Name

groff_mdoc — compose BSD-style manual (man) pages with GNU *roff*

Synopsis

groff -mdoc *file* ...

Description

The GNU implementation of the *mdoc* macro package is part of the *groff*(1) document formatting system. *mdoc* is a structurally- and semantically-oriented package for writing Unix manual pages with *troff*(1). Its predecessor, the *man*(7) package, primarily addressed page layout and presentational concerns, leaving the selection of fonts and other typesetting details to the individual author. This discretion has led to divergent styling practices among authors using it.

mdoc organizes its macros into *domains*. The *page structure domain* lays out the page and comprises titles, section headings, displays, and lists. The *general text domain* supplies macros to quote or style text, or to interpolate common noun phrases. The *manual domain* offers semantic macros corresponding to the terminology used by practitioners in discussion of Unix commands, routines, and files. Manual domain macros distinguish command-line arguments and options, function names, function parameters, pathnames, variables, cross references to other manual pages, and so on. These terms are meaningful both to the author and the readers of a manual page. It is hoped that the resulting increased consistency of the man page corpus will enable easier translation to future documentation tools.

Throughout Unix documentation, a manual entry is referred to simply as a "man page", regardless of its length, without gendered implication, and irrespective of the macro package selected for its composition.

Getting started

The *mdoc* package attempts to simplify man page authorship and maintenance without requiring mastery of the *roff* language. This document presents only essential facts about *roff*. For further background, including a discussion of basic typographical concepts like "breaking", "filling", and "adjustment", see *roff*(7). Specialized units of measurement also arise, namely ens, vees, inches, and points, abbreviated "n", "v", "i", and "p", respectively; see section "Measurements" of *groff*(7).

For brief examples, we employ an arrow notation illustrating a transformation of input on the left to rendered output on the right. Consider the **.Dq** macro, which double-quotes its arguments.

```
.Dq man page → "man page"
```

Usage

An *mdoc macro* is *called* by placing the *roff* control character, '**.**' (dot) at the beginning of a line followed by its name. In this document, we often discuss a macro name with this leading dot to identify it clearly, but the dot is *not* part of its name. Space or tab characters can separate the dot from the macro name. Arguments may follow, separated from the macro name and each other by spaces, but *not* tabs. The dot at the beginning of the line prepares the formatter to expect a macro name. A dot followed immediately by a newline is ignored; this is called the *empty request*. To begin an input line with a dot (or a neutral apostrophe '**'**) in some context other than a macro call, precede it with the '**\&**' escape sequence; this is a dummy character, not formatted for output. The backslash is the *roff* escape character; it can appear anywhere and it always followed by at least one more character. If followed by a newline, the backslash escapes the input line break; you can thus keep input lines to a reasonable length without affecting their interpretation.

Macros in GNU *troff* accept an unlimited number of arguments, in contrast to other *troff*s that often can't handle more than nine. In limited cases, arguments may be continued or extended on the next input line without resort to the '*newline*' escape sequence; see subsection "Extended arguments" below. Neutral double quotes **"** can be used to group multiple words into an argument; see subsection "Passing space characters in an argument" below.

Most of *mdoc*'s general text and manual domain macros *parse* their argument lists for *callable* macro names. This means that an argument in the list matching a general text or manual domain macro name (and defined to be callable) will be called with the remaining arguments when it is encountered. In such cases, the argument, although the name of a macro, is not preceded by a dot. Macro calls can thus be nested. This approach to macro argument processing is a unique characteristic of the *mdoc* package, not a general feature of *roff* syntax.

For example, the option macro, .**Op**, may call the flag and argument macros, .**Fl** and .**Ar**, to specify an optional flag with an argument.

```
.Op Fl s Ar bytes      → [-s bytes]
```

To prevent a word from being interpreted as a macro name, precede it with the dummy character.

```
.Op \&Fl s \&Ar bytes → [Fl s Ar bytes]
```

In this document, macros whose argument lists are parsed for callable arguments are referred to as *parsed*, and those that may be called from an argument list are referred to as *callable*. This usage is a technical *faux pas*, since all *mdoc* macros are in fact interpreted (unless prevented with '\&'), but as it is cumbersome to constantly refer to macros as "being able to call other macros", we employ the term "parsed" instead. Except where explicitly stated, all *mdoc* macros are parsed and callable.

In the following, we term an *mdoc* macro that starts a line (with a leading dot) a *command* if a distinction from those appearing as arguments of other macros is necessary.

Passing space characters in an argument

Sometimes it is desirable to give a macro an argument containing one or more space characters, for instance to specify a particular arrangement of arguments demanded by the macro. Additionally, quoting multi-word arguments that are to be treated the same makes *mdoc* work faster; macros that parse arguments do so once (at most) for each. For example, the function command .**Fn** expects its first argument to be the name of a function and any remaining arguments to be function parameters. Because C language standards mandate the inclusion of types *and* identifiers in the parameter lists of function definitions, each 'Fn' parameter after the first will be at least two words in length, as in "*int foo*".

There are a few ways to embed a space in a macro argument. One is to use the unadjustable space escape sequence *space*. The formatter treats this escape sequence as if it were any other printable character, and will not break a line there as it would a word space when the output line is full. This method is useful for macro arguments that are not expected to straddle an output line boundary, but has a drawback: this space does not adjust as others do when the output line is formatted. An alternative is to use the unbreakable space escape sequence, '\~', which cannot break but does adjust. This *groff* extension is widely but not perfectly portable. Another method is to enclose the string in double quotes.

```
.Fn fetch char\ *str  → fetch(char *str)
.Fn fetch char\~*str  → fetch(char *str)
.Fn fetch "char *str" → fetch(char *str)
```

If the '\' before the space in the first example or the double quotes in the third example were omitted, .Fn would see three arguments, and the result would contain an undesired comma.

```
.Fn fetch char *str   → fetch(char, *str)
```

Trailing space characters

It is wise to remove trailing spaces from the ends of input lines. Should the need arise to put a formattable space at the end of a line, do so with the unadjustable or unbreakable space escape sequences.

Formatting the backslash glyph

When you need the *roff* escape character '\' to appear in the output, use '\e' or \(rs instead. Technically, '\e' formats the current escape character; it works reliably as long as no *roff* request is used to change it, which should never happen in man pages. \(rs is a *groff* special character escape sequence that explicitly formats the "reverse solidus" (backslash) glyph.

Other possible pitfalls

groff mdoc warns when an empty input line is found outside of a *display*, a topic presented in subsection "Examples and displays" below. Use empty requests to space the source document for maintenance.

Leading spaces cause a break and are formatted. Avoid this behaviour if possible. Similarly, do not put more than one space between words in an ordinary text line; they are not "normalized" to a single space as other text formatters might do.

Don't try to use the neutral double quote character '"' to represent itself in an argument. Use the special character escape sequence \(dq to format it. Further, this glyph should not be used for conventional quotation; *mdoc* offers several quotation macros. See subsection "Enclosure and quoting macros" below.

The formatter attempts to detect the ends of sentences and by default puts the equivalent of two spaces between sentences on the same output line; see *roff*(7). To defeat this detection in a parsed list of macro arguments, put '`\&`' before the punctuation mark. Thus,

```
The
.Ql .
character.
.Pp
The
.Ql \&.
character.
.Pp
.No test .
test
.Pp
.No test.
test
```

gives

The ''. character

The '.' character.

test. test

test. test

as output. As can be seen in the first and third output lines, *mdoc* handles punctuation characters specially in macro arguments. This will be explained in section "General syntax" below.

A comment in the source file of a man page can begin with '`.\"`' at the start of an input line, '`\"`' after other input, or '`\#`' anywhere (the last is a *groff* extension); the remainder of any such line is ignored.

A man page template

Use *mdoc* to construct a man page from the following template.

```
.\" The following three macro calls are required.
.Dd date
.Dt topic [section-identifier [section-keyword-or-title]]
.Os [package-or-operating system [version-or-release]]
.Sh Name
.Nm topic
.Nd summary-description
.\" The next heading is used in sections 2 and 3.
.\" .Sh Library
.\" The next heading is used in sections 1-4, 6, 8, and 9.
.Sh Synopsis
.Sh Description
.\" Uncomment and populate the following sections as needed.
.\" .Sh "Implementation notes"
.\" The next heading is used in sections 2, 3, and 9.
.\" .Sh "Return values"
.\" The next heading is used in sections 1, 3, 6, and 8.
.\" .Sh Environment
.\" .Sh Files
.\" The next heading is used in sections 1, 6, and 8.
.\" .Sh "Exit status"
.\" .Sh Examples
.\" The next heading is used in sections 1, 4, 6, 8, and 9.
.\" .Sh Diagnostics
.\" .Sh Compatibility
```

```
.\" The next heading is used in sections 2, 3, 4, and 9.
.\" .Sh Errors
.\" .Sh "See also"
.\" .Sh Standards
.\" .Sh History
.\" .Sh Authors
.\" .Sh Caveats
.\" .Sh Bugs
```

The first items in the template are the commands **.Dd**, **.Dt**, and **.Os**. They identify the page and are discussed below in section "Title macros".

The remaining items in the template are section headings (**.Sh**); of which "Name" and "Description" are mandatory. These headings are discussed in section "Page structure domain", which follows section "Manual domain". Familiarize yourself with manual domain macros first; we use them to illustrate the use of page structure domain macros.

Conventions

In the descriptions of macros below, square brackets surround optional arguments. An ellipsis ('...') represents repetition of the preceding argument zero or more times. Alternative values of a parameter are separated with '|'. If a mandatory parameter can take one of several alternative values, use braces to enclose the set, with spaces and '|' separating the items.

ztar {c | x} [-w [-y | -z]] [-f *archive*] *member* ...

An alternative to using braces is to separately synopsize distinct operation modes, particularly if the list of valid optional arguments is dependent on the user's choice of a mandatory parameter.

ztar c [-w [-y | -z]] [-f *archive*] *member* ...
ztar x [-w [-y | -z]] [-f *archive*] *member* ...

Most macros affect subsequent arguments until another macro or a newline is encountered. For example, '**.Li ls Bq Ar file**' doesn't produce '`ls [file]`', but '`ls [`*file*`]`'. Consequently, a warning message is emitted for many commands if the first argument is itself a macro, since it cancels the effect of the preceding one. On rare occasions, you might want to format a word along with surrounding brackets as a literal.

`.Li "ls [file]"` → `ls [file]` *# list any files named e, f, i, or l*

Many macros possess an implicit width, used when they are contained in lists and displays. If you avoid relying on these default measurements, you escape potential conflicts with site-local modifications of the *mdoc* package. Explicit `-width` and `-offset` arguments to the **.Bl** and **.Bd** macros are preferable.

Title macros

We present the **mandatory** title macros first due to their importance even though they formally belong to the page structure domain macros. They designate the topic, date of last revision, and the operating system or software project associated with the page. Call each once at the beginning of the document. They populate the page headers and footers, which are in *roff* parlance termed "titles".

`.Dd` *date*

This first macro of any *mdoc* manual records the last modification date of the document source. Arguments are concatenated and separated with space characters.

Historically, *date* was written in U.S. traditional format, "*Month day , year*" where *Month* is the full month name in English, *day* an integer without a leading zero, and *year* the four-digit year. This localism is not enforced, however. You may prefer ISO 8601 format, *YYYY-MM-DD*. A *date* of the form '`$Mdocdate: Month day year $`' is also recognized. It is used in OpenBSD manuals to automatically insert the current date when committing.

This macro is neither callable nor parsed.

`.Dt` *topic* [*section-identifier* [*section-keyword-or-title*]]

topic is the subject of the man page. A *section-identifier* that begins with an integer in the range 1–9 or is one of the words `unass`, `draft`, or `paper` selects a predefined section title. This use of "section" has nothing to do with the section headings otherwise discussed in this page; it arises from the organizational scheme of printed and bound Unix manuals.

In this implementation, the following titles are defined for integral section numbers.

`1`	General Commands Manual
`2`	System Calls Manual
`3`	Library Functions Manual
`4`	Kernel Interfaces Manual
`5`	File Formats Manual
`6`	Games Manual
`7`	Miscellaneous Information Manual
`8`	System Manager's Manual
`9`	Kernel Developer's Manual

A section title may be arbitrary or one of the following abbreviations.

`USD`	User's Supplementary Documents
`PS1`	Programmer's Supplementary Documents
`AMD`	Ancestral Manual Documents
`SMM`	System Manager's Manual
`URM`	User's Reference Manual
`PRM`	Programmer's Manual
`KM`	Kernel Manual
`IND`	Manual Master Index
`LOCAL`	Local Manual
`CON`	Contributed Software Manual

For compatibility, `MMI` can be used for `IND`, and `LOC` for `LOCAL`. Values from the previous table will specify a new section title. If *section-keyword-or-title* designates a computer architecture recognized by *groff mdoc*, its value is prepended to the default section title as specified by the second parameter. By default, the following architecture keywords are defined.

acorn26, acorn32, algor, alpha, amd64, amiga, amigappc, arc, arm, arm26, arm32, armish, atari, aviion, beagle, bebox, cats, cesfic, cobalt, dreamcast, emips, evbarm, evbmips, evbppc, evbsh3, ews4800mips, hp300, hp700, hpcarm, hpcmips, hpcsh, hppa, hppa64, i386, ia64, ibmnws, iyonix, landisk, loongson, luna68k, luna88k, m68k, mac68k, macppc, mips, mips64, mipsco, mmeye, mvme68k, mvme88k, mvmeppc, netwinder, news68k, newsmips, next68k, ofppc, palm, pc532, playstation2, pmax, pmppc, powerpc, prep, rs6000, sandpoint, sbmips, sgi, sgimips, sh3, shark, socppc, solbourne, sparc, sparc64, sun2, sun3, tahoe, vax, x68k, x86_64, xen, zaurus

If a section title is not determined after the above matches have been attempted, *section-keyword-or-title* is used.

The effects of varying `.Dt` arguments on the page header content are shown below. Observe how '`\&`' prevents the numeral 2 from being used to look up a predefined section title.

`.Dt foo 2`	→	foo(2)	System Calls Manual	foo(2)
`.Dt foo 2 m68k`	→	foo(2)	m68k System Calls Manual	foo(2)
`.Dt foo 2 baz`	→	foo(2)	System Calls Manual	foo(2)
`.Dt foo \&2 baz`	→	foo(2)	baz	foo(2)
`.Dt foo "" baz`	→	foo	baz	foo
`.Dt foo M Z80`	→	foo(M)	Z80	foo(M)

roff strings define section titles and architecture identifiers. Site-specific additions might be found in the file *mdoc.local*; see section "Files" below.

This macro is neither callable nor parsed.

`.Os` [*operating-system-or-package-name* [*version-or-release*]]

This macro associates the document with a software distribution. When composing a man page to be included in the base installation of an operating system, do not provide an argument; *mdoc* will supply it. In this implementation, that default is "FreeBSD 13.2". It may be overridden in the site configuration file, *mdoc.local*; see section "Files" below. A portable software package maintaining its own man pages can supply its name and version number or release identifier as optional arguments. A *version-or-release* argument should use the standard nomenclature for the software specified. In the following table, recognized *version-or-release* arguments for some predefined operating systems are listed. As with **.Dt**, site additions might be defined in *mdoc.local*.

ATT	7th, 7, III, 3, V, V.2, V.3, V.4
BSD	3, 4, 4.1, 4.2, 4.3, 4.3t, 4.3T, 4.3r, 4.3R, 4.4
NetBSD	0.8, 0.8a, 0.9, 0.9a, 1.0, 1.0a, 1.1, 1.2, 1.2a, 1.2b, 1.2c, 1.2d, 1.2e, 1.3, 1.3a, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1, 1.6.2, 1.6.3, 2.0, 2.0.1, 2.0.2, 2.0.3, 2.1, 3.0, 3.0.1, 3.0.2, 3.0.3, 3.1, 3.1.1, 4.0, 4.0.1, 5.0, 5.0.1, 5.0.2, 5.1, 5.1.2, 5.1.3, 5.1.4, 5.2, 5.2.1, 5.2.2, 6.0, 6.0.1, 6.0.2, 6.0.3, 6.0.4, 6.0.5, 6.0.6, 6.1, 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 7.0, 7.0.1, 7.0.2, 7.1, 7.1.1, 7.1.2, 7.2, 8.0, 8.1
FreeBSD	1.0, 1.1, 1.1.5, 1.1.5.1, 2.0, 2.0.5, 2.1, 2.1.5, 2.1.6, 2.1.7, 2.2, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 4.9, 4.10, 4.11, 5.0, 5.1, 5.2, 5.2.1, 5.3, 5.4, 5.5, 6.0, 6.1, 6.2, 6.3, 6.4, 7.0, 7.1, 7.2, 7.3, 7.4, 8.0, 8.1, 8.2, 8.3, 8.4, 9.0, 9.1, 9.2, 9.3, 10.0, 10.1, 10.2, 10.3, 10.4, 11.0, 11.1, 11.2, 11.3, 12.0, 12.1
OpenBSD	2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6
DragonFly	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.8.1, 1.9, 1.10, 1.11, 1.12, 1.12.2, 1.13, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.9.1, 2.10, 2.10.1, 2.11, 2.12, 2.13, 3.0, 3.0.1, 3.0.2, 3.1, 3.2, 3.2.1, 3.2.2, 3.3, 3.4, 3.4.1, 3.4.2, 3.4.3, 3.5, 3.6, 3.6.1, 3.6.2, 3.7, 3.8, 3.8.1, 3.8.2, 4.0, 4.0.1, 4.0.2, 4.0.3, 4.0.4, 4.0.5, 4.0.6, 4.1, 4.2, 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.3, 4.4, 4.4.1, 4.4.2, 4.4.3, 4.5, 4.6, 4.6.1, 4.6.2, 4.7, 4.8, 4.8.1, 4.9, 5.0, 5.0.1, 5.0.2, 5.1, 5.2, 5.2.1, 5.2.2, 5.3, 5.4, 5.4.1, 5.4.2, 5.4.3, 5.5, 5.6, 5.6.1, 5.6.2
Darwin	8.0.0, 8.1.0, 8.2.0, 8.3.0, 8.4.0, 8.5.0, 8.6.0, 8.7.0, 8.8.0, 8.9.0, 8.10.0, 8.11.0, 9.0.0, 9.1.0, 9.2.0, 9.3.0, 9.4.0, 9.5.0, 9.6.0, 9.7.0, 9.8.0, 10.0.0, 10.1.0, 10.2.0, 10.3.0, 10.4.0, 10.5.0, 10.6.0, 10.7.0, 10.8.0, 11.0.0, 11.1.0, 11.2.0, 11.3.0, 11.4.0, 11.5.0, 12.0.0, 12.1.0, 12.2.0, 13.0.0, 13.1.0, 13.2.0, 13.3.0, 13.4.0, 14.0.0, 14.1.0, 14.2.0, 14.3.0, 14.4.0, 14.5.0, 15.0.0, 15.1.0, 15.2.0, 15.3.0, 15.4.0, 15.5.0, 15.6.0, 16.0.0, 16.1.0, 16.2.0, 16.3.0, 16.4.0, 16.5.0, 16.6.0, 17.0.0, 17.1.0, 17.2.0, 17.3.0, 17.4.0, 17.5.0, 17.6.0, 17.7.0, 18.0.0, 18.1.0, 18.2.0, 18.3.0, 18.4.0, 18.5.0, 18.6.0, 18.7.0, 19.0.0, 19.1.0, 19.2.0

Historically, the first argument used with **.Dt** was `BSD` or `ATT`. An unrecognized version argument after `ATT` is replaced with "Unix"; for other predefined abbreviations, it is ignored and a warning diagnostic emitted. Otherwise, unrecognized arguments are displayed verbatim in the page footer. For instance, this page uses "`.Os groff 1.23.0`" whereas a locally produced page might employ "`.Os "UXYZ CS Department"`", omitting versioning.

This macro is neither callable nor parsed.

Introduction to manual and general text domains

What's in a Name . . .

The manual domain macro names are derived from the day to day informal language used to describe commands, subroutines and related files. Slightly different variations of this language are used to describe the three different aspects of writing a man page. First, there is the description of *mdoc* macro command usage. Second is the description of a Unix command *with mdoc* macros, and third, the description of a command to a user in the verbal sense; that is, discussion of a command in the text of a man page.

In the first case, *troff* macros are themselves a type of command; the general syntax for a *troff* command is:

```
.Xx argument1 argument2 ...
```

`.Xx` is a macro command, and anything following it are arguments to be processed. In the second case, the description of a Unix command using the manual domain macros is a bit more involved; a typical "Synopsis" command line might be displayed as:

filter [-flag] ⟨*infile*⟩ ⟨*outfile*⟩

Here, **filter** is the command name and the bracketed string `-flag` is a *flag* argument designated as optional by the option brackets. In *mdoc* terms, ⟨*infile*⟩ and ⟨*outfile*⟩ are called *meta arguments*; in this example, the user has to replace the meta expressions given in angle brackets with real file names. Note that in this document meta arguments are used to describe *mdoc* commands; in most man pages, meta variables are not specifically written with angle brackets. The macros that formatted the above example:

```
.Nm filter
.Op Fl flag
.Ao Ar infile Ac Ao Ar outfile Ac
```

In the third case, discussion of commands and command syntax includes both examples above, but may add more detail. The arguments ⟨*infile*⟩ and ⟨*outfile*⟩ from the example above might be referred to as *operands* or *file arguments*. Some command-line argument lists are quite long:

make [-eiknqrstv] [-D *variable*] [-d *flags*] [-f *makefile*] [-I *directory*] [-j *max_jobs*] [*variable*=*value*] [*target* ...]

Here one might talk about the command *make* and qualify the argument, *makefile*, as an argument to the flag, `-f`, or discuss the optional file operand *target*. In the verbal context, such detail can prevent confusion, however the *mdoc* package does not have a macro for an argument *to* a flag. Instead the '`Ar`' argument macro is used for an operand or file argument like *target* as well as an argument to a flag like *variable*. The make command line was produced from:

```
.Nm make
.Op Fl eiknqrstv
.Op Fl D Ar variable
.Op Fl d Ar flags
.Op Fl f Ar makefile
.Op Fl I Ar directory
.Op Fl j Ar max_jobs
.Op Ar variable Ns = Ns Ar value
.Bk
.Op Ar target ...
.Ek
```

The `.Bk` and `.Ek` macros are explained in "Keeps".

General Syntax

The manual domain and general text domain macros share a similar syntax with a few minor deviations; most notably, `.Ar`, `.Fl`, `.Nm`, and `.Pa` differ only when called without arguments; and `.Fn` and `.Xr` impose an order on their argument lists. All manual domain macros are capable of recognizing and properly handling punctuation, provided each punctuation character is separated by a leading space. If a command is given:

```
.Ar sptr, ptr),
```

The result is:

sptr, ptr),

The punctuation is not recognized and all is output in the font used by `.Ar`. If the punctuation is separated by a leading white space:

```
.Ar sptr , ptr ) ,
```

The result is:

sptr , *ptr*) ,

The punctuation is now recognized and output in the default font distinguishing it from the argument strings. To remove the special meaning from a punctuation character, escape it with '`\&`'.

The following punctuation characters are recognized by *mdoc*:

```
.       ,       :       ;       (
)       [       ]       ?       !
```

troff is limited as a macro language, and has difficulty when presented with a string containing certain mathematical, logical, or quotation character sequences:

```
{+,-,/,*,%,<,>,<=,>=,=,==,&,`,',"}
```

The problem is that *troff* may assume it is supposed to actually perform the operation or evaluation suggested by the characters. To prevent the accidental evaluation of these characters, escape them with '`\&`'. Typical syntax is shown in the first manual domain macro displayed below, `.Ad`.

Manual domain

Addresses

The address macro identifies an address construct.

`Usage: .Ad` ⟨address⟩ ...

```
.Ad addr1           addr1
.Ad addr1 .         addr1.
.Ad addr1 , file2   addr1, file2
.Ad f1 , f2 , f3 :  f1, f2, f3:
.Ad addr ) ) ,      addr)),
```

The default width is 12n.

Author Name

The `.An` macro is used to specify the name of the author of the item being documented, or the name of the author of the actual manual page.

`Usage: .An` ⟨author name⟩ ...

```
.An "Joe Author"          Joe Author
.An "Joe Author" ,        Joe Author,
.An "Joe Author" Aq nobody@FreeBSD.org
                          Joe Author <nobody@FreeBSD.org>
.An "Joe Author" ) ) ,    Joe Author)),
```

The default width is 12n.

In a section titled "Authors", '`An`' causes a break, allowing each new name to appear on its own line. If this is not desirable,

```
.An -nosplit
```

call will turn this off. To turn splitting back on, write

```
.An -split
```

Arguments

The `.Ar` argument macro may be used whenever an argument is referenced. If called without arguments, '*file ...*' is output. This places the ellipsis in italics, which is ugly and incorrect, and will be noticed on terminals that underline text instead of using an oblique typeface. We recommend using `.Ar file No ...` instead.

```
Usage: .Ar [⟨argument⟩] ...
```

```
.Ar              file ...
.Ar file No ...
                 file ...
.Ar file1        file1
.Ar file1 .      file1.
.Ar file1 file2  file1 file2
.Ar f1 f2 f3 :   f1 f2 f3:
.Ar file ) ) ,   file)),
```

The default width is 12n.

Configuration Declaration (Section Four Only)

The `.Cd` macro is used to demonstrate a *config*(8) declaration for a device interface in a section four manual.

```
Usage: .Cd ⟨argument⟩ ...
```

```
.Cd "device le0 at scode?" device le0 at scode?
```

In a section titled "Synopsis", '`Cd`' causes a break before and after its arguments.

The default width is 12n.

Command Modifiers

The command modifier is identical to the `.Fl` (flag) command with the exception that the `.Cm` macro does not assert a dash in front of every argument. Traditionally flags are marked by the preceding dash, however, some commands or subsets of commands do not use them. Command modifiers may also be specified in conjunction with interactive commands such as editor commands. See "Flags".

The default width is 10n.

Defined Variables

A variable (or constant) that is defined in an include file is specified by the macro `.Dv`.

```
Usage: .Dv ⟨defined-variable⟩ ...
```

```
.Dv MAXHOSTNAMELEN MAXHOSTNAMELEN
.Dv TIOCGPGRP )    TIOCGPGRP)
```

The default width is 12n.

Errnos

The `.Er` errno macro specifies the error return value for section 2, 3, and 9 library routines. The second example below shows `.Er` used with the `.Bq` general text domain macro, as it would be used in a section two manual page.

```
Usage: .Er ⟨errno type⟩ ...
```

```
.Er ENOENT     ENOENT
.Er ENOENT ) ; ENOENT);
.Bq Er ENOTDIR [ENOTDIR]
```

The default width is 17n.

Environment Variables

The `.Ev` macro specifies an environment variable.

```
Usage: .Ev ⟨argument⟩ ...

       .Ev DISPLAY       DISPLAY
       .Ev PATH .        PATH.
       .Ev PRINTER ) ) , PRINTER)),
```

The default width is 15n.

Flags

The `.Fl` macro handles command-line flags. It prepends a dash, '−', to the flag. For interactive command flags that are not prepended with a dash, the `.Cm` (command modifier) macro is identical, but without the dash.

```
Usage: .Fl ⟨argument⟩ ...

       .Fl          −
       .Fl cfv      −cfv
       .Fl cfv .    −cfv.
       .Cm cfv .    cfv.
       .Fl s v t    −s −v −t
       .Fl − ,      −−,
       .Fl xyz ) ,  −xyz),
       .Fl |        − |
```

The `.Fl` macro without any arguments results in a dash representing stdin/stdout. Note that giving `.Fl` a single dash will result in two dashes.

The default width is 12n.

Function Declarations

The `.Fd` macro is used in the "Synopsis" section with section two or three functions. It is neither callable nor parsed.

```
Usage: .Fd ⟨argument⟩ ...

       .Fd "#include <sys/types.h>" #include <sys/types.h>
```

In a section titled "Synopsis", '`Fd`' causes a break if a function has already been presented and a break has not occurred, leaving vertical space between one function declaration and the next.

In a section titled "Synopsis", the '`In`' macro represents the `#include` statement, and is the short form of the above example. It specifies the C header file as being included in a C program. It also causes a break.

While not in the "Synopsis" section, it represents the header file enclosed in angle brackets.

```
Usage: .In ⟨header file⟩

       .In stdio.h <stdio.h>
       .In stdio.h <stdio.h>
```

Function Types

This macro is intended for the "Synopsis" section. It may be used anywhere else in the man page without problems, but its main purpose is to present the function type (in BSD kernel normal form) for the "Synopsis" of sections two and three. (It causes a break, allowing the function name to appear on the next line.)

```
Usage: .Ft ⟨type⟩ ...

       .Ft struct stat struct stat
```

Functions (Library Routines)

The `.Fn` macro is modeled on ANSI C conventions.

`Usage: .Fn` ⟨function⟩ [⟨parameter⟩] ...

```
.Fn getchar              getchar()
.Fn strlen ) ,           strlen()),
.Fn align "char *ptr" ,  align(char *ptr),
```

Note that any call to another macro signals the end of the `.Fn` call (it will insert a closing parenthesis at that point).

For functions with many parameters (which is rare), the macros `.Fo` (function open) and `.Fc` (function close) may be used with `.Fa` (function argument).

Example:

```
.Ft int
.Fo res_mkquery
.Fa "int op"
.Fa "char *dname"
.Fa "int class"
.Fa "int type"
.Fa "char *data"
.Fa "int datalen"
.Fa "struct rrec *newrr"
.Fa "char *buf"
.Fa "int buflen"
.Fc
```

Produces:

int **res_mkquery**(*int op*, *char *dname*, *int class*, *int type*, *char *data*, *int datalen*, *struct rrec *newrr*, *char *buf*, *int buflen*)

Typically, in a "Synopsis" section, the function delcaration will begin the line. If more than one function is presented in the "Synopsis" section and a function type has not been given, a break will occur, leaving vertical space between the current and prior function names.

The default width values of `.Fn` and `.Fo` are 12n and 16n, respectively.

Function Arguments

The `.Fa` macro is used to refer to function arguments (parameters) outside of the "Synopsis" section of the manual or inside the "Synopsis" section if the enclosure macros `.Fo` and `.Fc` instead of `.Fn` are used. `.Fa` may also be used to refer to structure members.

`Usage: .Fa` ⟨function argument⟩ ...

```
.Fa d_namlen ) ) , d_namlen)),
.Fa iov_len        iov_len
```

The default width is 12n.

Return Values

The `.Rv` macro generates text for use in the "Return values" section.

`Usage: .Rv` [−std] [⟨function⟩ ...]

For example, `.Rv -std atexit` produces:

> The **atexit**() function returns the value 0 if successful; otherwise the value −1 is returned and the global variable *errno* is set to indicate the error.

The `-std` option is valid only for manual page sections 2 and 3. Currently, this macro does nothing if used without the `-std` flag.

Exit Status

The `.Ex` macro generates text for use in the "Diagnostics" section.

`Usage: .Ex` [−std] [⟨utility⟩ ...]

For example, `.Ex -std cat` produces:

> The **cat** utility exits 0 on success, and >0 if an error occurs.

The `-std` option is valid only for manual page sections 1, 6 and 8. Currently, this macro does nothing if used without the `-std` flag.

Interactive Commands

The `.Ic` macro designates an interactive or internal command.

`Usage: .Ic` ⟨argument⟩ ...

```
.Ic :wq                 :wq
.Ic "do while {...}"    do while {...}
.Ic setenv , unsetenv   setenv, unsetenv
```

The default width is 12n.

Library Names

The `.Lb` macro is used to specify the library where a particular function is compiled in.

`Usage: .Lb` ⟨argument⟩ ...

Available arguments to `.Lb` and their results are:

`libarchive`	Streaming Archive Library (libarchive, −larchive)
`libarm`	ARM Architecture Library (libarm, −larm)
`libarm32`	ARM32 Architecture Library (libarm32, −larm32)
`libbluetooth`	Bluetooth User Library (libbluetooth, −lbluetooth)
`libbsm`	Basic Security Module Library (libbsm, −lbsm)
`libc`	Standard C Library (libc, −lc)
`libc_r`	Reentrant C Library (libc_r, −lc_r)
`libcalendar`	Calendar Arithmetic Library (libcalendar, −lcalendar)
`libcam`	Common Access Method User Library (libcam, −lcam)
`libcdk`	Curses Development Kit Library (libcdk, −lcdk)
`libcipher`	FreeSec Crypt Library (libcipher, −lcipher)
`libcompat`	Compatibility Library (libcompat, −lcompat)
`libcrypt`	Crypt Library (libcrypt, −lcrypt)
`libcurses`	Curses Library (libcurses, −lcurses)
`libdevinfo`	Device and Resource Information Utility Library (libdevinfo, −ldevinfo)
`libdevstat`	Device Statistics Library (libdevstat, −ldevstat)
`libdisk`	Interface to Slice and Partition Labels Library (libdisk, −ldisk)
`libdwarf`	DWARF Access Library (libdwarf, −ldwarf)
`libedit`	Line Editor and History Library (libedit, −ledit)
`libelf`	ELF Parsing Library (libelf, −lelf)
`libevent`	Event Notification Library (libevent, −levent)
`libfetch`	File Transfer Library (libfetch, −lfetch)
`libform`	Curses Form Library (libform, −lform)
`libgeom`	Userland API Library for kernel GEOM subsystem (libgeom, −lgeom)
`libgpib`	General-Purpose Instrument Bus (GPIB) library (libgpib, −lgpib)
`libi386`	i386 Architecture Library (libi386, −li386)
`libintl`	Internationalized Message Handling Library (libintl, −lintl)
`libipsec`	IPsec Policy Control Library (libipsec, −lipsec)
`libipx`	IPX Address Conversion Support Library (libipx, −lipx)

`libiscsi`	iSCSI protocol library (libiscsi, –liscsi)
`libjail`	Jail Library (libjail, –ljail)
`libkiconv`	Kernel side iconv library (libkiconv, –lkiconv)
`libkse`	N:M Threading Library (libkse, –lkse)
`libkvm`	Kernel Data Access Library (libkvm, –lkvm)
`libm`	Math Library (libm, –lm)
`libm68k`	m68k Architecture Library (libm68k, –lm68k)
`libmagic`	Magic Number Recognition Library (libmagic, –lmagic)
`libmd`	Message Digest (MD4, MD5, etc.) Support Library (libmd, –lmd)
`libmemstat`	Kernel Memory Allocator Statistics Library (libmemstat, –lmemstat)
`libmenu`	Curses Menu Library (libmenu, –lmenu)
`libnetgraph`	Netgraph User Library (libnetgraph, –lnetgraph)
`libnetpgp`	Netpgp signing, verification, encryption and decryption (libnetpgp, –lnetpgp)
`libossaudio`	OSS Audio Emulation Library (libossaudio, –lossaudio)
`libpam`	Pluggable Authentication Module Library (libpam, –lpam)
`libpcap`	Packet Capture Library (libpcap, –lpcap)
`libpci`	PCI Bus Access Library (libpci, –lpci)
`libpmc`	Performance Monitoring Counters Interface Library (libpmc, –lpmc)
`libposix`	POSIX Compatibility Library (libposix, –lposix)
`libprop`	Property Container Object Library (libprop, –lprop)
`libpthread`	POSIX Threads Library (libpthread, –lpthread)
`libpuffs`	puffs Convenience Library (libpuffs, –lpuffs)
`librefuse`	File System in Userspace Convenience Library (librefuse, –lrefuse)
`libresolv`	DNS Resolver Library (libresolv, –lresolv)
`librpcsec_gss`	RPC GSS-API Authentication Library (librpcsec_gss, –lrpcsec_gss)
`librpcsvc`	RPC Service Library (librpcsvc, –lrpcsvc)
`librt`	POSIX Real-time Library (librt, –lrt)
`libsdp`	Bluetooth Service Discovery Protocol User Library (libsdp, –lsdp)
`libssp`	Buffer Overflow Protection Library (libssp, –lssp)
`libSystem`	System Library (libSystem, –lSystem)
`libtermcap`	Termcap Access Library (libtermcap, –ltermcap)
`libterminfo`	Terminal Information Library (libterminfo, –lterminfo)
`libthr`	1:1 Threading Library (libthr, –lthr)
`libufs`	UFS File System Access Library (libufs, –lufs)
`libugidfw`	File System Firewall Interface Library (libugidfw, –lugidfw)
`libulog`	User Login Record Library (libulog, –lulog)
`libusbhid`	USB Human Interface Devices Library (libusbhid, –lusbhid)
`libutil`	System Utilities Library (libutil, –lutil)
`libvgl`	Video Graphics Library (libvgl, –lvgl)
`libx86_64`	x86_64 Architecture Library (libx86_64, –lx86_64)
`libz`	Compression Library (libz, –lz)

Site-specific additions might be found in the file *mdoc.local*; see section “Files” below.

In a section titled “Library”, ‘`Lb`’ causes a break before and after its arguments.

Literals

The ‘`Li`’ literal macro may be used for special characters, symbolic constants, and other syntactical items that should be typed exactly as displayed.

`Usage: .Li` ⟨argument⟩ ...

```
.Li \en         \n
.Li M1 M2 M3 ;  M1 M2 M3;
.Li cntrl-D ) , cntrl-D),
```

```
.Li 1024 ...    1024 ...
```

The default width is 16n.

Names

The '`Nm`' macro is used for the document title or page topic. Upon its first call, it has the peculiarity of remembering its argument, which should always be the topic of the man page. When subsequently called without arguments, '`Nm`' regurgitates this initial name for the sole purpose of making less work for the author. Use of '`Nm`' is also appropriate when presenting a command synopsis for the topic of a man page in section 1, 6, or 8. Its behavior changes when presented with arguments of various forms.

```
.Nm groff_mdoc groff_mdoc
.Nm            groff_mdoc
.Nm \-mdoc     -mdoc
.Nm foo ) ) ,  foo)),
.Nm :          groff_mdoc:
```

By default, the topic is set in boldface to reflect its prime importance in the discussion. Cross references to other man page topics should use '`Xr`'; including a second argument for the section number enables them to be hyperlinked. By default, cross-referenced topics are set in italics to avoid cluttering the page with boldface.

The default width is 10n.

Options

The `.Op` macro places option brackets around any remaining arguments on the command line, and places any trailing punctuation outside the brackets. The macros `.Oo` and `.Oc` (which produce an opening and a closing option bracket, respectively) may be used across one or more lines or to specify the exact position of the closing parenthesis.

`Usage: .Op` [⟨option⟩] ...

```
.Op                                 []
.Op Fl k                            [-k]
.Op Fl k ) .                        [-k]).
.Op Fl k Ar kookfile                [-k kookfile]
.Op Fl k Ar kookfile ,              [-k kookfile],
.Op Ar objfil Op Ar corfil          [objfil [corfil]]
.Op Fl c Ar objfil Op Ar corfil ,   [-c objfil [corfil]],
.Op word1 word2                     [word1 word2]
.Li .Op Oo Ao option Ac Oc ...      .Op [⟨option⟩] ...
```

Here a typical example of the `.Oo` and `.Oc` macros:

```
.Oo
.Op Fl k Ar kilobytes
.Op Fl i Ar interval
.Op Fl c Ar count
.Oc
```

Produces:

[[`-k` *kilobytes*][`-i` *interval*][`-c` *count*]]

The default width values of `.Op` and `.Oo` are 14n and 10n, respectively.

Pathnames

The `.Pa` macro formats file specifications. If called without arguments, '~' (recognized by many shells) is output, representing the user's home directory.

`Usage: .Pa` [⟨pathname⟩] ...

```
.Pa                     ~
.Pa /usr/share          /usr/share
.Pa /tmp/fooXXXXX ) .   /tmp/fooXXXXX).
```

The default width is 32n.

Standards

The `.St` macro replaces standard abbreviations with their formal names.

`Usage: .St` ⟨abbreviation⟩ ...

Available pairs for "Abbreviation/Formal Name" are:

ANSI/ISO C

`-ansiC`	ANSI X3.159-1989 ("ANSI C89")
`-ansiC-89`	ANSI X3.159-1989 ("ANSI C89")
`-isoC`	ISO/IEC 9899:1990 ("ISO C90")
`-isoC-90`	ISO/IEC 9899:1990 ("ISO C90")
`-isoC-99`	ISO/IEC 9899:1999 ("ISO C99")
`-isoC-2011`	ISO/IEC 9899:2011 ("ISO C11")

POSIX Part 1: System API

`-iso9945-1-90`	ISO/IEC 9945-1:1990 ("POSIX.1")
`-iso9945-1-96`	ISO/IEC 9945-1:1996 ("POSIX.1")
`-p1003.1`	IEEE Std 1003.1 ("POSIX.1")
`-p1003.1-88`	IEEE Std 1003.1-1988 ("POSIX.1")
`-p1003.1-90`	ISO/IEC 9945-1:1990 ("POSIX.1")
`-p1003.1-96`	ISO/IEC 9945-1:1996 ("POSIX.1")
`-p1003.1b-93`	IEEE Std 1003.1b-1993 ("POSIX.1")
`-p1003.1c-95`	IEEE Std 1003.1c-1995 ("POSIX.1")
`-p1003.1g-2000`	IEEE Std 1003.1g-2000 ("POSIX.1")
`-p1003.1i-95`	IEEE Std 1003.1i-1995 ("POSIX.1")
`-p1003.1-2001`	IEEE Std 1003.1-2001 ("POSIX.1")
`-p1003.1-2004`	IEEE Std 1003.1-2004 ("POSIX.1")
`-p1003.1-2008`	IEEE Std 1003.1-2008 ("POSIX.1")

POSIX Part 2: Shell and Utilities

`-iso9945-2-93`	ISO/IEC 9945-2:1993 ("POSIX.2")
`-p1003.2`	IEEE Std 1003.2 ("POSIX.2")
`-p1003.2-92`	IEEE Std 1003.2-1992 ("POSIX.2")
`-p1003.2a-92`	IEEE Std 1003.2a-1992 ("POSIX.2")

X/Open

`-susv1`	Version 1 of the Single UNIX Specification ("SUSv1")
`-susv2`	Version 2 of the Single UNIX Specification ("SUSv2")
`-susv3`	Version 3 of the Single UNIX Specification ("SUSv3")
`-susv4`	Version 4 of the Single UNIX Specification ("SUSv4")
`-svid4`	System V Interface Definition, Fourth Edition ("SVID4")
`-xbd5`	X/Open Base Definitions Issue 5 ("XBD5")
`-xcu5`	X/Open Commands and Utilities Issue 5 ("XCU5")
`-xcurses4.2`	X/Open Curses Issue 4, Version 2 ("XCURSES4.2")
`-xns5`	X/Open Networking Services Issue 5 ("XNS5")
`-xns5.2`	X/Open Networking Services Issue 5.2 ("XNS5.2")
`-xpg3`	X/Open Portability Guide Issue 3 ("XPG3")
`-xpg4`	X/Open Portability Guide Issue 4 ("XPG4")

`-xpg4.2`	X/Open Portability Guide Issue 4, Version 2 (“XPG4.2”)
`-xsh5`	X/Open System Interfaces and Headers Issue 5 (“XSH5”)

Miscellaneous

`-ieee754`	IEEE Std 754-1985
`-iso8601`	ISO 8601
`-iso8802-3`	ISO/IEC 8802-3:1989

Variable Types

The `.Vt` macro may be used whenever a type is referenced. In a section titled “Synopsis”, ‘`Vt`’ causes a break (useful for old-style C variable declarations).

`Usage: .Vt` ⟨type⟩ ...

```
.Vt extern char *optarg ; extern char *optarg;
.Vt FILE *                FILE *
```

Variables

Generic variable reference.

`Usage: .Va` ⟨variable⟩ ...

```
.Va count             count
.Va settimer ,        settimer,
.Va "int *prt" ) :    int *prt):
.Va "char s" ] ) ) ,  char s])),
```

The default width is 12n.

Manual Page Cross References

The `.Xr` macro expects the first argument to be a manual page name. The optional second argument, if a string (defining the manual section), is put into parentheses.

`Usage: .Xr` ⟨man page name⟩ [⟨section⟩] ...

```
.Xr mdoc       mdoc
.Xr mdoc ,     mdoc,
.Xr mdoc 7     mdoc(7)
.Xr xinit 1x ; xinit(1x);
```

The default width is 10n.

General text domain

AT&T Macro

`Usage: .At` [⟨version⟩] ...

```
.At       AT&T UNIX
.At v6 .  Version 6 AT&T UNIX.
```

The following values for ⟨version⟩ are possible:

```
32v, v1, v2, v3, v4, v5, v6, v7, III, V, V.1, V.2, V.3, V.4
```

BSD Macro

`Usage: .Bx` {−alpha | −beta | −devel} ...
`.Bx` [⟨version⟩ [⟨release⟩]] ...

```
.Bx          BSD
.Bx 4.3 .    4.3BSD.
.Bx -devel   BSD (currently under development)
```

⟨version⟩ will be prepended to the string ‘BSD’. The following values for ⟨release⟩ are possible:

```
Reno, reno, Tahoe, tahoe, Lite, lite, Lite2, lite2
```

NetBSD Macro

`Usage: .Nx` [⟨version⟩] ...

```
.Nx          NetBSD
.Nx 1.4 .    NetBSD 1.4.
```

For possible values of ⟨version⟩ see the description of the `.Os` command above in section "Title macros".

FreeBSD Macro

`Usage: .Fx` [⟨version⟩] ...

```
.Fx          FreeBSD
.Fx 2.2 .    FreeBSD 2.2.
```

For possible values of ⟨version⟩ see the description of the `.Os` command above in section "Title macros".

DragonFly Macro

`Usage: .Dx` [⟨version⟩] ...

```
.Dx          DragonFly
.Dx 1.4 .    DragonFly 1.4.
```

For possible values of ⟨version⟩ see the description of the `.Os` command above in section "Title macros".

OpenBSD Macro

`Usage: .Ox` [⟨version⟩] ...

```
.Ox 1.0 OpenBSD 1.0
```

BSD/OS Macro

`Usage: .Bsx` [⟨version⟩] ...

```
.Bsx 1.0 BSD/OS 1.0
```

Unix Macro

`Usage: .Ux ...`

```
.Ux Unix
```

Emphasis Macro

Text may be stressed or emphasized with the `.Em` macro. The usual font for emphasis is italic.

`Usage: .Em` ⟨argument⟩ ...

```
.Em does not          does not
.Em exceed 1024 .     exceed 1024.
.Em vide infra ) ) ,  vide infra)),
```

The default width is 10n.

Font Mode

The `.Bf` font mode must be ended with the `.Ef` macro (the latter takes no arguments). Font modes may be nested within other font modes.

`.Bf` has the following syntax:

`.Bf` ⟨font mode⟩

⟨font mode⟩ must be one of the following three types:

Em | `-emphasis`	Same as if the `.Em` macro was used for the entire block of text.
Li | `-literal`	Same as if the `.Li` macro was used for the entire block of text.
Sy | `-symbolic`	Same as if the `.Sy` macro was used for the entire block of text.

Both macros are neither callable nor parsed.

Enclosure and Quoting Macros

The concept of enclosure is similar to quoting. The object being to enclose one or more strings between a pair of characters like quotes or parentheses. The terms quoting and enclosure are used interchangeably

throughout this document. Most of the one-line enclosure macros end in small letter ‘q’ to give a hint of quoting, but there are a few irregularities. For each enclosure macro, there is a pair of opening and closing macros that end with the lowercase letters ‘o’ and ‘c’ respectively.

Quote	Open	Close	Function	Result
.Aq	.Ao	.Ac	Angle Bracket Enclosure	<string>
.Bq	.Bo	.Bc	Bracket Enclosure	[string]
.Brq	.Bro	.Brc	Brace Enclosure	{string}
.Dq	.Do	.Dc	Double Quote	“string”
.Eq	.Eo	.Ec	Enclose String (in XY)	XstringY
.Pq	.Po	.Pc	Parenthesis Enclosure	(string)
.Ql			Quoted Literal	“string” or string
.Qq	.Qo	.Qc	Straight Double Quote	"string"
.Sq	.So	.Sc	Single Quote	‘string’

All macros ending with ‘q’ and ‘o’ have a default width value of 12n.

`.Eo`, `.Ec` These macros expect the first argument to be the opening and closing strings, respectively.

`.Es`, `.En` To work around the nine-argument limit in the original *troff* program, *mdoc* supports two other macros that are now obsolete. `.Es` uses its first and second parameters as opening and closing marks which are then used to enclose the arguments of `.En`. The default width value is 12n for both macros.

`.Eq` The first and second arguments of this macro are the opening and closing strings respectively, followed by the arguments to be enclosed.

`.Ql` The quoted literal macro behaves differently in *troff* and *nroff* modes. If formatted with *nroff*(1), a quoted literal is always quoted. If formatted with *troff*, an item is only quoted if the width of the item is less than three constant-width characters. This is to make short strings more visible where the font change to literal (constant-width) is less noticeable.

The default width is 16n.

`.Pf` The prefix macro suppresses the whitespace between its first and second argument:

```
.Pf ( Fa name2 (name2
```

The default width is 12n.

The `.Ns` macro (see below) performs the analogous suffix function.

`.Ap` The `.Ap` macro inserts an apostrophe and exits any special text modes, continuing in `.No` mode.

Examples of quoting:

```
.Aq                     ⟨⟩
.Aq Pa ctype.h ) ,      ⟨ctype.h⟩),
.Bq                     []
.Bq Em Greek , French . [Greek, French].
.Dq                     “”
.Dq string abc .        “string abc”.
.Dq '\[ha][A-Z]'        “'^[A-Z]'”
.Ql man mdoc            man mdoc
.Qq                     ""
.Qq string ) ,          "string"),
.Qq string Ns ),        "string),"
.Sq                     ‘’
```

```
.Sq string              'string'
.Em or Ap ing           or'ing
```

For a good example of nested enclosure macros, see the `.Op` option macro. It was created from the same underlying enclosure macros as those presented in the list above. The `.Xo` and `.Xc` extended argument list macros are discussed below.

Normal text macro

'`No`' formats subsequent argument(s) normally, ending the effect of '`Em`' and similar. Parsing is *not* suppressed, so you must prefix words like '`No`' with '`\&`' to avoid their interpretation as *mdoc* macros.

Usage: `.No` *argument* ...

```
.Em Use caution No here .  → Use caution here.
.Em No dogs allowed .      → No dogs allowed.
.Em \&No dogs allowed .    → No dogs allowed.
```

The default width is 12n.

No-Space Macro

The `.Ns` macro suppresses insertion of a space between the current position and its first parameter. For example, it is useful for old style argument lists where there is no space between the flag and argument:

Usage: `...` ⟨argument⟩ Ns [⟨argument⟩] ...
`.Ns` ⟨argument⟩ ...

```
.Op Fl I Ns Ar directory [-Idirectory]
```

Note: The `.Ns` macro always invokes the `.No` macro after eliminating the space unless another macro name follows it. If used as a command (i.e., the second form above in the 'Usage' line), `.Ns` is identical to `.No`.

(Sub)section cross references

Use the `.Sx` macro to cite a (sub)section heading within the given document.

Usage: `.Sx` ⟨section-reference⟩ ...

```
.Sx Files → "Files"
```

The default width is 16n.

Symbolics

The symbolic emphasis macro is generally a boldface macro in either the symbolic sense or the traditional English usage.

Usage: `.Sy` ⟨symbol⟩ ...

```
.Sy Important Notice → Important Notice
```

The default width is 6n.

Mathematical Symbols

Use this macro for mathematical symbols and similar things.

Usage: `.Ms` ⟨math symbol⟩ ...

```
.Ms sigma → sigma
```

The default width is 6n.

References and Citations

The following macros make a modest attempt to handle references. At best, the macros make it convenient to manually drop in a subset of *refer*(1) style references.

`.Rs` Reference start (does not take arguments). In a section titled "See also", it causes a break and begins collection of reference information until the reference end macro is read.

`.Re` Reference end (does not take arguments). The reference is printed.
`.%A` Reference author name; one name per invocation.
`.%B` Book title.
`.%C` City/place.
`.%D` Date.
`.%I` Issuer/publisher name.
`.%J` Journal name.
`.%N` Issue number.
`.%O` Optional information.
`.%P` Page number.
`.%Q` Corporate or foreign author.
`.%R` Report name.
`.%T` Title of article.
`.%U` Optional hypertext reference.
`.%V` Volume.

Macros beginning with '`%`' are not callable but accept multiple arguments in the usual way. Only the `.Tn` macro is handled properly as a parameter; other macros will cause strange output. `.%B` and `.%T` can be used outside of the `.Rs`/`.Re` environment.

Example:

```
.Rs
.%A "Matthew Bar"
.%A "John Foo"
.%T "Implementation Notes on foobar(1)"
.%R "Technical Report ABC-DE-12-345"
.%Q "Drofnats College"
.%C "Nowhere"
.%D "April 1991"
.Re
```

produces

> Matthew Bar and John Foo, *Implementation Notes on foobar(1)*, Technical Report ABC-DE-12-345, Drofnats College, Nowhere, April 1991.

Trade Names or Acronyms

The trade name macro prints its arguments at a smaller type size. It is intended to imitate a small caps fonts for fully capitalized acronyms.

`Usage: .Tn` ⟨symbol⟩ ...

```
.Tn DEC   DEC
.Tn ASCII ASCII
```

The default width is 10n.

Extended Arguments

The `.Xo` and `.Xc` macros allow one to extend an argument list on a macro boundary for the `.It` macro (see below). Note that `.Xo` and `.Xc` are implemented similarly to all other macros opening and closing an enclosure (without inserting characters, of course). This means that the following is true for those macros also.

Here is an example of `.Xo` using the space mode macro to turn spacing off:

```
.Bd -literal -offset indent
.Sm off
.It Xo Sy I Ar operation
.No \en Ar count No \en
.Xc
```

```
.Sm on
.Ed
```

produces

I*operation*\n*count*\n

Another one:

```
.Bd -literal -offset indent
.Sm off
.It Cm S No / Ar old_pattern Xo
.No / Ar new_pattern
.No / Op Cm g
.Xc
.Sm on
.Ed
```

produces

S/*old_pattern*/*new_pattern*/[g]

Another example of `.Xo` and enclosure macros: Test the value of a variable.

```
.Bd -literal -offset indent
.It Xo
.Ic .ifndef
.Oo \&! Oc Ns Ar variable Oo
.Ar operator variable No ...
.Oc Xc
.Ed
```

produces

.ifndef [!]*variable* [*operator variable ...*]

Page structure domain

Section headings

The following `.Sh` section heading macros are required in every man page. The remaining section headings are recommended at the discretion of the author writing the manual page. The `.Sh` macro is parsed but not generally callable. It can be used as an argument in a call to `.Sh` only; it then reactivates the default font for `.Sh`.

The default width is 8n.

`.Sh Name` The `.Sh Name` macro is mandatory. If not specified, headers, footers, and page layout defaults will not be set and things will be rather unpleasant. The *Name* section consists of at least three items. The first is the `.Nm` name macro naming the subject of the man page. The second is the name description macro, `.Nd`, which separates the subject name from the third item, which is the description. The description should be the most terse and lucid possible, as the space available is small.

`.Nd` first prints '−', then all its arguments.

`.Sh Library` This section is for section two and three function calls. It should consist of a single `.Lb` macro call; see "Library Names".

`.Sh Synopsis` The "Synopsis" section describes the typical usage of the subject of a man page. The macros required are either `.Nm`, `.Cd`, or `.Fn` (and possibly `.Fo`, `.Fc`, `.Fd`, and `.Ft`). The function name macro `.Fn` is required for manual page sections 2 and 3; the command and general name macro `.Nm` is required for sections 1, 5, 6, 7, and 8. Section 4 manuals require a `.Nm`, `.Fd` or a `.Cd`

configuration device usage macro. Several other macros may be necessary to produce the synopsis line as shown below:

cat [-benstuv] [-] *file* ...

The following macros were used:

```
.Nm cat
.Op Fl benstuv
.Op Fl
.Ar file No ...
```

`.Sh Description` In most cases the first text in the "Description" section is a brief paragraph on the command, function or file, followed by a lexical list of options and respective explanations. To create such a list, the `.Bl` (begin list), `.It` (list item) and `.El` (end list) macros are used (see "Lists and Columns" below).

`.Sh Implementation notes`
Implementation specific information should be placed here.

`.Sh Return values` Sections 2, 3 and 9 function return values should go here. The `.Rv` macro may be used to generate text for use in the "Return values" section for most section 2 and 3 library functions; see "Return Values".

The following `.Sh` section headings are part of the preferred manual page layout and must be used appropriately to maintain consistency. They are listed in the order in which they would be used.

`.Sh Environment` The *Environment* section should reveal any related environment variables and clues to their behavior and/or usage.

`.Sh Files` Files which are used or created by the man page subject should be listed via the `.Pa` macro in the "Files" section.

`.Sh Examples` There are several ways to create examples. See subsection "Examples and Displays" below for details.

`.Sh Diagnostics` Diagnostic messages from a command should be placed in this section. The `.Ex` macro may be used to generate text for use in the "Diagnostics" section for most section 1, 6 and 8 commands; see "Exit Status".

`.Sh Compatibility` Known compatibility issues (e.g. deprecated options or parameters) should be listed here.

`.Sh Errors` Specific error handling, especially from library functions (man page sections 2, 3, and 9) should go here. The `.Er` macro is used to specify an error (errno).

`.Sh See also` References to other material on the man page topic and cross references to other relevant man pages should be placed in the "See also" section. Cross references are specified using the `.Xr` macro. Currently *refer*(1) style references are not accommodated.

It is recommended that the cross references be sorted by section number, then alphabetically by name within each section, then separated by commas. Example:

ls(1), *ps*(1), *group*(5), *passwd*(5)

`.Sh Standards` If the command, library function, or file adheres to a specific implementation such as IEEE Std 1003.2 ("POSIX.2") or ANSI X3.159-1989 ("ANSI C89") this should be noted here. If the command does not adhere to any standard, its history should be noted in the *History* section.

`.Sh History` Any command which does not adhere to any specific standards should be outlined historically in this section.

`.Sh Authors` Credits should be placed here. Use the `.An` macro for names and the `.Aq` macro for email addresses within optional contact information. Explicitly indicate whether the person authored the initial manual page or the software or whatever the person is being credited for.

`.Sh Bugs` Blatant problems with the topic go here.

User-specified `.Sh` sections may be added; for example, this section was set with:

```
.Sh "Page structure domain"
```

Subsection headings

Subsection headings have exactly the same syntax as section headings: `.Ss` is parsed but not generally callable. It can be used as an argument in a call to `.Ss` only; it then reactivates the default font for `.Ss`.

The default width is 8n.

Paragraphs and Line Spacing

`.Pp` The `.Pp` paragraph command may be used to specify a line space where necessary. The macro is not necessary after a `.Sh` or `.Ss` macro or before a `.Bl` or `.Bd` macro (which both assert a vertical distance unless the `-compact` flag is given).

The macro is neither callable nor parsed and takes no arguments; an alternative name is `.Lp`.

Keeps

The only keep that is implemented at this time is for words. The macros are `.Bk` (begin keep) and `.Ek` (end keep). The only option that `.Bk` currently accepts is `-words` (also the default); this prevents breaks in the middle of options. In the example for **`make`** command-line arguments (see "What's in a Name"), the keep prevents *nroff* from placing the flag and the argument on separate lines.

Neither macro is callable or parsed.

More work needs to be done on the keep macros; specifically, a `-line` option should be added.

Examples and Displays

There are seven types of displays.

`.D1` (This is D-one.) Display one line of indented text. This macro is parsed but not callable.

```
-ldghfstru
```

The above was produced by: `.D1 Fl ldghfstru`.

`.Dl` (This is D-ell.) Display one line of indented *literal* text. The `.Dl` example macro has been used throughout this file. It allows the indentation (display) of one line of text. Its default font is set to constant width (literal). `.Dl` is parsed but not callable.

```
% ls -ldg /usr/local/bin
```

The above was produced by: `.Dl % ls \-ldg /usr/local/bin`.

`.Bd` Begin display. The `.Bd` display must be ended with the `.Ed` macro. It has the following syntax:

`.Bd` {−literal | −filled | −unfilled | −ragged | −centered} [−offset ⟨string⟩] [−file ⟨file name⟩] [−compact]

`-ragged`	Fill, but do not adjust the right margin (only left-justify).
`-centered`	Center lines between the current left and right margin. Note that each single line is centered.
`-unfilled`	Do not fill; break lines where their input lines are broken. This can produce overlong lines without warning messages.
`-filled`	Display a filled block. The block of text is formatted (i.e., the text is justified on both the left and right side).

`-literal` Display block with literal font (usually fixed-width). Useful for source code or simple tabbed or spaced text.

`-file` ⟨*file name*⟩ The file whose name follows the `-file` flag is read and displayed before any data enclosed with `.Bd` and `.Ed`, using the selected display type. Any *troff/mdoc* commands in the file will be processed.

`-offset` ⟨*string*⟩ If `-offset` is specified with one of the following strings, the string is interpreted to indicate the level of indentation for the forthcoming block of text:

- *left* Align block on the current left margin; this is the default mode of `.Bd`.
- *center* Supposedly center the block. At this time unfortunately, the block merely gets left aligned about an imaginary center margin.
- *indent* Indent by one default indent value or tab. The default indent value is also used for the `.D1` and `.Dl` macros, so one is guaranteed the two types of displays will line up. The indentation value is normally set to 6n or about two thirds of an inch (six constant width characters).
- *indent-two* Indent two times the default indent value.
- *right* This *left* aligns the block about two inches from the right side of the page. This macro needs work and perhaps may never do the right thing within *troff*.

If ⟨string⟩ is a valid numeric expression instead (*with a scaling indicator other than* '*u*'), use that value for indentation. The most useful scaling indicators are 'm' and 'n', specifying the so-called *Em* and *En square*. This is approximately the width of the letters 'm' and 'n' respectively of the current font (for *nroff* output, both scaling indicators give the same values). If ⟨string⟩ isn't a numeric expression, it is tested whether it is an *mdoc* macro name, and the default offset value associated with this macro is used. Finally, if all tests fail, the width of ⟨string⟩ (typeset with a fixed-width font) is taken as the offset.

`-compact` Suppress insertion of vertical space before begin of display.

`.Ed` End display (takes no arguments).

Lists and Columns

There are several types of lists which may be initiated with the `.Bl` begin-list macro. Items within the list are specified with the `.It` item macro, and each list must end with the `.El` macro. Lists may be nested within themselves and within displays. The use of columns inside of lists or lists inside of columns is untested.

In addition, several list attributes may be specified such as the width of a tag, the list offset, and compactness (blank lines between items allowed or disallowed). Most of this document has been formatted with a tag style list (`-tag`).

It has the following syntax forms:

```
.Bl {-hang | -ohang | -tag | -diag | -inset} [-width ⟨string⟩] [-offset ⟨string⟩] [-compact]
.Bl -column [-offset ⟨string⟩] ⟨string1⟩ ⟨string2⟩ ...
.Bl {-item | -enum [-nested] | -bullet | -hyphen | -dash} [-offset ⟨string⟩] [-compact]
```

And now a detailed description of the list types.

`-bullet` A bullet list.

```
.Bl -bullet -offset indent -compact
.It
Bullet one goes here.
```

```
.It
Bullet two here.
.El
```

Produces:

- Bullet one goes here.
- Bullet two here.

`-dash` (or `-hyphen`)
A dash list.

```
.Bl -dash -offset indent -compact
.It
Dash one goes here.
.It
Dash two here.
.El
```

Produces:

- Dash one goes here.
- Dash two here.

`-enum` An enumerated list.

```
.Bl -enum -offset indent -compact
.It
Item one goes here.
.It
And item two here.
.El
```

The result:

1. Item one goes here.
2. And item two here.

If you want to nest enumerated lists, use the `-nested` flag (starting with the second-level list):

```
.Bl -enum -offset indent -compact
.It
Item one goes here
.Bl -enum -nested -compact
.It
Item two goes here.
.It
And item three here.
.El
.It
And item four here.
.El
```

Result:

1. Item one goes here.
   1.1. Item two goes here.
   1.2. And item three here.
2. And item four here.

`-item` A list of type `-item` without list markers.

```
.Bl -item -offset indent
.It
Item one goes here.
Item one goes here.
Item one goes here.
.It
Item two here.
Item two here.
Item two here.
.El
```

Produces:

Item one goes here. Item one goes here. Item one goes here.

Item two here. Item two here. Item two here.

`-tag` A list with tags. Use `-width` to specify the tag width.

SL sleep time of the process (seconds blocked)

PAGEIN
: number of disk I/O operations resulting from references by the process to pages not loaded in core.

UID numerical user-id of process owner

PPID numerical id of parent of process priority (non-positive when in non-interruptible wait)

The raw text:

```
.Bl -tag -width "PPID" -compact -offset indent
.It SL
sleep time of the process (seconds blocked)
.It PAGEIN
number of disk I/O operations resulting from references
by the process to pages not loaded in core.
.It UID
numerical user-id of process owner
.It PPID
numerical id of parent of process priority
(non-positive when in non-interruptible wait)
.El
```

`-diag` Diag lists create section four diagnostic lists and are similar to inset lists except callable macros are ignored. The `-width` flag is not meaningful in this context.

Example:

```
.Bl -diag
.It You can't use Sy here.
The message says all.
.El
```

produces

You can't use Sy here. The message says all.

`-hang` A list with hanging tags.

Hanged labels appear similar to tagged lists when the label is smaller than the label width.

Longer hanged list labels blend into the paragraph unlike tagged paragraph labels.

And the unformatted text which created it:

```
.Bl -hang -offset indent
.It Em Hanged
labels appear similar to tagged lists when the
label is smaller than the label width.
.It Em Longer hanged list labels
blend into the paragraph unlike
tagged paragraph labels.
.El
```

`-ohang` Lists with overhanging tags do not use indentation for the items; tags are written to a separate line.

SL
sleep time of the process (seconds blocked)

PAGEIN
number of disk I/O operations resulting from references by the process to pages not loaded in core.

UID
numerical user-id of process owner

PPID
numerical id of parent of process priority (non-positive when in non-interruptible wait)

The raw text:

```
.Bl -ohang -offset indent
.It Sy SL
sleep time of the process (seconds blocked)
.It Sy PAGEIN
number of disk I/O operations resulting from references
by the process to pages not loaded in core.
.It Sy UID
numerical user-id of process owner
.It Sy PPID
numerical id of parent of process priority
(non-positive when in non-interruptible wait)
.El
```

`-inset` Here is an example of inset labels:

Tag The tagged list (also called a tagged paragraph) is the most common type of list used in the Berkeley manuals. Use a `-width` attribute as described below.

Diag Diag lists create section four diagnostic lists and are similar to inset lists except callable macros are ignored.

Hang Hanged labels are a matter of taste.

Ohang Overhanging labels are nice when space is constrained.

Inset Inset labels are useful for controlling blocks of paragraphs and are valuable for converting *mdoc* manuals to other formats.

Here is the source text which produced the above example:

```
.Bl -inset -offset indent
.It Em Tag
The tagged list (also called a tagged paragraph)
```

```
is the most common type of list used in the
Berkeley manuals.
.It Em Diag
Diag lists create section four diagnostic lists
and are similar to inset lists except callable
macros are ignored.
.It Em Hang
Hanged labels are a matter of taste.
.It Em Ohang
Overhanging labels are nice when space is constrained.
.It Em Inset
Inset labels are useful for controlling blocks of
paragraphs and are valuable for converting
.Xr mdoc
manuals to other formats.
.El
```

`-column` This list type generates multiple columns. The number of columns and the width of each column is determined by the arguments to the `-column` list, ⟨*string1*⟩, ⟨*string2*⟩, etc. If ⟨*stringN*⟩ starts with a '`.`' (dot) immediately followed by a valid *mdoc* macro name, interpret ⟨*stringN*⟩ and use the width of the result. Otherwise, the width of ⟨*stringN*⟩ (typeset with a fixed-width font) is taken as the *N*th column width.

Each `.It` argument is parsed to make a row, each column within the row is a separate argument separated by a tab or the `.Ta` macro.

The table:

String	Nroff	Troff
<=	<=	≤
>=	>=	≥

was produced by:

```
.Bl -column -offset indent ".Sy String" ".Sy Nroff" ".Sy Troff"
.It Sy String Ta Sy Nroff Ta Sy Troff
.It Li <= Ta <= Ta \*(<=
.It Li >= Ta >= Ta \*(>=
.El
```

Don't abuse this list type! For more complicated cases it might be far better and easier to use *tbl*(1), the table preprocessor.

Other keywords:

`-width` ⟨*string*⟩ If ⟨*string*⟩ starts with a '`.`' (dot) immediately followed by a valid *mdoc* macro name, interpret ⟨*string*⟩ and use the width of the result. Almost all lists in this document use this option.

Example:

```
.Bl -tag -width ".Fl test Ao Ar string Ac"
.It Fl test Ao Ar string Ac
This is a longer sentence to show how the
.Fl width
flag works in combination with a tag list.
.El
```

gives:

`-test` ⟨*string*⟩ This is a longer sentence to show how the `-width` flag works in combination with a tag list.

(Note that the current state of *mdoc* is saved before ⟨*string*⟩ is interpreted; afterwards, all variables are restored again. However, boxes (used for enclosures) can't be saved in GNU *troff*(1); as a consequence, arguments must always be *balanced* to avoid nasty errors. For example, do not write `.Ao Ar string` but `.Ao Ar string Xc` instead if you really need only an opening angle bracket.)

Otherwise, if ⟨*string*⟩ is a valid numeric expression (*with a scaling indicator other than* '*u*'), use that value for indentation. The most useful scaling indicators are 'm' and 'n', specifying the so-called *Em* and *En square*. This is approximately the width of the letters 'm' and 'n' respectively of the current font (for *nroff* output, both scaling indicators give the same values). If ⟨*string*⟩ isn't a numeric expression, it is tested whether it is an *mdoc* macro name, and the default width value associated with this macro is used. Finally, if all tests fail, the width of ⟨*string*⟩ (typeset with a fixed-width font) is taken as the width.

If a width is not specified for the tag list type, '6n' is used.

`-offset` ⟨*string*⟩ If ⟨*string*⟩ is *indent*, a default indent value (normally set to 6n, similar to the value used in `.Dl` or `.Bd`) is used. If ⟨*string*⟩ is a valid numeric expression instead (*with a scaling indicator other than* '*u*'), use that value for indentation. The most useful scaling indicators are 'm' and 'n', specifying the so-called *Em* and *En square*. This is approximately the width of the letters 'm' and 'n' respectively of the current font (for *nroff* output, both scaling indicators give the same values). If ⟨*string*⟩ isn't a numeric expression, it is tested whether it is an *mdoc* macro name, and the default offset value associated with this macro is used. Finally, if all tests fail, the width of ⟨*string*⟩ (typeset with a fixed-width font) is taken as the offset.

`-compact` Suppress insertion of vertical space before the list and between list items.

Miscellaneous macros

A double handful of macros fit only uncomfortably into one of the above sections. Of these, we couldn't find attested examples for '`Me`' or '`Ot`'. They are documented here for completeness—if you know their proper usage, please send a mail to *groff@gnu.org* and include a specimen with its provenance.

`.Bt` formats boilerplate text.

`.Bt` → is currently in beta test.

It is neither callable nor parsed and takes no arguments. Its default width is 6n.

`.Fr` is an obsolete means of specifying a function return value.

Usage: .**Fr** *return-value ...*

'`Fr`' allows a break right before the return value (usually a single digit) which is bad typographical behaviour. Instead, set the return value with the rest of the code, using '`\~`' to tie the return value to the previous word.

Its default width is 12n.

`.Hf` Inlines the contents of a (header) file into the document.

Usage: .**Hf** *file*

It first prints `File:` followed by the file name, then the contents of *file*. It is neither callable nor parsed.

`.Lk` Embed hyperlink.

Usage: .**Lk** *uri* [*link-text*]

Its default width is 6n.

`.Me` Usage unknown. The *mdoc* sources describe it as a macro for "menu entries".

Its default width is 6n.

`.Mt` Embed email address.

Usage: .**Mt** *email-address*

Its default width is 6n.

`.Ot` Usage unknown. The *mdoc* sources describe it as "old function type (fortran)".

`.Sm` Manipulate or toggle argument-spacing mode.

Usage: .**Sm** [`on` | `off`] ...

If argument-spacing mode is off, no spaces between macro arguments are inserted. If called without a parameter (or if the next parameter is neither '`on`' nor `off`), '`Sm`' toggles argument-spacing mode.

Its default width is 8n.

`.Ud` formats boilerplate text.

`.Ud` → currently under development.

It is neither callable nor parsed and takes no arguments. Its default width is 8n.

Predefined strings

The following strings are predefined for compatibility with legacy *mdoc* documents. Contemporary ones should use the alternatives shown in the "Prefer" column below. See *groff_char*(7) for a full discussion of these special character escape sequences.

String	7-bit	8-bit	UCS	Prefer	Meaning
*(<=	<=	<=	≤	\(<=	less than or equal to
*(>=	>=	>=	≥	\(>=	greater than or equal to
*(Rq	"	"	”	\(rq	right double quote
*(Lq	"	"	“	\(lq	left double quote
*(ua	^	^	↑	\(ua	vertical arrow up
*(aa	'	´	´	\(aa	acute accent
*(ga	`	`	`	\(ga	grave accent
*(q	"	"	"	\(dq	neutral double quote
*(Pi	pi	pi	π	\(*p	lowercase pi
*(Ne	!=	!=	≠	\(!=	not equals
*(Le	<=	<=	≤	\(<=	less than or equal to
*(Ge	>=	>=	≥	\(>=	greater than or equal to
*(Lt	<	<	<	<	less than
*(Gt	>	>	>	>	greater than
*(Pm	+-	±	±	\(+-	plus or minus
*(If	infinity	infinity	∞	\(if	infinity
*(Am	&	&	&	&	ampersand
*(Na	*NaN*	*NaN*	*NaN*	NaN	not a number
*(Ba	|	|	|	|	bar

Some column headings are shorthand for standardized character encodings; "7-bit" for ISO 646:1991 IRV (US-ASCII), "8-bit" for ISO 8859-1 (Latin-1) and IBM code page 1047, and "UCS" for ISO 10646 (Unicode character set). Historically, *mdoc* configured the string definitions to fit the capabilities expected of the output device. Old typesetters lacked directional double quotes, producing repeated directional single quotes ''like this''; early versions of *mdoc* in fact defined the '`Lq`' and '`Rq`' strings this way. Nowadays, output drivers take on the responsibility of glyph substitution, as they possess relevant knowledge of their available repertoires.

Diagnostics

The debugging macro `.Db` offered by previous versions of *mdoc* is unavailable in GNU *troff*(1) since the latter provides better facilities to check parameters; additionally, *groff mdoc* implements many error and warning messages, making the package more robust and more verbose.

The remaining debugging macro is `.Rd`, which dumps the package's global register and string contents to the standard error stream. A normal user will never need it.

Options

The following *groff* options set registers (with `-r`) and strings (with `-d`) recognized and used by the *mdoc* macro package. To ensure rendering consistent with output device capabilities and reader preferences, man pages should never manipulate them.

Setting string '`AD`' configures the adjustment mode for most formatted text. Typical values are '`b`' for adjustment to both margins (the default), or '`l`' for left alignment (ragged right margin). Any valid argument to *groff*'s '`ad`' request may be used. See *groff*(7) for less-common choices.

```
groff -Tutf8 -dAD=l -mdoc groff_mdoc.7 | less -R
```

Setting register '`C`' to 1 numbers output pages consecutively, rather than resetting the page number to 1 (or the value of register '`P`') with each new *mdoc* document.

By default, the package inhibits page breaks, headers, and footers in the midst of the document text if it is being displayed with a terminal device such as 'latin1' or 'utf8', to enable more efficient viewing of the page. This behavior can be changed to format the page as if for 66-line Teletype output by setting the continuous rendering register '`cR`' to zero while calling *groff*(1).

```
groff -Tlatin1 -rcR=0 -mdoc foo.man > foo.txt
```

On HTML devices, it cannot be disabled.

Section headings (defined with `.Sh`) and page titles in headers (defined with `.Dt`) can be presented in full capitals by setting the registers '`CS`' and '`CT`', respectively, to 1. These transformations are off by default because they discard case distinction information.

Setting register '`D`' to 1 enables double-sided page layout, which is only distinct when not continuously rendering. It places the page number at the bottom right on odd-numbered (recto) pages, and at the bottom left on even-numbered (verso) pages, swapping places with the arguments to `.Os`.

```
groff -Tps -rD1 -mdoc foo.man > foo.ps
```

The value of the '`FT`' register determines the footer's distance from the page bottom; this amount is always negative and should specify a scaling unit. At one half-inch above this location, the page text is broken before writing the footer. It is ignored if continuous rendering is enabled. The default is –0.5i.

The '`HF`' string sets the font used for section and subsection headings; the default is '`B`' (bold style of the default family). Any valid argument to *groff*'s '`ft`' request may be used.

Normally, automatic hyphenation is enabled using a mode appropriate to the *groff* locale; see section "Localization" of *groff*(7). It can be disabled by setting the '`HY`' register to zero.

```
groff -Tutf8 -rHY=0 -mdoc foo.man | less -R
```

The paragraph and subsection heading indentation amounts can be changed by setting the registers '`IN`' and '`SN`'.

```
groff -Tutf8 -rIN=5n -rSN=2n -mdoc foo.man | less -R
```

The default paragraph indentation is 7.2n on typesetters and 7n on terminals. The default subsection heading indentation amount is 3n; section headings are set with an indentation of zero.

The line and title lengths can be changed by setting the registers '`LL`' and '`LT`', respectively:

```
groff -Tutf8 -rLL=100n -rLT=100n -mdoc foo.man | less -R
```

If not set, both registers default to 78n for terminal devices and 6.5i otherwise.

Setting the '`P`' register starts enumeration of pages at its value. The default is 1.

To change the document font size to 11p or 12p, set register '`S`' accordingly:

```
groff -Tdvi -rS11 -mdoc foo.man > foo.dvi
```

Register '`S`' is ignored when formatting for terminal devices.

Setting the ‘X’ register to a page number *p* numbers its successors as *p*a, *p*b, *p*c, and so forth. The register tracking the suffixed page letter uses format ‘a’ (see the ‘af’ request in *groff*(7)).

Files

/usr/local/share/groff/1.23.0/tmac/andoc.tmac
: This brief *groff* program detects whether the *man* or *mdoc* macro package is being used by a document and loads the correct macro definitions, taking advantage of the fact that pages using them must call TH or Dd, respectively, before any other macros. A user typing, for example,

```
groff -mandoc page.1
```

need not know which package the file *page.1* uses. Multiple man pages, in either format, can be handled; *andoc.tmac* reloads each macro package as necessary.

/usr/local/share/groff/1.23.0/tmac/doc.tmac
: implements the bulk of the *groff mdoc* package and loads further components as needed from the *mdoc* subdirectory.

/usr/local/share/groff/1.23.0/tmac/mdoc.tmac
: is a wrapper that loads *doc.tmac*.

/usr/local/share/groff/1.23.0/tmac/mdoc/doc-common
: defines macros, registers, and strings concerned with the production of formatted output. It includes strings of the form doc-volume-ds-*X* and doc-volume-as-*X* for manual section titles and architecture identifiers, respectively, where *X* is an argument recognized by **.Dt**.

/usr/local/share/groff/1.23.0/tmac/mdoc/doc-nroff
: defines parameters appropriate for rendering to terminal devices.

/usr/local/share/groff/1.23.0/tmac/mdoc/doc-ditroff
: defines parameters appropriate for rendering to typesetter devices.

/usr/local/share/groff/1.23.0/tmac/mdoc/doc-syms
: defines many strings and macros that interpolate formatted text, such as names of operating system releases, ∗BSD libraries, and standards documents. The string names are of the form doc-str-*O*-*V*, doc-str-St--*S*-*I* (observe the double dashes), or doc-str-Lb-*L*, where *O* is one of the operating system macros from section “General text domain” above, *V* is an encoding of an operating system release (sometimes omitted along with the ‘-’ preceding it), *S* an identifier for a standards body or committee, *I* one for an issue of a standard promulgated by *S*, and *L* a keyword identifying a ∗BSD library.

/usr/local/share/groff/site-tmac/mdoc.local
: This file houses local additions and customizations to the package. It can be empty.

See also

The *mandoc*: https://mandoc.bsd.lv/ project maintains an independent implementation of the *mdoc* language and a renderer that directly parses its markup as well as that of *man*.

groff(1), *man*(1), *troff*(1), *groff_man*(7), *mdoc*(7)

Bugs

Section 3f has not been added to the header routines.

.Fn needs to have a check to prevent splitting up the line if its length is too short. Occasionally it separates the last parenthesis, and sometimes looks ridiculous if output lines are being filled.

The list and display macros do not do any keeps and certainly should be able to.

As of *groff* 1.23, ‘Tn’ no longer changes the type size; this functionality may return in the next release.