Name

groff – GNU *roff* language reference

Description

groff is short for GNU *roff*, a free reimplementation of the AT&T device-independent *troff* typesetting system. See *roff* (7) for a survey of and background on *roff* systems.

This document is intended as a reference. The primary *groff* manual, *Groff: The GNU Implementation of troff*, by Trent A. Fisher and Werner Lemberg, is a better resource for learners, containing many examples and much discussion. It is written in Texinfo; you can browse it interactively with "info groff". Additional formats, including plain text, HTML, DVI, and PDF, may be available in */usr/local/share/doc/groff−1.23.0*.

groff is also a name for an extended dialect of the *roff* language. We use "roff" to denote features that are universal, or nearly so, among implementations of this family. We apply the term "groff" to the language documented here, the GNU implementation of the overall system, the project that develops that system, and the command of that name.

GNU *troff*, installed on this system as *troff* (1), is the *formatter:* a program that reads device and font descriptions (*groff_font*(5)), interprets the *groff* language expressed in text input files, and translates that input into a device-independent output format (*groff_out*(5)) that is usually then post-processed by an output driver to produce PostScript, PDF, HTML, DVI, or terminal output.

Input format

Input to GNU *troff* is organized into lines separated by the Unix newline character (U+000A), and must be in one of two character encodings it can recognize: IBM code page 1047 on EBCDIC systems, and ISO Latin-1 (8859-1) otherwise. Use of ISO 646-1991:IRV ("US-ASCII") or (equivalently) the "Basic Latin" subset of ISO 10646 ("Unicode") is recommended; see *groff_char*(7). The *preconv*(1) preprocessor transforms other encodings, including UTF-8, to satisfy *troff*'s requirements.

Syntax characters

Several input characters are syntactically significant to *groff*.

. A dot at the beginning of an input line marks it as a *control line.* It can also follow the **.el** and **.nop** requests, and the condition in **.if**, **.ie**, and **.while** requests. The control character invokes requests and calls macros by the name that follows it. The **.cc** request can change the control character.

' The neutral apostrophe is the *no-break control character,* recognized where the control character is. It suppresses the (first) break implied by the **.bp**, **.cf**, **.fi**, **.fl**, **.in**, **.nf**, **.rj**, **.sp**, **.ti**, and **.trf** requests. The requested operation takes effect at the next break. It makes **.br** nilpotent. The no-break control character can be changed with the **.c2** request. When formatted, "'" may be typeset as a typographical quotation mark; use the **\[aq]** special character escape sequence to format a neutral apostrophe glyph.

" The neutral double quote can be used to enclose arguments to macros and strings, and is required if those arguments contain space or tab characters. In the **.ds**, **.ds1**, **.as**, and **.as1** requests, an initial neutral double quote in the second argument is stripped off to allow embedding of leading spaces. To include a double quote inside a quoted argument, use the **\[dq]** special character escape sequence (which also serves to typeset the glyph in text).

\ A backslash introduces an escape sequence. The escape character can be changed with the **.ec** request; **.eo** disables escape sequence recognition. Use the **\[rs]** special character escape sequence to format a backslash glyph, and **\e** to typeset the glyph of the current escape character.

(An opening parenthesis is special only in certain escape sequences; when recognized, it introduces an argument of exactly two characters. *groff* offers the more flexible square bracket syntax.

[An opening bracket is special only in certain escape sequences; when recognized, it introduces an argument (list) of any length, not including a closing bracket.

] A closing bracket is special only when an escape sequence using an opening bracket as an argument delimiter is being interpreted. It ends the argument (list).

Additionally, the Control+A character (U+0001) in text is interpreted as a *leader* (see below).

Horizontal white space characters are significant to *groff,* but trailing spaces on text lines are ignored.

space Space characters separate arguments in request invocations, macro calls, and string interpolations. In text, they separate words. Multiple adjacent space characters in text cause *groff* to attempt end-of-sentence detection on the preceding word (and trailing punctuation). The amount of space between words and sentences is controlled by the **.ss** request. When filling is enabled (the default), a line may be broken at a space. When adjustment is enabled (the default), inter-word spaces are expanded until the output line reaches the configured length. An adjustable but non-breaking space is available with **\~**. To get a space of fixed width, use one of the escape sequences '**\ **' (the escape character followed by a space), **\0**, **\|**, **\^**, or **\h**; see section "Escape sequences" below.

newline In text, a newline puts an inter-word space onto the output and, if filling is enabled, triggers end-of-sentence recognition on the preceding text. See section "Line continuation" below.

tab A tab character in text causes the drawing position to advance to the next defined tab stop.

Tabs and leaders

The formatter interprets input horizontal tab characters ("tabs") and Control+A characters ("leaders") into movements to the next tab stop. Tabs simply move to the next tab stop; leaders place enough periods to fill the space. Tab stops are by default located every half inch measured from the drawing position corresponding to the beginning of the input line; see section "Page geometry" of *roff* (7). Tabs and leaders do not cause breaks and therefore do not interrupt filling. Tab stops can be configured with the **ta** request, and tab and leader glyphs with the **tc** and **lc** requests, respectively.

Line continuation

When filling is enabled, input and output line breaks generally do not correspond. The *roff* language therefore distinguishes input and output line continuation.

A backslash \ immediately followed by a newline, sometimes discussed as *****newline*, suppresses the effects of that newline on the input. The next input line thus retains the classification of its predecessor as a control or text line. *****newline* is useful for managing line lengths in the input during document maintenance; you can break an input line in the middle of a request invocation, macro call, or escape sequence. Input line continuation is invisible to the formatter, with two exceptions: the | operator recognizes the new input line, and the input line counter register **.c** is incremented.

The **\c** escape sequence continues an *output* line. Nothing on the input line after it is formatted. In contrast to *****newline*, a line after **\c** is treated as a new input line, so a control character is recognized at its beginning. The visual results depend on whether filling is enabled. An intervening control line that causes a break overrides **\c**, flushing out the pending output line in the usual way. The register **.int** contains a positive value if the last output line was continued with **\c**; this datum is associated with the environment.

Colors

groff supports color output with a variety of color spaces and up to 16 bits per channel. Some devices, particularly terminals, may be more limited. When color support is enabled, two colors are current at any given time: the *stroke color,* with which glyphs, rules (lines), and geometric objects like circles and polygons are drawn, and the *fill color,* which can be used to paint the interior of a closed geometric figure. The **color**, **defcolor**, **gcolor**, and **fcolor** requests; **\m** and **\M** escape sequences; and **.color**, **.m**, and **.M** registers exercise color support.

Each output device has a color named "**default**", which cannot be redefined. A device's default stroke and fill colors are not necessarily the same. For the **dvi**, **html**, **pdf**, **ps**, and **xhtml** output devices, *troff* automatically loads a macro file defining many color names at startup. By the same mechanism, the devices supported by *grotty*(1) recognize the eight standard ISO 6429/ECMA-48 color names (also known vulgarly as "ANSI colors").

Measurements

Numeric parameters that specify measurements are expressed as integers or decimal fractions with an optional *scaling unit* suffixed. A scaling unit is a letter that immediately follows the last digit of a number. Digits after the decimal point are optional.

Measurements are scaled by the scaling unit and stored internally (with any fractional part discarded) in basic units. The device resolution can therefore be obtained by storing a value of "**1i**" to a register. The only constraint on the basic unit is that it is at least as small as any other unit.

u Basic unit.
i Inch; defined as 2.54 centimeters.
c Centimeter.
p Point; a typesetter's unit used for measuring type size. There are 72 points to an inch.
P Pica; another typesetter's unit. There are 6 picas to an inch and 12 points to a pica.
s, **z** Scaled points and multiplication by the output device's *sizescale* parameter, respectively.
f Multiplication by 65,536; scales decimal fractions in the interval [0, 1] to 16-bit unsigned integers.

The magnitudes of other scaling units depend on the text formatting parameters in effect.

m Em; an em is equal to the current type size in points.
n En; an en is one-half em.
v Vee; distance between text baselines.
M Hundredth of an em.

Motion quanta

An output device's basic unit **u** is not necessarily its smallest addressable length; **u** can be smaller to avoid problems with integer roundoff. The minimum distances that a device can work with in the horizontal and vertical directions are termed its *motion quanta,* stored in the **.H** and **.V** registers, respectively. Measurements are rounded to applicable motion quanta. Half-quantum fractions round toward zero.

Default units

A general-purpose register (one created or updated with the **nr** request; see section "Registers" below) is implicitly dimensionless, or reckoned in basic units if interpreted in a measurement context. But it is convenient for many requests and escape sequences to infer a scaling unit for an argument if none is specified. An explicit scaling unit (not after a closing parenthesis) can override an undesirable default. Effectively, the default unit is suffixed to the expression if a scaling unit is not already present. GNU *troff*'s use of integer arithmetic should also be kept in mind; see below.

Numeric expressions

A *numeric expression* evaluates to an integer. The following operators are recognized.

	+	addition
	−	subtraction
	*	multiplication
	/	truncating division
	%	modulus
unary	+	assertion, motion, incrementation
unary	−	negation, motion, decrementation
	;	scaling
	>?	maximum
	<?	minimum
	<	less than
	>	greater than
	<=	less than or equal
	>=	greater than or equal
	=	equal
	==	equal
	&	logical conjunction ("and")
	:	logical disjunction ("or")
	!	logical complementation ("not")
	( )	precedence
	|	boundary-relative motion

troff provides a set of mathematical and logical operators familiar to programmers—as well as some unusual ones—but supports only integer arithmetic. (Provision is made for interpreting and reporting decimal fractions in certain cases.) The internal data type used for computing results is usually a 32-bit signed integer, which suffices to represent magnitudes within a range of ±2 billion. (If that's not enough, see *groff_tmac*(5) for the *62bit.tmac* macro package.)

Arithmetic infix operators perform a function on the numeric expressions to their left and right; they are **+** (addition), **−** (subtraction), ***** (multiplication), **/** (truncating division), and **%** (modulus). *Truncating division* rounds to the integer nearer to zero, no matter how large the fractional portion. Overflow and division (or modulus) by zero are errors and abort evaluation of a numeric expression.

Arithmetic unary operators operate on the numeric expression to their right; they are **−** (negation) and **+** (assertion—for completeness; it does nothing). The unary minus must often be used with parentheses to avoid confusion with the decrementation operator, discussed below.

The sign of the modulus of operands of mixed signs is determined by the sign of the first. Division and modulus operators satisfy the following property: given a dividend *a* and a divisor *b*, a quotient *q* formed by "**(a / b)**" and a remainder *r* by "**(a % b)**", then $qb + r = a$.

GNU *troff*'s scaling operator, used with parentheses as (*c***;***e***)**, evaluates a numeric expression *e* using *c* as the default scaling unit. If *c* is omitted, scaling units are ignored in the evaluation of *e*. GNU *troff* also provides a pair of operators to compute the extrema of two operands: **>?** (maximum) and **<?** (minimum).

Comparison operators comprise **<** (less than), **>** (greater than), **<=** (less than or equal), **>=** (greater than or equal), and **=** (equal). **==** is a synonym for **=**. When evaluated, a comparison is replaced with "**0**" if it is false and "**1**" if true. In the *roff* language, positive values are true, others false.

We can operate on truth values with the logical operators **&** (logical conjunction or "and") and **:** (logical disjunction or "or"). They evaluate as comparison operators do. A logical complementation ("not") operator, **!,** works only within "**if**", "**ie**", and "**while**" requests. Furthermore, **!** is recognized only at the beginning of a numeric expression not contained by another numeric expression. In other words, it must be the "outermost" operator. Including it elsewhere in the expression produces a warning in the "**number**" category (see *troff*(1)), and its expression evaluates false. This unfortunate limitation maintains compatibility with AT&T *troff*. Test a numeric expression for falsity by comparing it to a false value.

The *roff* language has no operator precedence: expressions are evaluated strictly from left to right, in contrast to schoolhouse arithmetic. Use parentheses **( )** to impose a desired precedence upon subexpressions.

For many requests and escape sequences that cause motion on the page, the unary operators **+** and **−** work differently when leading a numeric expression. They then indicate a motion relative to the drawing position: positive is down in vertical contexts, right in horizontal ones.

+ and **−** are also treated differently by the following requests and escape sequences: **bp**, **in**, **ll**, **pl**, **pn**, **po**, **ps**, **pvs**, **rt**, **ti**, **\H**, **\R**, and **\s**. Here, leading plus and minus signs serve as incrementation and decrementation operators, respectively. To negate an expression, subtract it from zero or include the unary minus in parentheses with its argument.

A leading **|** operator indicates a motion relative not to the drawing position but to a boundary. For horizontal motions, the measurement specifies a distance relative to a drawing position corresponding to the beginning of the *input* line. By default, tab stops reckon movements in this way. Most escape sequences do not; **|** tells them to do so. For vertical motions, the **|** operator specifies a distance from the first text baseline on the page or in the current diversion, using the current vertical spacing.

The **\B** escape sequence tests its argument for validity as a numeric expression.

A register interpolated as an operand in a numeric expression must have an Arabic format; luckily, this is the default.

Due to the way arguments are parsed, spaces are not allowed in numeric expressions unless the (sub)expression containing them is surrounded by parentheses.

Identifiers

An *identifier* labels a GNU *troff* datum such as a register, name (macro, string, or diversion), typeface, color, special character, character class, environment, or stream. Valid identifiers consist of one or more ordinary characters. An *ordinary character* is an input character that is not the escape character, a leader, tab, newline, or invalid as GNU *troff* input.

Invalid input characters are subset of control characters (from the sets "C0 Controls" and "C1 Controls" as Unicode describes them). When *troff* encounters one in an identifier, it produces a warning in category "**input**" (see section "Warnings" in *troff*(1)). They are removed during interpretation: an identifier "foo", followed by an invalid character and then "bar", is processed as "foobar".

On a machine using the ISO 646, 8859, or 10646 character encodings, invalid input characters are **0x00**, **0x08**, **0x0B**, **0x0D–0x1F**, and **0x80–0x9F**. On an EBCDIC host, they are **0x00–0x01**, **0x08**, **0x09**, **0x0B**, **0x0D–0x14**, **0x17–0x1F**, and **0x30–0x3F**. Some of these code points are used by *troff* internally, making it non-trivial to extend the program to accept UTF-8 or other encodings that use characters from these ranges.

An identifier with a closing bracket ("]") in its name can't be accessed with bracket-form escape sequences that expect an identifier as a parameter. Similarly, the identifier "(" can't be interpolated *except* with bracket forms.

If you begin a macro, string, or diversion name with either of the characters "[" or "]", you foreclose use of the *refer*(1) preprocessor, which recognizes ".[" and ".]" as bibliographic reference delimiters.

The escape sequence **\A** tests its argument for validity as an identifier.

How GNU *troff* handles the interpretation of an undefined identifier depends on the context. There is no way to invoke an undefined request; such syntax is interpreted as a macro call instead. If the identifier is interpreted as a string, macro, or diversion, *troff* emits a warning in category "**mac**", defines it as empty, and interpolates nothing. If the identifier is interpreted as a register, *troff* emits a warning in category "**reg**", initializes it to zero, and interpolates that value. See section "Warnings" in *troff*(1), and subsection "Interpolating registers" and section "Strings" below. Attempting to use an undefined typeface, style, special character, color, character class, environment, or stream generally provokes an error diagnostic.

Identifiers for requests, macros, strings, and diversions share one name space; special characters and character classes another. No other object types do.

Control characters

Control characters are recognized only at the beginning of an input line, or at the beginning of the branch of a control structure request; see section "Control structures" below.

A few requests cause a break implicitly; use the no-break control character to prevent the break. Break suppression is its sole behavioral distinction. Employing the no-break control character to invoke requests that don't cause breaks is harmless but poor style.

The control character "**.**" and the no-break control character "**'**" can be changed with the **cc** and **c2** requests, respectively. Within a macro definition, register **.br** indicates the control character used to call it.

Invoking requests

A control character is optionally followed by tabs and/or spaces and then an identifier naming a request or macro. The invocation of an unrecognized request is interpreted as a macro call. Defining a macro with the same name as a request replaces the request. Deleting a request name with the **rm** request makes it unavailable. The **als** request can alias requests, permitting them to be wrapped or non-destructively replaced. See section "Strings" below.

There is no inherent limit on argument length or quantity. Most requests take one or more arguments, and ignore any they do not expect. A request may be separated from its arguments by tabs or spaces, but only spaces can separate an argument from its successor. Only one between arguments is necessary; any excess is ignored. GNU *troff* does not allow tabs for argument separation.

Generally, a space *within* a request argument is not relevant, not meaningful, or is supported by bespoke provisions, as with the **tl** request's delimiters. Some requests, like **ds**, interpret the remainder of the control line as a single argument. See section "Strings" below.

Spaces and tabs immediately after a control character are ignored. Commonly, authors structure the source of documents or macro files with them.

Calling macros

If a macro of the desired name does not exist when called, it is created, assigned an empty definition, and a warning in category "**mac**" is emitted. Calling an undefined macro *does* end a macro definition naming it as its end macro (see section "Writing macros" below).

To embed spaces *within* a macro argument, enclose the argument in neutral double quotes '**"**'. Horizontal motion escape sequences are sometimes a better choice for arguments to be formatted as text.

The foregoing raises the question of how to embed neutral double quotes or backslashes in macro arguments when *those* characters are desired as literals. In GNU *troff*, the special character escape sequence **\[rs]** produces a backslash and **\[dq]** a neutral double quote.

In GNU *troff*'s AT&T compatibility mode, these characters remain available as **\(rs** and **\(dq**, respectively. AT&T *troff* did not consistently define these special characters, but its descendants can be made to support them. See *groff_font*(5). If even that is not feasible, see the "Calling Macros" section of the *groff* Texinfo manual for the complex macro argument quoting rules of AT&T *troff*.

Using escape sequences

Whereas requests must occur on control lines, escape sequences can occur intermixed with text and may appear in arguments to requests, macros, and other escape sequences. An escape sequence is introduced by the escape character, a backslash ****. The next character selects the escape's function.

Escape sequences vary in length. Some take an argument, and of those, some have different syntactical forms for a one-character, two-character, or arbitrary-length argument. Others accept *only* an arbitrary-length argument. In the former scheme, a one-character argument follows the function character immediately, an opening parenthesis "(" introduces a two-character argument (no closing parenthesis is used), and an argument of arbitrary length is enclosed in brackets "[]". In the latter scheme, the user selects a delimiter character. A few escape sequences are idiosyncratic, and support both of the foregoing conventions (**\s**), designate their own termination sequence (**\?**), consume input until the next newline (**\!**, **\"**, **\#**), or support an additional modifier character (**\s** again, and **\n**).

If an escape character is followed by a character that does not identify a defined operation, the escape character is ignored (producing a diagnostic of the "**escape**" warning category, which is not enabled by default) and the following character is processed normally.

Escape sequence interpolation is of higher precedence than escape sequence argument interpretation. This rule affords flexibility in using escape sequences to construct parameters to other escape sequences.

The escape character can be interpolated (**\e**). Requests permit the escape mechanism to be deactivated (**eo**) and restored, or the escape character changed (**ec**), and to save and restore it (**ecs** and **ecr**).

Delimiters

Some escape sequences that require parameters use delimiters. The neutral apostrophe **'** is a popular choice and shown in this document. The neutral double quote **"** is also commonly seen. Letters, numerals, and leaders can be used. Punctuation characters are likely better choices, except for those defined as infix operators in numeric expressions; see below.

The following escape sequences don't take arguments and thus are allowed as delimiters: *****space*, **\%**, **\|**, **\^**, **\{**, **\}**, **\'**, **\`**, **\–**, **_**, **\!**, **\?**, **\)**, **\/**, **\,**, **\&**, **\:**, **\~**, **\0**, **\a**, **\c**, **\d**, **\e**, **\E**, **\p**, **\r**, **\t**, and **\u**. However, using them this way is discouraged; they can make the input confusing to read.

A few escape sequences, **\A**, **\b**, **\o**, **\w**, **\X**, and **\Z**, accept a newline as a delimiter. Newlines that serve as delimiters continue to be recognized as input line terminators. Use of newlines as delimiters in escape sequences is also discouraged.

Finally, the escape sequences **\D**, **\h**, **\H**, **\l**, **\L**, **\N**, **\R**, **\s**, **\S**, **\v**, and **\x** prohibit many delimiters.

- the numerals 0–9 and the decimal point "**.**"

- the (single-character) operators **+–/*%<>=&:()**
- any escape sequences other than **\%**, **\:**, **\{**, **\}**, **\'**, **\`**, **\–**, **_**, **\!**, **\/**, **\c**, **\e**, and **\p**

Delimiter syntax is complex and flexible primarily for historical reasons; the foregoing restrictions need be kept in mind mainly when using *groff* in AT&T compatibility mode. GNU *troff* keeps track of the nesting depth of escape sequence interpolations, so the only characters you need to avoid using as delimiters are those that appear in the arguments you input, not any that result from interpolation. Typically, **'** works fine. See section "Implementation differences" in *groff_diff* (7).

Dummy characters

As discussed in *roff* (7), the first character on an input line is treated specially. Further, formatting a glyph has many consequences on formatter state (see section "Environments" below). Occasionally, we want to escape this context or embrace some of those consequences without actually rendering a glyph to the output. **\&** interpolates a dummy character, which is constitutive of output but invisible. Its presence alters the interpretation context of a subsequent input character, and enjoys several applications: preventing the insertion of extra space after an end-of-sentence character, preventing interpretation of a control character at the beginning of an input line, preventing kerning between two glyphs, and permitting the **tr** request to remap a character to "nothing". **\)** works as **\&** does, except that it does not cancel a pending end-of-sentence state.

Control structures

groff has "if" and "while" control structures like other languages. However, the syntax for grouping multiple input lines in the branches or bodies of these structures is unusual.

They have a common form: the request name is (except for **.el** "else") followed by a conditional expression *cond-expr*; the remainder of the line, *anything*, is interpreted as if it were an input line. Any quantity of spaces between arguments to requests serves only to separate them; leading spaces in *anything* are therefore not seen. *anything* effectively *cannot* be omitted; if *cond-expr* is true and *anything* is empty, the newline at the end of the control line is interpreted as a blank line (and therefore a blank text line).

It is frequently desirable for a control structure to govern more than one request, macro call, or text line, or a combination of the foregoing. The opening and closing brace escape sequences **\{** and **\}** perform such grouping. Brace escape sequences outside of control structures have no meaning and produce no output.

\{ should appear (after optional spaces and tabs) immediately subsequent to the request's conditional expression. **\}** should appear on a line with other occurrences of itself as necessary to match **\{** sequences. It can be preceded by a control character, spaces, and tabs. Input after any quantity of **\}** sequences on the same line is processed only if all the preceding conditions to which they correspond are true. Furthermore, a **\}** closing the body of a **.while** request must be the last such escape sequence on an input line.

Conditional expressions

The **.if**, **.ie**, and **.while** requests test the truth values of numeric expressions. They also support several additional Boolean operators; the members of this expanded class are termed *conditional expressions*; their truth values are as shown below.

***cond-expr*...**	**...is true if...**
'*s1***'***s2***'**	*s1* produces the same formatted output as *s2*.
c *g*	a glyph *g* is available.
d *m*	a string, macro, diversion, or request *m* is defined.
e	the current page number is even.
F *f*	a font named *f* is available.
m *c*	a color named *c* is defined.
n	the formatter is in *nroff* mode.
o	the current page number is odd.
r *n*	a register named *n* is defined.
S *s*	a font style named *s* is available.
t	the formatter is in *troff* mode.
v	n/a (historical artifact; always false).

If the first argument to an **.if**, **.ie**, or **.while** request begins with a non-alphanumeric character apart from **!** (see below); it performs an *output comparison test.* Shown first in the table above, the *output comparison operator* interpolates a true value if formatting its comparands *s1* and *s2* produces the same output commands. Other delimiters can be used in place of the neutral apostrophes. *troff* formats *s1* and *s2* in separate environments; after the comparison, the resulting data are discarded. The resulting glyph properties, including font family, style, size, and slant, must match, but not necessarily the requests and/or escape sequences used to obtain them. Motions must match in orientation and magnitude to within the applicable horizontal or vertical motion quantum of the device, after rounding.

Surround the comparands with **\?** to avoid formatting them; this causes them to be compared character by character, as with string comparisons in other programming languages. Since comparands protected with **\?** are read in copy mode, they need not even be valid *groff* syntax. The escape character is still lexically recognized, however, and consumes the next character.

The above operators can't be combined with most others, but a leading "**!**", not followed immediately by spaces or tabs, complements an expression. Spaces and tabs are optional immediately after the "**c**", "**d**", "**F**", "**m**", "**r**", and "**S**" operators, but right after "**!**", they end the predicate and the conditional evaluates true. (This bizarre behavior maintains compatibility with AT&T *troff*.)

Syntax reference conventions

In the following request and escape sequence specifications, most argument names were chosen to be descriptive. A few denotations may require introduction.

c	denotes a single input character.
font	a font either specified as a font name or a numeric mounting position.
anything	all characters up to the end of the line, to the ending delimiter for the escape sequence, or within **\{** and **\}**. Escape sequences may generally be used freely in *anything*, except when it is read in copy mode.
message	is a character sequence to be emitted on the standard error stream. Special character escape sequences are *not* interpreted.
n	is a numeric expression that evaluates to a non-negative integer.
npl	is a numeric expression constituting a count of subsequent *productive* input lines; that is, those that directly produce formatted output. Text lines produce output, as do control lines containing requests like **.tl** or escape sequences like **\D**. Macro calls are not themselves productive, but their interpolated contents can be.
±*N*	is a numeric expression with a meaning dependent on its sign.

If a numeric expression presented as ±*N* starts with a '**+**' sign, an increment in the amount of of *N* is applied to the value applicable to the request or escape sequence. If it starts with a '**−**' sign, a decrement of magnitude *N* is applied instead. Without a sign, *N* replaces any existing value. A leading minus sign in *N* is always interpreted as a decrementation operator, not an algebraic sign. To assign a register a negative value or the negated value of another register, enclose it with its operand in parentheses or subtract it from zero. If a prior value does not exist (the register was undefined), an increment or decrement is applied as if to 0.

Request short reference

Not all details of request behavior are outlined here. See the *groff* Texinfo manual or, for features new to GNU *troff*, *groff_diff*(7).

.ab Abort processing; exit with failure status.

.ab *message*
: Abort processing; write *message* to the standard error stream and exit with failure status.

.ad Enable output line alignment and adjustment using the mode stored in **\n[.j]**.

.ad *c* Enable output line alignment and adjustment in mode *c* (*c*=**b**,**c**,**l**,**n**,**r**). Sets **\n[.j]**.

.af *register c*
: Assign format *c* to *register*, where *c* is "**i**", "**I**", "**a**", "**A**", or a sequence of decimal digits whose quantity denotes the minimum width in digits to be used when the register is interpolated. "**i**" and "**a**" indicate Roman numerals and basic Latin alphabetics, respectively, in the lettercase specified. The default is **0**.

.aln *new old*
Create alias (additional name) *new* for existing register named *old*.
.als *new old*
Create alias (additional name) *new* for existing request, string, macro, or diversion *old*.
.am *macro*
Append to *macro* until **..** is encountered.
.am *macro end*
Append to *macro* until **.***end* is called.
.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.
.ami *macro*
Append to a macro whose name is contained in the string *macro* until **..** is encountered.
.ami *macro end*
Append to a macro indirectly. *macro* and *end* are strings whose contents are interpolated for the macro name and the end macro, respectively.
.ami1 *macro*
Same as **.ami** but with compatibility mode switched off during macro expansion.
.ami1 *macro end*
Same as **.ami** but with compatibility mode switched off during macro expansion.
.as *name*
Create string *name* with empty contents; no operation if *name* already exists.
.as *name contents*
Append *contents* to string *name*.
.as1 *string*
.as1 *string contents*
As **.as**, but with compatibility mode disabled when *contents* interpolated.
.asciify *diversion*
Unformat ASCII characters, spaces, and some escape sequences in *diversion*.
.backtrace
Write the state of the input stack to the standard error stream. See the **−b** option of *groff* (1).
.bd *font* Stop emboldening font *font.*
.bd *font n*
Embolden *font* by overstriking its glyphs offset by $n-1$ units. See register **.b**.
.bd *special-font font*
Stop emboldening *special-font* when *font* is selected.
.bd *special-font font n*
Embolden *special-font,* overstriking its glyphs offset by $n-1$ units when *font* is selected. See register **.b**.
.blm Unset blank line macro (trap). Restore default handling of blank lines.
.blm *name*
Set blank line macro (trap) to *name*.
.box Stop directing output to current diversion; any pending output line is discarded.
.box *name*
Direct output to diversion *name*, omitting a partially collected line.
.boxa Stop appending output to current diversion; any pending output line is discarded.
.boxa *name*
Append output to diversion *name*, omitting a partially collected line.
.bp Break page and start a new one.
.bp $\pm N$ Break page, starting a new one numbered $\pm N$.
.br Break output line.

.brp Break output line; adjust if applicable.

.break Break out of a while loop.

.c2 Reset no-break control character to "**'**".

.c2 *o* Recognize ordinary character *o* as no-break control character.

.cc Reset control character to '**.**'.

.cc *o* Recognize ordinary character *o* as the control character.

.ce Break, center the output of the next productive input line without filling, and break again.

.ce *npl* Break, center the output of the next *npl* productive input lines without filling, then break again. If *npl* ≤ 0, stop centering.

.cf *file* Copy contents of *file* without formatting to the (top-level) diversion.

.cflags *n c1 c2 ...*
Assign properties encoded by *n* to characters *c1*, *c2*, and so on.

.ch *name*
Unplant page location trap *name*.

.ch *name vpos*
Change page location trap *name* planted by **.wh** by moving its location to *vpos* (default scaling unit **v**).

.char *c contents*
Define ordinary or special character *c* as *contents*.

.chop *object*
Remove the last character from the macro, string, or diversion named *object*.

.class *name c1 c2 ...*
Define a (character) class *name* comprising the characters or range expressions *c1*, *c2*, and so on.

.close *stream*
Close the *stream*.

.color Enable output of color-related device-independent output commands.

.color *n*
If *n* is zero, disable output of color-related device-independent output commands; otherwise, enable them.

.composite *from to*
Map glyph name *from* to glyph name *to* while constructing a composite glyph name.

.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 Continuously underline the output of the next productive input line.

.cu *npl* Continuously underline the output of the next *npl* productive input lines. If *npl*=0, stop continuously underlining.

.da Stop appending output to current diversion.

.da *name*
Append output to diversion *name*.

.de *macro*
Define or redefine *macro* until "**..**" occurs at the start of a control line in the current conditional block.

.de *macro end*
Define or redefine *macro* until *end* is invoked or called at the start of a control line in the current conditional block.

.de1 *macro*
As **.de**, but disable compatibility mode during macro expansion.

.de1 *macro end*
As "**.de** *macro end*", but disable compatibility mode during macro expansion.

.defcolor *ident scheme color-component* ...
Define a color named *ident*. *scheme* identifies a color space and determines the number of required *color-component*s; it must be one of "**rgb**" (three components), "**cmy**" (three), "**cmyk**" (four), or "**gray**" (one). "**grey**" is accepted as a synonym of "**gray**". The color components can be encoded as a single hexadecimal value starting with **#** or **##**. The former indicates that each component is in the range 0–255 (0–FF), the latter the range 0–65,535 (0–FFFF). Alternatively, each color component can be specified as a decimal fraction in the range 0–1, interpreted using a default scaling unit of "**f**", which multiplies its value by 65,536 (but clamps it at 65,535). Each output device has a color named "**default**", which cannot be redefined. A device's default stroke and fill colors are not necessarily the same.

.dei *macro*
Define macro indirectly. As **.de**, but use interpolation of string *macro* as the name of the defined macro.

.dei *macro end*
Define macro indirectly. As **.de**, but use interpolations of strings *macro* and *end* as the names of the defined and end macros.

.dei1 *macro*
As **.dei**, but disable compatibility mode during macro expansion.

.dei1 *macro end*
As **.dei** *macro end*, but disable compatibility mode during macro expansion.

.device *anything*
Write *anything*, read in copy mode, to *troff* output as a device control command. An initial neutral double quote is stripped to allow embedding of leading spaces.

.devicem *name*
Write contents of macro or string *name* to *troff* output as a device control command.

.di Stop directing output to current diversion.

.di *name*
Direct output to diversion *name*.

.do *name* ...
Interpret the string, request, diversion, or macro *name* (along with any arguments) with compatibility mode disabled. Compatibility mode is restored (only if it was active) when the *expansion* of *name* is interpreted.

.ds *name*
Create empty string *name*.

.ds *name contents*
Create a string *name* containing *contents*.

.ds1 *name*

.ds1 *name contents*
As **.ds**, but with compatibility mode disabled when *contents* interpolated.

.dt Clear diversion trap.

.dt *vertical-position name*
Set the diversion trap to macro *name* at *vertical-position* (default scaling unit **v**).

.ec Recognize **** as the escape character.

.ec *o* Recognize ordinary character *o* as the escape character.

.ecr Restore escape character saved with **.ecs**.

.ecs Save the escape character.

.el *anything*
Interpret *anything* as if it were an input line if the conditional expression of the corresponding **.ie** request was false.

.em *name*
Call macro *name* after the end of input.

.eo Disable the escape mechanism in interpretation mode.
.ev Pop environment stack, returning to previous one.
.ev *env* Push current environment onto stack and switch to *env*.
.evc *env* Copy environment *env* to the current one.
.ex Exit with successful status.
.fam Set default font family to previous value.
.fam *name*
Set default font family to *name*.
.fc Disable field mechanism.
.fc *a* Set field delimiter to *a* and pad glyph to space.
.fc *a b* Set field delimiter to *a* and pad glyph to *b*.
.fchar *c contents*
Define fallback character (or glyph) *c* as *contents*.
.fcolor Restore previous fill color.
.fcolor *c*
Set fill color to *c*.
.fi Enable filling of output lines; a pending output line is broken. Sets **\n[.u]**.
.fl Flush output buffer.
.fp *pos id*
Mount font with font description file name *id* at non-negative position *n*.
.fp *pos id font-description-file-name*
Mount font with *font-description-file-name* as name *id* at non-negative position *n*.
.fschar *f c anything*
Define fallback character (or glyph) *c* for font *f* as string *anything*.
.fspecial *font*
Reset list of special fonts for *font* to be empty.
.fspecial *font s1 s2 ...*
When the current font is *font*, then the fonts *s1*, *s2*, ... are special.
.ft
.ft P Select previous font mounting position (abstract style or font); same as **\f[]** or **\fP**.
.ft *font* Select typeface *font,* which can be a mounting position, abstract style, or font name; same as **\f[***font***]** escape sequence. *font* cannot be **P**.
.ftr *font1 font2*
Translate *font1* to *font2*.
.fzoom *font*
.fzoom *font* **0**
Stop magnifying *font*.
.fzoom *font z*
Set zoom factor for *font* to *z* (in thousandths; default: 1000).
.gcolor Restore previous stroke color.
.gcolor *c*
Set stroke color to *c*.
.hc Reset the hyphenation character to **\%** (the default).
.hc *char* Change the hyphenation character to *char*.
.hcode *c1 code1* [*c2 code2*] ...
Set the hyphenation code of character *c1* to *code1*, that of *c2* to *code2*, and so on.
.hla *lang*
Set the hyphenation language to *lang*.
.hlm *n* Set the maximum quantity of consecutive hyphenated lines to *n*.
.hpf *pattern-file*
Read hyphenation patterns from *pattern-file*.
.hpfa *pattern-file*
Append hyphenation patterns from *pattern-file*.

.hpfcode *a b* [*c d*] . . .
Define mappings for character codes in hyphenation pattern files read with **.hpf** and **.hpfa**.

.hw *word* . . .
Define hyphenation overrides for each *word;* a hyphen "**–**" indicates a hyphenation point.

.hy Set automatic hyphenation mode to **1**.

.hy 0 Disable automatic hyphenation; same as **.nh**.

.hy *mode*
Set automatic hyphenation mode to *mode*; see section "Hyphenation" below.

.hym Set the (right) hyphenation margin to **0** (the default).

.hym *length*
Set the (right) hyphenation margin to *length* (default scaling unit **m**).

.hys Set the hyphenation space to **0** (the default).

.hys *hyphenation-space*
Suppress automatic hyphenation in adjustment modes "**b**" or "**n**" if the line can be justified with the addition of up to *hyphenation-space* to each inter-word space (default scaling unit **m**).

.ie *cond-expr anything*
If *cond-expr* is true, interpret *anything* as if it were an input line, otherwise skip to a corresponding **.el** request.

.if *cond-expr anything*
If *cond-expr* is true, then interpret *anything* as if it were an input line.

.ig Ignore input (except for side effects of **\R** on auto-incrementing registers) until "**..**" occurs at the start of a control line in the current conditional block.

.ig *end* Ignore input (except for side effects of **\R** on auto-incrementing registers) until **.***end* is called at the start of a control line in the current conditional block.

.in Set indentation amount to previous value.

.in ±*N* Set indentation to ±*N* (default scaling unit **m**).

.it Cancel any pending input line trap.

.it *npl name*
Set (or replace) an input line trap in the environment, calling macro *name*, after the next *npl* productive input lines have been read. Lines interrupted with the **\c** escape sequence are counted separately.

.itc Cancel any pending input line trap.

.itc *npl name*
As **.it**, except that input lines interrupted with the **\c** escape sequence are not counted.

.kern Enable pairwise kerning.

.kern *n* If *n* is zero, disable pairwise kerning, otherwise enable it.

.lc Unset leader repetition character.

.lc *c* Set leader repetition character to *c* (default: "**.**").

.length *reg anything*
Compute the number of characters of *anything* and store the count in the register *reg*.

.linetabs
Enable line-tabs mode (calculate tab positions relative to beginning of output line).

.linetabs 0
Disable line-tabs mode.

.lf *n* Set number of next input line to *n*.

.lf *n file* Set number of next input line to *n* and input file name to *file*.

.lg *m* Set ligature mode to *m* (**0** = disable, **1** = enable, **2** = enable for two-letter ligatures only).

.ll Set line length to previous value. Does not affect a pending output line.

.ll ±*N* Set line length to ±*N* (default length 6.5**i**, default scaling unit **m**). Does not affect a pending output line.

.lsm Unset the leading space macro (trap). Restore default handling of lines with leading spaces.

.lsm *name*
Set the leading space macro (trap) to *name*.

.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 Set length of title lines to previous value.

.lt ±*N* Set length of title lines (default length 6.5**i**, default scaling unit **m**).

.mc Cease writing margin character.

.mc *c* Begin writing margin character *c* to the right of each output line.

.mc *c d* Begin writing margin character *c* on each output line at distance *d* to the right of the right margin (default distance 10**p**, default scaling unit **m**).

.mk Mark vertical drawing position in an internal register; see **.rt**.

.mk *register*
Mark vertical drawing position in *register*.

.mso *file* As **.so**, except that *file* is sought in the *tmac* directories.

.msoquiet *file*
As **.mso**, but no warning is emitted if *file* does not exist.

.na Disable output line adjustment.

.ne Break page if distance to next page location trap is less than one vee.

.ne *d* Break page if distance to next page location trap is less than distance *d* (default scaling unit **v**).

.nf Disable filling of output lines; a pending output line is broken. Clears **\n[.u]**.

.nh Disable automatic hyphenation; same as "**.hy 0**".

.nm Deactivate output line numbering.

.nm ±*N*

.nm ±*N m*

.nm ±*N m s*

.nm ±*N m s i*
Activate output line numbering: number the next output line ±*N,* writing numbers every *m* lines, with *s* numeral widths (**\0**) between the line number and the output (default 1), and indenting the line number by *i* numeral widths (default 0).

.nn Suppress numbering of the next output line to be numbered with **nm**.

.nn *n* Suppress numbering of the next *n* output lines to be numbered with **nm**. If *n*=0, cancel suppression.

.nop *anything*
Interpret *anything* as if it were an input line.

.nr *reg* ±*N*
Define or update register *reg* with value *N*.

.nr *reg* ±*N I*
Define or update register *reg* with value *N* and auto-increment *I*.

.nroff Make the conditional expressions **n** true and **t** false.

.ns Enable *no-space mode*, ignoring **.sp** requests until a glyph or **\D** primitive is output. See **.rs**.

.nx Immediately jump to end of current file.

.nx *file* Stop formatting current file and begin reading *file.*

.open *stream file*
Open *file* for writing and associate the stream named *stream* with it. Unsafe request; disabled by default.

.opena *stream file*
As **.open**, but append to *file.* Unsafe request; disabled by default.

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

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

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

.pc *c* Page number character.

.pev Report the state of the current environment followed by that of all other environments to the standard error stream.

.pi *program*
Pipe output to *program* (*nroff* only). Unsafe request; disabled by default.

.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 unit **v**).

.pm Report, to the standard error stream, the names and sizes in bytes of defined macros, strings, and diversions.

.pn ±*N* Next page number *N*.

.pnr Write the names and contents of all defined registers to the standard error stream.

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

.po ±*N* Page offset *N*.

.ps Return to previous type size.

.ps ±*N* Set/increase/decrease the type size to/by *N* scaled points (a non-positive resulting type size is set to 1 u); also see **\s[**±*N***]**.

.psbb *file*
Retrieve the bounding box of the PostScript image found in *file,* which must conform to Adobe's Document Structuring Conventions (DSC). See registers **llx**, **lly**, **urx**, **ury**.

.pso *command-line*
Execute *command-line* with *popen*(3) and interpolate its output. Unsafe request; disabled by default.

.ptr Report names and positions of all page location traps to the standard error stream.

.pvs Change to previous post-vertical line spacing.

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

.rchar *c1 c2 ...*
Remove definition of each ordinary or special character *c1*, *c2*, ... defined by a **.char**, **.fchar**, or **.schar** request.

.rd *prompt*
Read insertion.

.return Return from a macro.

.return *anything*
Return twice, namely from the macro at the current level and from the macro one level higher.

.rfschar *f c1 c2 ...*
Remove the font-specific definitions of glyphs *c1*, *c2*, ... for font *f*.

.rj *npl* Break, right-align the output of the next productive input line without filling, then break again.

.rj *npl* Break, right-align the output of the next *npl* productive input lines without filling, then break again. If *npl* ≤ 0, stop right-aligning.

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

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

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

.rr *ident* Remove register *ident*.

.rs Restore spacing; disable no-space mode. See **.ns**.

.rt Return *(upward only)* to vertical position marked by **.mk** on the current page.

.rt *N* Return *(upward only)* to vertical position *N* (default scaling unit **v**).

.schar *c contents*
Define global fallback character (or glyph) *c* as *contents*.

.shc Reset the soft hyphen character to **\[hy]**.

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

.shift *n*
In a macro definition, left-shift arguments by *n* positions.

.sizes *s1 s2 . . . sn* [**0**]
: Set available type sizes similarly to the **sizes** directive in a *DESC* file. Each *s*i is interpreted in units of scaled points (**z**).

.so *file*
: Replace the request's control line with the contents of *file*, "sourcing" it.

.soquiet *file*
: As **.so**, but no warning is emitted if *file* does not exist.

.sp
: Break and move the next text baseline down by one vee, or until springing a page location trap.

.sp *dist*
: Break and move the next text baseline down by *dist*, or until springing a page location trap (default scaling unit **v**). A negative *dist* will not reduce the position of the text baseline below zero. Prefixing *dist* with the | operator moves to a position relative to the page top for positive *N*, and the bottom if *N* is negative; in all cases, one line height (vee) is added to *dist*. *dist* is ignored inside a diversion.

.special
: Reset global list of special fonts to be empty.

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

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

.spreadwarn *N*
: Emit a **break** warning if the additional space inserted for each space between words in an adjusted output line is greater than or equal to *N*. A negative *N* is treated as 0. The default scaling unit is **m**. At startup, **.spreadwarn** is inactive and *N* is 3 m **.**

.ss *n*
: Set minimal inter-word spacing to *n* 12ths of current font's space width.

.ss *n m*
: As "**.ss** *n*", and set additional inter-sentence space to *m* 12ths of current font's space width.

.stringdown *stringvar*
: Replace each byte in the string named *stringvar* with its lowercase version.

.stringup *stringvar*
: Replace each byte in the string named *stringvar* with its uppercase version.

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

.substring *str start* [*end*]
: Replace the string named *str* with its substring bounded by the indices *start* and *end*, inclusive. Negative indices count backwards from the end of the string.

.sv
: As **.ne**, but save 1 v for output with **.os** request.

.sv *d*
: As **.ne**, but save distance *d* for later output with **.os** request (default scaling unit **v**).

.sy *command-line*
: Execute *command-line* with *system*(3). Unsafe request; disabled by default.

.ta *n1 n2 . . . n*n **T** *r1 r2 . . . r*n
: Set tabs at positions *n1*, *n2*, . . ., *n*n, then set tabs at *n*n+*m*×*r*n+*r1* through *n*n+*m*×*r*n+*r*n, where *m* increments from 0, 1, 2, . . . to the output line length. Each *n* argument can be prefixed with a "**+**" to place the tab stop *ni* at a distance relative to the previous, *n*(*i*−1). Each argument *ni* or *ri* can be suffixed with a letter to align text within the tab column bounded by tab stops *i* and *i*+1; "**L**" for left-aligned (the default), "**C**" for centered, and "**R**" for right-aligned.

.tag

.taga
: Reserved for internal use.

.tc
: Unset tab repetition character.

.tc *c*
: Set tab repetition character to *c* (default: none).

.ti ±*N*
: Temporarily indent next output line (default scaling unit **m**).

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

.tl **'***left***'***center***'***right***'**
: Format three-part title.

.tm *message*
: Write *message,* followed by a newline, to the standard error stream.

.tm1 *message*
: As **.tm**, but an initial neutral double quote in *message* is removed, allowing it to contain leading spaces.

.tmc *message*
: As **.tm1**, without emitting a newline.

.tr *abcd...*
: Translate ordinary or special characters *a* to *b*, *c* to *d*, and so on prior to output.

.trf *file*
: Transparently output the contents of *file.* Unlike **.cf**, invalid input characters in *file* are rejected.

.trin *abcd...*
: As **.tr**, except that **.asciify** ignores the translation when a diversion is interpolated.

.trnt *abcd...*
: As **.tr**, except that translations are suppressed in the argument to **\!**.

.troff
: Make the conditional expressions **t** true and **n** false.

.uf *font*
: Set underline font used by **.ul** to *font.*

.ul
: Underline (italicize in *troff* mode) the output of the next productive input line.

.ul *npl*
: Underline (italicize in *troff* mode) the output of the next *npl* productive input line. If *npl*=0, stop underlining.

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

.vpt
: Enable vertical position traps.

.vpt 0
: Disable vertical position traps.

.vs
: Change to previous vertical spacing.

.vs ±*N*
: Set vertical spacing to ±*N* (default scaling unit **p**).

.warn
: Enable all warning categories.

.warn 0
: Disable all warning categories.

.warn *n*
: Enable warnings in categories whose codes sum to *n*; see *troff* (1).

.warnscale *su*
: Set scaling unit used in certain warnings to *su* (one of **u**, **i**, **c**, **p**, or **P**; default: **i**).

.wh *vpos*
: Remove visible page location trap at *vpos* (default scaling unit **v**).

.wh *vpos name*
: Plant macro *name* as page location trap at *vpos* (default scaling unit **v**), removing any visible trap already there.

.while *cond-expr anything*
: Repeatedly execute *anything* unless and until *cond-expr* evaluates false.

.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*.

Escape sequence short reference

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

\"
: Comment. Everything up to the end of the line is ignored.

\#
: Comment. Everything up to and including the next newline is ignored.

*****s*
: Interpolate string with one-character name *s*.

\(st
: Interpolate string with two-character name *st*.

\[*string***]**
: Interpolate string with name *string* (of arbitrary length).

\[*string arg ...***]**
: Interpolate string with name *string* (of arbitrary length), taking *arg* ... as arguments.

\$0 Interpolate name by which currently executing macro was invoked.
\$*n* Interpolate macro or string parameter numbered *n* ($1 \le n \le 9$).
\$(*nn* Interpolate macro or string parameter numbered *nn* ($01 \le nn \le 99$).
\$[*nnn***]**
Interpolate macro or string parameter numbered *nnn* ($nnn \ge 1$).
\$* Interpolate concatenation of all macro or string parameters, separated by spaces.
\$@ Interpolate concatenation of all macro or string parameters, with each surrounded by double quotes and separated by spaces.
\$^ Interpolate concatenation of all macro or string parameters as if they were arguments to the **.ds** request.
\' is a synonym for **\[aa]**, the acute accent special character.
\` is a synonym for **\[ga]**, the grave accent special character.
\- is a synonym for **\[-]**, the minus sign special character.
_ is a synonym for **\[ul]**, the underrule special character.
\% Control hyphenation.
\! Transparent line. The remainder of the input line is interpreted (1) when the current diversion is read; or (2) if in the top-level diversion, by the postprocessor (if any).
\?*anything***\?**
Transparently embed *anything*, read in copy mode, in a diversion, or unformatted as an output comparand in a conditional expression.
*****space Move right one word space.
\~ Insert an unbreakable, adjustable space.
\0 Move right by the width of a numeral in the current font.
\| Move one-sixth em to the right on typesetters.
\^ Move one-twelfth em to the right on typesetters.
\& Interpolate a dummy character.
\) Interpolate a dummy character that is transparent to end-of-sentence recognition.
\/ Apply italic correction. Use between an immediately adjacent oblique glyph on the left and an upright glyph on the right.
\, Apply left italic correction. Use between an immediately adjacent upright glyph on the left and an oblique glyph on the right.
\: Non-printing break point (similar to **\%**, but never produces a hyphen glyph).
*****newline
Continue current input line on the next.
\{ Begin conditional input.
\} End conditional input.
\(*gl* Interpolate glyph with two-character name *gl*.
\[*glyph***]**
Interpolate glyph with name *glyph* (of arbitrary length).
\[*base-char comp* . . .**]**
Interpolate composite glyph constructed from *base-char* and each component *comp*.
\[char*nnn***]**
Interpolate glyph of eight-bit encoded character *nnn*, where $0 \le nnn \le 255$.
\[u*nnnn*[*n*[*n*]]**]**
Interpolate glyph of Unicode character with code point *nnnn*[*n*[*n*]] in uppercase hexadecimal.
\[u*base-char*[**_***combining-component*]. . .**]**
Interpolate composite glyph from Unicode character *base-char* and *combining-components*.
\a Interpolate a leader in copy mode.
\A'*anything***'**
Interpolate 1 if *anything* is a valid identifier, and 0 otherwise.
\b'*string***'**
Build bracket: pile a sequence of glyphs corresponding to each character in *string* vertically, and center it vertically on the output line.

\B'*anything***'**
Interpolate 1 if *anything* is a valid numeric expression, and 0 otherwise.

\c Continue output line at next input line.

\C'*glyph***'**
As **\[***glyph***]**, but compatible with other *troff* implementations.

\d Move downward ½ em on typesetters.

\D'*drawing-command***'**
See subsection "Drawing commands" below.

\e Interpolate the escape character.

\E As **\e**, but not interpreted in copy mode.

\fP Select previous font mounting position (abstract style or font); same as "**.ft**" or "**.ft P**".

\f*F* Select font mounting position, abstract style, or font with one-character name or one-digit position *F*. *F* cannot be **P**.

\f(*ft* Select font mounting position, abstract style, or font with two-character name or two-digit position *ft*.

\f[*font***]**
Select font mounting position, abstract style, or font with arbitrarily long name or position *font*. *font* cannot be **P**.

\f[] Select previous font mounting position (abstract style or font).

\F*f* Set default font family to that with one-character name *f*.

\F(*fm* Set default font family to that with two-character name *fm*.

\F[*fam***]**
Set default font family to that with arbitrarily long name *fam*.

\F[] Set default font family to previous value.

\g*r* Interpolate format of register with one-character name *r*.

\g(*rg* Interpolate format of register with two-character name *rg*.

\g[*reg***]**
Interpolate format of register with arbitrarily long name *reg*.

\h'*N***'**
Horizontally move the drawing position by *N* ems (or specified units); | may be used. Positive motion is rightward.

\H'*N***'**
Set height of current font to *N* scaled points (or specified units).

\k*r* Mark horizontal position in one-character register name *r*.

\k(*rg* Mark horizontal position in two-character register name *rg*.

\k[*reg***]**
Mark horizontal position in register with arbitrarily long name *reg*.

\l'*N*[*c*]**'**
Draw horizontal line of length *N* with character **c** (default: **\[ru]**; default scaling unit **m**).

\L'*N*[*c*]**'**
Draw vertical line of length *N* with character **c** (default: **\[br]**; default scaling unit **v**).

\m*c* Set stroke color to that with one-character name *c*.

\m(*cl* Set stroke color to that with two-character name *cl*.

\m[*color***]**
Set stroke color to that with arbitrarily long name *color*.

\m[] Restore previous stroke color.

\M*c* Set fill color to that with one-character name *c*.

\M(*cl* Set fill color to that with two-character name *cl*.

\M[*color***]**
Set fill color to that with arbitrarily long name *color*.

\M[] Restore previous fill color.

\n*r* Interpolate contents of register with one-character name *r*.

\n(*rg* Interpolate contents of register with two-character name *rg*.

\n[*reg***]**
Interpolate contents of register with arbitrarily long name *reg*.

\N'*n***'** Interpolate glyph with index *n* in the current font.

\o'*abc*...**'**
Overstrike centered glyphs of characters *a*, *b*, *c*, and so on.

\O0 At the outermost suppression level, disable emission of glyphs and geometric objects to the output driver.

\O1 At the outermost suppression level, enable emission of glyphs and geometric objects to the output driver.

\O2 At the outermost suppression level, enable glyph and geometric primitive emission to the output driver and write to the standard error stream the page number, four bounding box registers enclosing glyphs written since the previous **\O** escape sequence, the page offset, line length, image file name (if any), horizontal and vertical device motion quanta, and input file name.

\O3 Begin a nested suppression level.

\O4 End a nested suppression level.

\O[5*P file***]**
At the outermost suppression level, write the name *file* to the standard error stream at position *P*, which must be one of **l**, **r**, **c**, or **i**.

\p Break output line at next word boundary; adjust if applicable.

\r Move "in reverse" (upward) 1 em.

\R'*name* ±*N***'**
Set, increment, or decrement register *name* by *N*.

\s±*N* Set/increase/decrease the type size to/by *N* scaled points. *N* must be a single digit; 0 restores the previous type size. (In compatibility mode only, a non-zero *N* must be in the range 4–39.) Otherwise, as **.ps** request.

\s(±*N*
\s±**(***N*
Set/increase/decrease the type size to/by *N* scaled points; *N* is a two-digit number ≥1. As **.ps** request.

\s[±*N***]**
\s±**[***N***]**
\s'±*N***'**
\s±**'***N***'**
Set/increase/decrease the type size to/by *N* scaled points. As **.ps** request.

\S'*N***'**
Slant output glyphs by *N* degrees; the direction of text flow is positive.

\t Interpolate a tab in copy mode.

\u Move upward ½ em on typesetters.

\v'*N***'**
Vertically move the drawing position by *N* vees (or specified units); | may be used. Positive motion is downward.

\V*e* Interpolate contents of environment variable with one-character name *e*.

\V(*ev* Interpolate contents of environment variable with two-character name *ev*.

\V[*env***]**
Interpolate contents of environment variable with arbitrarily long name *env*.

\w'*anything***'**
Interpolate width of *anything*, formatted in a dummy environment.

\x'*N***'**
Increase vertical spacing of pending output line by *N* vees (or specified units; negative before, positive after).

\X' *anything* **'**
Write *anything* to *troff* output as a device control command. Within *anything*, the escape sequences **\&**, **\)**, **\%**, and **\:** are ignored; **** *space* and **\~** are converted to single space characters; and **** has its escape character stripped. So that the basic Latin subset of the Unicode character set can be reliably encoded in *anything,* the special character escape sequences **\–**, **\[aq]**, **\[dq]**, **\[ga]**, **\[ha]**, **\[rs]**, and **\[ti]** are mapped to basic Latin characters; see *groff_char*(7). For this transformation, character translations and special character definitions are ignored.

\Y *n* Write contents of macro or string *n* to *troff* output as a device control command.

\Y(*nm* Write contents of macro or string *nm* to *troff* output as a device control command.

\Y[*name* **]**
Write contents of macro or string *name* to *troff* output as a device control command.

\z *c* Format character *c* with zero width—without advancing the drawing position.

\Z' *anything* **'**
Save the drawing position, format *anything*, then restore it.

Drawing commands

Drawing commands direct the output device to render geometrical objects rather than glyphs. Specific devices may support only a subset, or may feature additional ones; consult the man page for the output driver in use. Terminal devices in particular implement almost none.

Rendering starts at the drawing position; when finished, the drawing position is left at the rightmost point of the object, even for closed figures, except where noted. GNU *troff* draws stroked (outlined) objects with the stroke color, and shades filled ones with the fill color. See section "Colors" above. Coordinates *h* and *v* are horizontal and vertical motions relative to the drawing position or previous point in the command. The default scaling unit for horizontal measurements (and diameters of circles) is **m**; for vertical ones, **v**.

Circles, ellipses, and polygons can be drawn stroked or filled. These are independent properties; if you want a filled, stroked figure, you must draw the same figure twice using each drawing command. A filled figure is always smaller than an outlined one because the former is drawn only within its defined area, whereas strokes have a line thickness (set with **\D't'**).

\D'~ *h1 v1 . . . hn vn***'**
Draw B-spline to each point in sequence, leaving drawing position at (*hn*, *vn*).

\D'a *hc vc h v***'**
Draw circular arc centered at (*hc*, *vc*) counterclockwise from the drawing position to a point (*h*, *v*) relative to the center. (*hc*, *vc*) is adjusted to the point nearest the perpendicular bisector of the arc's chord.

\D'c *d***'** Draw circle of diameter *d* with its leftmost point at the drawing position.

\D'C *d***'**
As **\D'C'**, but the circle is filled.

\D'e *h v***'**
Draw ellipse of width *h* and height *v* with its leftmost point at the drawing position.

\D'E *h v***'**
As **\D'e'**, but the ellipse is filled.

\D'l *h v***'**
Draw line from the drawing position to (*h*, *v*).

\D'p *h1 v1 . . . hn vn***'**
Draw polygon with vertices at drawing position and each point in sequence. GNU *troff* closes the polygon by drawing a line from (*hn*, *vn*) back to the initial drawing position. Afterward, the drawing position is left at (*hn*, *vn*).

\D'P *h1 v1 . . . hn vn***'**
As **\D'p'**, but the polygon is filled.

\D't *n***'** Set stroke thickness of geometric objects to to *n* basic units. A zero *n* selects the minimal supported thickness. A negative *n* selects a thickness proportional to the type size; this is the default.

Device control commands

The **.device** and **.devicem** requests, and **\X** and **\Y** escape sequences, enable documents to pass information directly to a postprocessor. These are useful for exercising device-specific capabilities that the *groff* language does not abstract or generalize; such functions include the embedding of hyperlinks and image files. Device-specific functions are documented in each output driver's man page.

Strings

groff supports strings primarily for user convenience. Conventionally, if one would define a macro only to interpolate a small amount of text, without invoking requests or calling any other macros, one defines a string instead. Only one string is predefined by the language.

*[**.T**] Contains the name of the output device (for example, "**utf8**" or "**pdf**").

The **.ds** request creates a string with a specified name and contents. If the identifier named by **.ds** already exists as an alias, the target of the alias is redefined. If **.ds** is called with only one argument, the named string becomes empty. Otherwise, *troff* stores the remainder of the control line in copy mode; see subsection "Copy mode" below.

The ***** escape sequence dereferences a string's name, interpolating its contents. If the name does not exist, it is defined as empty, nothing is interpolated, and a warning in category "**mac**" is emitted. See section "Warnings" in *troff*(1). The bracketed interpolation form accepts arguments that are handled as macro arguments are; see section "Calling macros" above. In contrast to macro calls, however, if a closing bracket **]** occurs in a string argument, that argument must be enclosed in double quotes. ***** is interpreted even in copy mode. When defining strings, argument interpolations must be escaped if they are to reference parameters from the calling context; see section "Parameters" below.

An initial neutral double quote **"** in the string contents is stripped to allow embedding of leading spaces. Any other **"** is interpreted literally, but it is wise to use the special character escape sequence **\[dq]** instead if the string might be interpolated as part of a macro argument; see section "Calling macros" above. Strings are not limited to a single input line of text. *****_newline_ works just as it does elsewhere. The resulting string is stored *without* the newlines. Care is therefore required when interpolating strings while filling is disabled. It is not possible to embed a newline in a string that will be interpreted as such when the string is interpolated. To achieve that effect, use ***** to interpolate a macro instead.

The **.as** request is similar to **.ds** but appends to a string instead of redefining it. If **.as** is called with only one argument, no operation is performed (beyond dereferencing the string).

Because strings are similar to macros, they too can be defined to suppress AT&T *troff* compatibility mode enablement when interpolated; see section "Compatibility mode" below. The **.ds1** request defines a string that suspends compatibility mode when the string is later interpolated. **.as1** is likewise similar to **.as**, with compatibility mode suspended when the appended portion of the string is later interpolated.

Caution: Unlike other requests, the second argument to these requests consumes the remainder of the input line, including trailing spaces. Ending string definitions (and appendments) with a comment, even an empty one, prevents unwanted space from creeping into them during source document maintenance.

Several requests exist to perform rudimentary string operations. Strings can be queried (**.length**) and modified (**.chop**, **.substring**, **.stringup**, **.stringdown**), and their names can be manipulated through renaming, removal, and aliasing (**.rn**, **.rm**, **.als**).

When a request, macro, string, or diversion is aliased, redefinitions and appendments "write through" alias names. To replace an alias with a separately defined object, you must use the **rm** request on its name first.

Registers

In the *roff* language, numbers can be stored in *registers*. Many built-in registers exist, supplying anything from the date to details of formatting parameters. You can also define your own. See section "Identifiers" above for information on constructing a valid name for a register.

Define registers and update their values with the **nr** request or the **\R** escape sequence.

Registers can also be incremented or decremented by a configured amount at the time they are interpolated. The value of the increment is specified with a third argument to the **.nr** request, and a special interpolation

syntax, **\n**± is used to alter and then retrieve the register's value. Together, these features are called *auto-increment*. (A negative auto-increment can be considered an "auto-decrement".)

Many predefined registers are available. In the following presentation, the register interpolation syntax **\n[***name***]** is used to refer to a register *name* to clearly distinguish it from a string or request *name*. The register name space is separate from that used for requests, macros, strings, and diversions. Bear in mind that the symbols **\n[]** are *not* part of the register name.

Read-only registers

Predefined registers whose identifiers start with a dot are read-only. Many are Boolean-valued. Some are string-valued, meaning that they interpolate text. A register name (without the dot) is often associated with a request of the same name; exceptions are noted.

\n[**.$**] Count of arguments passed to currently interpolated macro or string.
\n[**.a**] Amount of extra post-vertical line space; see **\x**.
\n[**.A**] Approximate output is being formatted (Boolean-valued); see *troff* **−a** option.
\n[**.b**] Font emboldening offset; see **.bd**.
\n[**.br**] The normal control character was used to call the currently interpolated macro (Boolean-valued).
\n[**.c**] Input line number; see **.lf** and register "**c.**".
\n[**.C**] Compatibility mode is enabled (Boolean-valued); see **.cp**. Always false when processing **.do**; see register **.cp**.
\n[**.cdp**] Depth of last glyph formatted in the environment; positive if glyph extends below the baseline.
\n[**.ce**] Count of output lines remaining to be centered.
\n[**.cht**] Height of last glyph formatted in the environment; positive if glyph extends above the baseline.
\n[**.color**] Color output is enabled (Boolean-valued).
\n[**.cp**] Within **.do**, the saved value of compatibility mode; see register **.C**.
\n[**.csk**] Skew of the last glyph formatted in the environment; skew is how far to the right of the center of a glyph the center of an accent over that glyph should be placed.
\n[**.d**] Vertical drawing position in diversion.
\n[**.ev**] Name of environment (string-valued).
\n[**.f**] Mounting position of selected font; see **.ft** and **\f**.
\n[**.F**] Name of input file (string-valued); see **.lf**.
\n[**.fam**] Name of default font family (string-valued).
\n[**.fn**] Resolved name of selected font (string-valued); see **.ft** and **\f**.
\n[**.fp**] Next non-zero free font mounting position index.
\n[**.g**] Always true in GNU *troff* (Boolean-valued).
\n[**.h**] Text baseline high-water mark on page or in diversion.
\n[**.H**] Horizontal motion quantum of output device in basic units.
\n[**.height**] Font height; see **\H**.
\n[**.hla**] Hyphenation language in environment (string-valued).
\n[**.hlc**] Count of immediately preceding consecutive hyphenated lines in environment.
\n[**.hlm**] Maximum quantity of consecutive hyphenated lines allowed in environment.
\n[**.hy**] Automatic hyphenation mode in environment.
\n[**.hym**] Hyphenation margin in environment.
\n[**.hys**] Hyphenation space adjustment threshold in environment.
\n[**.i**] Indentation amount; see **.in**.
\n[**.in**] Indentation amount applicable to the pending output line; see **.ti**.
\n[**.int**] Previous output line was "interrupted" or continued with **\c** (Boolean-valued).
\n[**.j**] Adjustment mode encoded as an integer; see **.ad** and **.na**. Do not interpret or perform arithmetic on its value.
\n[**.k**] Horizontal drawing position relative to indentation.

\n[**.kern**] Pairwise kerning is enabled (Boolean-valued).
\n[**.l**] Line length; see **.ll**.
\n[**.L**] Line spacing; see **.ls**.
\n[**.lg**] Ligature mode.
\n[**.linetabs**] Line-tabs mode is enabled (Boolean-valued).
\n[**.ll**] Line length applicable to the pending output line.
\n[**.lt**] Title length.
\n[**.m**] Stroke color (string-valued); see **.gcolor** and **\m**. Empty if the stroke color is the default.
\n[**.M**] Fill color (string-valued); see **.fcolor** and **\M**. Empty if the fill color is the default.
\n[**.n**] Length of formatted output on previous output line.
\n[**.ne**] Amount of vertical space required by last **.ne** that caused a trap to be sprung; also see register **.trunc**.
\n[**.nm**] Output line numbering is enabled (Boolean-valued).
\n[**.nn**] Count of output lines remaining to have numbering suppressed.
\n[**.ns**] No-space mode is enabled (Boolean-valued).
\n[**.o**] Page offset; see **.po**.
\n[**.O**] Output suppression nesting level; see **\O**.
\n[**.p**] Page length; see **.pl**.
\n[**.P**] The page is selected for output (Boolean-valued); see *troff* **−o** option.
\n[**.pe**] Page ejection is in progress (Boolean-valued).
\n[**.pn**] Number of the next page.
\n[**.ps**] Type size in scaled points.
\n[**.psr**] Most recently requested type size in scaled points; see **.ps** and **\s**.
\n[**.pvs**] Post-vertical line spacing.
\n[**.R**] Count of available unused registers; always 10,000 in GNU *troff*.
\n[**.rj**] Count of lines remaining to be right-aligned.
\n[**.s**] Type size in points as a decimal fraction (string-valued); see **.ps** and **\s**.
\n[**.slant**] Slant of font in degrees; see **\S**.
\n[**.sr**] Most recently requested type size in points as a decimal fraction (string-valued); see **.ps** and **\s**.
\n[**.ss**] Size of minimal inter-word space in twelfths of the space width of the selected font.
\n[**.sss**] Size of additional inter-sentence space in twelfths of the space width of the selected font.
\n[**.sty**] Selected abstract style (string-valued); see **.ft** and **\f**.
\n[**.t**] Distance to next vertical position trap; see **.wh** and **.ch**.
\n[**.T**] An output device was explicitly selected (Boolean-valued); see *troff* **−T** option.
\n[**.tabs**] Representation of tab settings suitable for use as argument to **.ta** (string-valued).
\n[**.trunc**] Amount of vertical space truncated by the most recently sprung vertical position trap, or, if the trap was sprung by an **.ne**, minus the amount of vertical motion produced by **.ne**; also see register **.ne**.
\n[**.u**] Filling is enabled (Boolean-valued); see **.fi** and **.nf**.
\n[**.U**] Unsafe mode is enabled (Boolean-valued); see *troff* **−U** option.
\n[**.v**] Vertical line spacing; see **.vs**.
\n[**.V**] Vertical motion quantum of the output device in basic units.
\n[**.vpt**] Vertical position traps are enabled (Boolean-valued).
\n[**.w**] Width of previous glyph formatted in the environment.
\n[**.warn**] Sum of the numeric codes of enabled warning categories.
\n[**.x**] Major version number of the running *troff* formatter.
\n[**.y**] Minor version number of the running *troff* formatter.
\n[**.Y**] Revision number of the running *troff* formatter.
\n[**.z**] Name of diversion (string-valued). Empty if output is directed to the top-level diversion.
\n[**.zoom**] Zoom multiplier of current font (in thousandths; zero if no magnification); see **.fzoom**.

Writable predefined registers

Several registers are predefined but also modifiable; some are updated upon interpretation of certain requests or escape sequences. Date- and time-related registers are set to the local time as determined by

localtime(3) when the formatter launches. This initialization can be overridden by *SOURCE_DATE_EPOCH* and *TZ*; see section "Environment" of *groff* (1).

\n[**$$**]	Process ID of *troff*.
\n[**%**]	Page number.
\n[**c.**]	Input line number.
\n[**ct**]	Union of character types of each glyph rendered into dummy environment by **\w**.
\n[**dl**]	Width of last closed diversion.
\n[**dn**]	Height of last closed diversion.
\n[**dw**]	Day of the week (1–7; 1 is Sunday).
\n[**dy**]	Day of the month (1–31).
\n[**hours**]	Count of hours elapsed since midnight (0–23).
\n[**hp**]	Horizontal drawing position relative to start of input line.
\n[**llx**]	Lower-left *x* coordinate (in PostScript units) of PostScript image; see **.psbb**.
\n[**lly**]	Lower-left *y* coordinate (in PostScript units) of PostScript image; see **.psbb**.
\n[**ln**]	Output line number; see **.nm**.
\n[**lsn**]	Count of leading spaces on input line.
\n[**lss**]	Amount of horizontal space corresponding to leading spaces on input line.
\n[**minutes**]	Count of minutes elapsed in the hour (0–59).
\n[**mo**]	Month of the year (1–12).
\n[**nl**]	Vertical drawing position.
\n[**opmaxx**]	
\n[**opmaxy**]	
\n[**opminx**]	
\n[**opminy**]	These four registers mark the top left- and bottom right-hand corners of a rectangle encompassing all formatted output on the page. They are reset to −1 by **\O0** or **\O1**.
\n[**rsb**]	As register **sb**, adding maximum glyph height to measurement.
\n[**rst**]	As register **st**, adding maximum glyph depth to measurement.
\n[**sb**]	Maximum displacement of text baseline below its original position after rendering into dummy environment by **\w**.
\n[**seconds**]	Count of seconds elapsed in the minute (0–60).
\n[**skw**]	Skew of last glyph rendered into dummy environment by **\w**.
\n[**slimit**]	The maximum depth of *troff*'s internal input stack. If ≤0, there is no limit: recursion can continue until available memory is exhausted. The default is 1,000.
\n[**ssc**]	Subscript correction of last glyph rendered into dummy environment by **\w**.
\n[**st**]	Maximum displacement of text baseline above its original position after rendering into dummy environment by **\w**.
\n[**systat**]	Return value of *system()* function; see **.sy**.
\n[**urx**]	Upper-right *x* coordinate (in PostScript units) of PostScript image; see **.psbb**.
\n[**ury**]	Upper-right *y* coordinate (in PostScript units) of PostScript image; see **.psbb**.
\n[**year**]	Gregorian year.
\n[**yr**]	Gregorian year minus 1900.

Using fonts

In digital typography, a *font* is a collection of characters in a specific typeface that a device can render as glyphs at a desired size. (Terminals and some output devices have fonts that render at only one or two sizes. As examples of the latter, take the *groff* **lj4** device's Lineprinter, and **lbp**'s Courier and Elite faces.) A *roff* formatter can change typefaces at any point in the text. The basic faces are a set of *styles* combining upright and slanted shapes with normal and heavy stroke weights: "**R**", "**I**", "**B**", and "**BI**"—these stand for *roman, bold, italic,* and *bold-italic.* For linguistic text, GNU *troff* groups typefaces into *families* containing each of these styles. (Font designers prepare families such that the styles share esthetic properties.) A *text font* is thus often a family combined with a style, but it need not be: consider the **ps** and **pdf** devices' **ZCMI** (Zapf Chancery Medium italic)—often, no other style of Zapf Chancery Medium is provided. On typesetting devices, at least one *special font* is available, comprising *unstyled* glyphs for mathematical operators and other purposes.

Like AT&T *troff,* GNU *troff* does not itself load or manipulate a digital font file; instead it works with a *font description file* that characterizes it, including its glyph repertoire and the *metrics* (dimensions) of each glyph. This information permits the formatter to accurately place glyphs with respect to each other. Before using a font description, the formatter associates it with a *mounting position,* a place in an ordered list of available typefaces. So that a document need not be strongly coupled to a specific font family, in GNU *troff* an output device can associate a style in the abstract sense with a mounting position. Thus the default family can be combined with a style dynamically, producing a *resolved font name.*

Fonts often have trademarked names, and even Free Software fonts can require renaming upon modification. *groff* maintains a convention that a device's serif font family is given the name **T** ("Times"), its sans-serif family **H** ("Helvetica"), and its monospaced family **C** ("Courier"). Historical inertia has driven *groff*'s font identifiers to short uppercase abbreviations of font names, as with **TR**, **TB**, **TI**, **TBI**, and a special font **S**.

The default family used with abstract styles can be changed at any time; initially, it is **T**. Typically, abstract styles are arranged in the first four mounting positions in the order shown above. The default mounting position, and therefore style, is always **1** (**R**). By issuing appropriate formatter instructions, you can override these defaults before your document writes its first glyph.

Terminal output devices cannot change font families and lack special fonts. They support style changes by overstriking, or by altering ISO 6429/ECMA-48 *graphic renditions* (character cell attributes).

Hyphenation

When filling, *groff* hyphenates words as needed at user-specified and automatically determined hyphenation points. Explicitly hyphenated words such as "mother-in-law" are always eligible for breaking after each of their hyphens. The hyphenation character **\%** and non-printing break point **\:** escape sequences may be used to control the hyphenation and breaking of individual words. The **.hw** request sets user-defined hyphenation points for specified words at any subsequent occurrence. Otherwise, *groff* determines hyphenation points automatically by default.

Several requests influence automatic hyphenation. Because conventions vary, a variety of hyphenation modes is available to the **.hy** request; these determine whether hyphenation will apply to a word prior to breaking a line at the end of a page (more or less; see below for details), and at which positions within that word automatically determined hyphenation points are permissible. The default is "**1**" for historical reasons, but this is not an appropriate value for the English hyphenation patterns used by *groff*; localization macro files loaded by *troffrc* and macro packages often override it.

0 disables hyphenation.

1 enables hyphenation except after the first and before the last character of a word.

The remaining values "imply" **1**; that is, they enable hyphenation under the same conditions as "**.hy 1**", and then apply or lift restrictions relative to that basis.

2 disables hyphenation of the last word on a page. (Hyphenation is prevented if the next page location trap is closer to the vertical drawing position than the next text baseline would be. See section "Traps" below.)

4 disables hyphenation before the last two characters of a word.

8 disables hyphenation after the first two characters of a word.

16 enables hyphenation before the last character of a word.

32 enables hyphenation after the first character of a word.

Apart from value 2, restrictions imposed by the hyphenation mode are *not* respected for words whose hyphenations have been specified with the hyphenation character ("**\%**" by default) or the **.hw** request.

Nonzero values are additive. For example, mode 12 causes *groff* to hyphenate neither the last two nor the first two characters of a word. Some values cannot be used together because they contradict; for instance, values 4 and 16, and values 8 and 32. As noted, it is superfluous to add 1 to any non-zero even mode.

The places within a word that are eligible for hyphenation are determined by language-specific data (**.hla**, **.hpf**, and **.hpfa**) and lettercase relationships (**.hcode** and **.hpfcode**). Furthermore, hyphenation of a word might be suppressed due to a limit on consecutive hyphenated lines (**.hlm**), a minimum line length threshold (**.hym**), or because the line can instead be adjusted with additional inter-word space (**.hys**).

Localization

The set of hyphenation patterns is associated with the hyphenation language set by the **.hla** request. The **.hpf** request is usually invoked by a localization file loaded by the *troffrc* file. *groff* provides localization files for several languages; see *groff_tmac*(5).

Writing macros

The **.de** request defines a macro named for its argument. If that name already exists as an alias, the target of the alias is redefined; see section "Strings" above. *troff* enters "copy mode" (see below), storing subsequent input lines as the definition. If the optional second argument is not specified, the definition ends with the control line "**..**" (two dots). Alternatively, a second argument names a macro whose call syntax ends the definition; this "end macro" is then called normally. Spaces or tabs are permitted after the first control character in the line containing this ending token, but a tab immediately after the token prevents its recognition as the end of a macro definition. Macro definitions can be nested if they use distinct end macros or if their ending tokens are sufficiently escaped. An end macro need not be defined until it is called. This fact enables a nested macro definition to begin inside one macro and end inside another.

Variants of **.de** disable compatibility mode and/or indirect the names of the macros specified for definition or termination: these are **.de1**, **.dei**, and **.dei1**. Append to macro definitions with **.am**, **.am1**, **.ami**, and **.ami1**. The **.als**, **.rm**, and **.rn** requests create an alias of, remove, and rename a macro, respectively. **.return** stops the execution of a macro immediately, returning to the enclosing context.

Parameters

Macro call and string interpolation parameters can be accessed using escape sequences starting with "**\$**". The **\n[.$]** read-only register stores the count of parameters available to a macro or string; its value can be changed by the **.shift** request, which dequeues parameters from the current list. The **\$0** escape sequence interpolates the name by which a macro was called. Applying string interpolation to a macro does not change this name.

Copy mode

When *troff* processes certain requests, most importantly those which define or append to a macro or string, it does so in *copy mode*: it copies the characters of the definition into a dedicated storage region, interpolating the escape sequences **\n**, **\g**, **\$**, *****, **\V**, and **\?** normally; interpreting *\newline* immediately; discarding comments **\"** and **\#**; interpolating the current leader, escape, or tab character with **\a**, **\e**, and **\t**, respectively; and storing all other escape sequences in an encoded form. The complement of copy mode—a *roff* formatter's behavior when not defining or appending to a macro, string, or diversion—where all macros are interpolated, requests invoked, and valid escape sequences processed immediately upon recognition, can be termed *interpretation mode*.

The escape character, **** by default, can escape itself. This enables you to control whether a given **\n**, **\g**, **\$**, *****, **\V**, or **\?** escape sequence is interpreted at the time the macro containing it is defined, or later when the macro is called.

You can think of **** as a "delayed" backslash; it is the escape character followed by a backslash from which the escape character has removed its special meaning. Consequently, **** is not an escape sequence in the usual sense. In any escape sequence *\X* that *troff* does not recognize, the escape character is ignored and *X* is output. An unrecognized escape sequence causes a warning in category "**escape**", with two exceptions, **** being one. The other is **\.**, which escapes the control character. It is used to permit nested macro definitions to end without a named macro call to conclude them. Without a syntax for escaping the control character, this would not be possible. *roff* documents should not use the **** or **\.** character sequences outside of copy mode; they serve only to obfuscate the input. Use **\e** to represent the escape character, **\[rs]** to obtain a backslash glyph, and **\&** before **.** and **'** where *troff* expects them as control characters if you mean to use them literally.

Macro definitions can be nested to arbitrary depth. In "\\", each escape character is interpreted twice—once in copy mode, when the macro is defined, and once in interpretation mode, when the macro is called. This fact leads to exponential growth in the quantity of escape characters required to delay interpolation of **\n**, **\g**, **\$**, *****, **\V**, and **\?** at each nesting level. An alternative is to use **\E**, which represents an escape character that is not interpreted in copy mode. Because **\.** is not a true escape sequence, we can't use **\E** to keep "**..**" from ending a macro definition prematurely. If the multiplicity of backslashes complicates maintenance, use end macros.

Traps

Traps are locations in the output, or conditions on the input that, when reached or fulfilled, call a specified macro. A *vertical position trap* calls a macro when the formatter's vertical drawing position reaches or passes, in the downward direction, a certain location on the output page or in a diversion. Its applications include setting page headers and footers, body text in multiple columns, and footnotes. These traps can occur at a given location on the page (**.wh**, **.ch**); at a given location in the current diversion (**.dt**)—together, these are known as vertical position traps, which can be disabled and re-enabled (**.vpt**).

A diversion is not formatted in the context of a page, so it lacks page location traps; instead it can have a *diversion trap.* There can exist at most one such vertical position trap per diversion.

Other kinds of trap can be planted at a blank line (**.blm**); at a line with leading space characters (**.lsm**); after a certain number of productive input lines (**.it**, **.itc**); or at the end of input (**.em**). Macros called by traps are passed no arguments. Setting a trap is also called *planting* one. It is said that a trap is *sprung* if its condition is fulfilled.

Registers associated with trap management include vertical position trap enablement status (**\n[.vpt]**), distance to the next trap (**\n[.t]**), amount of needed (**.ne**-requested) space that caused the most recent vertical position trap to be sprung (**\n[.ne]**), amount of needed space truncated from the amount requested (**\n[.trunc]**), page ejection status (**\n[.pe]**), and leading space count (**\n[.lsn]**) with its corresponding amount of motion (**\n[.lss]**).

Page location traps

A *page location trap* is a vertical position trap that applies to the page; that is, to undiverted output. Many can be present; manage them with the **wh** and **ch** requests. Non-negative page locations given to these requests set the trap relative to the top of the page; negative values set the trap relative to the bottom of the page. It is not possible to plant a trap less than one basic unit from the page bottom: a location of "−0" is interpreted as "0", the top of the page. An existing *visible* trap (see below) at the same location is removed; this is **.wh**'s sole function if its second argument is missing.

A trap is sprung only if it is *visible,* meaning that its location is reachable on the page and it is not hidden by another trap at the same location already planted there. (A trap planted at "20i" or "−30i" will not be sprung on a page of length "11i".)

A trap above the top or at or below the bottom of the page can be made visible by either moving it into the page area or increasing the page length so that the trap is on the page. Negative trap values always use the *current* page length; they are not converted to an absolute vertical position. Use **.ptr** to dump page location traps to the standard error stream; their positions are reported in basic units.

The implicit page trap

An *implicit page trap* always exists in the top-level diversion; it works like a trap in some ways but not others. Its purpose is to eject the current page and start the next one. It has no name, so it cannot be moved or deleted with **wh** or **ch** requests. You cannot hide it by placing another trap at its location, and can move it only by redefining the page length with **.pl**. Its operation is suppressed when vertical page traps are disabled with the **vpt** request.

Diversions

In *roff* systems it is possible to format text as if for output, but instead of writing it immediately, one can *divert* the formatted text into a named storage area. It is retrieved later by specifying its name after a control character. The same name space is used for such *diversions* as for strings and macros; see section "Identifiers" above. Such text is sometimes said to be "stored in a macro", but this coinage obscures the important distinction between macros and strings on one hand and diversions on the other; the former store

unformatted input text, and the latter capture *formatted* output. Diversions also do not interpret arguments. Applications of diversions include "keeps" (preventing a page break from occurring at an inconvenient place by forcing a set of output lines to be set as a group), footnotes, tables of contents, and indices. For orthogonality it is said that GNU *troff* is in the *top-level diversion* if no diversion is active (that is, formatted output is being "diverted" immediately to the output device.

Dereferencing an undefined diversion will create an empty one of that name and cause a warning in category **mac** to be emitted. (see section "Warnings" in *troff* (1)). A diversion does not exist for the purpose of testing with the **d** conditional operator until its initial definition ends (see subsection "Conditional expressions" above).

The **di** request creates a diversion, including any partially collected line. **da** appends to a diversion, creating one if it does not already exist. If the diversion's name already exists as an alias, the target of the alias is replaced or appended to; see section "Strings" above. **box** and **boxa** works similarly, but ignore partially collected lines. Call any of these macros again without an argument to end the diversion.

Diversions can be nested. The registers **.d**, **.z**, **dn**, and **dl** report information about the current (or last closed) diversion. **.h** is meaningful in diversions, including the top level.

The **\!** and **\?** escape sequences and **output** request escape from a diversion, the first two to the enclosing level and the last to the top level. This facility is termed *transparent embedding*.

The **asciify** and **unformat** requests reprocess diversions.

Punning names

Macros, strings, and diversions share a name space; see section "Identifiers" above. Internally, the same mechanism is used to store them. You can thus call a macro with string interpolation syntax and vice versa. Interpolating a string does not hide existing macro arguments. The sequence **** can be placed at the end of a line in a macro definition or, within a macro definition, immediately after the interpolation of a macro as a string to suppress the effect of a newline.

Environments

Environments store most of the parameters that control text processing. A default environment named "**0**" exists when *troff* starts up; it is modified by formatting-related requests and escape sequences.

You can create new environments and switch among them. Only one is current at any given time. Active environments are managed using a *stack,* a data structure supporting "push" and "pop" operations. The current environment is at the top of the stack. The same environment name can be pushed onto the stack multiple times, possibly interleaved with others. Popping the environment stack does not destroy the current environment; it remains accessible by name and can be made current again by pushing it at any time. Environments cannot be renamed or deleted, and can only be modified when current. To inspect the environment stack, use the **pev** request; see section "Debugging" below.

Environments store the following information.

- a partially collected line, if any
- data about the most recently output glyph and line (registers **.cdp**, **.cht**, **.csk**, **.n**, **.w**)
- typeface parameters (size, family, style, height and slant, inter-word and inter-sentence space sizes)
- page parameters (line length, title length, vertical spacing, line spacing, indentation, line numbering, centering, right-alignment, underlining, hyphenation parameters)
- filling enablement; adjustment enablement and mode
- tab stops; tab, leader, escape, control, no-break control, hyphenation, and margin characters
- input line traps
- stroke and fill colors

The **ev** request pushes to and pops from the environment stack, while **evc** copies a named environment's contents to the current one.

Underlining

In *RUNOFF* (see *roff* (7)), underlining, even of lengthy passages, was straightforward because only fixed-pitch printing devices were targeted. Typesetter output posed a greater challenge. There exists a *groff* request **.ul** (see above) that underlines subsequent source lines on terminal devices, but on typesetters, it selects an italic font style instead. The *ms* macro package (see *groff_ms*(7)) offers a macro **.UL**, but it too produces the desired effect only on typesetters, and has other limitations.

One could adapt *ms*'s approach to the construction of a macro as follows.

```
.de UNDERLINE
. ie n \\$1\f[I]\\$2\f[P]\\$3
. el \\$1\Z'\\$2'\v'.25m'\D'l \w'\\$2'u 0'\v'-.25m'\\$3
..
```

If *doclifter*(1) makes trouble, change the macro name **UNDERLINE** into some 2-letter word, like **Ul**. Moreover, change the form of the font selection escape sequence from **\f[P]** to **\fP**.

Underlining without macro definitions

If one does not want to use macro definitions, e.g., when *doclifter* gets lost, use the following.

```
.ds u1 before
.ds u2 in
.ds u3 after
.ie n \*[u1]\f[I]\*[u2]\f[P]\*[u3]
.el \*[u1]\Z'\*[u2]'\v'.25m'\D'l \w'\*[u2]'u 0'\v'-.25m'\*[u3]
```

When using *doclifter*, it might be necessary to change syntax forms such as **\[xy]** and ***[xy]** to those supported by AT&T *troff*: ***(xy** and **\(xy**, and so on.

Then these lines could look like

```
.ds u1 before
.ds u2 in
.ds u3 after
.ie n \*[u1]\fI\*(u2\fP\*(u3
.el \*(u1\Z'\*(u2'\v'.25m'\D'l \w'\*(u2'u 0'\v'-.25m'\*(u3
```

The result looks like

```
before in after
```

Underlining by overstriking with \(ul

The **\z** escape sequence writes a glyph without advancing the drawing position, enabling overstriking. Thus, **\z***c***\(ul** formats *c* with an underrule glyph on top of it. Video terminals implement the underrule by setting a character cell's underline attribute, so this technique works in both *nroff* and *troff* modes.

Long words may then look intimidating in the input; a clarifying approach might be to use the input line continuation escape sequence *****newline* to place each underlined character on its own input line. Thus,

```
.nf
\&\fB: ${\fIvar\fR\c
\zo\(ul\
\zp\(ul\c
\&\fIvalue\fB}
.fi
```

produces

: ${*var***:-***value***}**

as output.

Compatibility mode

The differences between the *roff* language recognized by GNU *troff* and that of AT&T *troff*, as well as the device, font, and device-independent intermediate output formats described by CSTR #54 are documented in *groff_diff* (7). *groff* provides an AT&T compatibility mode. The **.cp** request and registers **.C** and **.cp** set and test the enablement of this mode.

Debugging

Preprocessors use the **.lf** request to preserve the identities of line numbers and names of input files. *groff* emits a variety of error diagnostics and supports several categories of warning; the output of these can be selectively suppressed with **.warn** (and see the **−E**, **−w**, and **−W** options of *troff* (1)). A trace of the formatter's input processing stack can be emitted when errors or warnings occur by means of *troff* (1)'s **−b** option, or produced on demand with the **.backtrace** request. **.tm**, **.tmc**, and **.tm1** can be used to emit customized diagnostic messages or for instrumentation while troubleshooting. **.ex** and **.ab** cause early termination with successful and error exit codes respectively, to halt further processing when continuing would be fruitless. Examine the state of the formatter with requests that write lists of defined names—macros, strings, and diversions—(**.pm**); environments (**.pev**), registers (**.pnr**), and page location traps (**.ptr**) to the standard error stream.

Authors

This document was written by by Trent A. Fisher, Werner Lemberg, and G. Branden Robinson ⟨g.branden .robinson@gmail.com⟩. Section "Underlining" was primarily written by Bernd Warken ⟨groff−bernd .warken−72@web.de⟩.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54, widely called simply "CSTR #54", documents the language, device and font description file formats, and device-independent output format referred to collectively in *groff* documentation as "AT&T *troff*".

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97 (CSTR #97), provides additional insights into the device and font description file formats and device-independent output format.

groff (1)
: is the preferred interface to the *groff* system; it manages the pipeline that carries a source document through preprocessors, the *troff* formatter, and an output driver to viewable or printable form. It also exhaustively lists the man pages provided with the GNU *roff* system.

groff_char(7)
: discusses character encoding issues, escape sequences that produce glyphs, and enumerates *groff*'s predefined special character escape sequences.

groff_diff (7)
: covers differences between the GNU *troff* formatter, its device and font description file formats, its device-independent output format, and those of AT&T *troff*, whose design it reimplements.

groff_font(5)
: describes the formats of the files that describe devices (*DESC*) and fonts.

groff_tmac(5)
: surveys macro packages provided with *groff*, describes how documents can take advantage of them, offers guidance on writing macro packages and using diversions, and includes historical information on macro package naming conventions.

roff (7) presents a detailed history of *roff* systems and summarizes concepts common to them.