date-format***)T**

A **%(***date-format***)T** format can be use to treat an argument as a date/time string and to format the date/time according to the *date-format* as defined for the date(1) command.

%Z A **%Z** format will output a byte whose value is 0.

%d The precision field of the **%d** format can be followed by a **.** and the output base. In this case, the # flag character causes *base*# to be prepended.

The # flag, when used with the **%d** format without an output base, displays the output in powers of 1000 indicated by one of the following suffixes: **k M G T P E**, and when used with the **%i** format displays the output in powers of 1024 indicated by one of the following suffixes: **Ki Mi Gi Ti Pi Ei**.

= The = flag centers the output within the specified field width.

L The **L** flag, when used with the **%c** or **%s** formats, treats precision as character width instead of byte count.

, The **,** flag, when used with the **%d** or **%f** formats, separates groups of digits with the grouping delimiter (**,** on groups of 3 in the C locale.)

pwd [**-LP**]

Outputs the value of the current working directory. The**-L** option is the default; it prints the logical name of the current directory. If the**-P** option is giv en, all symbolic links are resolved from the name. The last instance of **-L** or **-P** on the command line determines which method is used.

read [**-ACSprsv**] [**-d** *delim*] [**-n** *n*] [[**-N** *n*] [[**-t** *time out*] [**-u** *unit*] [*vname***?***pr ompt*] [*vname* ...]

The shell input mechanism. One line is read and is broken up into fields using the characters in **IFS** as separators. The escape character, , is used to remove any special meaning for the next character and for line continuation. The**-d** option causes the read to continue to the first character of *delim* rather than new-line. The **-n** option causes at most *n* bytes to read rather a full line but will return when reading from a slow device as soon as any characters have been read. The **-N** option causes exactly *n* to be read unless an end-of-file has been encountered or the read times out because of the **-t** option. In raw mode, **-r,** the character is not treated specially. The first field is assigned to the first *vname*, the second field to the second *vname*, etc., with leftover fields assigned to the last *vname*. When*vname* has the binary attribute and **-n** or **-N** is specified, the b ytes that are read are stored directly into the variable. If the**-v** is specified, then the v alue of the first *vname* will be used as a default value when reading from a terminal device. The **-A** option causes the variable *vname* to be unset and each field that is read to be stored in successive elements of the indexed array *vname*. The **-C** option causes the variable *vname* to be read as a compound variable. Blanks will be ignored when finding the beginning open parenthesis. The -S option causes the line to be treated like a record in a **.csv** format file so that double quotes can be used to allow the delimiter character and the new-line character to appear within a field. 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** option is present, the input will be saved as a command in the history file. The option **-u** can be used to specify a one digit file descriptor unit *unit* to read from. The file descriptor can be opened with the **exec** special built-in command. The default value of unit *n* is 0.

The option **-t** is used to specify a timeout in seconds when reading from a terminal or pipe. If*vname* is omitted, then **REPL Y** is used as the default*vname*. 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* on standard error when the shell is interactive. The exit status is 0 unless an end-of-file is encountered or **read** has timed out.

readonly [**-p**] [*vname*[=*value*]] ...
If *vname* is not given, the names and values of each variable with the readonly attribute is printed with the values quoted in a manner that allows them to be re-inputted. The **-p** option causes the word **readonly** to be inserted before each one. Otherwise, the given *vname*s are marked readonly and these names cannot be changed by subsequent assignment. When defining a type, if the value of a readonly sub-variable is not defined the value is required when creating each instance.

return [*n*]
Causes a shell *function* or **.** script to return to the invoking script with the exit status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted, then the return status is that of the last command executed. If **return** is invoked while not in a *function* or a **.** script, then it behaves the same as **exit**.

set [±**BCGabefhkmnoprstuvx**] [±**o** [*option*]] ... [±**A** *vname*] [*arg* ...]
The options for this command have meaning as follows:

- **-A** Array assignment. Unset the variable *vname* and assign values sequentially from the *arg* list. If +**A** is used, the variable *vname* is not unset first.
- **-B** Enable brace pattern field generation. This is the default behavior.
- **-B** Enable brace group expansion. On by default.
- **-C** Prevents redirection > from truncating existing files. Files that are created are opened with the O_EXCL mode. Requires >| to truncate a file when turned on.
- **-G** Causes the pattern ** by itself to match files and zero or more directories and sub-directories when used for file name generation. If followed by a / only directories and sub-directories are matched.
- **-a** All subsequent variables that are defined are automatically exported.
- **-b** Prints job completion messages as soon as a background job changes state rather than waiting for the next prompt.
- **-e** Unless contained in a || or **&&** command, or the command following an **if** **while** or **until** command or in the pipeline following **!**, if a command has a non-zero exit status, execute the **ERR** trap, if set, and exit. This mode is disabled while reading profiles.
- **-f** Disables file name generation.
- **-h** Each command becomes a tracked alias when first encountered.
- **-k** (Obsolete). All variable 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 option is turned on automatically for interactive shells.
- **-n** Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.
- **-o** The following argument can be one of the following option names:
  - **allexport** Same as **-a**.
  - **errexit** Same as **-e**.
  - **bgnice** All background jobs are run at a lower priority. This is the default mode.

braceexpand
Same as **-B**.

emacs Puts you in an *emacs* style in-line editor for command entry.

globstar
Same as **-G**.

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**.

letoctal
The **let** command allows octal constants starting with **0**.

markdirs
All directory names resulting from file name generation have a trailing / appended.

monitor
Same as **-m**.

multiline
The built-in editors will use multiple lines on the screen for lines that are longer than the width of the screen. This may not work for all terminals.

noclobber
Same as **-C**.

noexec Same as **-n**.

noglob Same as **-f**.

nolog Do not save function definitions in the history file.

notify Same as **-b**.

nounset
Same as **-u**.

pipefail
A pipeline will not complete until all components of the pipeline have completed, and the return value will be the value of the last non-zero command to fail or zero if no command has failed.

showme
When enabled, simple commands or pipelines preceded by a semicolon (**;**) will be displayed as if the **xtrace** option were enabled but will not be executed. Otherwise, the leading **;** will be ignored.

privileged
Same as **-p**.

verbose
Same as **-v**.

trackall
Same as **-h**.

vi Puts you in insert mode of a *vi* style in-line editor until you hit the escape character **033**. This puts you in control 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.

-p Disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.

-r Enables the restricted shell. This option cannot be unset once set.
-s Sort the positional parameters lexicographically.
-t (Obsolete). 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.
-- Do not change any of the options; useful in setting **$1** to a value beginning with **-**. If no arguments follow this option then the positional parameters are unset.

As an obsolete feature, if the first *arg* is **-** then the **-x** and **-v** options are turned off and the next *arg* is treated as the first argument. Using+ rather than **-** causes these options to be turned off. These options can also be used upon invocation of the shell. The current set of options may be found in **$-**. Unless**-A** is specified, the remaining argumen ts are positional parameters and are assigned, in order, to **$1 $2** .... If no arguments are given, then the names and values of all variables are printed on the standard output.

shift [*n*]
The positional parameters from **$***n***+1** ... are renamed **$1** ... , default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

sleep *seconds*
Suspends execution for the number of decimal seconds or fractions of a second given by *seconds*.

trap [**-p**] [*action*] [*sig*] ...
The **-p** option causes the trap action associated with each trap as specified by the arguments to be printed with appropriate quoting. Otherwise, *action* will be processed as if it were an argument to **eval** when the shell receives signal(s) *sig*. 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. If*action* is omitted and the first *sig* is a n umber, or if *action* is **-**, then the trap(s) for each *sig* are reset to their original values. If*action* is the n ull string then this signal is ignored by the shell and by the commands it invokes. If*sig* is **ERR** then *action* will be executed whenever a command has a non-zero exit status. If *sig* is **DEBUG** then*action* will be executed before eac h command. The variable **.sh.command** will contain the contents of the current command line when *action* is running. If the exit status of the trap is **2** the command will not be executed. If the exit status of the trap is **255** and inside a function or a dot script, the function or dot script will return. If*sig* is **0** or **EXIT** and the**trap** statemen t is executed inside the body of a function defined with the **function** *name* syntax, then the command *action* is executed after the function completes. If *sig* is **0** or **EXIT** for a**trap** set outside an y function then the command *action* is executed on exit from the shell. If *sig* is **KEYBD**, then *action* will be executed whenever a key is read while in **emacs**, **gmacs**, or **vi** mode. The **trap** command with no arguments prints a list of commands associated with each signal number.

An **exit** or **return** without an argument in a trap action will preserve the exit status of the command that invoked the trap.

true Does nothing, and exits 0. Used with **while** for infinite loops.

typeset [**±ACHSfblmnprtux**] [**±EFLRXZi**[*n*]] [**+-M** [*mapname*]] [**-T** [*tname*=(*assign_list*)]] [**-h***str*] [**-a** [*typ e*]] [*vname*[=*value*]] **...**
Sets attributes and values for shell variables and functions. When invoked inside a function defined with the **function** *name* syntax, a new instance of the variable *vname* is created, and the variable's value and type are restored when the function completes. The following list of attributes may be specified:
-A Declares *vname* to be an associative array. Subscripts are strings rather than arithmetic expressions.

-C causes each *vname* to be a compound variable. *value* names a compound variable it is copied into *vname*. Otherwise, it unsets each *vname*.

-a Declares *vname* to be an indexed array. If*typ e* is specified, it must be the name of an enumeration type created with the **enum** command and it allows enumeration constants to be used as subscripts.

-E Declares *vname* to be a double precision floating point number. If*n* is non-zero, it defines the number of significant figures that are used when expanding *vname*. Otherwise, ten significant figures will be used.

-F Declares *vname* to be a double precision floating point number. If*n* is non-zero, it defines the number of places after the decimal point that are used when expanding *vname*. Otherwise ten places after the decimal point will be used.

-H This option 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 variable is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. The **-R** option is turned off.

-M Use the character mapping *mapping* defined by wctrans(3). such as **tolower** and **toupper** when assigning a value to each of the specified operands. When *mapping* is specified and there are not operands, all variables that use this mapping are written to standard output. When *mapping* is omitted and there are no operands, all mapped variables are written to standard output.

-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 variable is reassigned. The**-L** option is turned off.

-S When used within the *assign_list* of a type definition, it causes the specified subvariable to be shared by all instances of the type. When used inside a function defined with the **function** reserved word, the specified variables will have *function static* scope. Otherwise, the variable is unset prior to processing the assignment list.

-T If followed by *tname*, it creates a type named by *tname* using the compound assignment *assign_list* to *tname*. Otherwise, it writes all the type definitions to standard output.

-X Declares *vname* to be a double precision floating point number and expands using the **%a** format of ISO-C99. If *n* is non-zero, it defines the number of hex digits after the radix point that is used when expanding *vname*. The default is 10.

-Z Right justify and fill with leading zeros if the first non-blank character is a digit and the **-L** option has not been set. Remove leading zeros if the **-L** option is also 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.

-f The names refer to function names rather than variable names. No assignments can be made and the only other valid options are **-S**, **-t**, **-u** and **-x**. The -S can be used with discipline functions defined in a type to indicate that the function is static. For a static function, the same method will be used by all instances of that type no matter which instance references it. In addition, it can only use value of variables from the original type definition. These discipline functions cannot be redefined in any type instance. The **-t** option turns on execution tracing for this function. The**-u** option causes this function to be mark ed undefined. The **FPATH** variable will be searched to find the function definition when the function is referenced. If no options other than **-f** is specified, then the function definition will be displayed on standard output. If **+f** is specified, then a line containing the function name followed by a shell comment containing the line number and path name of the file where this function was defined, if any, is displayed. The exit status can be used to determine whether the function is defined so that

typeset -f .sh.math.*name* will return 0 when math function *name* is defined and non-zero otherwise.

-b The variable can hold any number of bytes of data. The data can be text or binary. The value is represented by the base64 encoding of the data. If **-Z** is also specified, the size in bytes of the data in the buffer will be determined by the size associated with the **-Z**. If the base64 string assigned results in more data, it will be truncated. Otherwise, it will be filled with bytes whose value is zero. The **printf** format **%B** can be used to output the actual data in this buffer instead of the base64 encoding of the data.

-h Used within type definitions to add information when generating information about the sub-variable on the man page. It is ignored when used outside of a type definition. When used with **-f** the information is associated with the corresponding discipline function.

-i Declares *vname* to be represented internally as integer. The right hand side of an assignment is evaluated as an arithmetic expression when assigning to an integer. If *n* is non-zero, it defines the output arithmetic base, otherwise the output base will be ten.

-l Used with **-i**, **-E** or **-F**, to indicate long integer, or long float. Otherwise, all upper-case characters are converted to lower-case. The upper-case option,**-u**, is turned off. Equivalent to **-M tolower .**

-m moves or renames the variable. The value is the name of a variable whose value will be moved to *vname*. The original variable will be unset. Cannot be used with any other options.

-n Declares *vname* to be a reference to the variable whose name is defined by the value of variable *vname*. This is usually used to reference a variable inside a function whose name has been passed as an argument. Cannot be used with any other options.

-p The name, attributes and values for the given *vname*s are written on standard output in a form that can be used as shell input. If **+p** is specified, then the values are not displayed.

-r The given *vname*s are marked readonly and these names cannot be changed by subsequent assignment.

-t Tags the variables. Tags are user definable and have no special meaning to the shell.

-u When given along with **-i**, specifies unsigned integer. Otherwise, all lower-case characters are converted to upper-case. The lower-case option, **-l**, is turned off. Equivalent to **-M toupper .**

-x The given *vname*s are marked for automatic export to the *environment* of subsequently-executed commands. Variables whose names contain a **.** cannot be exported.

The **-i** attribute cannot be specified along with **-R**, **-L**, **-Z**, or **-f**.

Using + rather than **-** causes these options to be turned off. If no *vname* arguments are given, a list of *vnames* (and optionally the *values*) of the *variables* is printed. (Using+ rather than **-** keeps the values from being printed.) The**-p** option causes **t ypeset** followed by the option letters to be printed before each name rather than the names of the options. If any option other than **-p** is given, only those variables which have all of the given options are printed. Otherwise, the*vname*s and *attributes* of all *variables* that ha ve attributes are printed.

ulimit [**-HSacdfmnpstv**] [*limit*]

Set or display a resource limit. The available resource limits are listed below. Many systems do not support one or more of these limits. The limit for a specified resource is set when *limit* is specified. The value of *limit* can be a number in the unit specified below with each resource, or the value **unlimited**. The**-H** and **-S** options specify whether the

hard limit or the soft limit for the given resource is set. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the **H** nor **S** option is specified, the limit applies to both. The current resource limit is printed when *limit* is omitted. In this case, the soft limit is printed unless **H** is specified. When more than one resource is specified, then the limit name and unit is printed before the value.

-a Lists all of the current resource limits.
-c The number of 512-byte blocks on the size of core dumps.
-d The number of K-bytes on the size of the data area.
-f The number of 512-byte blocks on files that can be written by the current process or by child processes (files of any size may be read).
-m The number of K-bytes on the size of physical memory.
-n The number of file descriptors plus 1.
-p The number of 512-byte blocks for pipe buffering.
-s The number of K-bytes on the size of the stack area.
-t The number of CPU seconds to be used by each process.
-v The number of K-bytes for virtual memory.

If no option is given, **-f** is assumed.

umask [**-S**] [*mask*]
The user file-creation mask is set to *mask* (see umask(2)). *mask* can either be an octal number or a symbolic value as described in chmod(1). If a symbolic value is given, the new umask value is the complement of the result of applying *mask* to the complement of the previous umask value. If*mask* is omitted, the curren t value of the mask is printed. The **-S** option causes the mode to be printed as a symbolic value. Otherwise, the mask is printed in octal.

unalias [**-a**] *name* ...
The aliases given by the list of *name*s are removed from the alias list. The **-a** option causes all the aliases to be unset.

unset [**-fnv**] *vname* ...
The variables given by the list of *vname*s are unassigned, i.e., except for sub-variables within a type, their values and attributes are erased. For sub-variables of a type, the values are reset to the default value from the type definition. Readonly variables cannot be unset. If the**-f** option is set, then the names refer to *function* names. If the **-v** option is set, then the names refer to *variable* names. The **-f** option overrides **-v**. If**-n** is set and *name* is a name reference, then *name* will be unset rather than the variable that it references. The default is equivalent to **-v**. Unsetting **LINENO**, **MAILCHECK**, **OPTARG**, **OPTIND**, **RANDOM**, **SECONDS**, **TMOUT**, and _ removes their special meaning even if they are subsequently assigned to.

wait [*job* ...]
Wait for the specified *job* and report its termination status. If *job* is not given, then all currently active child processes are waited for. The exit status from this command is that of the last process waited for if *job* is specified; otherwise it is zero. See *Jobs* for a description of the format of *job*.

whence [**-afpv**] *name* ...
For each *name*, indicate how it would be interpreted if used as a command name.

The **-v** option produces a more verbose report. The **-f** option skips the search for functions. The**-p** option does a path searc h for *name* even if name is an alias, a function, or a reserved word. The**-p** option turns off the **-v** option. The **-a** option is similar to the **-v** option but causes all interpretations of the given name to be reported.

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, for interactive shells, commands are read from the file named by performing parameter expansion, command substitution, and arithmetic substitution on the value of the environment variable **ENV** if the file exists. If the**-s** option is not presen t and *arg* and a file by the name of *arg* exists, then it reads and executes this script. Otherwise, if the first *arg* does not contain a /, a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have execute permission and any *setuid* and *setgid* settings will be ignored. If the script is not found on the path, *arg* is processed as if it named a built-in command or function. Commands are then read as described below; the following options are interpreted by the shell when it is invoked:

-D Do not execute the script, but output the set of double quoted strings preceded by a **$**. These strings are needed for localization of the script to different locales.

-E Reads the file named by the **ENV** variable or by $HOME/**.**kshrc if not defined after the profiles.

-c If the **-c** option is present, then commands are read from the first *arg.* Any remaining arguments become positional parameters starting at **0**.

-s If the **-s** option is present or if no arguments remain, then commands are read from the standard input. Shell output, except for the output of the *Special Commands* listed above, is written to file descriptor 2.

-i If the **-i** option is present or if the shell input and output are attached to a terminal (as told by *tcgetattr(2)),* then this shell is *interactive.* In this case TERM is ignored (so that **kill 0** does not kill an interactive shell) and INTR is caught and ignored (so that **wait** is). In all cases, QUIT is ignored by the shell.

-r If the **-r** option is present, the shell is a restricted shell.

-D A list of all double quoted strings that are preceded by a **$** will be printed on standard output and the shell will exit. This set of strings will be subject to language translation when the locale is not C or POSIX. No commands will be executed.

-P If **-P** or **-o profile** is present, the shell is a profile shell (see *pfexec(1)).*

-R *filename*

The **-R** *filename* option is used to generate a cross reference database that can be used by a separate utility to find definitions and references for variables and commands.

The remaining options and arguments are described under the **set** command above. An optional **-** as the first argument is ignored.

Rksh Only.

Rksh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rksh** are identical to those of **ksh**, except that the following are disallowed:

Unsetting the restricted option.
changing directory (see *cd(1)),*
setting or unsetting the value or attributes of **SHELL**, **ENV**, **FPATH**, or **PATH,**
specifying path or command names containing /,
redirecting output (>, >|, <>, and >>).
adding or deleting built-in commands.
using **command -p** to invoke a command.

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, **rksh** invokes *ksh* 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 (e.g., **/usr/rbin**) that can be safely invoked by **rksh**.

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 unless the error occurs inside a subshell in which case the subshell is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above). Run time errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets ([]) after the command or function name.

FILES

/etc/profile
The system wide initialization file, executed for login shells.

$HOME/.profile
The personal initialization file, executed for login shells after /etc/profile.

$HOME/**.**kshrc
Default personal initialization file, executed for interactive shells when **ENV** is not set.

/etc/suid_profile
Alternative initialization file, executed instead of the personal initialization file when the real and effective user or group id do not match.

/dev/null
NULL device

SEE ALSO

cat(1), cd(1), chmod(1), cut(1), egrep(1), echo(1), emacs(1), env(1), fgrep(1), gmacs(1), grep(1), newgrp(1), pfexec(1), stty(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), getpwnam(3), ioctl(2), lseek(2), paste(1), pathconf(2), pipe(2), sysconf(2), umask(2), ulimit(2), wait(2), wctrans(3), rand(3), a.out(5), profile(5), environ(7)

Morris I. Bolsky and David G. Korn, *The New KornShell Command and Programming Language*, Prentice Hall, 1995.

POSIX - Part 2: Shell and Utilities, IEEE Std 1003.2-1992, ISO/IEC 9945-2, IEEE, 1993.

CAVEATS

If a command 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.

Some very old shell scripts contain a ^ as a synonym for the pipe character |.

Using the **hist** built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command **.** *file* reads the whole file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to any commands defined in the file.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on **CHLD** won't be executed until the foreground job terminates.

It is a good idea to leave a space after the comma operator in arithmetic expressions to prevent the comma from being interpreted as the decimal point character in certain locales.