How to Write a Plan 9 Manual Page

Geoff Collyer
Russ Cox
(following Henry Spencer)

geoff@collyer.net
rsc@swtch.com

This document is an introduction to writing a Plan 9 manual page using the *troff* manual page macros. It does not cover every last detail; judgement and good taste are still necessary for writing readable documentation, especially when striving for clarity in the extremely concise format encouraged by Plan 9. This document is derived originally from a similar document written for Unix by Henry Spencer.

Manual pages are stored as *troff* source files. The manual page is a single file even if the printed form is several pages long. The formatting macros are documented in *man* (6). This document is a more detailed explanation of them, with examples of their usage.

This document first describes the low-level text formatting and then discusses higher-level manual page considerations.

Fonts

Manual pages make fairly heavy use of *italic* and fixed-width fonts. There are standard macros for switching fonts. All of them change the font of a small piece of text; the text is either the arguments to the macro (up to 6 words or strings in quotes) or, if no argument is given, the next text line. None of these macros break the current line, so they may be used anywhere in text.

.I , .B and .L provide italic, fixed-width (formerly bold), and literal (fixed-width with single quotes around it in *nroff*) text. While the .B macro uses fixed-width font, the B font (selected using \fB) is still bold. The fixed-width font is named L (for literal). Explicit font changes using \f are eschewed in favor of the macros when possible.

Situations often arise where it is necessary to have one part of a word in one font and another part in another. For these situations, there are several macros which merge the words from their text input into a single word, alternating from one font to another from word to word. For example, .IR alternates between italic and roman. .BR , .IB , .IR , .RB , and .RI exhaust the remaining combinations of roman, fixed-width, and italic. There are also .L, .LR and .RL macros.

For example,

```
.I Snprint
is like
.IR sprint ,
but will not place more than
.I len
bytes in
.IR s .
```

Snprint is like *sprint* , but will not place more than *len* bytes in *s*.

```
The numeric verbs
.BR d ,
.BR o ,
.BR b ,
.BR x ,
and
.B X
format their arguments in decimal.
```

The numeric verbs d, o, b, x, and X format their arguments in decimal.

```
.I Utflen
returns the number of runes that are represented by the
.SM UTF
string
.IR s .
```

Utflen returns the number of runes that are represented by the UTF string *s*.

The .SM macro emits its arguments in smaller text and is the usual method for typesetting all-capital names such as UTF, ASCII, and NUL.

Typing Conventions

Quotation marks are written using pairs of left and right quotes ( ) ( ) rather than the double quotes ("). There are three reasons for this. First, the double quotes are often used to enclose macro arguments; there is no way to put them inside such arguments. Second, output on devices like laser printers looks much better that way. Third, the paired-quotes convention *is* the correct English usage.

The hyphen, the dash, and the minus sign are three different characters, even though most keyboards only have one key for all three. The hyphen is simply -, the dash \-is, and the minus sign is \(em . In the fixed-width font, there is no distinction between - and \- and thus no need to use the latter.

Paragraphs

Text sections are normally a sequence of paragraphs. Simple paragraphs are separated by .PP , which outputs a small vertical space, checks that enough paper remains on the page for a few more lines, and resets indents and the like. The appearance of paragraphs produced by .PP is similar to that of the paragraphs you are now reading. It is not necessary to use .PP to begin a new paragraph after a subheader (.SH).

A tagged paragraph is one indented an extra amount and preceded by a short *tag* in the space of the indent. Tagged paragraphs are commonly used to discuss a list of items, such as files or command line options. Each tag is an item.

The .TP macro begins a tagged paragraph; the first line after it is the tag, and subsequent lines are the paragraph text. An optional argument to the .TP macro sets the amount of the indent. Setting the indent this way sets the default for future .TP invocations. The .PP macro resets the indent to its default setting.

Explicit indent lengths are rarely used. Instead, the .TF macro sets the .TP indent to the width of its argument plus two spaces in the fixed-width font. This is useful when the paragraph tags are file names, control messages, or other text formatted in fixed-width font.

A sequence of .TP paragraphs must end with a .PD request to restore the usual inter-paragraph spacing. For example:

```
.TF /adm/users
.TP
.B /adm/users
The user names known to the file server
.TP
.B /sys/games/lib/fortunes
Pithy comments
.PD
```

/adm/users   The user names known to the file server

/sys/games/lib/fortunes
    Pithy comments

The .IP macro behaves identically to .TP , except that its first argument is the tag and its second (optional) argument is the indent distance.

Title Heading

A manual page consists of a title heading, several subheadings, and indented text paragraphs.

The title heading is defined by the .TH macro, which takes two arguments:

.TH *name section*

Name and *section* are the name of the manual page and the section number in which it appears. *Name* and *section* appear in the top corners of all manual pages.

```
.TH TROFF 1
.TH QSORT 2
.TH MAN 6
```

The .TH line must be the first in the file.

The *name* argument to the .TH is usually the name of the entity (program, function, library, etc.) being described. Sometimes a manual page describes several entities; in such cases, the one used as the argument to .TH is the first one in the NAME list (to be discussed in a moment). A manual page which is not associated with any particular program, function, etc., is named by what it describes, preferably one short word (e.g., *booting* (8)).

The *section* in the .TH is the manual section number. The Plan 9 manual has eight sections:

Section 1 General publicly accessible commands
Section 2 Library functions, including system calls
Section 3 Kernel devices (accessed via *bind*(1))
Section 4 File services (accessed via *mount*)
Section 5 The Plan 9 file protocol, 9P
Section 6 File formats
Section 7 Databases and database access programs
Section 8 Things related to administering Plan 9

A more detailed explanation of these chapters can be found in the manual itself, specifically the *intro* manual page in each section.

The manual page source is stored in the file /sys/man/*section*/*name*.

Page Sections

The sections in a manual page always appear in the following order:

NAME
SYNOPSIS
DESCRIPTION
EXAMPLE (or EXAMPLES)
FILES
SOURCE
SEE ALSO
DIAGNOSTICS
BUGS

Not every section is needed for every manual page. The Plan 9 manual intentionally omits many sections now common on modern Unix systems, such as AUTHOR, HISTORY, COPYRIGHT, and REPORTING BUGS.

A section heading is given, and a section begun, by a .SH macro with the heading as its argument. The header is printed at the left margin; the section is indented a short distance.

NAME

The NAME section is present in every manual page. It lists the exact names of the things discussed in the entry followed by a short description. If there is more than one name, the first should be the one which best evokes visions of the whole list, since that will also be the name of the manual entry as a whole. (This criterion is admittedly a bit vague.)

The name(s) (separated by commas) are followed by a minus sign (\-) and then the description. Keep the description brief, less than a line. Avoid font changes, special symbols, and cryptic buzzwords. (The NAME section is used by other programs, such as the one which prepares the indices for the manual, and those programs do not parse arbitrary *troff* input.) Examples (troff source, not printed output):

```
awk \- pattern-directed scanning and processing language
bind, mount, unmount \- change name space
calendar \- print upcoming events
cmp \- compare two files
```

SYNOPSIS

Next, on most manual pages, is the SYNOPSIS section. This is absent only in manual entries not discussing identifiable programs, functions, etc., but rather general concepts like booting. The rule is: if there is any conceivable way to type it, the synopsis section should say how.

Command synopses use the following notations:

Fixed-width text is literal, to be typed just as is appears.

Italic text is a placeholder, indicating a place where an argument such as a number or file name is to be typed.

Square brackets [ ] around something mean that it is optional.

A pipe symbol | between two things indicate that only one should be used.

An ellipsis ... means that the previous thing can be repeated.

Because they are set in fixed-width font, command options are typeset with a simple hyphen rather than the minus that would be necessary in variable-width fonts.

By convention, options without arguments are listed first in a single bracketing, followed by the options taking arguments. Both should usually be alphabetized.

For example:

```
.B hget
[
.B -dhv
] [
.B -o
.I ofile
] [
.B -p
.I body
] [
.B -x
.I netmntpt
]
.I url
```

hget [-dhv] [-o *ofile*] [-p *body*] [-x *netmntpt*] *url*

The same sort of conventions apply to SYNOPSIS sections for things other than commands, although such sections tend to use fixed-width text exclusively, since there is seldom much choice about how to call a function. If a manual entry describes more than one program, function, etc., the synopses are separated by a paragraph break (.PP).

Synopses in section 2 begin with the #include lines that must be used to load the given prototypes:

```
.B #include <u.h>
.br
.B #include <libc.h>
.PP
.B
int runetochar(char *s, Rune *r)
.PP
.B
int chartorune(Rune *r, char *s)
```

```
#include <u.h>
#include <libc.h>

int runetochar(char *s, Rune *r)

int chartorune(Rune *r, char *s)
```

DESCRIPTION

The DESCRIPTION section is next and is present in all manual pages. It is typically several paragraphs of narrative text describing the details of what goes on. It is helpful if the first paragraph is a capsule summary of what the program (function, etc.) does and what its inputs and outputs are.

Within narrative text in a manual entry, the basic rules are those of good English: clarity and conciseness. Paragraphs should be short. Tables, lists, etc. should be used whenever they make something clearer. Use the active voice. Omit needless words. For further guidance, see *The Elements of Style* by Strunk & White.

Frequently a narrative has cause to name programs, variables, macros, etc., and to reproduce pieces of the SYNOPSIS section.

Pieces of the SYNOPSIS are reproduced as they occurred, complete with font changes; the same applies to any place where a similar notation is useful in expanding on what is meant by something mentioned in the synopsis.

There is one exception to this: names of programs, functions, files, and variables, even the ones described in the synopsis, are treated like foreign words: they are written in italics. Such names are capitalized when they occur at the beginning of a sentence. The italics rule applies even to name-and-chapter manual references in the text: within a DESCRIPTION section, the proper way to refer to the manual entry for, say, the mail program, is *mail*(1).

References to programs or functions documented on pages with different names should give the page reference as a parenthetical, as in:

```
The type of compression is inferred from the file name extension:
.I bzip2
(see
.IR gzip (1))
for
.BR .tar.bz ,
.BR .tbz ,
.BR .tar.bz2 ,
and
.BR .tbz2 .
```

The exact reference to *gzip*(1), as compared with a reference to the non-existent page *bzip2*(1), creates a valid hyperlink in the HTML version of the manual.

Constants, *troff* macros, file names, and shell environment variables are generally written in fixed-width font.

EXAMPLE

An EXAMPLE section can be helpful when something (especially some common usage) is tricky or non-obvious.

Avoid verbosity: one of the major virtues of the UNIX and Plan 9 manual style is its compactness. If there is more than one example, name the section EXAMPLES.

FILES

The FILES section gives the names of the files which are built into the program. The names are generally given one to a line, with a comment following indicating what the file's significance is. The list is often formatted using tagged paragraphs, discussed above.

SOURCE

The SOURCE section names the source files (or directory) providing each command or function.

```
.SH SOURCE
.B /sys/src/9/port/devcons.c
```

SEE ALSO

The SEE ALSO section gives pointers to related information, usually other manual pages but sometimes external documents as well. Manual page references are formatted in italics, as discussed above. A list of references should be separated by commas.

```
.SH SEE ALSO
.IR ed (1),
.IR sed (1),
.IR grep (1),
.IR rio (1),
.IR regexp (6)
.PP
Rob Pike,
``The text editor sam''.
```

DIAGNOSTICS

The DIAGNOSTICS section explains diagnostics such as the exit status of commands or the return value of functions. If the diagnostics are considered to be sufficiently explained in the description, this section is omitted.

```
.SH DIAGNOSTICS
If
.I echo
draws an error while writing to standard output,
the exit status is
.LR "write error" .
Otherwise the exit status is empty.
```

```
.SH DIAGNOSTICS
.I Abs
and
.I labs
return
the most negative integer or long
when the true result is unrepresentable.
```

BUGS

The BUGS section briefly lists shortcomings or other gotchas that the user should be aware of when using the program. This is the place to mention things which are unsatisfactory or tricky about the program, even if it is not clear that they are bugs. Mentioning something in a BUGS section does not imply a commitment to fix it.

```
.SH BUGS
.I Bundle
will not create directories and is
unsatisfactory for non-text files.
.PP
Beware of gift horses.
```

In general, if in doubt as to how to format something, it is better to look for an existing manual page and imitate it than to invent a new and unique style. Standardization of style is a strong aid to readability.

Formatting the Manual

Once a manual page has been installed, it can be displayed with *man* (1). By default, *man* prints the manual page as text. The -P option instructs *man* to typeset the manual page and display it in *page*(1), the PostScript viewer.

Before the manual page is installed, it can be displayed by running *troff* directly:

```
troff -man file | page
nroff -man file
troff -man file | lp
```

Complete Examples

The appendices show the source and final typeset versions of *src* (1) and *pipe* (2) as reference examples. The entire manual is a good source of further examples.

```
.TH SRC 1
.SH NAME
src   find source code for executable
.SH SYNOPSIS
.B src
[
.B -n
]
[
.B -s
.I symbol
]
.I file
.B ...
.SH DESCRIPTION
.I Src
examines the named
.I files
to find the corresponding source code,
which is then sent to the editor using
.B B
(see
.IR sam (1)).
If
.I file
is an
.IR rc (1)
script, the source is the file itself.
If
.I file
is an executable, the source is defined
to be the single file containing the
definition of
.B main
and
.I src
will point the editor at the line that
begins the definition.
.I Src
uses
.IR db (1)
to extract the symbol table information
that identifies the source.
.PP
.I Src
looks for each
.I file
in the current directory, in
.BR /bin ,
and in the subdirectories of
.BR /bin ,
in that order.
.PP
The
.B -n
flag causes
.B src
to print the file name but not send it
to the editor.
The
.B -s
flag identifies a
.I symbol
other than
.B main
to locate.
.SH EXAMPLES
Find the source to the
.B main
routine in
.BR /bin/ed :
.IP
.EX
src ed
.EE
.PP
Find the source for
.BR strcmp :
.IP
.EX
src -s strcmp rc
.EE
.SH SOURCE
.B /rc/bin/src
.SH SEE ALSO
.IR db (1),
.IR plumb (1),
.IR sam (1).
```

NAME

src – find source code for executable

SYNOPSIS

src [-n] [-s *symbol*] *file* ...

DESCRIPTION

Src examines the named *files* to find the corresponding source code, which is then sent to the editor using B (see *sam* (1)). If *file* is an *rc* (1) script, the source is the file itself. If *file* is an executable, the source is defined to be the single file containing the definition of main and *src* will point the editor at the line that begins the definition. *Src* uses *db* (1) to extract the symbol table information that identifies the source.

Src looks for each *file* in the current directory, in /bin, and in the subdirectories of /bin, in that order.

The -n flag causes src to print the file name but not send it to the editor. The -s flag identifies a *symbol* other than main to locate.

EXAMPLES

Find the source to the main routine in /bin/ed:

```
src ed
```

Find the source for strcmp:

```
src -s strcmp rc
```

SOURCE

/rc/bin/src

SEE ALSO

db (1), *plumb* (1), *sam* (1).

/sys/man/2/pipe

```
.TH PIPE 2
.SH NAME
pipe  create an interprocess channel
.SH SYNOPSIS
.B #include <u.h>
.br
.B #include <libc.h>
.PP
.B
int pipe(int fd[2])
.SH DESCRIPTION
.I Pipe
creates a buffered channel for
interprocess I/O communication.
Two file descriptors are returned in
.IR fd .
Data written to
.B fd[1]
is available for reading from
.B fd[0]
and data written to
.B fd[0]
is available for reading from
.BR fd[1] .
.PP
After the pipe has been established,
cooperating processes
created by subsequent
.IR fork (2)
calls may pass data through the
pipe with
.I read
and
.I write
calls.
The bytes placed on a pipe
by one
.I write
are contiguous even if many processes
are writing.
Write boundaries are preserved:
each read terminates when the read
buffer is full or after reading the
last byte of a write, whichever comes
first.
.PP
The number of bytes available to a
.IR read (2)
is reported
in the
.B Length
field returned by
.I fstat
or
.I dirfstat
on a pipe (see
.IR stat (2)).
.PP
When all the data has been read from a
pipe and the writer has closed the pipe
or exited,
.IR read (2)
will return 0 bytes. Writes to a pipe
with no reader will generate a note
.BR "sys: write on closed pipe" .
.SH SOURCE
.B /sys/src/libc/9syscall
.SH SEE ALSO
.IR intro (2),
.IR read (2),
.IR pipe (3)
.SH DIAGNOSTICS
Sets
.IR errstr .
.SH BUGS
If a read or a write of a pipe is
interrupted, some unknown number
of bytes may have been transferred.
.br
When a read from a pipe returns 0 bytes,
it usually means end of file but is
indistinguishable from reading the result
of an explicit write of zero bytes.
```

NAME

pipe – create an interprocess channel

SYNOPSIS

```
#include <u.h>
#include <libc.h>

int pipe(int fd[2])
```

DESCRIPTION

Pipe creates a buffered channel for interprocess I/O communication. Two file descriptors are returned in *fd*. Data written to fd[1] is available for reading from fd[0] and data written to fd[0] is available for reading from fd[1].

After the pipe has been established, cooperating processes created by subsequent *fork*(2) calls may pass data through the pipe with *read* and *write* calls. The bytes placed on a pipe by one *write* are contiguous even if many processes are writing. Write boundaries are preserved: each read terminates when the read buffer is full or after reading the last byte of a write, whichever comes first.

The number of bytes available to a *read*(2) is reported in the Length field returned by *fstat* or *dirfstat* on a pipe (see *stat*(2)).

When all the data has been read from a pipe and the writer has closed the pipe or exited, *read*(2) will return 0 bytes. Writes to a pipe with no reader will generate a note sys: write on closed pipe.

SOURCE

/sys/src/libc/9syscall

SEE ALSO

intro(2), *read*(2), *pipe*(3)

DIAGNOSTICS

Sets *errstr*.

BUGS

If a read or a write of a pipe is interrupted, some unknown number of bytes may have been transferred.
When a read from a pipe returns 0 bytes, it usually means end of file but is indistinguishable from reading the result of an explicit write of zero bytes.