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 package's **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.

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).

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).

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.

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 being used in the internet (World Wide Web) pages; this includes URL links and mail addresses; see **groff_www**(7).

NAMING

In classical roff systems, there was a funny naming scheme for macro packages, due to a simplistic design in option parsing. Macro packages were always included by 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 optically working for macro package names, all classical macro packages choose a name 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 talking; 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 according to **tmac.***name*. In modern operating systems, the type of a file is specified as 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 makes the contents of the macro package *name* available. In groff, the file *name***.tmac** is searched within the tmac path; if not found, **tmac.***name* will be 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/1.18.1.4/tmac/macros.tmac* and is used in some document called *docu.roff*.

At run-time, the formatter call for this is

```
sh#  groff −m macrofile document.roff
```

To include the macro file directly in the document either

```
.mso macrofile.tmac
```

is used or

```
.so /usr/share/groff/1.18.1.4/tmac/macros.tmac
```

In both cases, the formatter is called with

```
sh#  troff −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 **0**; see **groff**(7).

Copy-in Mode

The phase when groff reads a macro is called *copy-in 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 expression, 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
```

Let's analyze each backslash in the macro definition. As the positional parameters and the number of arguments will 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 will 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 will 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 will fail 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 \)\\$*[rs]
  ..
  ```

- 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 empty lines for a better structuring.
- To increase readability, use groff's indentation facility for requests and macro calls (arbitrary whitespace after the leading dot).

Diversions

Diversions can be used to realize 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 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 are conscious about the fact that diversions always deal with complete lines. If diversions are used when the line buffer has not been flashed, 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 amply 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/groff's **−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/lib64/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/1.18.1.4/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.

AUTHOR

This document is part of *groff*, the GNU roff distribution. It was written by Bernd Warken ⟨`bwarken@mayn.de`⟩; it is maintained by Werner Lemberg ⟨`wl@gnu.org`⟩.

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_www(7).
the groff tmac macro packages.

groff(7)
the groff language.

The Filesystem Hierarchy Standard is available at the FHS web site ⟨`http://www.pathname.com/fhs/`⟩.