NAME

groff − a short reference for the GNU roff language

DESCRIPTION

The name *groff* stands for *GNU roff* and is the free implementation of the roff type-setting system. See **roff**(7) for a survey and the background of the groff system.

This document gives only short descriptions of the predefined roff language elements as used in groff. Both the classical features and the groff extensions are provided.

Historically, the *roff language* was called *troff*. *groff* is compatible with the classical system and provides proper extensions. So in GNU, the terms *roff*, *troff*, and *groff language* could be used as synonyms. However *troff* slightly tends to refer more to the classical aspects, whereas *groff* emphasizes the GNU extensions, and *roff* is the general term for the language.

This file is only a short version of the complete documentation that is found in the *groff* **info**(1) file, which contains more detailed, actual, and concise information.

The general syntax for writing groff documents is relatively easy, but writing extensions to the roff language can be a bit harder.

The roff language is line-oriented. There are only two kinds of lines, control lines and text lines. The control lines start with a control character, by default a period "**.**" or a single quote "**'**"; all other lines are text lines.

Control lines represent commands, optionally with arguments. They have the following syntax. The leading control character can be followed by a command name; arguments, if any, are separated by blanks from the command name and among themselves, for example,

```
.command_name arg1 arg2
```

For indentation, any number of space or tab characters can be inserted between the leading control character and the command name, but the control character must be on the first position of the line.

Text lines represent the parts that will be printed. They can be modified by escape sequences, which are recognized by a leading backslash '****'. These are in-line or even in-word formatting elements or functions. Some of these take arguments separated by single quotes "**'**", others are regulated by a length encoding introduced by an open parenthesis '**(**' or enclosed in brackets '**[**' and '**]**'.

The roff language provides flexible instruments for writing language extension, such as macros. When interpreting macro definitions, the roff system enters a special operating mode, called the **copy mode**.

The copy mode behavior can be quite tricky, but there are some rules that ensure a safe usage.

1. Printable backslashes must be denoted as **\e**. To be more precise, **\e** represents the current escape character. To get a backslash glyph, use **\(rs** or **\[rs]**.
2. Double all backslashes.
3. Begin all text lines with the special non-spacing character **\&**.

This does not produce the most efficient code, but it should work as a first measure. For better strategies, see the groff info file and **groff_tmac**(5).

Reading roff source files is easier, just reduce all double backslashes to a single one in all macro definitions.

GROFF ELEMENTS

The roff language elements add formatting information to a text file. The fundamental elements are predefined commands and variables that make roff a full-blown programming language.

There are two kinds of roff commands, possibly with arguments. **Requests** are written on a line of their own starting with a dot '**.**' or a "**'**", whereas **Escape sequences** are in-line functions and in-word formatting elements starting with a backslash '****'.

The user can define her own formatting commands using the **de** request. These commands are called **macros**, but they are used exactly like requests. Macro packages are pre-defined sets of macros written in

the groff language. A user's possibilities to create escape sequences herself is very limited, only special characters can be mapped.

The groff language provides several kinds of variables with different interfaces. There are pre-defined variables, but the user can define her own variables as well.

String variables store character sequences. They are set with the **ds** request and retrieved by the ***** escape sequences. Strings can have variables.

Register variables can store numerical values, numbers with a scale unit, and occasionally string-like objects. They are set with the **nr** request and retrieved by the **\n** escape sequences.

Environments allow the user to temporarily store global formatting parameters like line length, font size, etc. for later reuse. This is done by the **ev** request.

Fonts are identified either by a name or by an internal number. The current font is chosen by the **ft** request or by the **\f** escape sequences. Each device has special fonts, but the following fonts are available for all devices. **R** is the standard font Roman. **B** is its **bold** counterpart. The *italic* font is called **I** and is available everywhere, but on text devices it is displayed as an underlined Roman font. For the graphical output devices, there exist constant-width pendants of these fonts, **CR**, **CI**, and **CB**. On text devices, all characters have a constant width anyway.

Moreover, there are some advanced roff elements. A **diversion** stores information into a macro for later usage. A **trap** is a positional condition like a certain number of lines from page top or in a diversion or in the input. Some action can be prescribed to be run automatically when the condition is met.

More detailed information and examples can be found in the groff info file.

CONTROL CHARACTERS

There is a small set of characters that have a special controlling task in certain conditions.

. A dot is only special at the beginning of a line or after the condition in the requests **if**, **ie**, **el**, and **while**. There it is the control character that introduces a request (or macro). The special behavior can be delayed by using the **\.** escape. By using the **cc** request, the control character can be set to a different character, making the dot '**.**' a non-special character.

In all other positions, it just means a dot character. In text paragraphs, it is advantageous to start each sentence at a line of its own.

' The single quote has two controlling tasks. At the beginning of a line and in the conditional requests it is the non-breaking control character. That means that it introduces a request like the dot, but with the additional property that this request doesn't cause a linebreak. By using the **c2** request, the non-break control character can be set to a different character.

As a second task, it is the most commonly used argument separator in some functional escape sequences (but any pair of characters not part of the argument will work). In all other positions, it denotes the single quote or apostrophe character. Groff provides a printable representation with the **\(cq** escape sequence.

" The double quote is used to enclose arguments in requests, macros, and strings. In the **ds** and **as** requests, a leading double quote in the argument will be stripped off, making everything else afterwards the string to be defined (enabling leading whitespace). The escaped double quote **\"** introduces a comment. Otherwise, it is not special. Groff provides a printable representation with the **\(dq** escape sequence.

**** The backslash usually introduces an escape sequence (this can be changed with the **ec** request). A printed version of the escape character is the **\e** escape; a backslash glyph can be obtained by **\(rs**.

(The open parenthesis is only special in escape sequences when introducing an escape name or argument consisting of exactly two characters. In groff, this behavior can be replaced by the **[]** construct.

[The opening bracket is only special in groff escape sequences; there it is used to introduce a long escape name or long escape argument. Otherwise, it is non-special, e.g. in macro calls.

] The closing bracket is only special in groff escape sequences; there it terminates a long escape name or long escape argument. Otherwise, it is non-special.

space Space characters are only functional characters. They separate the arguments in requests, macros, and strings, and the words in text lines. They are subject to groff's horizontal spacing calculations. To get a defined space width, escape sequences like '**\ **' (this is the escape character followed by a space), \|, \^, or **h** should be used.

newline
In text paragraphs, newlines mostly behave like space characters. Continuation lines can be specified by an escaped newline, i.e., by specifying a backslash '****' as the last character of a line.

tab If a tab character occurs during text the interpreter makes a horizontal jump to the next pre-defined tab position. There is a sophisticated interface for handling tab positions.

NUMERICAL EXPRESSIONS

A **numerical value** is a signed or unsigned integer or float with or without an appended scaling indicator. A **scaling indicator** is a one-character abbreviation for a unit of measurement. A number followed by a scaling indicator signifies a size value. By default, numerical values do not have a scaling indicator, i.e., they are normal numbers.

The *roff* language defines the following scaling indicators.

c	Centimeter
i	Inch
P	Pica = 1/6 inch
p	Point = 1/72 inch
m	Em = the font size in points (width of letter 'm')
M	100th of an Em
n	En = Em/2
u	Basic unit for actual output device
v	Vertical line space in basic units scaled point = 1/*sizescale* of a point (defined in font *DESC* file)
f	Scale by 65536.

Numerical expressions are combinations of the numerical values defined above with the following arithmetical operators already defined in classical troff.

+	Addition
−	Subtraction
*****	Multiplication
/	Division
%	Modulo
=	Equals
==	Equals
<	Less than
>	Greater than
<=	Less or equal
>=	Greater or equal
&	Logical and
:	Logical or
!	Logical not
(	Grouping of expressions
)	Close current grouping

Moreover, *groff* added the following operators for numerical expressions:

*e1***>?***e2*	The maximum of *e1* and *e2*.
*e1***<?***e2*	The minimum of *e1* and *e2*.

(*c***;***e***)** Evaluate *e* using *c* as the default scaling indicator.

For details see the groff info file.

CONDITIONS

Conditions occur in tests raised by the **if**, **ie**, and the **while** requests. The following table characterizes the different types of conditions.

N	A numerical expression *N* yields true if its value is greater than 0.
!*N*	True if the value of *I* is 0.
'*s1***'***s2***'**	True if string *s1* is identical to string *s2*.
!'*s1***'***s2***'**	True if string *s1* is not identical to string *s2*.
c*ch*	True if there is a character *ch* available.
d*name*	True if there is a string, macro, diversion, or request called *name*.
e	Current page number is even.
o	Current page number is odd.
m*name*	True if there is a color called *name*.
n	Formatter is **nroff**.
r*reg*	True if there is a register named *reg*.
t	Formatter is **troff**.

REQUESTS

This section provides a short reference for the predefined requests. In groff, request and macro names can be arbitrarily long. No bracketing or marking of long names is needed.

Most requests take one or more arguments. The arguments are separated by space characters (no tabs!); there is no inherent limit for their length or number. An argument can be enclosed by a pair of double quotes. This is very handy if an argument contains space characters, e.g., "*arg with space*" denotes a single argument.

Some requests have optional arguments with a different behaviour. Not all of these details are outlined here. Refer to the groff info file and **groff_diff**(7) for all details.

In the following request specifications, most argument names were chosen to be descriptive. Only the following denotations need clarification.

c	denotes a single character.
font	a font either specified as a font name or a font number.
anything	all characters up to the end of the line or within **\{** and **\}**.
n	is a numerical expression that evaluates to an integer value.
N	is an arbitrary numerical expression, signed or unsigned.
±*N*	has three meanings depending on its sign, described below.

If an expression defined as ±*N* starts with a '**+**' sign the resulting value of the expression will be added to an already existing value inherent to the related request, e.g. adding to a number register. If the expression starts with a '**-**' the value of the expression will be subtracted from the request value.

Without a sign, *N* replaces the existing value directly. To assign a negative number either prepend 0 or enclose the negative number in parentheses.

Request Short Reference

. Empty line, ignored. Useful for structuring documents.

.\" *anything*
Complete line is a comment.

.ab *string*
Print *string* on standard error, exit program.

.ad Begin line adjustment for output lines in current adjust mode.

.ad *c* Start line adjustment in mode *c* (*c* = `l,r,b,n`).

.af *register c*
Assign format *c* to *register* (*c* = `l,i,I,a,A`).

.aln *alias register*
Create alias name for *register*.
.als *alias object*
Create alias name for request, string, macro, or diversion *object*.
.am *macro*
Append to *macro* until **..** is encountered.
.am *macro end*
Append to *macro* until **.end** is called.
.ami *macro*
Append to a macro whose name is contained in the string register *macro* until **..** is encountered.
.ami *macro end*
Append to a macro indirectly. *macro* and *end* are string registers whose contents are interpolated for the macro name and the end macro, respectively.
.am1 *macro*
Same as **.am** but with compatibility mode switched off during macro expansion.
.am1 *macro end*
Same as **.am** but with compatibility mode switched off during macro expansion.
.as *stringvar anything*
Append *anything* to *stringvar*.
.asciify *diversion*
Unformat ASCII characters, spaces, and some escape sequences in *diversion*.
.as1 *stringvar anything*
Same as **.as** but with compatibility mode switched off during string expansion.
.backtrace
Print a backtrace of the input on stderr.
.bd *font N*
Embolden *font* by *N*-1 units.
.bd *S font N*
Embolden Special Font *S* when current font is *font*.
.blm Unset the blank line macro.
.blm *macro*
Set the blank line macro to *macro*.
.box End current diversion.
.box *macro*
Divert to *macro*, omitting a partially filled line.
.boxa End current diversion.
.boxa *macro*
Divert and append to *macro*, omitting a partially filled line.
.bp Eject current page and begin new page.
.bp ±*N* Eject current page; next page number ±*N*.
.br Line break.
.brp Break and spread output line. Same as **\p**.
.break Break out of a while loop.
.c2 Reset no-break control character to "**'**".
.c2 *c* Set no-break control character to *c*.
.cc Reset control character to '**.**'.
.cc *c* Set control character to *c*.
.ce Center the next input line.
.ce *N* Center following *N* input lines.
.cf *filename*
Copy contents of file *filename* unprocessed to stdout or to the diversion.
.cflags *mode c1 c2 . . .*
Treat characters *c1*, *c2*, . . . according to *mode* number.

.ch *trap N*
Change *trap* location to *N* .

.char *c anything*
Define character *c* as string *anything*.

.chop *object*
Chop the last character off macro, string, or diversion *object*.

.close *stream*
Close the *stream*.

.color Enable colors.

.color *N*
If *N* is zero disable colors, otherwise enable them.

.continue
Finish the current iteration of a while loop.

.cp Enable compatibility mode.

.cp *N* If *N* is zero disable compatibility mode, otherwise enable it.

.cs *font N M*
Set constant character width mode for *font* to *N*/36 ems with em *M*.

.cu *N* Continuous underline in nroff, like **.ul** in troff.

.da End current diversion.

.da *macro*
Divert and append to *macro*.

.de *macro*
Define or redefine *macro* until **..** is encountered.

.de *macro end*
Define or redefine *macro* until **.end** is called.

.de1 *macro*
Same as **.de** but with compatibility mode switched off during macro expansion.

.de1 *macro end*
Same as **.de** but with compatibility mode switched off during macro expansion.

.defcolor *color scheme component*
Define or redefine a color with name *color*. *scheme* can be **rgb**, **cym**, **cymk**, **gray**, or **grey**. *component* can be single components specified as fractions in the range 0 to 1 (default scaling indicator **f**), as a string of two-digit hexadecimal color components with a leading #, or as a string of four-digit hexadecimal components with two leading #. The color **default** can't be redefined.

.dei *macro*
Define or redefine a macro whose name is contained in the string register *macro* until **..** is encountered.

.dei *macro end*
Define or redefine a macro indirectly. *macro* and *end* are string registers whose contents are interpolated for the macro name and the end macro, respectively.

.di End current diversion.

.di *macro*
Divert to *macro* .

.do *name*
Interpret **.name** with compatibility mode disabled.

.ds *stringvar anything*
Set *stringvar* to *anything*.

.ds1 *stringvar anything*
Same as **.ds** but with compatibility mode switched off during string expansion.

.dt *N trap*
Set diversion trap to position *N* (default scaling indicator **v**).

.ec Reset escape character to '****'.

.ec *c* Set escape character to *c*.
.ecr Restore escape character saved with **.ecs**.
.ecs Save current escape character.
.el *anything*
Else part for if-else (**ie**) request.
.em *macro*
The *macro* will be run after the end of input.
.eo Turn off escape character mechanism.
.ev Switch to previous environment.
.ev *env* Push down environment number or name *env* and switch to it.
.evc *env* Copy the contents of environment *env* to the current environment. No pushing or popping.
.ex Exit from roff processing.
.fam Return to previous font family.
.fam *name*
Set the current font family to *name*.
.fc Disable field mechanism.
.fc *a* Set field delimiter to *a* and pad character to space.
.fc *a b* Set field delimiter to *a* and pad character to *b*.
.fchar *c anything*
Define fallback character *c* as string *anything*.
.fi Fill output lines.
.fl Flush output buffer.
.fp *n font*
Mount *font* on position *n*.
.fp *n internal external*
Mount font with long *external* name to short *internal* name on position *n*.
.fspecial *font s1 s2 . . .*
When the current font is *font*, then the fonts *s1*, *s2*, . . . will be special.
.ft Return to previous font. Same as **\f[]** or **\fP**.
.ft *font* Change to font name or number *font*; same as **\f[***font***]** escape sequence.
.ftr *font1 font2*
Translate *font1* to *font2*.
.hc Remove additional hyphenation indicator character.
.hc *c* Set up additional hyphenation indicator character *c*.
.hcode *c1 code1 c2 code2 . . .*
Set the hyphenation code of character *c1* to *code1*, that of *c2* to *code2*, etc.
.hla *lang*
Set the current hyphenation language to *lang*.
.hlm *n* Set the maximum number of consecutive hyphenated lines to *n*.
.hpf *file* Read hyphenation patterns from *file*.
.hpfa *file*
Append hyphenation patterns from *file*.
.hpfcode *file*
Set input mapping for **.hpf**.
.hw *words*
List of *words* with exceptional hyphenation.
.hy *N* Switch to hyphenation mode *N*.
.hym *n* Set the hyphenation margin to *n* (default scaling indicator **m**).
.hys *n* Set the hyphenation space to *n*.
.ie *cond anything*
If *cond* then *anything* else goto **.el**.
.if *cond anything*
If *cond* then *anything*; otherwise do nothing.

.ig Ignore text until **..** is encountered.
.ig *end* Ignore text until **.end**.
.in Change to previous indent value.
.in ±*N* Change indent according to ±*N* (default scaling indicator **m**).
.it *N trap*
Set an input-line count trap for the next *N* lines.
.itc *N trap*
Same as **.it** but count lines interrupted with **\c** as one line.
.kern Enable pairwise kerning.
.kern *n* If *n* is zero, disable pairwise kerning, otherwise enable it.
.lc Remove leader repetition character.
.lc *c* Set leader repetition character to *c*.
.length *register anything*
Write the length of the string *anything* in *register*.
.linetabs
Enable line-tabs mode (i.e., calculate tab positions relative to output line).
.linetabs *n*
If *n* is zero, disable line-tabs mode, otherwise enable it.
.lf *N file*
Set input line number to *N* and filename to *file*.
.lg *N* Ligature mode on if *N*>0.
.ll Change to previous line length.
.ll ±*N* Set line length according to ±*N* (default size 6.5**i**, default scaling indicator **m**).
.ls Change to the previous value of additional intra-line skip.
.ls *N* Set additional intra-line skip value to *N*, i.e., *N*-1 blank lines are inserted after each text output line.
.lt ±*N* Length of title (default scaling indicator **m**).
.mc Margin character off.
.mc *c* Print character *c* after each text line at actual distance from right margin.
.mc *c N* Set margin character to *c* and distance to *N* from right margin (default scaling indicator **m**).
.mk *register*
Mark current vertical position in *register*.
.mso *file* The same as the .so request except that *file* is searched in the tmac directories.
.na No output-line adjusting.
.ne Need a one-line vertical space.
.ne *N* Need *N* vertical space (default scaling indicator **v**).
.nf No filling or adjusting of output-lines.
.nh No hyphenation.
.nm Number mode off.
.nm ±*N* [*M* [*S* [*I*]]]
In line number mode, set number, multiple, spacing, and indent.
.nn Do not number next line.
.nn *N* Do not number next *N* lines.
.nop *anything*
Always execute *anything*.
.nr *register* ±*N M*
Define or modify *register* using ±*N* with auto-increment *M*.
.nroff Make the built-in condition **n** true and **t** false.
.ns Turn no-space mode on.
.nx Immediately jump to end of current file.
.nx *filename*
Next file.

.open *stream filename*
: Open register **filename** for writing and associate the stream named register **stream** with it.

.opena *stream filename*
: Like **.open** but append to it.

.os
: Output vertical distance that was saved by the **sv** request.

.output *string*
: Emit *string* directly to intermediate output, allowing leading whitespace if *string* starts with **"** (which will be stripped off).

.pc
: Reset page number character to '**%**'.

.pc *c*
: Page number character.

.pi *program*
: Pipe output to *program* (nroff only).

.pl
: Set page length to default 11**i**. The current page length is stored in register **.p**.

.pl ±*N*
: Change page length to ±*N* (default scaling indicator **v**).

.pm
: Print macro names and sizes (number of blocks of 128 bytes).

.pm t
: Print only total of sizes of macros (number of 128 bytes blocks).

.pn ±*N*
: Next page number *N*.

.pnr
: Print the names and contents of all currently defined number registers on stderr.

.po
: Change to previous page offset. The current page offset is available in register **.o**.

.po ±*N*
: Page offset *N*.

.ps
: Return to previous point-size.

.ps ±*N*
: Point size; same as **\s[**±*N***]**.

.psbb *filename*
: Get the bounding box of a PostScript image *filename*.

.pso *command*
: This behaves like the **so** request except that input comes from the standard output of *command*.

.ptr
: Print the names and positions of all traps (not including input line traps and diversion traps) on stderr.

.pvs
: Change to previous post-vertical line spacing.

.pvs ±*N*
: Change post-vertical line spacing according to ±*N* (default scaling indicator **p**).

.rchar *c1 c2* . . .
: Remove the definitions of characters *c1*, *c2*, . . .

.rd *prompt*
: Read insertion.

.return
: Return from a macro.

.rj *n*
: Right justify the next *n* input lines.

.rm *name*
: Remove request, macro, or string *name*.

.rn *old new*
: Rename request, macro, or string *old* to *new*.

.rnn *reg1 reg2*
: Rename register *reg1* to *reg2*.

.rr *register*
: Remove *register*.

.rs
: Restore spacing; turn no-space mode off.

.rt ±*N*
: Return *(upward only)* to marked vertical place (default scaling indicator **v**).

.shc
: Reset soft hyphen character to **\(hy**.

.shc *c*
: Set the soft hyphen character to *c*.

.shift *n*
: In a macro, shift the arguments by *n* positions.

.sizes *s1 s2* . . . *sn* **[0]**
: Set available font sizes similar to the **sizes** command in a **DESC** file.

.so *filename*
: Include source file.

.sp
: Skip one line vertically.

.sp *N*
: Space vertical distance *N* up or down according to sign of *N* (default scaling indicator **v**).

.special *s1 s2 ...*
: Fonts *s1*, *s2*, etc. are special and will be searched for characters not in the current font.

.spreadwarn
: Toggle the spread warning on and off without changing its value.

.spreadwarn *limit*
: Emit a warning if each space in an output line is widened by *limit* or more (default scaling indicator **m**).

.ss *N*
: Space-character size set to *N*/12 of the spacewidth in the current font.

.ss *N M*
: Space-character size set to *N*/12 and sentence space size set to *M*/12 of the spacewidth in the current font (=`1/3 em`).

.sty *n style*
: Associate *style* with font position *n*.

.substring *xx n1 n2*
: Replace the string named *xx* with the substring defined by the indices *n1* and *n2*.

.sv
: Save 1**v** of vertical space.

.sv *N*
: Save the vertical distance *N* for later output with **os** request.

.sy *command-line*
: Execute program *command-line*.

.ta T *N*
: Set tabs after every position that is a multiple of *N* (default scaling indicator **m**).

.ta *n1 n2 ... nn* **T** *r1 r2 ... rn*
: Set tabs at positions *n1*, *n2*, ..., *nn*, then set tabs at *nn+r1*, *nn+r2*, ..., *nn+rn*, then at *nn+rn+r1*, *nn+rn+r2*, ..., *nn+rn+rn*, and so on.

.tc
: Remove tab repition character.

.tc *c*
: Set tab repetition character to *c*.

.ti ±*N*
: Temporary indent next line (default scaling indicator **m**).

.tkf *font s1 n1 s2 n2*
: Enable track kerning for *font*.

.tl *'left'center'right'*
: Three-part title.

.tm *anything*
: Print *anything* on terminal (UNIX standard message output).

.tm1 *anything*
: Print *anything* on terminal (UNIX standard message output), allowing leading whitespace if *anything* starts with **"** (which will be stripped off).

.tmc *anything*
: Similar to **.tm1** without emitting a final newline.

.tr *abcd...*
: Translate *a* to *b*, *c* to *d*, etc. on output.

.trf *filename*
: Transparently output the contents of file *filename*.

.trin *abcd...*
: This is the same as the **tr** request except that the **asciify** request will use the character code (if any) before the character translation.

.trnt *abcd...*
: This is the same as the **tr** request except that the translations do not apply to text that is transparently throughput into a diversion with **\!**.

.troff
: Make the built-in condition **t** true and **n** false.

.uf *font*
: Underline font set to *font* (to be switched to by **.ul**).

.ul *N*
: Underline (italicize in troff) *N* input lines.

.unformat *diversion*
Unformat space characters and tabs, preserving font information in *diversion*.

.vpt *n* Enable vertical position traps if *n* is non-zero, disable them otherwise.

.vs Change to previous vertical base line spacing.

.vs ±*N* Set vertical base line spacing according to ±*N* (default scaling indicator **p**). Default value is 12**p**.

.warn *n* Set warnings code to *n*.

.warnscale *si*
Set scaling indicator used in warnings to *si*.

.wh *N* Remove (first) trap at position *N*.

.wh *N trap*
Set location trap; negative means from page bottom.

.while *cond anything*
While condition *cond* is true, accept *anything* as input.

.write *stream anything*
Write *anything* to the stream named *stream*.

.writec *stream anything*
Similar to **.write** without emitting a final newline.

.writem *stream xx*
Write contents of macro or string *xx* to the stream named *stream*.

Besides these standard groff requests, there might be further macro calls. They can originate from a macro package (see **roff**(7) for an overview) or from a preprocessor.

Preprocessor macros are easy to be recognized. They enclose their code into a pair of characteristic macros.

preprocessor	start macro	end macro
eqn	**.PS**	**.PE**
grap	**.G1**	**.G2**
grn	**.GS**	**.GE**
pic	**.PS**	**.PE**
refer	**.R1**	**.R2**
soelim	*none*	*none*
tbl	**.TS**	**.TE**

ESCAPE SEQUENCES

Escape sequences are in-line language elements usually introduced by a backslash '****' and followed by an escape name and sometimes by a required argument. Input processing is continued directly after the escaped character or the argument resp. without an intervening separation character. So there must be a way to determine the end of the escape name and the end of the argument.

This is done by enclosing names (escape name and arguments consisting of a variable name) by a pair of brackets **[***name***]** and constant arguments (number expressions and characters) by apostrophes (ASCII 0x27) like **'***constant***'**.

There are abbreviations for short names. Two character escape names can be specified by an opening parenthesis like **\(**xy without a closing counterpart. And all one-character names different from the special characters '**[**' and '**(**' can even be specified without a marker in the form **\c**.

Constant arguments of length 1 can omit the marker apostrophes, too, but there is no two-character analogue.

While 1-character escape sequences are mainly used for in-line functions and system related tasks, the 2-letter names following the **\(** construct are used for special characters predefined by the roff system. Escapes sequences with names of more than two characters **\[***name***]** denote user defined named characters (see the **char** request).

Single Character Escapes

\" Beginning of a comment. Everything up to the end of the line is ignored.

\# Everything up to and including the next newline is ignored. This is interpreted in copy mode. This is like \" except that the terminating newline is ignored as well.

***** *s* The string stored in the string variable with 1-character name *s*.

***(** *st* The string stored in the string variable with 2-character name *st*.

***[** *stringvar arg1 arg2 . . .* **]**
The string stored in the string variable with arbitrary length name *stringvar*, taking *arg1*, *arg2*, . . . as arguments.

\$0 The name by which the current macro was invoked. The **als** request can make a macro have more than one name.

\$ *x* Macro or string argument with 1-place number *x*, where *x* is a digit between 1 and 9.

\$(*xy* Macro or string argument with 2-digit number *xy*.

\$[*nexp* **]**
Macro or string argument with number *nexp*, where *nexp* is a numerical expression evaluating to an integer ≥1.

\$* In a macro or string, the concatenation of all the arguments separated by spaces.

\$@ In a macro or string, the concatenation of all the arguments with each surrounded by double quotes, and separated by spaces.

**** reduces to a single backslash; useful to delay its interpretation as escape character in copy mode. For a printable backslash, use **\e**, or even better **\[rs]**, to be independent from the current escape character.

\' The acute accent ´; same as **\(aa**. Unescaped: apostrophe, right quotation mark, single quote (ASCII 0x27).

\`** The grave accent `; same as **\(ga. Unescaped: left quote, backquote (ASCII 0x60).

\- The − sign in the current font.

\. An uninterpreted dot (period), even at start of line.

\% Default optional hyphenation character.

\! Transparent line indicator.

\? *anything* **?**
In a diversion, this will transparently embed *anything* in the diversion. *anything* is read in copy mode. See also the escape sequences **\!** and **\?**.

**** *space* Unpaddable space-size space character (no line break).

\0 Digit width.

\| 1/6 em narrow space character; zero width in nroff.

\^ 1/12 em half-narrow space character; zero width in nroff.

\& Non-printable, zero width character.

\) Like **\&** except that it behaves like a character declared with the cflags request to be transparent for the purposes of end of sentence recognition.

\/ Increases the width of the preceding character so that the spacing between that character and the following character will be correct if the following character is a roman character.

\, Modifies the spacing of the following character so that the spacing between that character and the preceding character will correct if the preceding character is a roman character.

\~ Unbreakable space that stretches like a normal inter-word space when a line is adjusted.

\: Inserts a zero-width break point (similar to **\%** but without a soft hyphen character).

**** *newline*
Ignored newline, for continuation lines.

\{ Begin conditional input.

\} End conditional input.

\(*sc* The special character with 2-character name *sc*, see section **Special Characters**.

\[*name* **]**
The named character with arbitrary length name *name*.

\a Non-interpreted leader character.

\A'*anything***'**
If *anything* is acceptable as a name of a string, macro, diversion, register, environment or font it expands to 1, and to 0 otherwise.

\b'*abc...***'**
Bracket building function.

\B'*anything***'**
If *anything* is acceptable as a valid numeric expression it expands to 1, and to 0 otherwise.

\c Interrupt text processing.

\C'*char***'**
The character called *char*; same as **\[***char***]**, but compatible to other roff versions.

\d Forward (down) 1/2 em vertical unit (1/2 line in nroff).

\D'*charseq***'**
Draw a graphical element defined by the characters in *charseq*; see groff info file for details.

\e Printable version of the current escape character.

\E Equivalent to an escape character, but is not interpreted in copy-mode.

\f*F* Change to font with 1-character name or 1-digit number *F*.

\fP Switch back to previous font.

\f(*fo* Change to font with 2-character name or 2-digit number *fo*.

\f[*font***]**
Change to font with arbitrary length name or number expression *font*.

\f[] Switch back to previous font.

\F *f* Change to font family with 1-character name *f*.

\F(*fm* Change to font family with 2-character name *fm*.

\F[*fam***]**
Change to font family with arbitrary length name *fam*.

\F[] Switch back to previous font family.

\g[*reg***]**
Return format of register with name *reg* suitable for **.af**. Alternative forms **\g(***xy* and **\g***x*.

\h'*N***'** Local horizontal motion; move right *N* (left if negative).

\H'*N***'** Set height of current font to *N*.

\k[*reg***]**
Mark horizontal input place in register with arbitrary length name *reg*. Alternative forms **\k(***xy* and **\k***x*.

\l'*Nc***'**
Horizontal line drawing function (optionally using character *c*).

\L'*Nc***'**
Vertical line drawing function (optionally using character *c*).

\m[*color***]**
Change to color *color*. Alternative forms **\m(***co* and **\m***c*.

\m[] Switch back to previous color.

\M[*color***]**
Change filling color for closed drawn objects to color *color*. Alternative forms **\M(***co* and **\M***c*.

\M[] Switch to previous fill color.

\n*r* The numerical value stored in the register variable with the 1-character name *r*.

\n(*re* The numerical value stored in the register variable with the 2-character name *re*.

\n[*reg***]**
The numerical value stored in the register variable with arbitrary length name *reg*.

\N'*n***'** Typeset the character with code *n* in the current font, no special fonts are searched. Useful for adding characters to a font using the **char** request.

\o'*abc...***'**
Overstrike characters *a*, *b*, *c*, etc.

\O*0* Disable glyph output. Mainly for internal use.

\O*1* Enable glyph output. Mainly for internal use.

\p Break and spread output line.
\r Reverse 1 em vertical motion (reverse line in nroff).
\R'*name* ±*n*'
The same as **.nr** *name* ±*n*.
\s[±*N***]**
Set the point size to *N* scaled points. Note the alternative forms **\s±[***N***]**, **\s'**±*N*', **\s±'***N*', **\s(**±*xy*, **\s±(***xy*, **\s±**(*x*. Same as **ps** request.
\S'*N*' Slant output *N* degrees.
\t Non-interpreted horizontal tab.
\u Reverse (up) 1/2 em vertical motion (1/2 line in nroff).
\v'*N*' Local vertical motion; move down *N* (up if negative).
\V[*env***]**
The contents of the environment variable *env*. Alternative forms **\V(***xy* and **\V***x*.
\w'*string*'
The width of the character sequence *string*.
\x'*N*' Extra line-space function (negative before, positive after).
\X'*string*'
Output *string* as device control function.
\Y[*name***]**
Output string variable or macro *name* uninterpreted as device control function. Alternative forms **\Y(***xy* and **\Y***x*.
\z*c* Print *c* with zero width (without spacing).
\Z'*anything*'
Print *anything* and then restore the horizontal and vertical position; *anything* may not contain tabs or leaders.

The escape sequences **\e**, **\.**, **\"**, **\$**, *****, **\a**, **\n**, **\t**, **\g**, and *****newline* are interpreted in copy mode.

Escape sequences starting with **\(** or **\[** do not represent single character escape sequences, but introduce escape names with two or more characters.

If a backslash is followed by a character that does not constitute a defined escape sequence the backslash is silently ignored and the character maps to itself.

Special Characters

Common special characters are predefined by escape sequences of the form **\(***xy* with characters *x* and *y*. Some of these exist in the usual font while most of them are only available in the special font. Below you'll find a selection of the most important glyphs; a complete list can be found in **groff_char**(7).

\(bu Bullet sign
\(co Copyright
\(ct Cent
\(dd Double dagger
\(de Degree
\(dg Dagger
\(rs Printable double quote
\(em Em-dash
\(hy Hyphen
\(rg Registered sign
\(rs Printable backslash character
\(sc Section sign
\(ul Underline character
\(== Identical
\(>= Larger or equal
\(<= Less or equal
\(!= Not equal

\(-> Right arrow
\(<- Left arrow
\(+- Plus-minus sign

Strings

Strings are defined by the **ds** request and can be retrieved by the ***** escape sequence.

Strings share their name space with macros. So strings and macros without arguments are roughly equivalent; it is possible to call a string like a macro and vice-versa, but this often leads to unpredictable results. The following strings are predefined in groff.

*[**.T**] The name of the current output device as specified by the **-T** command line option.

REGISTERS

Registers are variables that store a value. In groff, most registers store numerical values (see section **NUMERICAL EXPRESSIONS** above), but some can also hold a string value.

Each register is given a name. Arbitrary registers can be defined and set with the request **nr** *register*.

The value stored in a register can be retrieved by the escape sequences introduced by **\n**.

Most useful are predefined registers. In the following the notation *name* is used to refer to a register called register **name** to make clear that we speak about registers. Please keep in mind that the **\n[]** decoration is not part of the register name.

Read-only Registers

The following registers have predefined values that should not be modified by the user (usually, registers starting with a dot a read-only). Mostly, they provide information on the current settings or store results from request calls.

\n[**.$**] Number of arguments in the current macro or string.
\n[**.a**] Post-line extra line-space most recently utilized using **\x'***N***'**.
\n[**.A**] Set to 1 in **troff** if option **−A** is used; always 1 in **nroff**.
\n[**.c**] Current input line number.
\n[**.C**] 1 if compatibility mode is in effect, 0 otherwise.
\n[**.cdp**] The depth of the last character added to the current environment. It is positive if the character extends below the baseline.
\n[**.ce**] The number of lines remaining to be centered, as set by the **ce** request.
\n[**.cht**] The height of the last character added to the current environment. It is positive if the character extends above the baseline.
\n[**.color**] 1 if colors are enabled, 0 otherwise.
\n[**.csk**] The skew of the last character added to the current environment. The skew of a character is how far to the right of the center of a character the center of an accent over that character should be placed.
\n[**.d**] Current vertical place in current diversion; equal to register register **nl**.
\n[**.ev**] The name or number of the current environment (string-valued).
\n[**.f**] Current font number.
\n[**.fam**] The current font family (string-valued).
\n[**.fn**] The current (internal) real font name (string-valued).
\n[**.fp**] The number of the next free font position.
\n[**.g**] Always 1 in GNU troff. Macros should use it to test if running under groff.
\n[**.h**] Text base-line high-water mark on current page or diversion.
\n[**.H**] Available horizontal resolution in basic units.
\n[**.hla**] The current hyphenation language as set by the **.hla** request.
\n[**.hlc**] The number of immediately preceding consecutive hyphenated lines.
\n[**.hlm**] The maximum allowed number of consecutive hyphenated lines, as set by the **hlm** request.
\n[**.hy**] The current hyphenation flags (as set by the **hy** request).
\n[**.hym**] The current hyphenation margin (as set by the **hym** request).

\n[**.hys**] The current hyphenation space (as set by the **hys** request).
\n[**.i**] Current ident.
\n[**.in**] The indent that applies to the current output line.
\n[**.int**] Positive if last output line contains **c**.
\n[**.kern**] 1 if pairwise kerning is enabled, 0 otherwise.
\n[**.l**] Current line length.
\n[**.lg**] The current ligature mode (as set by the **lg** request).
\n[**.linetabs**]
The current line-tabs mode (as set by the **linetabs** request).
\n[**.ll**] The line length that applies to the current output line.
\n[**.lt**] The title length (as set by the **lt** request).
\n[**.n**] Length of text portion on previous output line.
\n[**.ne**] The amount of space that was needed in the last **ne** request that caused a trap to be sprung. Useful in conjunction with register **.trunc**.
\n[**.ns**] 1 if in no-space mode, 0 otherwise.
\n[**.o**] Current page offset.
\n[**.p**] Current page length.
\n[**.pn**] The number of the next page: either the value set by a **pn** request, or the number of the current page plus 1.
\n[**.ps**] The current pointsize in scaled points.
\n[**.psr**] The last-requested pointsize in scaled points.
\n[**.pvs**] The current post-vertical line spacing.
\n[**.rj**] The number of lines to be right-justified as set by the rj request.
\n[**.s**] Current point size as a decimal fraction.
\n[**.sr**] The last requested pointsize in points as a decimal fraction (string-valued).
\n[**.t**] Distance to the next trap.
\n[**.T**] Set to 1 if option **−T** is used.
\n[**.tabs**] A string representation of the current tab settings suitable for use as an argument to the **ta** request.
\n[**.trunc**]
The amount of vertical space truncated by the most recently sprung vertical position trap, or, if the trap was sprung by a **ne** request, minus the amount of vertical motion produced by **.ne**. In other words, at the point a trap is sprung, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is. Useful in conjunction with the register **.ne** register.
\n[**.ss**] The value of the parameters set by the first argument of the **ss** request.
\n[**.sss**] The value of the parameters set by the second argument of the **ss** request.
\n[**.u**] Equal to 1 bin fill mode and 0 in nofill mode.
\n[**.v**] Current vertical line spacing.
\n[**.V**] Available vertical resolution in basic units.
\n[**.vpt**] 1 if vertical position traps are enabled, 0 otherwise.
\n[**.w**] Width of previous character.
\n[**.warn**] The sum of the number codes of the currently enabled warnings.
\n[**.x**] The major version number.
\n[**.y**] The minor version number.
\n[**.Y**] The revision number of groff.
\n[**.z**] Name of current diversion.

Writable Registers

The following registers can be read and written by the user. They have predefined default values, but these can be modified for customizing a document.

\n[**%**] Current page number.
\n[**c.**] Current input line number.
\n[**ct**] Character type (set by width function **w**).

\n[**dl**] Maximal width of last completed diversion.
\n[**dn**] Height of last completed diversion.
\n[**dw**] Current day of week (1-7).
\n[**dy**] Current day of month (1-31).
\n[**hours**] The number of hours past midnight. Initialized at start-up.
\n[**hp**] Current horizontal position at input line.
\n[**llx**] Lower left x-coordinate (in PostScript units) of a given PostScript image (set by **.psbb**).
\n[**lly**] Lower left y-coordinate (in PostScript units) of a given PostScript image (set by **.psbb**).
\n[**ln**] Output line number.
\n[**minutes**]
The number of minutes after the hour. Initialized at start-up.
\n[**mo**] Current month (1-12).
\n[**nl**] Vertical position of last printed text base-line.
\n[**rsb**] Like register **sb**, but takes account of the heights and depths of characters.
\n[**rst**] Like register **st**, but takes account of the heights and depths of characters.
\n[**sb**] Depth of string below base line (generated by width function **\w**).
\n[**seconds**]
The number of seconds after the minute. Initialized at start-up.
\n[**skw**] Right skip width from the center of the last character in the **\w** argument.
\n[**slimit**]
If greater than 0, the maximum number of objects on the input stack. If ≤0 there is no limit, i.e., recursion can continue until virtual memory is exhausted.
\n[**ssc**] The amount of horizontal space (possibly negative) that should be added to the last character before a subscript (generated by width function **\w**).
\n[**st**] Height of string above base line (generated by width function **\w**).
\n[**systat**]
The return value of the *system()* function executed by the last **sy** request.
\n[**urx**] Upper right x-coordinate (in PostScript units) of a given PostScript image (set by **.psbb**).
\n[**ury**] Upper right y-coordinate (in PostScript units) of a given PostScript image (set by **.psbb**).
\n[**year**] The current year (year 2000 compliant).
\n[**yr**] Current year minus 1900. For Y2K compliance use register register **year** instead.

COMPATIBILITY

The differences of the groff language in comparison to classical troff as defined by *[CSTR #54]* are documented in **groff_diff**(7).

The groff system provides a compatibility mode, see **groff**(1) on how to invoke this.

BUGS

Report bugs to the groff bug mailing list ⟨bug-groff@gnu.org⟩. Include a complete, self-contained example that will allow the bug to be reproduced, and say which version of groff you are using.

AUTHORS

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

SEE ALSO

The main source of information for the groff language is the **groff info**(1) file. Besides the gory details, it contains many examples.

groff(1)
the usage of the groff program and pointers to the documentation and availability of the groff system.

groff_diff(7)
: the differences of the groff language as compared to classical roff. This is the authoritative document for the predefined language elements that are specific to groff.

groff_char(7)
: the predefined groff characters (glyphs).

groff_font(5)
: the specification of fonts and the DESC file.

roff(7) the history of roff, the common parts shared by all roff systems, and pointers to further documentation.

[CSTR #54]
: Nroff/Troff User's Manual by Osanna & Kernighan ⟨`http://cm.bell-labs.com/cm/cs/54.ps`⟩ — the bible for classical troff.