NAME

groff_tmac - macro files in the roff typesetting system

DESCRIPTION

The roff(7) type-setting system provides a set of macro packages suitable for special kinds of documents. Each macro package stores its macros and definitions in a file called the packages **tmac file**. The name is deduced from '**T***roff***MAC***ros*'.

The tmac files are normal roff source documents, except that they usually contain only definitions and setup commands, but no text. All tmac files are kept in a single or a small number of directories, the **tmac** directories.

GROFF MACRO PACKAGES

groff provides all classical macro packages, some more full packages, and some secondary packages for special purposes. Note that it is not possible to use multiple primary macro packages at the same time; saying e.g.

```
sh# groff -m man -m ms foo
```

or

```
sh# groff -m man foo -m ms bar
```

fails. Exception to this is the use of man pages written with either the**mdoc** or the **man** macro package. See below the description of the **andoc.tmac** file.

Man Pages

man This is the classical macro package for UNIX manual pages (man pages); it is quite handy and easy to use; see groff_man(7).

doc
mdoc An alternative macro package for man pages mainly used in BSD systems; it provides many new features, but it is not the standard for man pages; see groff_mdoc(7).

andoc
mandoc
Use this file in case you dont know whether the **man** macros or the **mdoc** package should be used. Multiple man pages (in either format) can be handled.

Full Packages

The packages in this section provide a complete set of macros for writing documents of any kind, up to whole books. They are similar in functionality; it is a matter of taste which one to use.

me The classical *me* macro package; see groff_me(7).

mm The semi-classical *mm* macro package; see groff_mm(7).

mom The new *mom* macro package, only available in groff. As this is not based on other packages, it can be freely designed. So it is expected to become quite a nice, modern macro package. See groff_mom(7).

ms The classical *ms* macro package; see groff_ms(7).

Language-specific Packages

cs This file adds support for Czech localization, including the main macro packages (me, mom, mm, and ms).

Note that **cs.tmac** sets the input encoding to latin-2.

de
den German localization support, including the main macro packages (me, mom, mm, and ms).

de.tmac selects hyphenation patterns for traditional orthography, and **den.tmac** does the same for the new orthography ('Rechtschreibreform'). It should be used as the last macro package on the command line.

fr This file adds support for French localization, including the main macro packages (me, mom, mm, and ms). Example:

```
sh# groff -ms -mfr foo.ms > foo.ps
```

Note that **fr.tmac** sets the input encoding to latin-9 to get proper support of the 'oe' ligature.

sv Swedish localization support, including the me, mom, and ms macro packages. Note that Swedish for the mm macros is handled separately; see **groff_mmse(7)** (only in Swedish locales). It should be used as the last macro package on the command line.

Input Encodings

latin1
latin2
latin5
latin9 Various input encodings supported directly by groff. Normally, this macro is loaded at the very beginning of a document or specified as the first macro argument on the command line. **roff** loads latin1 b y default at start-up. Note that these macro packages dont work on EBCDIC hosts.

cp1047
Encoding support for EBCDIC. On those platforms it is loaded automatically at start-up. Due to different character ranges used in **roff** it doesnt work on architectures which are based on ASCII.

Note that it can happen that some input encoding characters are not available for a particular output device. For example, saying

```
groff -Tlatin1 -mlatin9 ...
```

fails if you use the Euro character in the input. Usually, this limitation is present only for devices which have a limited set of output glyphs (**-Tascii**, **-Tlatin1**); for other devices it is usually sufficient to install proper fonts which contain the necessary glyphs.

Special Packages

The macro packages in this section are not intended for stand-alone usage, but can be used to add special functionality to any other macro package or to plain groff.

62bit Provide some macros for addition, multiplication, and division of 60bit integers (allowing safe multiplication of 31bit integers, for example).

ec Switch to the EC and TC font families. To be used with grodvi(1) – this man page also gives more details of how to use it.

papersize
This macro file is already loaded at start-up by **troff** so it isnt necessary to call it explicitly. It provides an interface to set the paper size on the command line with the option **-dpaper=***size*. Possible values for *size* are the same as the predefined **papersize** values in the DESC file (only lowercase; see groff_font(5) for more) except **a7**–**d7**. An appended **l** (ell) character denotes landscape orientation. Examples:**a4**, **c3l**, **letterl**.

Most output drivers need additional command line switches **-p** and **-l** to override the default paper length and orientation as set in the driver specific DESC file. For example, use the following for PS output on A4 paper in landscape orientation:

```
sh# groff -Tps -dpaper=a4l -P-pa4 -P-l -ms foo.ms > foo.ps
```

pic This file provides proper definitions for the macros **PS** and **PE**, needed for the pic(1) preprocessor. They center each picture. Use it only if your macro package doesnt provide proper definitions for those two macros (actually, most of them already do).

pspic A single macro is provided in this file, **PSPIC**, to include a PostScript graphic in a document. The following output devices support inclusion of PS images: **-Tps**, **-Tdvi**,

-Thtml, and **-Txhtml**; for all other devices the image is replaced with a hollow rectangle of the same size. This macro file is already loaded at start-up by **troff** so it isnt necessary to call it explicitly.

Syntax:

.PSPIC [**-L**|**-R**|**-C**|**-I** *n*] *file* [*width* [*height*]]

file is the name of the PostScript file; *width* and *height* give the desired width and height of the image. If neither a *width* nor a *height* argument is specified, the images natural width (as given in the files bounding box) or the current line length is used as the width, whatever is smaller. The *width* and *height* arguments may have scaling indicators attached; the default scaling indicator is **i**. This macro scales the graphic uniformly in the x and y directions so that it is no more than *width* wide and *height* high. Option **-C** centers the graphic horizontally, which is the default. The **-L** and **-R** options cause the graphic to be left-aligned and right-aligned, respectively. The**-I** option causes the graphic to be indented by *n* (default scaling indicator is **m**).

For use of **.PSPIC** within a diversion it is recommended to extend it with the following code, assuring that the diversions width completely covers the images width.

```
.am PSPIC
.  vpt 0
\h'(\\n[ps-offset]u + \\n[ps-deswid]u)'
.  sp -1
.  vpt 1
..
```

ptx A single macro is provided in this file, **xx**, for formatting permuted index entries as produced by the GNU ptx(1) program. In case you need a different formatting, copy the macro into your document and adapt it to your needs.

trace Use this for tracing macro calls. It is only useful for debugging. See groff_trace(7)

tty-char
Overrides the definition of standard troff characters and some groff characters for TTY devices. The optical appearance is intentionally inferior compared to that of normal TTY formatting to allow processing with critical equipment.

www Additions of elements known from the HTML format, as used in the internet (World Wide Web) pages; this includes URL links and mail addresses; see groff_www(7).

NAMING

Classical roff systems were designed before the conventions of the modern C getopt(3) call evolved, and used a naming scheme for macro packages that looks odd to modern eyes. Macro packages were always included with the option **-m**; when this option was directly followed by its argument without an intervening space, this looked like a long option preceded by a single minus — a sensation in the computer stone age. To make this invocation form work, classical troff macro packages used names that started with the letter 'm', which was omitted in the naming of the macro file.

For example, the macro package for the man pages was called *man*, while its macro file *tmac.an*. So it could be activated by the argument *an* to option **-m**, or **-man** for short.

For similar reasons, macro packages that did not start with an 'm' had a leading 'm' added in the documentation and in speech; for example, the package corresponding to *tmac.doc* was called *mdoc* in the documentation, although a more suitable name would be *doc*. For, when omitting the space between the option and its argument, the command line option for activating this package reads **-mdoc**.

To cope with all situations, actual versions of groff(1) are smart about both naming schemes by providing two macro files for the inflicted macro packages; one with a leading 'm' the other one

without it. So in *groff*, the *man* macro package may be specified as on of the following four methods:

```
sh# groff -m man
sh# groff -man
sh# groff -mman
sh# groff -m an
```

Recent packages that do not start with 'm' do not use an additional 'm' in the documentation. For example, the *www* macro package may be specified only as one of the two methods:

```
sh# groff -m www
sh# groff -mwww
```

Obviously, variants like *-mmwww* would not make much sense.

A second strange feature of classical troff was to name macro files in the form **tmac.***name*. In modern operating systems, the type of a file is specified as a postfix, the file name extension. Again, groff copes with this situation by searching both *anything***.tmac** and **tmac.***anything* if only *anything* is specified.

The easiest way to find out which macro packages are available on a system is to check the man page groff(1), or the contents of the *tmac* directories.

In *groff*, most macro packages are described in man pages called **groff_***name*(7), with a leading 'm' for the classical packages.

INCLUSION

There are several ways to use a macro package in a document. The classical way is to specify the troff/groff option**-m** *name* at run-time; this mak es the contents of the macro package *name* available. In groff, the file*name***.tmac** is searc hed within the tmac path; if not found, **tmac.***name* is searched for instead.

Alternatively, it is also possible to include a macro file by adding the request **.so** *filename* into the document; the argument must be the full file name of an existing file, possibly with the directory where it is kept. In groff, this was improved by the similar request **.mso** *package*, which added searching in the tmac path, just like option **-m** does.

Note that in order to resolve the **.so** and **.mso** requests, the roff preprocessor soelim(1) must be called if the files to be included need preprocessing. This can be done either directly by a pipeline on the command line or by using the troff/groff option **-s**. *man* calls soelim automatically.

For example, suppose a macro file is stored as

/usr/share/groff/1.22.3/tmac/macros.tmac

and is used in some document called *docu.roff*.

At run-time, the formatter call for this is

```
sh# groff -m macros docu.roff
```

To include the macro file directly in the document either

```
.mso macros.tmac
```

is used or

```
.so /usr/share/groff/1.22.3/tmac/macros.tmac
```

In both cases, the formatter should be called with option **-s** to invoke **soelim**.

```
sh# groff -s docu.roff
```

If you want to write your own groff macro file, call it *whatever***.tmac** and put it in some directory of the tmac path, see section **FILES**. Then documents can include it with the **.mso** request or the option **-m**.

WRITING MACROS

A roff(7) document is a text file that is enriched by predefined formatting constructs, such as requests, escape sequences, strings, numeric registers, and macros from a macro package. These elements are described in roff(7).

To give a document a personal style, it is most useful to extend the existing elements by defining some macros for repeating tasks; the best place for this is near the beginning of the document or in a separate file.

Macros without arguments are just like strings. But the full power of macros reveals when arguments are passed with a macro call. Within the macro definition, the arguments are available as the escape sequences **$1**, ..., **$9**, **$[...]**, **$***, and **$@**, the name under which the macro was called is in **$0**, and the number of arguments is in register **n[.$]**; see groff(7).

Copy-in Mode

The phase when groff reads a macro is called *copy-in mode* or *copy mode* in roff-talk. This is comparable to the C preprocessing phase during the development of a program written in the C language.

In this phase, groff interprets all backslashes; that means that all escape sequences in the macro body are interpreted and replaced by their value. For constant expressions, this is wanted, but strings and registers that might change between calls of the macro must be protected from being evaluated. This is most easily done by doubling the backslash that introduces the escape sequence. This doubling is most important for the positional parameters. For example, to print information on the arguments that were passed to the macro to the terminal, define a macro named '.print_args', say.

```
.ds midpart was called with
.de print_args
.  tm \f[I]\\$0\f[] \*[midpart] \\n[.$] arguments:
.  tm \\$*
..
```

When calling this macro by

```
.print_args arg1 arg2
```

the following text is printed to the terminal:

```
print_args was called with the following 2 arguments:
arg1 arg2
```

Lets analyze each backslash in the macro definition. As the positional parameters and the number of arguments change with each call of the macro their leading backslash must be doubled, which results in *** and *[.]*. The same applies to the macro name because it could be called with an alias name, so *0*.

On the other hand, *midpart* is a constant string, it does not change, so no doubling for **[midpart]*. The *f* escape sequences are predefined groff elements for setting the font within the text. Of course, this behavior does not change, so no doubling with *f[I]* and *f[]*.

Draft Mode

Writing groff macros is easy when the escaping mechanism is temporarily disabled. In groff, this is done by enclosing the macro definition(s) into a pair of **.eo** and **.ec** requests. Then the body in the macro definition is just like a normal part of the document — text enhanced by calls of requests, macros, strings, registers, etc. For example, the code above can be written in a simpler way by

```
.eo
.ds midpart was called with
.de print_args
.  tm \f[I]\$0\f[] \*[midpart] \n[.$] arguments:
```

```
.  tm \$*
..
.ec
```

Unfortunately, draft mode cannot be used universally. Although it is good enough for defining normal macros, draft mode fails with advanced applications, such as indirectly defined strings, registers, etc. An optimal way is to define and test all macros in draft mode and then do the backslash doubling as a final step; do not forget to remove the *.eo* request.

Tips for Macro Definitions

- Start every line with a dot, for example, by using the groff request **.nop** for text lines, or write your own macro that handles also text lines with a leading dot.

```
.de Text
.  if (\\n[.$] == 0) \
.    return
.  nop \)\\$*\)
..
```

- Write a comment macro that works both for copy-in and draft mode; for as escaping is off in draft mode, trouble might occur when normal comments are used. For example, the following macro just ignores its arguments, so it acts like a comment line:

```
.de c
..
.c This is like a comment line.
```

- In long macro definitions, make ample use of comment lines or almost-empty lines (this is, lines which have a leading dot and nothing else) for a better structuring.
- To increase readability, use groffs indentation facility for requests and macro calls (arbitrary whitespace after the leading dot).

Diversions

Diversions can be used to implement quite advanced programming constructs. They are comparable to pointers to large data structures in the C programming language, but their usage is quite different.

In their simplest form, diversions are multi-line strings, but they get their power when diversions are used dynamically within macros. The (formatted) information stored in a diversion can be retrieved by calling the diversion just like a macro.

Most of the problems arising with diversions can be avoided if you remain aware of the fact that diversions always store complete lines. If diversions are used when the line buffer has not been flushed, strange results are produced; not knowing this, many people get desperate about diversions. To ensure that a diversion works, line breaks should be added at the right places. To be on the secure side, enclose everything that has to do with diversions into a pair of line breaks; for example, by explicitly using **.br** requests. This rule should be applied to diversion definition, both inside and outside, and to all calls of diversions. This is a bit of overkill, but it works nicely.

[If you really need diversions which should ignore the current partial line, use environments to save the current partial line and/or use the **.box** request.]

The most powerful feature using diversions is to start a diversion within a macro definition and end it within another macro. Then everything between each call of this macro pair is stored within the diversion and can be manipulated from within the macros.

FILES

All macro names must be named *name***.tmac** to fully use the tmac mechanism. **tmac.***name* as with classical packages is possible as well, but deprecated.

The macro files are kept in the *tmac directories*; a colon separated list of these constitutes the *tmac path*.

The search sequence for macro files is (in that order):

- the directories specified with troff/groffs **-M** command line option
- the directories given in the **$GROFF_TMAC_PATH** environment variable
- the current directory (only if in unsafe mode, which is enabled by the **-U** command line switch)
- the home directory
- a platform-specific directory, being

  /usr/lib/groff/site-tmac

  in this installation
- a site-specific (platform-independent) directory, being

  /usr/share/groff/site-tmac

  in this installation
- the main tmac directory, being

  /usr/share/groff/1.22.3/tmac

  in this installation

ENVIRONMENT

$GROFF_TMAC_PATH

A colon separated list of additional tmac directories in which to search for macro files. See the previous section for a detailed description.

SEE ALSO

A complete reference for all parts of the groff system is found in the groff **info(1)** file.

groff(1) an overview of the groff system.

groff_man(7),
groff_mdoc(7),
groff_me(7),
groff_mm(7),
groff_mom(7),
groff_ms(7),
groff_trace(7),
groff_www(7). the groff tmac macro packages.

groff(7) the groff language.

The Filesystem Hierarchy Standard is available at the FHS web site.

COPYING

AUTHOR

This file was written by ⟨⟩groff-bernd.warken-72@web.de Bernd Warken ⟨⟩ and Werner Lemberg.