NAME

mdoc.samples − tutorial sampler for writing BSD manuals with **−mdoc**

SYNOPSIS

man mdoc.samples

DESCRIPTION

A tutorial sampler for writing BSD manual pages with the **−mdoc** macro package, a *content*−based and *domain*−based formatting package for `troff`(1). Its predecessor, the `−man`(7) package, addressed page layout leaving the manipulation of fonts and other typesetting details to the individual author. In **−mdoc**, page layout macros make up the *page structure domain* which consists of macros for titles, section headers, displays and lists. Essentially items which affect the physical position of text on a formatted page. In addition to the page structure domain, there are two more domains, the manual domain and the general text domain. The general text domain is defined as macros which perform tasks such as quoting or emphasizing pieces of text. The manual domain is defined as macros that are a subset of the day to day informal language used to describe commands, routines and related BSD files. Macros in the manual domain handle command names, command-line arguments and options, function names, function parameters, pathnames, variables, cross references to other manual pages, and so on. These domain items have value for both the author and the future user of the manual page. It is hoped the consistency gained across the manual set will provide easier translation to future documentation tools.

Throughout the UNIX manual pages, a manual entry is simply referred to as a man page, regardless of actual length and without sexist intention.

GETTING STARTED

Since a tutorial document is normally read when a person desires to use the material immediately, the assumption has been made that the user of this document may be impatient. The material presented in the remained of this document is outlined as follows:

1. TROFF IDIOSYNCRASIES
   - Macro Usage.
   - Passing Space Characters in an Argument.
   - Trailing Blank Space Characters (a warning).
   - Escaping Special Characters.
2. THE ANATOMY OF A MAN PAGE
   - A manual page template.
3. TITLE MACROS.
4. INTRODUCTION OF MANUAL AND GENERAL TEXT DOMAINS.
   - What’s in a name....
   - General Syntax.
5. MANUAL DOMAIN
   - Addresses.
   - Author name.
   - Arguments.
   - Configuration Declarations (section four only).
   - Command Modifier.
   - Defined Variables.

TROFF IDIOSYNCRASIES

The **-mdoc** package attempts to simplify the process of writing a man page. Theoretically, one should not have to learn the dirty details of `troff`(1) to use **-mdoc**; however, there are a few limitations which are unavoidable and best gotten out of the way. And, too, be forewarned, this package is *not* fast.

Macro Usage

As in `troff`(1), a macro is called by placing a '`.`' (dot character) at the beginning of a line followed by the two character name for the macro. Arguments may follow the macro separated by spaces. It is the dot character at the beginning of the line which causes `troff`(1) to interpret the next two characters as a macro name. To place a '`.`' (dot character) at the beginning of a line in some context other than a macro invocation, precede the '`.`' (dot) with the '`\&`' escape sequence. The '`\&`' translates literally to a zero width space, and is never displayed in the output.

In general, `troff`(1) macros accept up to nine arguments, any extra arguments are ignored. Most macros in **-mdoc** accept nine arguments and, in limited cases, arguments may be continued or extended on the next line (See **Extensions**). A few macros handle quoted arguments (see **Passing Space Characters in an Argument** below).

Most of the **-mdoc** general text domain and manual domain macros are special in that their argument lists are *parsed* for callable macro names. This means an argument on the argument list which matches a general text or manual domain macro name and is determined to be callable will be executed or called when it is processed. In this case the argument, although the name of a macro, is not preceded by a '`.`' (dot). It is in this manner that many macros are nested; for example the option macro, `.Op`, may *call* the flag and argument macros, '`Fl`' and '`Ar`', to specify an optional flag with an argument:

[**-s** *bytes*] is produced by `.Op Fl s Ar bytes`

To prevent a two character string from being interpreted as a macro name, precede the string with the escape sequence '`\&`':

[Fl s Ar bytes] is produced by `.Op \&Fl s \&Ar bytes`

Here the strings '`Fl`' and '`Ar`' are not interpreted as macros. Macros whose argument lists are parsed for callable arguments are referred to as parsed and macros which may be called from an argument list are referred to as callable throughout this document and in the companion quick reference manual `mdoc`(7). This is a technical *faux pas* as almost all of the macros in **-mdoc** are parsed, but as it was cumbersome to constantly refer to macros as being callable and being able to call other macros, the term parsed has been used.

Passing Space Characters in an Argument

Sometimes it is desirable to give as one argument a string containing one or more blank space characters. This may be necessary to defeat the nine argument limit or to specify arguments to macros which expect particular arrangement of items in the argument list. For example, the function macro `.Fn` expects the first argument to be the name of a function and any remaining arguments to be function parameters. As ANSI C stipulates the declaration of function parameters in the parenthesized parameter list, each parameter is guaranteed to be at minimum a two word string. For example, *`int foo`*.

There are two possible ways to pass an argument which contains an embedded space. *Implementation note*: Unfortunately, the most convenient way of passing spaces in between quotes by reassigning individual arguments before parsing was fairly expensive speed wise and space wise to implement in all the macros for AT&T `troff`. It is not expensive for `groff` but for the sake of portability, has been limited to the following macros which need it the most:

`Cd` Configuration declaration (section 4 **SYNOPSIS**)
`Bl` Begin list (for the width specifier).
`Em` Emphasized text.
`Fn` Functions (sections two and four).
`It` List items.
`Li` Literal text.
`Sy` Symbolic text.
`%B` Book titles.
`%J` Journal names.
`%O` Optional notes for a reference.
`%R` Report title (in a reference).
`%T` Title of article in a book or journal.

One way of passing a string containing blank spaces is to use the hard or unpaddable space character '`\ `', that is, a blank space preceded by the escape character '`\`'. This method may be used with any macro but has the side effect of interfering with the adjustment of text over the length of a line. `Troff` sees the hard space as if it were any other printable character and cannot split the string into blank or newline separated pieces as one would expect. The method is useful for strings which are not expected to overlap a line boundary. For example:

fetch(*char *str*) is created by `.Fn fetch char\ *str`

fetch(*char *str*) can also be created by `.Fn fetch "char *str"`

If the '`\`' or quotes were omitted, `.Fn` would see three arguments and the result would be:

fetch(*char*, **str*)

For an example of what happens when the parameter list overlaps a newline boundary, see the **BUGS** section.

Trailing Blank Space Characters

`Troff` can be confused by blank space characters at the end of a line. It is a wise preventive measure to globally remove all blank spaces from <blank-space><end-of-line> character sequences. Should the need arise to force a blank character at the end of a line, it may be forced with an unpaddable space and the '`\&`' escape character. For example, `string\ \&`.

Escaping Special Characters

Special characters like the newline character '`\n`', are handled by replacing the '`\`' with '`\e`' (e.g. `\en`) to preserve the backslash.

THE ANATOMY OF A MAN PAGE

The body of a man page is easily constructed from a basic template found in the file `/usr/share/misc/mdoc.template`. Several example man pages can also be found in `/usr/share/examples/mdoc`.

A manual page template

```
.\" The following requests are required for all man pages.
.Dd Month day, year
.Os OPERATING_SYSTEM [version/release]
.Dt DOCUMENT_TITLE [section number] [volume]
.Sh NAME
.Nm name
.Nd one line description of name
```

```
.Sh SYNOPSIS
.Sh DESCRIPTION
.\" The following requests should be uncommented and
.\" used where appropriate.  This next request is
.\" for sections 2 and 3 function return values only.
.\" .Sh RETURN VALUE
.\" This next request is for sections 1, 6, 7 & 8 only
.\" .Sh ENVIRONMENT
.\" .Sh FILES
.\" .Sh EXAMPLES
.\" This next request is for sections 1, 6, 7 & 8 only
.\"     (command return values (to shell) and
.\"       fprintf/stderr type diagnostics)
.\" .Sh DIAGNOSTICS
.\" The next request is for sections 2 and 3 error
.\" and signal handling only.
.\" .Sh ERRORS
.\" .Sh SEE ALSO
.\" .Sh CONFORMING TO
.\" .Sh HISTORY
.\" .Sh AUTHORS
.\" .Sh BUGS
```

The first items in the template are the macros (`.Dd`, `.Os`, `.Dt`); the document date, the operating system the man page or subject source is developed or modified for, and the man page title (*in upper case*) along with the section of the manual the page belongs in. These macros identify the page, and are discussed below in **TITLE MACROS**.

The remaining items in the template are section headers (`.Sh`); of which **NAME**, **SYNOPSIS** and **DESCRIPTION** are mandatory. The headers are discussed in **PAGE STRUCTURE DOMAIN**, after presentation of **MANUAL DOMAIN**. Several content macros are used to demonstrate page layout macros; reading about content macros before page layout macros is recommended.

TITLE MACROS

The title macros are the first portion of the page structure domain, but are presented first and separate for someone who wishes to start writing a man page yesterday. Three header macros designate the document title or manual page title, the operating system, and the date of authorship. These macros are one called once at the very beginning of the document and are used to construct the headers and footers only.

`.Dt DOCUMENT_TITLE section# [volume]`

The document title is the subject of the man page and must be in CAPITALS due to troff limitations. The section number may be 1, ..., 8, and if it is specified, the volume title may be omitted. A volume title may be arbitrary or one of the following:

```
AMD   UNIX Ancestral Manual Documents
SMM   UNIX System Manager's Manual
URM   UNIX Reference Manual
PRM   UNIX Programmer's Manual
```

The default volume labeling is `URM` for sections 1, 6, and 7; `SMM` for section 8; `PRM` for sections 2, 3, 4, and 5.

`.Os operating_system release#`

The name of the operating system should be the common acronym, for example, BSD or FreeBSD or ATT. The release should be the standard release nomenclature for the system specified, for example,

4.3, 4.3+Tahoe, V.3, V.4. Unrecognized arguments are displayed as given in the page footer. For instance, a typical footer might be:

```
.Os 4.3BSD
```

or

```
.Os FreeBSD 2.2
```

or for a locally produced set

```
.Os CS Department
```

The Berkeley default, `.Os` without an argument, has been defined as BSD in the site-specific file `/usr/share/tmac/mdoc/doc-common`. It really should default to LOCAL. Note, if the `.Os` macro is not present, the bottom left corner of the page will be ugly.

`.Dd month day, year`

The date should be written formally:

```
January 25, 1989
```

INTRODUCTION OF 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 request 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`(1) macros are themselves a type of command; the general syntax for a troff command is:

.Va argument1 argument2 ... argument9

The `.Va` is a macro command or request, and anything following it is an argument to be processed. In the second case, the description of a UNIX command using the content 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 *arguments*. The macros which formatted the above example:

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

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=value
.Bk -words
.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: `.Ar`, `.Fl`, `.Nm`, and `.Pa` differ only when called without arguments; `.Fn` and `.Xr` impose an order on their argument lists and the `.Op` and `.Fn` macros have nesting limitations. All content macros are capable of recognizing and properly handling punctuation, provided each punctuation character is separated by a leading space. If a request is given:

```
.Li sptr, ptr),
```

The result is:

```
sptr, ptr),
```

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

```
.Li sptr , ptr ) ,
```

The result is:

```
sptr , ptr ) ,
```

The punctuation is now recognized and is output in the default font distinguishing it from the strings in literal font.

To remove the special meaning from a punctuation character escape it with '`\&`'. `Troff` is limited as a macro language, and has difficulty when presented with a string containing a member of the mathematical, logical or quotation set:

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

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 content macro displayed below, `.Ad`.

MANUAL DOMAIN

Address Macro

The address macro identifies an address construct of the form addr1[,addr2[,addr3]].

```
Usage: .Ad address ...
       .Ad addr1
```

addr1

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

It is an error to call `.Ad` without arguments. `.Ad` is callable by other macros and is parsed.

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. Any remaining arguments after the name information are assumed to be punctuation.

```
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 `.An` macro is parsed and is callable. It is an error to call `.An` without any arguments.

Argument Macro

The `.Ar` argument macro may be used whenever a command-line argument is referenced.

```
Usage: .Ar argument ...
```
```
.Ar         file ...
.Ar file1   file1
.Ar file1 . file1.
.Ar file1 file2
            file1 file2
.Ar f1 f2 f3 :
            f1 f2 f3:
.Ar file ) ) ,
            file)),
```

If `.Ar` is called without arguments '*`file ...`*' is assumed. The `.Ar` macro is parsed and is callable.

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. This macro accepts quoted arguments (double quotes only).

`device le0 at scode?` produced by: `.Cd device le0 at scode?`.

Command Modifier

The command modifier is identical to the `.Fl` (flag) command with the exception the `.Cm` macro does not assert a dash in front of every argument. Traditionally flags are marked by the preceding dash, 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**.

Defined Variables

A variable which is defined in an include file is specified by the macro `.Dv`.

```
Usage: .Dv defined_variable ...
       .Dv MAXHOSTNAMELEN
                        MAXHOSTNAMELEN
       .Dv TIOCGPGRP )
                        TIOCGPGRP)
```

It is an error to call `.Dv` without arguments. `.Dv` is parsed and is callable.

Errno's (Section two only)

The `.Er` errno macro specifies the error return value for section two 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 ERRNOTYPE ...
       .Er ENOENT
                 ENOENT
       .Er ENOENT ) ;
                 ENOENT);
       .Bq Er ENOTDIR
                 [ENOTDIR]
```

It is an error to call `.Er` without arguments. The `.Er` macro is parsed and is callable.

Environment Variables

The `.Ev` macro specifies an environment variable.

```
Usage: .Ev argument ...
       .Ev DISPLAY
                 DISPLAY
       .Ev PATH .
                 PATH.
       .Ev PRINTER ) ) ,
                 PRINTER)),
```

It is an error to call `.Ev` without arguments. The `.Ev` macro is parsed and is callable.

Function Argument

The `.Fa` macro is used to refer to function arguments (parameters) outside of the **SYNOPSIS** section of the manual or inside the **SYNOPSIS** section should a parameter list be too long for the `.Fn` macro and the enclosure macros `.Fo` and `.Fc` must be 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
```

It is an error to call `.Fa` without arguments. `.Fa` is parsed and is callable.

Function Declaration

The `.Fd` macro is used in the **SYNOPSIS** section with section two or three functions. The `.Fd` macro does not call other macros and is not callable by other macros.

```
Usage: .Fd include_file (or defined variable)
```

In the **SYNOPSIS** section a `.Fd` request causes a line break if a function has already been presented and a break has not occurred. This leaves a nice vertical space in between the previous function call and the declaration for the next function.

Flags

The `.Fl` macro handles command-line flags. It prepends a dash, '**−**', to the flag. For interactive command flags, which 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.
       .Fl s v t    −s −v −t
       .Fl - ,      −−,
       .Fl xyz ) ,  −xyz),
```

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 `.Fl` macro is parsed and is callable.

Functions (library routines)

The .Fn macro is modeled on ANSI C conventions.

```
Usage: .Fn [type] function [[type] parameters ... ]
.Fn getchar                          getchar()
.Fn strlen ) ,                       strlen()),
.Fn "int align" "const * char *sptrs",
                                     int align(const * char *sptrs),
```

It is an error to call `.Fn` without any arguments. The `.Fn` macro is parsed and is callable, note that any call to another macro signals the end of the `.Fn` call (it will close-parenthesis at that point).

For functions that have more than eight parameters (and this is rare), the macros `.Fo` (function open) and `.Fc` (function close) may be used with `.Fa` (function argument) to get around the limitation. For example:

```
.Fo "int 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*)

The `.Fo` and `.Fc` macros are parsed and are callable. In the **SYNOPSIS** section, the function will always begin at the beginning of line. If there is more than one function presented in the **SYNOPSIS** section and a function type has not been given, a line break will occur, leaving a nice vertical space between the current

function name and the one prior. At the moment, `.Fn` does not check its word boundaries against troff line lengths and may split across a newline ungracefully. This will be fixed in the near future.

Function Type

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 kernel normal form for the **SYNOPSIS** of sections two and three (it causes a line break allowing the function name to appear on the next line).

```
Usage: .Ft type ...
       .Ft struct stat struct stat
```

The `.Ft` request is not callable by other macros.

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
```

It is an error to call `.Ic` without arguments. The `.Ic` macro is parsed and is callable.

Name Macro

The `.Nm` macro is used for the document title or subject name. It has the peculiarity of remembering the first argument it was called with, which should always be the subject name of the page. When called without arguments, `.Nm` regurgitates this initial name for the sole purpose of making less work for the author. Note: a section two or three document function name is addressed with the `.Nm` in the **NAME** section, and with `.Fn` in the **SYNOPSIS** and remaining sections. For interactive commands, such as the `while` command keyword in `csh`(1), the `.Ic` macro should be used. While the `.Ic` is nearly identical to `.Nm`, it can not recall the first argument it was invoked with.

```
Usage: .Nm argument ...
       .Nm mdoc.sample
                   mdoc.sample
       .Nm \-mdoc  -mdoc.
       .Nm foo ) ) ,
                   foo)),
       .Nm         mdoc.samples
```

The `.Nm` macro is parsed and is callable.

Options

The `.Op` macro places option brackets around the any remaining arguments on the command line, and places any trailing punctuation outside the brackets. The macros `.Oc` and `.Oo` may be used across one or more lines.

```
Usage: .Op options ...
.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]
```

The .Oc and .Oo macros:

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

Produce: [[**-k** *kilobytes*] [**-i** *interval*] [**-c** *count*]]

The macros .Op, .Oc and .Oo are parsed and are callable.

Pathnames

The .Pa macro formats pathnames or filenames.

```
Usage: .Pa pathname
       .Pa /usr/share /usr/share
       .Pa /tmp/fooXXXXX ) .
                     /tmp/fooXXXXX).
```

The .Pa macro is parsed and is callable.

Variables

Generic variable reference:

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

It is an error to call .Va without any arguments. The .Va macro is parsed and is callable.

Manual Page Cross References

The .Xr macro expects the first argument to be a manual page name, and the second argument, if it exists, to be either a section page number or punctuation. Any remaining arguments are assumed to be punctuation.

```
Usage: .Xr man_page [1,...,8]
       .Xr mdoc mdoc
       .Xr mdoc ,
                mdoc,
       .Xr mdoc 7
                mdoc(7)
```

```
.Xr mdoc 7 ) ) ,
```
mdoc(7))),

The `.Xr` macro is parsed and is callable. It is an error to call `.Xr` without any arguments.

GENERAL TEXT DOMAIN

AT&T Macro

```
Usage: .At [v6 | v7 | 32v | V.1 | V.4] ...
.At                 AT&T UNIX
.At v6 .            Version 6 AT&T UNIX.
```

The `.At` macro is *not* parsed and *not* callable It accepts at most two arguments.

BSD Macro

```
Usage: .Bx [Version/release] ...
.Bx     BSD
.Bx 4.3 .
        4.3BSD.
```

The `.Bx` macro is parsed and is callable.

FreeBSD Macro

```
Usage: .Fx Version.release ...
.Fx 2.2 .   FreeBSD 2.2.
```

The `.Fx` macro is *not* parsed and *not* callable It accepts at most two arguments.

UNIX Macro

```
Usage: .Ux ...
.Ux       UNIX
```

The `.Ux` macro is parsed and is callable.

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 also a pair of open and close macros which end in small letters ‘`o`’ and ‘`c`’ respectively. These can be used across one or more lines of text and while they have nesting limitations, the one line quote macros can be used inside of them.

Quote	*Close*	*Open*	*Function*	*Result*
.Aq	.Ac	.Ao	Angle Bracket Enclosure	<string>
.Bq	.Bc	.Bo	Bracket Enclosure	[string]
.Dq	.Dc	.Do	Double Quote	‘‘string’’
	.Ec	.Eo	Enclose String (in XX)	XXstringXX
.Pq	.Pc	.Po	Parenthesis Enclosure	(string)
.Ql			Quoted Literal	‘st’ or string
.Qq	.Qc	.Qo	Straight Double Quote	"string"
.Sq	.Sc	.So	Single Quote	‘string’

Except for the irregular macros noted below, all of the quoting macros are parsed and callable. All handle punctuation properly, as long as it is presented one character at a time and separated by spaces. The quoting

macros examine opening and closing punctuation to determine whether it comes before or after the enclosing string This makes some nesting possible.

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

`.Ql` The quoted literal macro behaves differently for `troff` than `nroff`. If formatted with `nroff`, 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.

`.Pf` The prefix macro is not callable, but it is parsed:

```
.Pf ( Fa name2
```
becomes (*name2*.

The `.Ns` (no space) macro performs the analogous suffix function.

Examples of quoting:

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

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 were also built from the same underlying routines and are a good example of **`-mdoc`** macro usage at its worst.

No−Op or Normal Text Macro

The macro `.No` is a hack for words in a macro command line which should *not* be formatted and follows the conventional syntax for content macros.

Space Macro

The `.Ns` macro eliminates unwanted spaces in between macro requests. It is useful for old style argument lists where there is no space between the flag and argument:

```
.Op Fl I Ns Ar directory
```
produces [**-I***directory*]

Note: the `.Ns` macro always invokes the `.No` macro after eliminating the space unless another macro name follows it. The macro `.Ns` is parsed and is callable.

Section Cross References

The `.Sx` macro designates a reference to a section header within the same document. It is parsed and is callable.

```
.Sx FILES     FILES
```

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 style references.

`.Rs`	Reference Start. Causes a line break and begins collection of reference information until the reference end macro is read.
`.Re`	Reference End. The reference is printed.
`.%A`	Reference author name, one name per invocation.
`.%B`	Book title.
`.%C`	City/place.
`.%D`	Date.
`.%J`	Journal name.
`.%N`	Issue number.
`.%O`	Optional information.
`.%P`	Page number.
`.%R`	Report name.
`.%T`	Title of article.
`.%V`	Volume(s).

The macros beginning with '%' are not callable, and are parsed only for the trade name macro which returns to its caller. (And not very predictably at the moment either.) The purpose is to allow trade names to be pretty printed in `troff`/`ditroff` output.

Return Values

The `.Rv` macro generates text for use in the **RETURN VALUE** section.

```
Usage: .Rv [-std function]
```

`.Rv -std atexit` will generate the following text:

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.

Trade Names (or Acronyms and Type Names)

The trade name macro is generally a small caps macro for all upper case words longer than two characters.

```
Usage: .Tn symbol ...
       .Tn DEC
            DEC
       .Tn ASCII
            ASCII
```

The `.Tn` macro is parsed and is callable by other macros.

Extended Arguments

The `.Xo` and `.Xc` macros allow one to extend an argument list on a macro boundary. Argument lists cannot be extended within a macro which expects all of its arguments on one line such as `.Op`.

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

```
.Sm off
.It Xo Sy I Ar operation
.No \en Ar count No \en
```

```
.Xc
.Sm on
```

Produces

I*operation*\n*count*\n

Another one:

```
.Sm off
.It Cm S No / Ar old_pattern Xo
.No / Ar new_pattern
.No / Op Cm g
.Xc
.Sm on
```

Produces

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

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

```
.It Xo
.Ic .ifndef
.Oo \&! Oc Ns Ar variable
.Op Ar operator variable ...
.Xc
```

Produces

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

All of the above examples have used the `.Xo` macro on the argument list of the `.It` (list-item) macro. The extend macros are not used very often, and when they are it is usually to extend the list-item argument list. Unfortunately, this is also where the extend macros are the most finicky. In the first two examples, spacing was turned off; in the third, spacing was desired in part of the output but not all of it. To make these macros work in this situation make sure the `.Xo` and `.Xc` macros are placed as shown in the third example. If the `.Xo` macro is not alone on the `.It` argument list, spacing will be unpredictable. The `.Ns` (no space macro) must not occur as the first or last macro on a line in this situation. Out of 900 manual pages (about 1500 actual pages) currently released with BSD only fifteen use the `.Xo` macro.

PAGE STRUCTURE DOMAIN

Section Headers

The first three `.Sh` section header macros list below are required in every man page. The remaining section headers are recommended at the discretion of the author writing the manual page. The `.Sh` macro can take up to nine arguments. It is parsed and but is not callable.

.Sh NAME The `.Sh NAME` macro is mandatory. If not specified, the 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.

.Sh SYNOPSIS

The **SYNOPSIS** section describes the typical usage of the subject of a man page. The macros required are either `.Nm`, `.Cd`, `.Fn`, (and possibly `.Fo`, `.Fc`, `.Fd`, `.Ft` macros). 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, 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
```

Note: The macros `.Op`, `.Fl`, and `.Ar` recognize the pipe bar character '|', so a command line such as:

```
.Op Fl a | Fl b
```

will not go orbital. `Troff` normally interprets a | as a special operator. See **PREDEFINED STRINGS** for a usable | character in other situations.

.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).

The following `.Sh` section headers 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 EXAMPLES

There are several ways to create examples. See the **EXAMPLES** section below for details.

.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 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. Cross references in the **SEE ALSO** section should be sorted by section number, and then placed in alphabetical order and comma separated. For example:

`ls`(1), `ps`(1), `group`(5), `passwd`(5).

At this time `refer`(1) style references are not accommodated.

.Sh CONFORMING TO

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 C") 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, if need be, should be placed here.

.Sh DIAGNOSTICS
: Diagnostics from a command should be placed in this section.

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

.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
```

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` macro. (The `.Bl` macro asserts a vertical distance unless the -compact flag is given).

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` accepts is **`-words`** and is useful for preventing line breaks in the middle of options. In the example for the make command-line arguments (see **What's in a name**), the keep prevented `nroff` from placing up the flag and the argument on separate lines. (Actually, the option macro used to prevent this from occurring, but was dropped when the decision (religious) was made to force right justified margins in `troff` as options in general look atrocious when spread across a sparse line. More work needs to be done with the keep macros, a **`-line`** option needs to be added.)

Examples and Displays

There are five types of displays, a quickie one line indented display `.D1`, a quickie one line literal display `.Dl`, and a block literal, block filled and block ragged which use the `.Bd` begin-display and `.Ed` end-display macros.

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

```
-ldghfstru
```

The above was produced by: `.Dl` **`-ldghfstru`**.

`.Dl` (D-ell) Display one line of indented *literal* text. The `.Dl` example macro has been used throughout this file. It allows the indent (display) of one line of text. Its default font is set to constant width (literal) however it is parsed and will recognized other macros. It is not callable however.

```
% 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. Displays may be nested within displays and lists. `.Bd` has the following syntax:

```
.Bd display-type [-offset offset_value] [-compact]
```

The display-type must be one of the following four types and may have an offset specifier for indentation: `.Bd`.

−ragged Display a block of text as typed, right (and left) margin edges are left ragged.

−filled Display a filled (formatted) block. The block of text is formatted (the edges are filled − not left unjustified).

−literal Display a literal block, useful for source code or simple tabbed or spaced text.

−file *file_name* The filename following the **−file** flag is read and displayed. Literal mode is asserted and tabs are set at 8 constant width character intervals, however any `troff`/**−mdoc** commands in 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 Indents by one default indent value or tab. The default indent value is also used for the `.D1` display so one is guaranteed the two types of displays will line up. This indent is normally set to 6n or about two thirds of an inch (six constant width characters).

indent-two Indents 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 by `troff`.

.Ed End-display.

Font Modes

There are five macros for changing the appearance of the manual page text:

.Em 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 `.Em` macro is parsed and is callable. It is an error to call `.Em` without arguments.

.Li The `.Li` literal macro may be used for special characters, variable constants, anything which should be displayed as it would be typed.

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

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

The .Li macro is parsed and is callable.

.Sy 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 .Sy macro is parsed and is callable. Arguments to .Sy may be quoted.

.Bf Begin font mode. The .Bf font mode must be ended with the .Ef macro. Font modes may be nested within other font modes. .Bf has the following syntax:

```
.Bf font-mode
```

The font-mode must be one of the following three types: .Bf.

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.

.Ef End font mode.

Tagged 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. Columns may be used inside of lists, but lists are unproven inside of columns.

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**). For a change of pace, the list-type used to present the list-types is an over-hanging list (**-ohang**). This type of list is quite popular with TeX users, but might look a bit funny after having read many pages of tagged lists. The following list types are accepted by .Bl:

-bullet
-item
-enum

These three are the simplest types of lists. Once the .Bl macro has been given, items in the list are merely indicated by a line consisting solely of the .It macro. For example, the source text for a simple enumerated list would look like:

```
.Bl -enum -compact
.It
Item one goes here.
.It
And item two here.
.It
Lastly item three goes here.
.El
```

The results:

1. Item one goes here.
2. And item two here.
3. Lastly item three goes here.

A simple bullet list construction:

```
.Bl -bullet -compact
.It
Bullet one goes here.
.It
Bullet two here.
.El
```

Produces:

- Bullet one goes here.
- Bullet two here.

−tag
−diag
−hang
−ohang
−inset

These list-types collect arguments specified with the `.It` macro and create a label which may be *inset* into the forthcoming text, *hanged* from the forthcoming text, *overhanged* from above and not indented or *tagged*. This list was constructed with the **−ohang** list-type. The `.It` macro is parsed only for the inset, hang and tag list-types and is not callable. 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.

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
```

```
.Nm \-mdoc
manuals to other formats.
.El
```

Here is a hanged list with two items:

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

Longer hanged list labels blend in to 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 in to the paragraph unlike
tagged paragraph labels.
.El
```

The tagged list which follows uses an optional width specifier to control the width of the tag.

SL sleep time of the process (seconds blocked)

PAGEIN number of disk I/O's 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 process priority (non-positive when in non-interruptible wait)

The raw text:

```
.Bl -tag -width "PAGEIN" -compact -offset indent
.It SL
sleep time of the process (seconds blocked)
.It PAGEIN
number of disk
.Tn I/O Ns 's
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 process priority
(non-positive when in non-interruptible wait)
.El
```

Acceptable width specifiers:

-width *Fl* sets the width to the default width for a flag. All callable macros have a default width value. The `.Fl`, value is presently set to ten constant width characters or about five sixth of an inch.

-width *24n*

sets the width to 24 constant width characters or about two inches. The 'n' is absolutely necessary for the scaling to work correctly.

-width *ENAMETOOLONG*
sets width to the constant width length of the string given.

-width *"int mkfifo"*
again, the width is set to the constant width of the string given.

If a width is not specified for the tag list type, the first time `.It` is invoked, an attempt is made to determine an appropriate width. If the first argument to `.It` is a callable macro, the default width for that macro will be used as if the macro name had been supplied as the width. However, if another item in the list is given with a different callable macro name, a new and nested list is assumed.

PREDEFINED STRINGS

The following strings are predefined as may be used by preceding with the troff string interpreting sequence `\*(xx` where *xx* is the name of the defined string or as `\*x` where *x* is the name of the string. The interpreting sequence may be used any where in the text.

String	Nroff	Troff
<=	<=	≤
>=	>=	≥
Rq	”	”
Lq	“	“
ua	^	⇑
aa	’	´
ga	`	`
q	"	"
Pi	pi	π
Ne	!=	≠
Le	<=	≤
Ge	>=	≥
Lt	<	>
Gt	>	<
Pm	+-	±
If	infinity	∞
Na	*NaN*	*NaN*
Ba	|	|

Note: The string named ‘`q`’ should be written as `\*q` since it is only one char.

DIAGNOSTICS

The debugging facilities for **-mdoc** are limited, but can help detect subtle errors such as the collision of an argument name with an internal register or macro name. (A what?) A register is an arithmetic storage class for `troff` with a one or two character name. All registers internal to **-mdoc** for `troff` and `ditroff` are two characters and of the form <upper_case><lower_case> such as ‘`Ar`’, <lower_case><upper_case> as ‘`aR`’ or <upper or lower letter><digit> as ‘`C1`’. And adding to the muddle, `troff` has its own internal registers all of which are either two lower case characters or a dot plus a letter or metacharacter character. In one of the introduction examples, it was shown how to prevent the interpretation of a macro name with the escape sequence ‘`\&`’. This is sufficient for the internal register names also.

If a non-escaped register name is given in the argument list of a request unpredictable behavior will occur. In general, any time huge portions of text do not appear where expected in the output, or small strings such as list tags disappear, chances are there is a misunderstanding about an argument type in the argument list. Your mother never intended for you to remember this evil stuff - so here is a way to find out whether or not your arguments are valid: The `.Db` (debug) macro displays the interpretation of the argument list for most macros. Macros such as the `.Pp` (paragraph) macro do not contain debugging information. All of the

callable macros do, and it is strongly advised whenever in doubt, turn on the .Db macro.

```
Usage: .Db [on | off]
```

An example of a portion of text with the debug macro placed above and below an artificially created problem (a flag argument 'aC' which should be \&aC in order to work):

```
.Db on
.Op Fl aC Ar file )
.Db off
```

The resulting output:

```
DEBUGGING ON
DEBUG(argv) MACRO: '.Op'  Line #: 2
        Argc: 1  Argv: 'Fl'  Length: 2
        Space: ''  Class: Executable
        Argc: 2  Argv: 'aC'  Length: 2
        Space: ''  Class: Executable
        Argc: 3  Argv: 'Ar'  Length: 2
        Space: ''  Class: Executable
        Argc: 4  Argv: 'file'  Length: 4
        Space: ' '  Class: String
        Argc: 5  Argv: ')'  Length: 1
        Space: ' '  Class: Closing Punctuation or suffix
        MACRO REQUEST: .Op Fl aC Ar file )
DEBUGGING OFF
```

The first line of information tells the name of the calling macro, here .Op, and the line number it appears on. If one or more files are involved (especially if text from another file is included) the line number may be bogus. If there is only one file, it should be accurate. The second line gives the argument count, the argument ('Fl') and its length. If the length of an argument is two characters, the argument is tested to see if it is executable (unfortunately, any register which contains a non-zero value appears executable). The third line gives the space allotted for a class, and the class type. The problem here is the argument aC should not be executable. The four types of classes are string, executable, closing punctuation and opening punctuation. The last line shows the entire argument list as it was read. In this next example, the offending 'aC' is escaped:

```
.Db on
.Em An escaped \&aC
.Db off


DEBUGGING ON
DEBUG(fargv) MACRO: '.Em'  Line #: 2
        Argc: 1  Argv: 'An'  Length: 2
        Space: ' '  Class: String
        Argc: 2  Argv: 'escaped'  Length: 7
        Space: ' '  Class: String
        Argc: 3  Argv: 'aC'  Length: 2
        Space: ' '  Class: String
        MACRO REQUEST: .Em An escaped &aC
DEBUGGING OFF
```

The argument \&aC shows up with the same length of 2 as the '\&' sequence produces a zero width, but a register named \&aC was not found and the type classified as string.

Other diagnostics consist of usage statements and are self explanatory.

GROFF, TROFF AND NROFF

The **-mdoc** package does not need compatibility mode with `groff`.

The package inhibits page breaks, and the headers and footers which normally occur at those breaks with `nroff`, to make the manual more efficient for viewing on-line. At the moment, `groff` with **-T***ascii* does eject the imaginary remainder of the page at end of file. The inhibiting of the page breaks makes `nroff`'d files unsuitable for hardcopy. There is a register named '`cR`' which can be set to zero in the site dependent style file `/usr/src/share/tmac/doc-nroff` to restore the old style behavior.

FILES

`/usr/share/tmac/doc.tmac`	manual macro package
`/usr/share/misc/mdoc.template`	template for writing a man page
`/usr/share/examples/mdoc/*`	several example man pages

BUGS

Undesirable hyphenation on the dash of a flag argument is not yet resolved, and causes occasional mishaps in the **DESCRIPTION** section. (line break on the hyphen).

Predefined strings are not declared in documentation.

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

`.Nm` font should be changed in **NAME** section.

`.Fn` needs to have a check to prevent splitting up if the line length is too short. Occasionally it separates the last parenthesis, and sometimes looks ridiculous if a line is in fill mode.

The method used to prevent header and footer page breaks (other than the initial header and footer) when using nroff occasionally places an unsightly partially filled line (blank) at the would be bottom of the page.

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

SEE ALSO

`man`(1), `troff`(1), `groff_mdoc`(7), `mdoc`(7)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.