ADDFTINFO

NAME

addftinfo − add information to troff font files for use with groff

SYNOPSIS

addftinfo [**−v**] [−*param value*...] *res unitwidth font*

DESCRIPTION

addftinfo reads a troff font file and adds some additional font-metric information that is used by the groff system. The font file with the information added is written on the standard output. The information added is guessed using some parametric information about the font and assumptions about the traditional troff names for characters. The main information added is the heights and depths of characters. The *res* and *unitwidth* arguments should be the same as the corresponding parameters in the DESC file; *font* is the name of the file describing the font; if *font* ends with **I** the font will be assumed to be italic.

OPTIONS

−v prints the version number.

All other options changes one of the parameters that is used to derive the heights and depths. Like the existing quantities in the font file, each *value* is in inches/*res* for a font whose point size is *unitwidth*. *param* must be one of:

x-height
The height of lowercase letters without ascenders such as x.

fig-height
The height of figures (digits).

asc-height
The height of characters with ascenders, such as b, d or l.

body-height
The height of characters such as parentheses.

cap-height
The height of uppercase letters such as A.

comma-depth
The depth of a comma.

desc-depth
The depth of characters with descenders, such as p,q, or y.

body-depth
The depth of characters such as parentheses.

addftinfo makes no attempt to use the specified parameters to guess the unspecified parameters. If a parameter is not specified the default will be used. The defaults are chosen to have the reasonable values for a Times font.

SEE ALSO

groff_font(5), **groff**(1), **groff_char**(7)

AFMTODIT

NAME

afmtodit – create font files for use with groff −Tps

SYNOPSIS

afmtodit [**−ckmnsvx**] [**−a** *n*] [**−d** *desc_file*] [**−e** *enc_file*] [**−f** *internal_name*] [**−i** *n*] *afm_file map_file font*

The whitespace between a command line option and its argument is optional.

DESCRIPTION

afmtodit creates a font file for use with groff and **grops**. **afmtodit** is written in perl; you must have perl version 5.004 or newer installed in order to run **afmtodit**.

afm_file is the AFM (Adobe Font Metric) file for the font.

map_file is a file that says which groff character names map onto each PostScript character name; this file should contain a sequence of lines of the form

ps_char groff_char

where *ps_char* is the PostScript name of the character and *groff_char* is the groff name of the character (as used in the groff font file). The same *ps_char* can occur multiple times in the file; each *groff_char* must occur at most once. Lines starting with **#** and blank lines are ignored. If the file isn't found in the current directory, it is searched in the 'devps/generate' subdirectory of the default font directory.

If a PostScript character is not mentioned in *map_file*, and a generic groff glyph name can't be deduced using the Adobe Glyph List (AGL, built into **afmtodit**), then **afmtodit** puts the PostScript character into the groff font file as an unnamed character which can only be accessed by the **\N** escape sequence in **troff**. In particular, this is true for glyph variants like 'foo.bar'; all glyph names containing one or more periods are mapped to unnamed entities. If option **−e** is not specified, the encoding defined in the AFM file (i.e., entries with non-negative character codes) is used. Please refer to section 'Using Symbols' in the groff info file which describes how groff glyph names are constructed.

Characters not encoded in the AFM file (i.e., entries which have −1 as the character code) are still available in groff; they get glyph index values greater than 255 (or greater than the biggest character code used in the AFM file in the unlikely case that it is greater than 255) in the groff font file. Glyph indices of unencoded characters don't have a specific order; it is best to access them with glyph names only.

The groff font file will be output to a file called *font*.

If there is a downloadable font file for the font, it may be listed in the file **c:/progra~1/groff/share/groff/1.20/font/devps/download**; see **grops**(1).

If the **−i** option is used, **afmtodit** will automatically generate an italic correction, a left italic correction and a subscript correction for each character (the significance of these parameters is explained in **groff_font**(5)); these parameters may be specified for individual characters by adding to the *afm_file* lines of the form:

italicCorrection *ps_char n*
leftItalicCorrection *ps_char n*
subscriptCorrection *ps_char n*

where *ps_char* is the PostScript name of the character, and *n* is the desired value of the corresponding parameter in thousandths of an em. These parameters are normally needed only for italic (or oblique) fonts.

OPTIONS

−a*n* Use *n* as the slant parameter in the font file; this is used by groff in the positioning of accents. By default **afmtodit** uses the negative of the ItalicAngle specified in the afm file; with true italic fonts it is sometimes desirable to use a slant that is less than this. If you find that characters from an italic font have accents placed too far to the right over them, then use the **−a** option to give the font a smaller slant.

−c Include comments in the font file in order to identify the PS font.

−d*desc_file*
The device description file is *desc_file* rather than the default **DESC**. If not found in the current directory, the 'devps' subdirectory of the default font directory is searched (this is true for both the default device description file and a file given with option **−d**).

−e*enc_file*
The PostScript font should be reencoded to use the encoding described in enc_file. The format of *enc_file* is described in **grops**(1). If not found in the current directory, the 'devps' subdirectory of the default font directory is searched.

−f*name*
The internal name of the groff font is set to *name*.

−i*n* Generate an italic correction for each character so that the character's width plus the character's italic correction is equal to *n* thousandths of an em plus the amount by which the right edge of the character's bounding box is to the right of the character's origin. If this would result in a negative italic correction, use a zero italic correction instead.

Also generate a subscript correction equal to the product of the tangent of the slant of the font and four fifths of the x-height of the font. If this would result in a subscript correction greater than the italic correction, use a subscript correction equal to the italic correction instead.

Also generate a left italic correction for each character equal to *n* thousandths of an em plus the amount by which the left edge of the character's bounding box is to the left of the character's origin. The left italic correction may be negative unless option **−m** is given.

This option is normally needed only with italic (or oblique) fonts. The font files distributed with groff were created using an option of **−i50** for italic fonts.

−k Omit any kerning data from the groff font. This should be used only for mono-spaced fonts.

−m Prevent negative left italic correction values. Roman font files distributed with groff were created with **−i0 −m** to improve spacing with **eqn**(1).

−n Don't output a **ligatures** command for this font. Use this with constant-width fonts.

−s The font is special. The effect of this option is to add the **special** command to the font file.

−v Print version.

−x Don't use the built-in Adobe Glyph List.

FILES

c:/progra˜1/groff/share/groff/1.20/font/devps/DESC
Device description file.

c:/progra˜1/groff/share/groff/1.20/font/devps/*F*
Font description file for font *F*.

c:/progra˜1/groff/share/groff/1.20/font/devps/download
List of downloadable fonts.

c:/progra˜1/groff/share/groff/1.20/font/devps/text.enc
Encoding used for text fonts.

c:/progra˜1/groff/share/groff/1.20/font/devps/generate/textmap
Standard mapping.

SEE ALSO

groff(1), **grops**(1), **groff_font**(5), **perl**(1)

The groff info file, section 'Using Symbols'.

CHEM

NAME

chem − groff preprocessor for producing chemical structure diagrams

SYNOPSIS

[*option*. . .] [**−−**] [*filespec*. . .] **−h | −−help −v | −−version**

OPTION USAGE

There are no other options than **−h**, **−−help**, **−v**, and **−−version**; these options provoke the printing of a version or usage information, respectively, and all *filespec* arguments are ignored. A *filespec* argument is either a file name of an existing file or a minus character **−**, meaning standard input. If no argument is specified then standard input is taken automatically.

DESCRIPTION

chem produces chemical structure diagrams. Today's version is best suited for organic chemistry (bonds, rings). The **chem** program is a **groff** preprocessor like **eqn**, **pic**, **tbl**, etc. It generates *pic* output such that all *chem* parts are translated into diagrams of the *pic* language.

The program **chem** originates from the Perl source file **`chem.pl`**. It tells **pic** to include a copy of the macro file **`chem.pic`**. Moreover the *groff* source file **`pic.tmac`** is loaded.

In a style reminiscent of *eqn* and *pic*, the *chem* diagrams are written in a special language.

A set of *chem* lines looks like this

.cstart
chem data
.cend

Lines containing the keywords **.cstart** and **.cend** start and end the input for **chem**, respectively. In *pic* context, i.e., after the call of **.PS**, *chem* input can optionally be started by the line **begin chem** and ended by the line with the single word **end** instead.

Anything outside these initialization lines is copied through without modification; all data between the initialization lines is converted into *pic* commands to draw the diagram.

As an example,

.cstart
CH3
bond
CH3
.cend

prints two **CH3** groups with a bond between them.

To actually view this, you must run **chem** followed by **groffer**:

chem [file. . .] | groffer

If you want to create just **groff** output, you must run **chem** followed by **groff** with the option **−p** for the activation of **pic**:

chem [file. . .] | groff -p . . .

THE LANGUAGE

The *chem* input language is rather small. It provides rings of several styles and a way to glue them together as desired, bonds of several styles, moieties (e.g., **C**, **NH3**, . . .), and strings.

Setting Variables

There are some variables that can be set by commands. Such commands have two possible forms, either

variable value

or

variable = *value*

This sets the given *variable* to the argument *value*. If more arguments are given only the last argument is taken, all other arguments are ignored.

There are only a few variables to be set by these commands:

textht *arg*
: Set the height of the text to *arg*; default is 0.16.

cwid *arg*
: Set the character width to *arg*; default is 0.12.

db *arg* Set the bond length to *arg*; default is 0.2.

size *arg*
: Scale the diagram to make it look plausible at point size *arg*; default is 10 point.

Bonds

This

[*direction*] [*length n*] [**from** *Name*|*picstuff*]

draws a single bond in direction from nearest corner of *Name*. **bond** can also be **double bond**, **front bond**, **back bond**, etc. (We will get back to *Name* soon.)

direction is the angle in degrees (0 up, positive clockwise) or a direction word like **up**, **down**, **sw** (= southwest), etc. If no direction is specified, the bond goes in the current direction (usually that of the last bond).

Normally the bond begins at the last object placed; this can be changed by naming a **from** place. For instance, to make a simple alkyl chain:

```
CH3
bond                  (this one goes right from the CH3)
C                     (at the right end of the bond)
double bond up        (from the C)
O                     (at the end of the double bond)
bond right from C
CH3
```

A length in inches may be specified to override the default length. Other *pic* commands can be tacked on to the end of a bond command, to created dotted or dashed bonds or to specify a **to** place.

Rings

There are lots of rings, but only 5 and 6-sided rings get much support. **ring** by itself is a 6-sided ring; **benzene** is the benzene ring with a circle inside. **aromatic** puts a circle into any kind of ring.

[**pointing** (**up**|**right**|**left**|**down**)] [**aromatic**] [**put Mol at** *n*] [**double** *i*,*j k*,*l . . .] [picstuff*]

The vertices of a ring are numbered 1, 2, . . . from the vertex that points in the natural compass direction. So for a hexagonal ring with the point at the top, the top vertex is 1, while if the ring has a point at the east side, that is vertex 1. This is expressed as

R1: ring pointing up
R2: ring pointing right

The ring vertices are named **.V1**, . . ., **.V***n*, with **.V1** in the pointing direction. So the corners of **R1** are **R1.V1** (the *top*), **R1.V2**, **R1.V3**, **R1.V4** (the *bottom*), etc., whereas for **R2**, **R2.V1** is the rightmost vertex and **R2.V4** the leftmost. These vertex names are used for connecting bonds or other rings. For example,

R1: benzene pointing right
R2: benzene pointing right with .V6 at R1.V2

creates two benzene rings connected along a side.

Interior double bonds are specified as **double** *n1***,***n2 n3***,***n4 . . .*; each number pair adds an interior bond. So the alternate form of a benzene ring is

ring double 1,2 3,4 5,6

Heterocycles (rings with something other than carbon at a vertex) are written as **put** *X* **at** *V*, as in

R: ring put N at 1 put O at 2

In this heterocycle, **R.N** and **R.O** become synonyms for **R.V1** and **R.V2**.

There are two 5-sided rings. **ring5** is pentagonal with a side that matches the 6-sided ring; it has four natural directions. A **flatring** is a 5-sided ring created by chopping one corner of a 6-sided ring so that it exactly matches the 6-sided rings.

The description of a ring has to fit on a single line.

Moieties and Strings

A moiety is a string of characters beginning with a capital letter, such as N(C2H5)2. Numbers are converted to subscripts (unless they appear to be fractional values, as in N2.5H). The name of a moiety is determined from the moiety after special characters have been stripped out: e.g., N(C2H5)2) has the name NC2H52.

Moieties can be specified in two kinds. Normally a moiety is placed right after the last thing mentioned, separated by a semicolon surrounded by spaces, e.g.,

B1: bond ; OH

Here the moiety is **OH**; it is set after a bond.

As the second kind a moiety can be positioned as the first word in a *pic*-like command, e.g.,

CH3 at C + (0.5,0.5)

Here the moiety is **CH3**. It is placed at a position relative to **C**, a moiety used earlier in the chemical structure.

So moiety names can be specified as *chem* positions everywhere in the *chem* code. Beneath their printing moieties are names for places.

The moiety **BP** is special. It is not printed but just serves as a mark to be referred to in later *chem* commands. For example,

bond ; BP

sets a mark at the end of the bond. This can be used then for specifying a place. The name **BP** is derived from *branch point* (i.e., line crossing).

A string within double quotes **"** is interpreted as a part of a *chem* command. It represents a string that should be printed (without the quotes). Text within quotes "..." is treated more or less like a moiety except that no changes are made to the quoted part.

Names

In the alkyl chain above, notice that the carbon atom **C** was used both to draw something and as the name for a place. A moiety always defines a name for a place; you can use your own names for places instead, and indeed, for rings you will have to. A name is just

Name: ...

Name is often the name of a moiety like **CH3**, but it need not to be. Any name that begins with a capital letter and which contains only letters and numbers is valid:

First: bond bond 30 from First

Miscellaneous

The specific construction

bond ... **; moiety**

is equivalent to

bond
moiety

Otherwise, each item has to be on a separate line (and only one line). Note that there must be whitespace after the semicolon which separates the commands.

A period character **.** or a single quote **'** in the first column of a line signals a *troff* command, which is copied through as-is.

A line whose first non-blank character is a hash character (**#**) is treated as a comment and thus ignored. However, hash characters within a word are kept.

A line whose first word is **pic** is copied through as-is after the word **pic** has been removed.

The command

size *n*

scales the diagram to make it look plausible at point size *n* (default is 10 point).

Anything else is assumed to be *pic* code, which is copied through with a label.

Since **chem** is a **pic** preprocessor, it is possible to include *pic* statements in the middle of a diagram to draw things not provided for by *chem* itself. Such *pic* statements should be included in *chem* code by adding **pic** as the first word of this line for clarity.

The following *pic* commands are accepted as *chem* commands, so no **pic** command word is needed:

define Start the definition of *pic* macro within *chem*.

[Start a block composite.

] End a block composite.

{ Start a macro definition block.

} End a macro definition block.

The macro names from **define** statements are stored and their call is accepted as a *chem* command as well.

WISH LIST

This TODO list was collected by Brian Kernighan.

Error checking is minimal; errors are usually detected and reported in an oblique fashion by *pic*.

There is no library or file inclusion mechanism, and there is no shorthand for repetitive structures.

The extension mechanism is to create *pic* macros, but these are tricky to get right and don't have all the properties of built-in objects.

There is no in-line chemistry yet (e.g., analogous to the $...$ construct of eqn).

There is no way to control entry point for bonds on groups. Normally a bond connects to the carbon atom if entering from the top or bottom and otherwise to the nearest corner.

Bonds from substituted atoms on heterocycles do not join at the proper place without adding a bit of *pic*.

There is no decent primitive for brackets.

Text (quoted strings) doesn't work very well.

A squiggle bond is needed.

FILES

`c:/progra~1/groff/share/groff/1.20/pic/chem.pic`
A collection of *pic* macros needed by **chem**.

`c:/progra~1/groff/share/groff/1.20/tmac/pic.tmac`
A macro file which redefines **.PS** and **.PE** to center *pic* diagrams.

`c:/progra~1/groff/share/doc/groff/1.20/examples/chem/*.chem`
Example files for *chem*.

`c:/progra~1/groff/share/doc/groff/1.20/examples/chem/122/*.chem`
Example files from the classical *chem* book **122.ps**.

BUGS

Report bugs to the bug-groff mailing list Include a complete, self-contained example that will allow the bug to be reproduced, and say which version of *groff* and *chem* you are using. You can get both version numbers by calling **chem --version**.

You can also use the groff mailing list but you must first subscribe to this list. You can do that by visiting the groff mailing list web page

See **groff**(1) for information on availability.

SEE ALSO

groff(1), **pic**(1), **groffer**(1).

You can still get the original chem awk source Its **`README`** file was used for this manual page.

The other classical document on *chem* is 122.ps

AUTHOR

This file was written by Bernd Warken. It is based on the documentation of Brian Kernighan original *awk* version of *chem*.

COPYING

DEROFF

NAME

deroff – remove roff, tbl, eqn, refer and pic constructs from documents

SYNOPSIS

deroff [**−w**] [**−s**] [**−ml**] [**−ms**] [**−mm**] [**−p**] [*file...*]
deroff [**−−word-list**] [**−−skip-headers**] [**−−skip-lists**] [**−−pretty-print**] [*file...*]
deroff −h|**−−help**
deroff −−version

DESCRIPTION

deroff reads roff documents and removes all *nroff*(1), *troff*(1), *refer*(1), *tbl*(1), *eqn*(1) and *pic*(1) constructs. The resulting text will be sent to standard output. **.so** and **.nx** requests are processed, but repeated requests to process an already read file will be ignored.

OPTIONS

−w, **−−word-list**
Output a word list, one word per line.

−s, **−−skip-headers**
Do not output headers. This is useful if you want to run text analysis tools on the output.

−ml, **−−suppress-lists**
Suppress lists. This option is useful with **−s**, if there are many incomplete sentences in lists.

−ms, **−mm**
These options are accepted for compatibility, but they are being ignored.

−i Ignore **.so** and **.nx** requests.

−p, **−−pretty−print**
Format the output more pretty by omitting and adding newline characters at certain places.

−h, **−−help**
Print a short usage message.

−−version
Print the version.

EXAMPLE

The following example does a simple spell check of a document:

```
deroff -w document.mm | sort -u |
comm -23 - /usr/share/words/en
```

RESTRICTIONS

deroff is not a complete roff parser, so it can be confused by complicated constructs. Often too much output is done in these cases.

AUTHOR

This program is copyright 1993–2004 Michael Haardt <michael@moria.de>.

SEE ALSO

soelim(1), troff(1), nroff(1), refer(1), tbl(1), pic(1), eqn(1)

EQN

NAME

eqn – format equations for troff or MathML

SYNOPSIS

[**–rvCNR**] [**–d** *xy*] [**–T** *name*] [**–M** *dir*] [**–f** *F*] [**–s** *n*] [**–p** *n*] [**–m** *n*] [*files*. . .]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

This manual page describes the GNU version of **eqn**, which is part of the groff document formatting system. **eqn** compiles descriptions of equations embedded within **troff** input files into commands that are understood by **troff**. Normally, it should be invoked using the **–e** option of **groff**. The syntax is quite compatible with Unix eqn. The output of GNU **eqn** cannot be processed with Unix troff; it must be processed with GNU troff. If no files are given on the command line, the standard input is read. A filename of **–** causes the standard input to be read.

eqn searches for the file **eqnrc** in the directories given with the **–M** option first, then in **c:/progra˜1/groff/lib/groff/site-tmac**, **c:/progra˜1/groff/share/groff/site-tmac**, and finally in the standard macro directory **c:/progra˜1/groff/share/groff/1.20/tmac**. If it exists, **eqn** processes it before the other input files. The **–R** option prevents this.

GNU **eqn** does not provide the functionality of neqn: it does not support low-resolution, typewriter-like devices (although it may work adequately for very simple input).

OPTIONS

–d*xy* Specify delimiters *x* and *y* for the left and right end, respectively, of in-line equations. Any **delim** statements in the source file overrides this.

–C Recognize **.EQ** and **.EN** even when followed by a character other than space or newline.

–N Don't allow newlines within delimiters. This option allows **eqn** to recover better from missing closing delimiters.

–v Print the version number.

–r Only one size reduction.

–m*n* The minimum point-size is *n*. **eqn** does not reduce the size of subscripts or superscripts to a smaller size than *n*.

–T*name*
The output is for device *name*. Normally, the only effect of this is to define a macro *name* with a value of **1**; **eqnrc** uses this to provide definitions appropriate for the output device. However, if the specified device is "MathML", the output is MathML markup rather than troff commands, and **eqnrc** is not loaded at all. The default output device is **ps**.

–M*dir* Search *dir* for **eqnrc** before the default directories.

–R Don't load **eqnrc**.

–f*F* This is equivalent to a **gfont** *F* command.

–s*n* This is equivalent to a **gsize** *n* command. This option is deprecated. **eqn** normally sets equations at whatever the current point size is when the equation is encountered.

–p*n* This says that subscripts and superscripts should be *n* points smaller than the surrounding text. This option is deprecated. Normally **eqn** sets subscripts and superscripts at 70% of the size of the surrounding text.

USAGE

Only the differences between GNU **eqn** and Unix eqn are described here.

GNU **eqn** emits Presentation MathML output when invoked with the **-T MathML** option.

GNU eqn sets the input token **"..."** as three periods or low dots, rather than the three centered dots of classic eqn. To get three centered dots, write **cdots** or **cdot cdot cdot**.

Most of the new features of the GNU **eqn** input language are based on TEX. There are some references to the differences between TEX and GNU **eqn** below; these may safely be ignored if you do not know TEX.

Automatic spacing

eqn gives each component of an equation a type, and adjusts the spacing between components using that type. Possible types are:

ordinary	an ordinary character such as '1' or '*x*';
operator	a large operator such as 'Σ';
binary	a binary operator such as '+';
relation	a relation such as '=';
opening	a opening bracket such as '(';
closing	a closing bracket such as ')';
punctuation	a punctuation character such as ',';
inner	a subformula contained within brackets;
suppress	spacing that suppresses automatic spacing adjustment.

Components of an equation get a type in one of two ways.

type *t e* This yields an equation component that contains *e* but that has type *t*, where *t* is one of the types mentioned above. For example, **times** is defined as

type "binary" \(mu

The name of the type doesn't have to be quoted, but quoting protects from macro expansion.

chartype *t text*
Unquoted groups of characters are split up into individual characters, and the type of each character is looked up; this changes the type that is stored for each character; it says that the characters in *text* from now on have type *t*. For example,

chartype "punctuation" .,;:

would make the characters '.,;:' have type punctuation whenever they subsequently appeared in an equation. The type *t* can also be **letter** or **digit**; in these cases **chartype** changes the font type of the characters. See the **Fonts** subsection.

New primitives

big *e* Enlarges the expression it modifies; intended to have semantics like CSS 'large'. In troff output, the point size is increased by 5; in MathML output, the expression uses

<mstyle mathsize='big'>

e1 **smallover** *e2*
This is similar to **over**; **smallover** reduces the size of *e1* and *e2*; it also puts less vertical space between *e1* or *e2* and the fraction bar. The **over** primitive corresponds to the TEX **\over** primitive in display styles; **smallover** corresponds to **\over** in non-display styles.

vcenter *e*
This vertically centers *e* about the math axis. The math axis is the vertical position about which characters such as '+' and '−' are centered; also it is the vertical position used for the bar of fractions. For example, **sum** is defined as

{ type "operator" vcenter size +5 \(*S }

(Note that vcenter is silently ignored when generating MathML.)

e1 **accent** *e2*
This sets *e2* as an accent over *e1*. *e2* is assumed to be at the correct height for a lowercase letter; *e2* is moved down according to whether *e1* is taller or shorter than a lowercase letter. For example, **hat** is defined as

accent { "^" }

dotdot, **dot**, **tilde**, **vec**, and **dyad** are also defined using the **accent** primitive.

e1 **uaccent** *e2*
This sets *e2* as an accent under *e1*. *e2* is assumed to be at the correct height for a character without a descender; *e2* is moved down if *e1* has a descender. **utilde** is pre-defined using

uaccent as a tilde accent below the baseline.

split "*text*"
This has the same effect as simply

text

but *text* is not subject to macro expansion because it is quoted; *text* is split up and the spacing between individual characters is adjusted.

nosplit *text*
This has the same effect as

"*text***"**

but because *text* is not quoted it is subject to macro expansion; *text* is not split up and the spacing between individual characters is not adjusted.

e **opprime**
This is a variant of **prime** that acts as an operator on *e*. It produces a different result from **prime** in a case such as **A opprime sub 1**: with **opprime** the **1** is tucked under the prime as a subscript to the **A** (as is conventional in mathematical typesetting), whereas with **prime** the **1** is a subscript to the prime character. The precedence of **opprime** is the same as that of **bar** and **under**, which is higher than that of everything except **accent** and **uaccent**. In unquoted text a **'** that is not the first character is treated like **opprime**.

special *text e*
This constructs a new object from *e* using a **troff**(1) macro named *text*. When the macro is called, the string **0s** contains the output for *e*, and the number registers **0w**, **0h**, **0d**, **0skern**, and **0skew** contain the width, height, depth, subscript kern, and skew of *e*. (The *subscript kern* of an object says how much a subscript on that object should be tucked in; the *skew* of an object says how far to the right of the center of the object an accent over the object should be placed.) The macro must modify **0s** so that it outputs the desired result with its origin at the current point, and increase the current horizontal position by the width of the object. The number registers must also be modified so that they correspond to the result.

For example, suppose you wanted a construct that 'cancels' an expression by drawing a diagonal line through it.

```
.EQ
define cancel 'special Ca'
.EN
.de Ca
. ds 0s \
\Z'\\*(0s'\
\v'\\n(0du'\
\D'l \\n(0wu -\\n(0hu-\\n(0du'\
\v'\\n(0hu'
..
```

Then you could cancel an expression *e* with **cancel {** *e* **}**

Here's a more complicated construct that draws a box round an expression:

```
.EQ
define box 'special Bx'
.EN
.de Bx
. ds 0s \
\Z'\h'1n'\\*(0s'\
\Z'\
\v'\\n(0du+1n'\
\D'l \\n(0wu+2n 0'\
\D'l 0 -\\n(0hu-\\n(0du-2n'\
\D'l -\\n(0wu-2n 0'\
\D'l 0 \\n(0hu+\\n(0du+2n'\
'\
\h'\\n(0wu+2n'
. nr 0w +2n
. nr 0d +1n
. nr 0h +1n
..
```

space *n*

A positive value of the integer *n* (in hundredths of an em) sets the vertical spacing before the equation, a negative value sets the spacing after the equation, replacing the default values. This primitive provides an interface to **groff**'s **\x** escape (but with opposite sign).

This keyword has no effect if the equation is part of a **pic** picture.

Extended primitives

col *n* { . . . }

ccol *n* { . . . } **lcol** *n* { . . . } **rcol** *n* { . . . } **pile** *n* { . . . } **cpile** *n* { . . . } **lpile** *n* { . . . } **rpile** *n* { . . . }
The integer value *n* (in hundredths of an em) increases the vertical spacing between rows, using **groff**'s **\x** escape (the value has no effect in MathML mode). Negative values are possible but have no effect. If there is more than a single value given in a matrix, the biggest one is used.

Customization

When **eqn** is generating troff markup, the appearance of equations is controlled by a large number of parameters. They have no effect when generating MathML mode, which pushes typesetting and fine motions downstream to a MathML rendering engine. These parameters can be set using the **set** command.

set *p n* This sets parameter *p* to value *n*; *n* is an integer. For example,

set x_height 45

says that **eqn** should assume an x height of 0.45 ems.

Possible parameters are as follows. Values are in units of hundredths of an em unless otherwise stated. These descriptions are intended to be expository rather than definitive.

minimum_size	**eqn** doesn't set anything at a smaller point-size than this. The value is in points.
fat_offset	The **fat** primitive emboldens an equation by overprinting two copies of the equation horizontally offset by this amount. This parameter is not used in MathML mode; instead, fat text uses <mstyle mathvariant='double-struck'>
over_hang	A fraction bar is longer by twice this amount than the maximum of the widths of the numerator and denominator; in other words, it overhangs the numerator and denominator by at least this amount.
accent_width	When **bar** or **under** is applied to a single character, the line is this long. Normally, **bar** or **under** produces a line whose length is the width of the object to which it applies; in the case of a single character, this tends to produce a line that looks too long.
delimiter_factor	Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth of at least this many thousandths of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis.
delimiter_shortfall	Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth not less than the difference of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis and this amount.
null_delimiter_space	This much horizontal space is inserted on each side of a fraction.
script_space	The width of subscripts and superscripts is increased by this amount.

thin_space	This amount of space is automatically inserted after punctuation characters.
medium_space	This amount of space is automatically inserted on either side of binary operators.
thick_space	This amount of space is automatically inserted on either side of relations.
x_height	The height of lowercase letters without ascenders such as 'x'.
axis_height	The height above the baseline of the center of characters such as '+' and '−'. It is important that this value is correct for the font you are using.
default_rule_thickness	This should set to the thickness of the **\(ru** character, or the thickness of horizontal lines produced with the **\D** escape sequence.
num1	The **over** command shifts up the numerator by at least this amount.
num2	The **smallover** command shifts up the numerator by at least this amount.
denom1	The **over** command shifts down the denominator by at least this amount.
denom2	The **smallover** command shifts down the denominator by at least this amount.
sup1	Normally superscripts are shifted up by at least this amount.
sup2	Superscripts within superscripts or upper limits or numerators of **smallover** fractions are shifted up by at least this amount. This is usually less than sup1.
sup3	Superscripts within denominators or square roots or subscripts or lower limits are shifted up by at least this amount. This is usually less than sup2.
sub1	Subscripts are normally shifted down by at least this amount.
sub2	When there is both a subscript and a superscript, the subscript is shifted down by at least this amount.
sup_drop	The baseline of a superscript is no more than this much amount below the top of the object on which the superscript is set.
sub_drop	The baseline of a subscript is at least this much below the bottom of the object on which the subscript is set.
big_op_spacing1	The baseline of an upper limit is at least this much above the top of the object on which the limit is set.
big_op_spacing2	The baseline of a lower limit is at least this much below the bottom of the object on which the limit is set.
big_op_spacing3	The bottom of an upper limit is at least this much above the top of the object on which the limit is set.
big_op_spacing4	The top of a lower limit is at least this much below the bottom of the object on which the limit is set.
big_op_spacing5	This much vertical space is added above and below limits.
baseline_sep	The baselines of the rows in a pile or matrix are normally this far apart. In most cases this should be equal to the sum of **num1** and **denom1**.
shift_down	The midpoint between the top baseline and the bottom baseline in a matrix or pile is shifted down by this much from the axis. In most cases this should be equal to **axis_height**.

column_sep	This much space is added between columns in a matrix.
matrix_side_sep	This much space is added at each side of a matrix.
draw_lines	If this is non-zero, lines are drawn using the **\D** escape sequence, rather than with the **\l** escape sequence and the **\(ru** character.
body_height	The amount by which the height of the equation exceeds this is added as extra space before the line containing the equation (using **\x**). The default value is 85.
body_depth	The amount by which the depth of the equation exceeds this is added as extra space after the line containing the equation (using **\x**). The default value is 35.
nroff	If this is non-zero, then **ndefine** behaves like **define** and **tdefine** is ignored, otherwise **tdefine** behaves like **define** and **ndefine** is ignored. The default value is 0 (This is typically changed to 1 by the **eqnrc** file for the **ascii**, **latin1**, **utf8**, and **cp1047** devices.)

A more precise description of the role of many of these parameters can be found in Appendix H of *The TEXbook*.

Macros

Macros can take arguments. In a macro body, **$***n* where *n* is between 1 and 9, is replaced by the *n-th* argument if the macro is called with arguments; if there are fewer than *n* arguments, it is replaced by nothing. A word containing a left parenthesis where the part of the word before the left parenthesis has been defined using the **define** command is recognized as a macro call with arguments; characters following the left parenthesis up to a matching right parenthesis are treated as comma-separated arguments; commas inside nested parentheses do not terminate an argument.

sdefine *name X anything X*
This is like the **define** command, but *name* is not recognized if called with arguments.

include "*file***"**
copy "*file***"** Include the contents of *file* (**include** and **copy** are synonyms). Lines of *file* beginning with **.EQ** or **.EN** are ignored.

ifdef *name X anything X*
If *name* has been defined by **define** (or has been automatically defined because *name* is the output device) process *anything*; otherwise ignore *anything*. *X* can be any character not appearing in *anything*.

undef *name*
Remove definition of *name*, making it undefined.

Besides the macros mentioned above, the following definitions are available: **Alpha**, **Beta**, . . ., **Omega** (this is the same as **ALPHA**, **BETA**, . . ., **OMEGA**), **ldots** (three dots on the base line), and **dollar**.

Fonts

eqn normally uses at least two fonts to set an equation: an italic font for letters, and a roman font for everything else. The existing **gfont** command changes the font that is used as the italic font. By default this is **I**. The font that is used as the roman font can be changed using the new **grfont** command.

grfont *f*
Set the roman font to *f*.

The **italic** primitive uses the current italic font set by **gfont**; the **roman** primitive uses the current roman font set by **grfont**. There is also a new **gbfont** command, which changes the font used by the **bold** primitive. If you only use the **roman**, **italic** and **bold** primitives to changes fonts within an equation, you can change all the fonts used by your equations just by using **gfont**, **grfont** and **gbfont** commands.

You can control which characters are treated as letters (and therefore set in italics) by using the **chartype** command described above. A type of **letter** causes a character to be set in italic type. A type of **digit** causes a character to be set in roman type.

FILES

c:/progra~1/groff/share/groff/1.20/tmac/eqnrc Initialization file.

MATHML MODE LIMITATIONS

MathML is designed on the assumption that it cannot know the exact physical characteristics of the media and devices on which it will be rendered. It does not support fine control of motions and sizes to the same degree troff does. Thus:

- **eqn** parameters have no effect on the generated MathML.
- The **special, up**, **down**, **fwd**, and **back** operations cannot be implemented, and yield a MathML '<merror>' message instead.
- The **vcenter** keyword is silently ignored, as centering on the math axis is the MathML default.
- Characters that **eqn** over troff sets extra large – notably the integral sign – may appear too small and need to have their '<mstyle>' wrappers adjusted by hand.

As in its troff mode, **eqn** in MathML mode leaves the **.EQ** and **.EN** delimiters in place for displayed equations, but emits no explicit delimiters around inline equations. They can, however, be recognized as strings that begin with '<math>' and end with '</math>' and do not cross line boundaries.

See the **BUGS** section for translation limits specific to **eqn**.

BUGS

Inline equations are set at the point size that is current at the beginning of the input line.

In MathML mode, the **mark** and **lineup** features don't work. These could, in theory, be implemented with '<maligngroup>' elements.

In MathML mode, each digit of a numeric literal gets a separate '<mn></mn>' pair, and decimal points are tagged with '<mo></mo>'. This is allowed by the specification, but inefficient.

SEE ALSO

groff(1), **troff**(1), **pic**(1), **groff_font**(5), *The TEXbook*

EQN2GRAPH

NAME

eqn2graph − convert an EQN equation into a cropped image

SYNOPSIS

eqn2graph [**−unsafe**] [**−format** *fmt*]

DESCRIPTION

Reads an EQN equation (one line) as input; produces an image file (by default in Portable Network Graphics format) suitable for the Web as output.

Your input EQN code should *not* have the .EQ/.EN preamble that that normally precedes it within **groff**(1) macros; nor do you need to have dollar-sign or other delimiters around the equation.

The output image will be clipped to the smallest possible bounding box that contains all the black pixels. Older versions of **convert**(1) will produce a black-on-white graphic; newer ones may produce a black-on-transparent graphic. By specifying command-line options to be passed to **convert**(1) you can give it a border, force the background transparent, set the image's pixel density, or perform other useful transformations.

This program uses **eqn**(1), **groff**(1), and the ImageMagick **convert**(1) program. These programs must be installed on your system and accessible on your $PATH for **eqn2graph** to work.

OPTIONS

−unsafe
: Run **groff**(1) in the 'unsafe' mode enabling the PIC macro **sh** to execute arbitrary commands. The default is to forbid this.

−format *fmt*
: Specify an output format; the default is PNG (Portable Network Graphics). Any format that **convert**(1) can emit is supported.

Command-line switches and arguments not listed above are passed to **convert**(1).

FILES

c:/progra~1/groff/share/groff/1.20/tmac/eqnrc The **eqn**(1) initialization file.

ENVIRONMENT

GROFF_TMPDIR
: The directory in which temporary files will be created. If this is not set **eqn2graph** searches the environment variables **TMPDIR**, **TMP**, and **TEMP** (in that order). Otherwise, temporary files will be created in **/tmp**.

BUGS

Due to changes in the behavior of ImageMagick **convert**(1) that are both forward and backward-incompatible, mismatches between your **eqn2graph** and **convert**(1) versions may produce zero-sized or untrimmed output images. For this version of **eqn2graph** you will need a version of **convert**(1) that supports the **−trim** option; older versions of **eqn2graph** used **−crop 0x0**, which no longer has trimming behavior.

SEE ALSO

pic2graph(1), **grap2graph**(1), **eqn**(1), **groff**(1), **gs**(1), **convert**(1).

AUTHOR

Eric S. Raymond <esr@thyrsus.com>.

GDIFFMK

NAME

gdiffmk − mark differences between groff/nroff/troff files

SYNOPSIS

gdiffmk [**−a***addmark*] [**−c***changemark*] [**−d***deletemark*] [**−D** [**−B**] [**−M***mark1 mark2*]]
[**−x***diffcmd*] [**−−**] [**−−help**] [**−−version**] *file1 file2* [*output*]

DESCRIPTION

gdiffmk compares two **groff**(1), **nroff**(1), or **troff**(1) documents, *file1* and *file2*, and creates an output which is *file2* with added 'margin character' (.mc) commands that indicate the differences.

If the *output* filename is present, the output is written there. If it is **−** or absent the output is written to the standard output.

If the *file1* or *file2* argument is **−** the standard input is read for that input. Clearly both cannot be **−**.

Note that the output is not necessarily compatible with all macro packages and all preprocessors. See the **BUGS** section below.

OPTIONS

−a*addmark*
Use the *addmark* for source lines not in *file1* but present in *file2*. Default: **+**.

−B By default, the deleted texts marked by the **−D** option end with an added troff break command, **.br**, to ensure that the deletions are marked properly. This is the only way to guarantee that deletions and small changes get flagged. This option directs the program not to insert these breaks; it makes no sense to use it without **−D**.

−c*changemark*
Use the *changemark* for changed source lines. Default: **|**.

−d*deletemark*
Use the *deletemark* for deleted source lines. Default: *****.

−D Show the deleted portions from changed and deleted text. Default delimiting marks: **[[....]]**.

−M*mark1 mark2*
Change the delimiting marks for the **−D** option. It makes no sense to use this option without **−D**.

−x*diffcmd*
Use the *diffcmd* command to perform the comparison of *file1* and *file2*. In particular, *diffcmd* should accept the GNU **diff −D***name* option. Default: **diff**(1).

−− All the following arguments are treated as file names, even if they begin with **−**.

−−help Print a usage message on standard error output and exit.

−−version
Print version information on the standard output and exit.

BUGS

The output is not necessarily compatible with all macro packages and all preprocessors. A workaround that is often successful against preprocessor problems is to run **gdiffmk** on the output of all the preprocessors instead of the input source.

gdiffmk relies on the **−D***name* option of GNU **diff**(1) to make a merged '#ifdef' output format. It hasn't been tested whether other versions of **diff**(1) do support this option. See also the **−x***diffcmd* option.

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

AUTHORS

This document was written and is maintained by Mike Bianchi

This document is distributed under the terms of the FDL (GNU Free Documentation License) version 1.1 or later. You should have received a copy of the FDL on your system, it is also available on-line at the GNU copyleft site

gdiffmk is part of the *groff* GNU free software project. All parts of the *groff package* are protected by GNU copyleft licenses. The software files are distributed under the terms of the GNU General Public License (GPL), while the documentation files mostly use the GNU Free Documentation License (FDL).

COPYRIGHT

SEE ALSO

groff(1), **nroff**(1), **gtroff**(1), **diff**(1) GRAP

NAME

grap — Kernighan and Bentley's language for typesetting graphs

SYNOPSIS

grap [**-d** *defines_file*] [**-D**] [**-l**] [**-M** *include path*] [**-R**] [**-r**] [**-v**] [**-u**] [**-C**] [**-c**] [**-h**] [*filename ...*]

DESCRIPTION

grap is an implementation of Kernighan and Bentley's language for typesetting graphs, as described in "Grap-A Language for Typesetting Graphs, Tutorial and User Manual," by Jon L. Bentley and Brian W. Kernighan, revised May 1991, which is the primary source for information on how to use **grap**. As of this writing, it is available electronically at http://www.kohala.com/start/troff/cstr114.ps. Additional documentation and examples, packaged with **grap**, may have been installed locally as well. If available, paths to them can be displayed using **grap -h** or **grap -v** (or **grap --help** / **grap --version**)

This version is a black box implementation of **grap**, and some inconsistencies are to be expected. The remainder of this manual page will briefly outline the **grap** language as implemented here.

grap is a pic(1) pre-processor. It takes commands embedded in a troff(1) source file which are surrounded by **.G1** and **.G2** macros, and rewrites them into pic commands to display the graph. Other lines are copied. Output is always to the standard output, which is usually redirected. Input is from the given *filename*s, which are read in order. A *filename* of **-** is the standard input. If no *filename*s are given, input is read from the standard input.

Because **grap** is a pic preprocessor, and GNU pic will output TeX, it is possible to use **grap** with TeX.

The **-d** option specifies a file of macro definitions to be read at startup, and defaults to /usr/local/share/grap/grap.defines . The **-D** option inhibits the reading of any initial macros file (the **-l** flag is a synonym for **-D**, though I do not remember why). The defines file can also be given using the GRAP_DEFINES environment variable. (See below).

-v prints the version information on the standard output and exits. **--version** is a synonym for **-v**.

-u makes labels unaligned by default. This version of **grap** uses new features of GNU pic to align the left and right labels with the axes, that is that the left and right labels run at right angles to the text of the paper. This may be useful in porting old **grap** programs. **-c** makes plot strings unclipped by default. Some versions of **grap** allow users to place a string anywhere in the coordinate space, rather than only in the frame. By default this version of **grap** does not plot any string centered outside the frame. **-c** allows strings to be placed anywhere. See also the **clipped** and **unclipped** string modifiers described in the **plot** statement.

-M is followed by a colon-separated list of directories used to search for relative pathnames included via **copy**. The path is also used to locate the defines file, so if the **-d** changes the defines file name to a relative name, it will be searched for in the path given by **-M**. The search path always includes the current directory, and by default that directory is searched last.

All numbers used internally by **grap** are double precision floating point values. Sometimes using floating point numbers has unintended consequences. To help avoid these problems, **grap** can use two thresholds for comparison of floating point numbers, set by **-R** or **-r**. The **-R** flag sets coarse comparison mode, which is suitable for most applications. If you are plotting small values – less than 1e-6 or so – consider using **-r** which uses very fine comparisons between numbers. You may also want to rescale your plotted values to be larger in magnitude. The coarse comarisons are used by default.

To be precise, the value by which two numbers must differ for **grap** to consider them not equal is called the comparison limit and the smallest non-zero number is called the minimum value. The values a given version of **grap** uses for these are included in the output of **-v** or **-h**.

All **grap** commands are included between **.G1** and **.G2** macros, which are consumed by **grap**. The output contains pic between **.PS** and **.PE** macros. Any arguments to the **.G1** macro in the input are arguments to the **.PS** macro in the output, so graphs can be scaled just like pic diagrams. If **-C** is given, any macro beginning with .G1 or .G2 is treated as a .G1 or .G2 macro, for compatibility with old versions of troff. Using **-C** also forces pure troff syntax on embedded font change commands when strings have the **size** attribute, and all strings to be **unclipped**.

The **-h** flag prints a brief help message and exits. **--help** is a synonym for **-h**.

It is possible for someone to cause **grap** to fail by passing a bad format string and data to the **sprintf** command. If **grap** is integrated as part of the printing system, this could conceivably provided a path to breaching security on the machine. If you choose to use **grap** as part of a printing system run by the super-user, you should disable **sprintf** commands. This can be done by calling **grap** with the **-S** flag, setting the GRAP_SAFER environment variable, or compiling with the GRAP_SAFER preprocessor symbol defined. (The GNU configure script included with **grap** will define that preprocessor symbol if the **--with-grap-safe** option is given.)

The **grap** commands are sketched below. Refer to Kernighan and Bentley's paper for the details.

New versions of groff(1) will invoke **grap** if **-G** is given.

Commands

Commands are separated from one another by newlines or semicolons (;).

frame [*line_description*] [**ht** *height* | **wid** *width*] [[(**top**|**bottom**|**left**| **right**) *line_description*] ...]

frame [**ht** *height* | **wid** *width*] [*line_description*] [[(**top**|**bottom**|**left**| **right**) *line_description*] ...]

This describes how the axes for the graph are drawn. A *line_description* is a pic line description, e.g., dashed 0.5, or the literal solid. It may also include a **color** keyword followed by the color to draw the string in double quotes. Any color understood by the underlying groff system can be used. Color can only be used under GNU pic, and is not available in compatibility mode. Similarly, for pic implementations that understand **thickness**, that attribute may be used with a real valued parameter. **Thickness** is not available in compatibility mode.

If the first *line_description* is given, the frame is drawn with that style. The default is solid. The height and width of the frame can also be specified in inches. The default line style can be over-ridden for sides of the frame by specifying additional parameters to **frame**.

If no plotting commands have been given before the **frame** command is issued, the frame will be output at that point in the plotting stream relative to embedded troff or pic commands. Otherwise the frame is output before the first plotted object (even invisible ones).

ht and **wid** are in inches by default, but can be any groff unit. If omitted, the dimensions are 2 inches high by 3 inches wide.

coord [*name*] [**x** *expr*, *expr*] [**y** *expr*, *expr*] [**log x** | **log y** | **log log**]

The **coord** command specifies a new coordinate system or sets limits on the default system. It defines the largest and smallest values that can be plotted, and therefore the scale of the data in the frame. The limits for the x and y coordinate systems can be given separately. If a *name* is given, that coordinate system is defined, if not the default system is modified.

A coordinate system created by one **coord** command may be modified by subsequent **coord** commands. A **grap** program may declare a coordinate space using **coord**, **copy** a file of data through a macro that plots the data and finds its maxima and minima, and then define the size of the coordinate system with a second **coord** statement.

This command also determines if a scale is plotted logarithmically. **log log** means the same thing as **log x log y**.

draw [*line_name*] [*line_description*] [*plot_string*]

The **draw** command defines the style with which a given line will be plotted. If *line_name* is given, the style is associated with that name, otherwise the default style is set. *line_description* is a pic line description, and the optional *plot_string* is a string to be centered at each point. The default line description is invis, and the default plotting string is a centered bullet, so by default each point is a filled circle, and they are unconnected. If points are being connected, each **draw** command ends any current line and begins a new one.

When defining a line style, that is the first **draw** command for a given line name, specifying no plot string means that there are to be no plot strings. Omitting the plot string on subsequent **draw** commands addressing the same named line means not to change the plot string. If a line has been defined with a plot string, and the format is changed by a subsequent **draw** statement, the plot string can be removed by specifying "" in the **draw** statement.

The plot string can have its format changed through several string_modifiers. String_modifiers are described in the description of the **plot** command.

The standard defines file includes several macros useful as plot strings, including **bullet**, **square**, and **delta**.

new is a synonym for **draw**.

next [*line_name*] **at** [*coordinates_name*] *expr*, *expr* [*line_description*]

The **next** command plots the given point using the line style given by *line_name*, or the default if none is given. If *line_name* is given, it should have been defined by an earlier **draw** command, if not a new line style with that name is created, initialized the same way as the default style. The two expressions give the point's x and y values, relative to the optional coordinate system. That system should have been defined by an earlier **coord** command, if not, grap will exit. If the optional *line_description* is given, it overrides the style's default line description. You cannot over-ride the plotting string. To use a different plotting string use the **plot** command.

The coordinates may optionally be enclosed in parentheses: (*expr*, *expr*)

quoted_string [*string_modifiers*] [, *quoted_string* [*string_modifiers*]] ... **at** [*coordinates_name*] *expr*, *expr*

plot *expr* [*format_string*] **at** [*coordinates_name*] *expr*, *expr*

These commands both plot a string at the given point. In the first case the literal strings are stacked above each other. The string_modifiers include the pic justification modifiers (**ljust**, **rjust**, **above**, and **below**), and absolute and relative size modifiers. See the pic documentation for the description of the justification modifiers. **grap** also supports the **aligned** and **unaligned** modifiers which are briefly noted in the description of the **label** command.

The standard defines file includes several macros useful as plot strings, including **bullet**, **square**, and **delta**.

Strings placed by either format of the **plot** command are restricted to being within the frame. This can be overriden by using the **unclipped** attribute, which allows a string to be plotted in or out of the frame. The **-c** and **-C** flags set **unclipped** on all strings, and to prevent a string from being plotted outside the frame when those flags are active, the **clipped** attribute can be used to retore clipping behavior. Though **clipped** or **unclipped** can be applied to any string, it only has meaning for **plot** statements.

size *expr* sets the string size to *expr* points. If *expr* is preceded by a + or -, the size is increased or decreased by that many points.

If **color** and a color name in double quotes appears, the string will be rendered in that color under a version of GNU troff that supports color. Color is not available in compatibility mode.

In the second version, the *expr* is converted to a string and placed on the graph. *format_string* is a `printf`(3) format string. Only formatting escapes for printing floating point numbers make sense. The format string is only respected if the **sprintf** command is also active. See the description of **sprintf** for the various ways to disable it. **Plot** and **sprintf** respond differently when **grap** is running safely. **Sprintf** ignores any arguments, passing the format string through without substitution. **plot** ignores the format string completely, plotting *expr* using the "%g" format.

Points are specified the same way as for **next** commands, with the same consequences for undefined coordinate systems.

The second form of this command is because the first form can be used with a **grap sprintf** expression (See **Expressions**).

ticks (**left**|**right**|**top**|**bottom**)[(**in**|**out**) [*expr*]] [**on**|**auto** *coord_name*]

ticks (**left**|**right**|**top**|**bottom**) (**in**|**out**) [*expr*] [**up** *expr* | **down** *expr* | **left** *expr* | **right** *expr*] **at** [*coord_name*] *expr* [*format_string*] [[, *expr* [*format_string*]] ...]

ticks (**left**|**right**|**top**|**bottom**) (**in**|**out**) [*expr*] [**up** *expr* | **down** *expr* | **left** *expr* | **right** *expr*] **from** [coord_name] *start_expr* **to** *end_expr* [**by** [+|-|*|/] *by_expr*] [format_string]

ticks [**left**|**right**|**top**|**bottom**] **off**

This command controls the placement of ticks on the frame. By default, ticks are automatically generated on the left and bottom sides of the frame.

The first version of this command turns on the automatic tick generation for a given side. The **in** or **out** parameter controls the direction and length of the ticks. If a *coord_name* is specified, the ticks are automatically generated using that coordinate system. If no system is specified, the default coordinate system is used. As with **next** and **plot**, the coordinate system must be declared before the **ticks** statement that references it. This syntax for requesting automatically generated ticks is an extension, and will not port to older **grap** implementations.

The second version of the **ticks** command overrides the automatic placement of the ticks by specifying a list of coordinates at which to place the ticks. If the ticks are not defined with respect to the default coordinate system, the *coord_name* parameter must be given. For each tick a `printf`(3) style format string can be given. The *format_string* defaults to "%g". The format string can also take string modifiers as described in the **plot** command. To place ticks with no labels, specify *format_string* as "".

If **sprintf** is disabled, **ticks** behaves as **plot** with respect to the format string.

The labels on the ticks may be shifted by specifying a direction and the distance in inches to offset the label. That is the optional direction and expression immediately preceding the **at**.

The third format of the **ticks** command over-rides the default tick generation with a set of ticks ar regular intervals. The syntax is reminiscent of programming language for loops. Ticks are placed starting at *start_expr* ending at *end_expr* one unit apart. If the **by** clause is specified, ticks are *by_expr* units apart. If an operator appears before *by_expr* each tick is operated on by that operator instead of +. For example

```
      ticks left out from 2 to 32 by *2
```

will put ticks at 2, 4, 8, 16, and 32. If *format_string* is specified, all ticks are formatted using it.

The parameters preceding the **from** act as described above.

The **at** and **for** forms of tick command may both be issued on the same side of a frame. For example:

```
ticks left out from 2 to 32 by *2
ticks left in 3, 5, 7
```

will put ticks on the left side of the frame pointing out at 2, 4, 8, 16, and 32 and in at 3, 5, and 7.

The final form of **ticks** turns off ticks on a given side. If no side is given the ticks for all sides are cancelled.

tick is a synonym for **ticks**.

grid (**left**|**right**|**top**|**bottom**) [`ticks off`] [*line_description*] [**up** *expr* | **down** *expr* | **left** *expr* | **right** *expr*] [**on**|**auto** [*coord_name*]]

grid (**left**|**right**|**top**|**bottom**) [`ticks off`] [*line_description*] [**up** *expr* | **down** *expr* | **left** *expr* | **right** *expr*] **at** [*coord_name*] *expr* [*format_string*] [[, *expr* [*format_string*]] ...]

grid (**left**|**right**|**top**|**bottom**) [`ticks off`] [*line_description*] [**up** *expr* | **down** *expr* | **left** *expr* | **right** *expr*] **from** [coord_name] *start_expr* **to** *end_expr* [**by** [+|-|*|/] *by_expr*] [format_string]

The **grid** command is similar to the **ticks** command except that **grid** specifies the placement of lines in the frame. The syntax is similar to **ticks** as well.

By specifying `ticks off` in the command, no ticks are drawn on that side of the frame. If ticks appear on a side by default, or have been declared by an earlier **ticks** command, **grid** does not cancel them unless `ticks off` is specified.

Instead of a direction for ticks, **grid** allows the user to pick a line description for the grid lines. The usual `pic` line descriptions are allowed.

Grids are labelled by default. To omit labels, specify the format string as "".

If **sprintf** is disabled, **grid** behaves as **plot** with respect to the format string.

label (**left**|**right**|**top**|**bottom**) *quoted_string* [*string_modifiers*] [, *quoted_string* [*string_modifiers*]] ... [**up** *expr* | **down** *expr* | **left** *expr* | **right** *expr*]

The **label** command places a label on the given axis. It is possible to specify several labels, which will be stacked over each other as in `pic`. The final argument, if present, specifies how many inches the label is shifted from the axis.

By default the labels on the left and right labels run parallel to the frame. You can cancel this by specifying `unaligned` as a *string_modifier*.

circle at [*coordinate_name*] *expr*, *expr* [**radius** *expr*] [*linedesc*]

This draws an circle at the point indicated. By default, the circle is small, 0.025 inches. This can be over-ridden by specifying a radius. The coordinates of the point are relative to the named coordinate system, or the default system if none is specified.

This command has been extended to take a line description, e.g., `dotted`. It also accepts the filling extensions described below in the **bar** command. It will also accept a **color** keyword that gives the color of the outline of the circle in double quotes and a **fillcolor** command that sets the color to fill the circle with similarly. Colors are only available when compatibility mode is off, and using a version of GNU pic that supports color.

line [*line_description*] **from** [*coordinate_name*] *expr*, *expr* **to** [*coordinate_name*] *expr*, *expr* [*line_description*]

arrow [*line_description*] **from** [*coordinate_name*] *expr*, *expr* **to** [*coordinate_name*] *expr*, *expr* [*line_description*]

This draws a line or arrow from the first point to the second using the given style. The default line style is `solid`. The *`line_description`* can be given either before the **`from`** or after the **`to`** clause. If both are given the second is used. It is possible to specify one point in one coordinate system and one in another, note that if both points are in a named coordinate system (even if they are in the same named coordinate system), both points must have *`coordinate_name`* given.

`copy` ["*`filename`*"] [**`until`** "*`string`*"] [**`thru`** *`macro`*]

The **`copy`** command imports data from another file into the current graph. The form with only a filename given is a simple file inclusion; the included file is simply read into the input stream and can contain arbitrary **`grap`** commands. The more common case is that it is a number list; see **Number Lists** below.

The second form takes lines from the file, splits them into words delimited by one or more spaces, and calls the given macro with those words as parameters. The macro may either be defined here, or be a macro defined earlier. See **Macros** for more information on macros.

The *`filename`* may be omitted if the **`until`** clause is present. If so the current file is treated as the input file until *`string`* is encountered at the beginning of the line.

`copy` is one of the workhorses of **`grap`**. Check out the paper and `/usr/local/share/examples/grap` for more details. Confirm the location of the examples directory using the **`-v`** flag.

`print` (*`expr`*|*`string`*)

Prints its argument to the standard error.

`sh` *`block`*

This passes *`block`* to `sh`(1). Unlike K&B **`grap`** no macro or variable expansion is done. I believe that this is also true for GNU `pic` version 1.10. See the **Macros** section for information on defining blocks.

`pic` *`pic_statement`*

This issues the given `pic` statements in the enclosing **`.PS`** and **`.PE`** at the point where the command is issued.

Statements that begin with a period are considered to be `troff`(statements) and are output in the enclosing **`.PS`** and **`.PE`** at the point where the command appears.

For the purposes of relative placement of `pic` or `troff` commands, the frame is output immediately before the first plotted object, or the **`frame`** statement, if any. If the user specifies `pic` or `troff` commands and neither any plotable object nor a **`frame`** command, the commands will not be output.

`graph` *`Name pic_commands`*

This command is used to position graphs with respect to each other. The current graph is given the `pic` name *`Name`* (names used by `pic` begin with capital letters). Any `pic` commands following the graph are used to position the next graph. The frame of the graph is available for use with `pic` name `Frame`. The following places a second graph below the first:

```
graph Linear
[ graph description ]
graph Exponential with .Frame.n at \
        Linear.Frame.s - (0, .05)
[ graph description ]
```

`name = expr`

This assigns *`expr`* to the variable *`name`*. **`grap`** has only numeric (double) variables.

Assignment creates a variable if it does not exist. Variables persist across graphs. Assignments can cascade; `a = b = 35` assigns 35 to `a` and `b`.

bar (**up**|**right**) [*coordinates_name*] *offset* **ht** *height* [**wid** *width*] [**base** *base_offset*] [*line_description*]

bar [*coordinates_name*] *expr*, *expr*, [*coordinates_name*] *expr*, *expr*, [*line_description*]

The **bar** command facilitates drawing bar graphs. The first form of the command describes the bar somewhat generally and has **grap** place it. The bar may extend up or to the right, is centered on *offset* and extends up or right *height* units (in the given coordinate system). For example

```
          bar up 3 ht 2
```

draws a 2 unit high bar sitting on the x axis, centered on x=3. By default bars are 1 unit wide, but this can be changed with the **wid** keyword. By default bars sit on the base axis, i.e., bars directed up will extend from y=0. That may be overridden by the **base** keyword. (The bar described above has corners (2.5, 0) and (3.5, 2).)

The line description has been extended to include a **fill** *expr* keyword that specifies the shading inside the bar. Bars may be drawn in any line style. They support the **color** and **fillcolor** keywords described under **circle**.

The second form of the command draws a box with the two points as corners. This can be used to draw boxes highlighting certain data as well as bar graphs. Note that filled bars will cover data drawn under them.

Control Flow

if *expr* **then** *block* [**else** *block*]

The **if** statement provides simple conditional execution. If *expr* is non-zero, the *block* after the **then** statement is executed. If not the *block* after the **else** is executed, if present. See **Macros** for the definition of blocks. Early versions of this implementation of **grap** treated the blocks as macros that were defined and expanded in place. This led to unnecessary confusion because explicit separators were sometimes called for. Now, **grap** inserts a separator (;) after the last character in *block*, so constructs like

```
if (x == 3) { y = y + 1 }
x = x + 1
```

behave as expected. A separator is also appended to the end of a **for** block.

for *name* **from** *from_expr* **to** *to_expr* [**by** [+|-|*|/] *by_expr*] **do** *block*

This command executes *block* iteratively. The variable *name* is set to *from_expr* and incremented by *by_expr* until it exceeds *to_expr*. The iteration has the semantics defined in the **ticks** command. The definition of *block* is discussed in **Marcos**. See also the note about implicit separators in the description of the **if** command.

An **=** can be used in place of **from**.

Expressions

grap supports most standard arithmetic operators: + - / * ^. The carat (^) is exponentiation. In an **if** statement **grap** also supports the C logical operators ==, !=, &&, || and unary !. Also in an **if**, == and != are overloaded for the comparison of quoted strings. Parentheses are used for grouping.

Assignment is not allowed in an expression in any context, except for simple cascading of assignments. `a = b = 35` works as expected; `a = 3.5 * (b = 10)` does not execute.

grap supports the following functions that take one argument: **log**, **exp**, **int**, **sin**, **cos**, **sqrt**, **rand**. The logarithms are base 10 and the trigonometric functions are in radians. **eexp** returns Euler's number to the given power and **ln** returns the natural logarithm. The natural log and exponentiation functions are extensions and are probably not available in other **grap** implementations.

rand returns a random number uniformly distributed on [0,1). The following two-argument functions are supported: **atan2**, **min**, **max**. **atan2** works just like atan2(3). The random number generator can be seeded by calling **srand** with a single parameter (converted internally to an integer). Because its return value is of no use, you must use **srand** as a separate statement, it is not part of a valid expression. **srand** is not portable.

The **getpid** function takes no arguments and returns the process id. This may be used to seed the random number generator, but do not expect cryptographically random values to result.

Other than string comparison, no expressions can use strings. One string valued function exists: **sprintf** (*format*, [*expr* [, *expr*]]). It operates like sprintf(3), except returning the value. It can be used anywhere a quoted string is used. If **grap** is run with **-S**, the environment variable GRAP_SAFER is defined, or **grap** has been compiled for safer operation, the **sprintf** command will return the format string. This mode of operation is only intended to be used only if **grap** is being used as part of a super-user enabled print system.

Macros

grap has a simple but powerful macro facility. Macros are defined using the **define** command :

define *name block*
undefine *name*

Every occurrence of *name* in the program text is replaced by the contents of *block*. *block* is defined by a series of statements in nested { }'s, or a series of statements surrounded by the same letter. An example of the latter is

```
define foo  X coord x 1,3 X
```

Each time foo appears in the text, it will be replaced by coord x 1,3. Macros are literal, and can contain newlines. If a macro does not span multiple lines, it should end in a semicolon to avoid parsing errors.

Macros can take parameters, too. If a macro call is followed by a parenthesized, comma-separated list the values starting with $1 will be replaced in the macro with the elements of the list. A $ not followed by a digit is left unchanged. This parsing is very rudimentary; no nesting or parentheses or escaping of commas is allowed. Also, there is no way to say argument 1 followed by a digit (${1}0 in sh(1)).

The following will draw a line with slope 1.

```
define foo { next at $1, $2 }
for i from 1 to 5 { foo(i,i) }
```

Macros persist across graphs. The file /usr/local/share/grap/grap.defines contains simple macros for plotting common characters. The **undefine** command deletes a macro.

See the directory /usr/local/share/examples/grap for more examples of macros. Confirm the location of the examples directory using the **-v** flag.

Number Lists

A whitespace-separated list of numbers is treated specially. The list is taken to be points to be plotted using the default line style on the default coordinate system. If more than two numbers are given, the extra numbers are taken to be additional y values to plot at the first x value. Number lists in DWB **grap** can be comma-separated, and this **grap** supports that as well. More precisely, numbers in number lists can be separated by either whitespace, commas, or both.

```
1 2 3
4 5 6
```

Will plot points using the default line style at (1,2), (1,3),(4,5) and (4,6). A simple way to plot a set of numbers in a file named `./data` is:

```
.G1
copy "./data"
.G2
```

Pic Macros

grap defines pic macros that can be used in embedded pic code to place elements in the graph. The macros are **x_gg**, **y_gg**, and **xy_gg**. These macros define pic distances that correspond to the given argument. They can be used to size boxes or to plot pic constructs on the graph. To place a given construct on the graph, you should add Frame.Origin to it. Other coordinate spaces can be used by replacing **gg** with the name of the coordinate space. A coordinate space named **gg** cannot be reliably accessed by these macros.

The macros are emitted immediately before the frame is drawn.

DWB **grap** may use these as part of its implementation. This **grap** provides them only for compatibility. Note that these are very simple macros, and may not do what you expect under complex conditions.

ENVIRONMENT VARIABLES

If the environment variable `GRAP_DEFINES` is defined, **grap** will look for its defines file there. If that value is a relative path name the path specified in the **-M** option will be searched for it. `GRAP_DEFINES` overrides the compiled in location of the defines file, but may be overridden by the **-d** or **-D** flags.

If `GRAP_SAFER` is set, **sprintf** is disabled to prevent forcing **grap** to core dump or smash the stack.

FILES

`/usr/local/share/grap/grap.defines`

SEE ALSO

`atan2`(3), `groff`(1), `pic`(1), `printf`(3), `sh`(1), `sprintf`(3), `troff`(1)

If documentation and examples have been installed, **grap --version** or **grap --help** will display the locations.

BUGS

There are several small incompatibilities with K&R **grap**. They include the **sh** command not expanding variables and macros, and a more strict adherence to parameter order in the internal commands.

Although much improved, the error reporting code can still be confused. Notably, an error in a macro is not detected until the macro is used, and it produces unusual output in the error message.

Iterating many times over a macro with no newlines can run **grap** out of memory.

AUTHOR

This implementation was done by Ted Faber ⟨faber@lunabase.org⟩. Bruce Lilly ⟨blilly@erols.com⟩ contributed many bug fixes, including a considerable revamp of the error reporting code. If you can actually find an error in your **grap** code, you can probably thank him. **grap** was designed and specified by Brian Kernighan and Jon Bentley.

GRAP2GRAPH

NAME

grap2graph − convert a grap diagram into a cropped bitmap image

SYNOPSIS

grap2graph [**−unsafe**] [**−resolution** *M*|*MxN*] [**−format** *fmt*]

DESCRIPTION

Reads a grap program as input; produces an image file (by default in Portable Network Graphics format) suitable for the Web as output. For a description of the grap language, see **grap**(1).

Your graph specification should *not* be wrapped with the .G1 and .G2 macros that normally guard it within **groff**(1) macros.

The output image will be a black-on-white graphic clipped to the smallest possible bounding box that contains all the black pixels. By specifying command-line options to be passed to **convert**(1) you can give it a border, set the background transparent, set the image's pixel density, or perform other useful transformations.

This program uses **grap**(1), **pic**(1), **groff**(1), and the ImageMagick **convert**(1) program. These programs must be installed on your system and accessible on your $PATH for **grap2graph** to work.

OPTIONS

−unsafe

Run **pic**(1) and **groff**(1) in the 'unsafe' mode enabling the PIC macro **sh** to execute arbitrary commands. The default is to forbid this.

−format *fmt*

Specify an output format; the default is PNG (Portable Network Graphics). Any format that **convert**(1) can emit is supported.

Command-line switches and arguments not listed above are passed to **convert**(1).

ENVIRONMENT

GROFF_TMPDIR

The directory in which temporary files will be created. If this is not set **grap2graph** searches the environment variables **TMPDIR**, **TMP**, and **TEMP** (in that order). Otherwise, temporary files will be created in **/tmp**.

SEE ALSO

pic2graph(1), **eqn2graph**(1), **pic**(1), **groff**(1), **gs**(1), **convert**(1).

AUTHOR

Eric S. Raymond <esr@thyrsus.com>

GRN

NAME

grn − groff preprocessor for gremlin files

SYNOPSIS

grn [**−Cv**] [**−T***dev*] [**−M***dir*] [**−F***dir*] [*file...*]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

grn is a preprocessor for including *gremlin* pictures in *groff* input. *grn* writes to standard output, processing only input lines between two that start with **.GS** and **.GE.** Those lines must contain *grn* commands (see below). These commands request a *gremlin* file, and the picture in that file is converted and placed in the *troff* input stream. The **.GS** request may be followed by a C, L, or R to center, left, or right justify the whole *gremlin* picture (default justification is center). If no *file* is mentioned, the standard input is read. At the end of the picture, the position on the page is the bottom of the *gremlin* picture. If the *grn* entry is ended with **.GF** instead of **.GE**, the position is left at the top of the picture.

Please note that currently only the −me macro package has support for **.GS**, **.GE**, and **.GF**.

The following command-line options are understood:

−T*dev* Prepare output for printer *dev*. The default device is **ps**. See **groff**(1) for acceptable devices.

−M*dir* Prepend *dir* to the default search path for *gremlin* files. The default path is (in that order) the current directory, the home directory, **c:/progra~1/groff/lib/groff/site-tmac**, **c:/progra~1/groff/share/groff/site-tmac**, and **c:/progra~1/groff/share/groff/1.20/tmac**.

−F*dir* Search *dir* for subdirectories **dev***name* (*name* is the name of the device) for the **DESC** file before the default font directories **c:/progra~1/groff/share/groff/site-font**, **c:/progra~1/groff/share/groff/1.20/font**, and **/usr/lib/font**.

−C Recognize **.GS** and **.GE** (and **.GF**) even when followed by a character other than space or newline.

−v Print the version number.

GRN COMMANDS

Each input line between **.GS** and **.GE** may have one *grn* command. Commands consist of one or two strings separated by white space, the first string being the command and the second its operand. Commands may be upper or lower case and abbreviated down to one character.

Commands that affect a picture's environment (those listed before **default**, see below) are only in effect for the current picture: The environment is reinitialized to the defaults at the start of the next picture. The commands are as follows:

1 *N*
2 *N*
3 *N*
4 *N* Set *gremlin*'s text size number 1 (2, 3, or 4) to *N* points. The default is 12 (16, 24, and 36, respectively).

roman *f*
italics *f*
bold *f*
special *f*
Set the roman (italics, bold, or special) font to *troff*'s font *f* (either a name or number). The default is R (I, B, and S, respectively).

l *f*
stipple *f*
Set the stipple font to *troff*'s stipple font *f* (name or number). The command **stipple** may be abbreviated down as far as 'st' (to avoid confusion with **special**). There is *no* default for stipples (unless one is set by the default command), and it is invalid to include a *gremlin* picture with polygons without specifying a stipple font.

x *N*

scale *N* Magnify the picture (in addition to any default magnification) by *N*, a floating point number larger than zero. The command **scale** may be abbreviated down to 'sc'.

narrow *N*
medium *N*
thick *N*
Set the thickness of *gremlin*'s narrow (medium and thick, respectively) lines to *N* times 0.15pt (this value can be changed at compile time). The default is 1.0 (3.0 and 5.0, respectively), which corresponds to 0.15pt (0.45pt and 0.75pt, respectively). A thickness value of zero selects the smallest available line thickness. Negative values cause the line thickness to be proportional to the current point size.

pointscale <*off/on*>
Scale text to match the picture. Gremlin text is usually printed in the point size specified with the commands **1**, **2**, **3**, or **4**, regardless of any scaling factors in the picture. Setting **pointscale** will cause the point sizes to scale with the picture (within *troff*'s limitations, of course). An operand of anything but *off* will turn text scaling on.

default Reset the picture environment defaults to the settings in the current picture. This is meant to be used as a global parameter setting mechanism at the beginning of the *troff* input file, but can be used at any time to reset the default settings.

width *N*
Forces the picture to be *N* inches wide. This overrides any scaling factors present in the same picture. '**width** *0*' is ignored.

height *N*
Forces picture to be *N* inches high, overriding other scaling factors. If both 'width' and 'height' are specified the tighter constraint will determine the scale of the picture. **Height** and **width** commands are not saved with a **default** command. They will, however, affect point size scaling if that option is set.

file *name*
Get picture from *gremlin* file *name* located the current directory (or in the library directory; see the **−M** option above). If two **file** commands are given, the second one overrides the first. If *name* doesn't exist, an error message is reported and processing continues from the **.GE** line.

NOTES ABOUT GROFF

Since *grn* is a preprocessor, it doesn't know about current indents, point sizes, margins, number registers, etc. Consequently, no *troff* input can be placed between the **.GS** and **.GE** requests. However, *gremlin* text is now processed by *troff*, so anything valid in a single line of *troff* input is valid in a line of *gremlin* text (barring '.' directives at the beginning of a line). Thus, it is possible to have equations within a *gremlin* figure by including in the *gremlin* file *eqn* expressions enclosed by previously defined delimiters (e.g. *$$*).

When using *grn* along with other preprocessors, it is best to run *tbl* before *grn*, *pic*, and/or *ideal* to avoid overworking *tbl*. *Eqn* should always be run last.

A picture is considered an entity, but that doesn't stop *troff* from trying to break it up if it falls off the end of a page. Placing the picture between 'keeps' in −me macros will ensure proper placement.

grn uses *troff*'s number registers **g1** through **g9** and sets registers **g1** and **g2** to the width and height of the *gremlin* figure (in device units) before entering the **.GS** request (this is for those who want to rewrite these macros).

GREMLIN FILE FORMAT

There exist two distinct *gremlin* file formats, the original format from the *AED* graphic terminal version, and the *SUN* or *X11* version. An extension to the *SUN/X11* version allowing reference points with negative coordinates is **not** compatible with the *AED* version. As long as a *gremlin* file does not contain negative coordinates, either format will be read correctly by either version of *gremlin* or *grn*. The other difference to the *SUN/X11* format is the use of names for picture objects (e.g., POLYGON, CURVE) instead of numbers. Files representing the same picture are shown in Table 1 in each format.

sungremlinfile	gremlinfile

```
0 240.00 128.00     0 240.00 128.00
CENTCENT            2
240.00 128.00       240.00 128.00
185.00 120.00       185.00 120.00
240.00 120.00       240.00 120.00
296.00 120.00       296.00 120.00
*                   -1.00 -1.00
2 3                 2 3
10 A Triangle       10 A Triangle
POLYGON             6
224.00 416.00       224.00 416.00
96.00 160.00        96.00 160.00
384.00 160.00       384.00 160.00
*                   -1.00 -1.00
5 1                 5 1
0                   0
-1                  -1
```

Table 1. File examples

- The first line of each *gremlin* file contains either the string **gremlinfile** (*AED* version) or **sungremlinfile** (*SUN/X11*)
- The second line of the file contains an orientation, and **x** and **y** values for a positioning point, separated by spaces. The orientation, either **0** or **1**, is ignored by the *SUN/X11* version. **0** means that *gremlin* will display things in horizontal format (drawing area wider than it is tall, with menu across top). **1** means that *gremlin* will display things in vertical format (drawing area taller than it is wide, with menu on left side). **x** and **y** are floating point values giving a positioning point to be used when this file is read into another file. The stuff on this line really isn't all that important; a value of "1 0.00 0.00" is suggested.
- The rest of the file consists of zero or more element specifications. After the last element specification is a line containing the string "-1".
- Lines longer than 127 characters are chopped to this limit.

ELEMENT SPECIFICATIONS

- The first line of each element contains a single decimal number giving the type of the element (*AED* version) or its ASCII name (*SUN/X11* version). See Table 2.

gremlin File Format – Object Type Specification

AED Number	*SUN/X11* Name	Description
0	BOTLEFT	bottom-left-justified text
1	BOTRIGHT	bottom-right-justified text
2	CENTCENT	center-justified text
3	VECTOR	vector
4	ARC	arc
5	CURVE	curve
6	POLYGON	polygon
7	BSPLINE	b-spline
8	BEZIER	Bézier
10	TOPLEFT	top-left-justified text
11	TOPCENT	top-center-justified text
12	TOPRIGHT	top-right-justified text
13	CENTLEFT	left-center-justified text
14	CENTRIGHT	right-center-justified text
15	BOTCENT	bottom-center-justified text

Table 2.

Type Specifications in *gremlin* Files

- After the object type comes a variable number of lines, each specifying a point used to display the element. Each line contains an x-coordinate and a y-coordinate in floating point format, separated by spaces. The list of points is terminated by a line containing the string "-1.0 -1.0" (*AED* version) or a single asterisk, "*" (*SUN/X11* version).
- After the points comes a line containing two decimal values, giving the brush and size for the element. The brush determines the style in which things are drawn. For vectors, arcs, and curves there are six valid brush values:

1 –	thin dotted lines
2 –	thin dot-dashed lines
3 –	thick solid lines
4 –	thin dashed lines
5 –	thin solid lines
6 –	medium solid lines

For polygons, one more value, 0, is valid. It specifies a polygon with an invisible border. For text, the brush selects a font as follows:

1 –	roman (R font in groff)
2 –	italics (I font in groff)
3 –	bold (B font in groff)
4 –	special (S font in groff)

If you're using *grn* to run your pictures through *groff*, the font is really just a starting font: The text string can contain formatting sequences like "\fI" or "\d" which may change the font (as well as do many other things). For text, the size field is a decimal value between 1 and 4. It selects the size of the font in which the text will be drawn. For polygons, this size field is interpreted as a stipple number to fill the polygon with. The number is used to index into a stipple font at print time.

- The last line of each element contains a decimal number and a string of characters, separated by a single space. The number is a count of the number of characters in the string. This information is only used for text elements, and contains the text string. There can be spaces inside the text. For arcs, curves, and vectors, this line of the element contains the string "0".

NOTES ON COORDINATES

gremlin was designed for *AED*s, and its coordinates reflect the *AED* coordinate space. For vertical pictures, x-values range 116 to 511, and y-values from 0 to 483. For horizontal pictures, x-values range from 0 to 511 and y-values range from 0 to 367. Although you needn't absolutely stick to this range, you'll get best results if you at least stay in this vicinity. Also, point lists are terminated by a point of (-1, -1), so you shouldn't ever use negative coordinates. *gremlin* writes out coordinates using format "%f1.2"; it's probably a good idea to use the same format if you want to modify the *grn* code.

NOTES ON SUN/X11 COORDINATES

There is no longer a restriction on the range of coordinates used to create objects in the *SUN/X11* version of *gremlin*. However, files with negative coordinates **will** cause problems if displayed on the *AED*.

FILES

c:/progra~1/groff/share/groff/1.20/font/dev*name***/DESC**
Device description file for device *name*.

SEE ALSO

gremlin(1), **groff**(1), **pic**(1), **ideal**(1)

HISTORY

David Slattengren and Barry Roitblat wrote the original Berkeley *grn*.

Daniel Senderowicz and Werner Lemberg modified it for *groff*.

GRODVI

NAME

grodvi – convert groff output to TeX dvi format

SYNOPSIS

grodvi [**–dlv**] [**–F***dir*] [**–p***papersize*] [**–w***n*] [*files* ...]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

grodvi is a driver for **groff** that produces TeX dvi format. Normally it should be run by **groff –Tdvi**. This will run **troff –Tdvi**; it will also input the macros in **c:/progra~1/groff/share/groff/1.20/tmac/dvi.tmac**.

The dvi file generated by **grodvi** can be printed by any correctly-written dvi driver. The troff drawing primitives are implemented using the tpic version 2 specials. If the driver does not support these, the **\D** commands will not produce any output.

There is an additional drawing command available:

\D'R *dh dv***'**
: Draw a rule (solid black rectangle), with one corner at the current position, and the diagonally opposite corner at the current position +(*dh*,*dv*). Afterwards the current position will be at the opposite corner. This produces a rule in the dvi file and so can be printed even with a driver that does not support the tpic specials unlike the other **\D** commands.

The groff command **\X'***anything***'** is translated into the same command in the dvi file as would be produced by **\special{***anything***}** in TeX; *anything* may not contain a newline.

For inclusion of EPS image files, **–Tdvi** loads **pspic.tmac** automatically, providing the **PSPIC** macro. Please check **groff_tmac**(5) for a detailed description.

Font files for **grodvi** can be created from tfm files using **tfmtodit**(1). The font description file should contain the following additional commands:

internalname *name*
: The name of the tfm file (without the **.tfm** extension) is *name*.

checksum *n* The checksum in the tfm file is *n*.

designsize *n* The designsize in the tfm file is *n*.

These are automatically generated by **tfmtodit.**

The default color for **\m** and **\M** is black. Currently, the drawing color for **\D** commands is always black, and fill color values are translated to gray.

In **troff** the **\N** escape sequence can be used to access characters by their position in the corresponding tfm file; all characters in the tfm file can be accessed this way.

By design, the DVI format doesn't care about physical dimensions of the output medium. Instead, **grodvi** emits the equivalent to TeX's **\special{papersize=***width***,***length***}** on the first page; **dvips** (and possibly other DVI drivers) then sets the page size accordingly. If either the page width or length is not positive, no papersize special is output.

OPTIONS

–d Do not use tpic specials to implement drawing commands. Horizontal and vertical lines will be implemented by rules. Other drawing commands will be ignored.

–F*dir* Prepend directory *dir***/dev***name* to the search path for font and device description files; *name* is the name of the device, usually **dvi**.

–l Specify landscape orientation.

–p*papersize*
: Specify paper dimensions. This overrides the **papersize**, **paperlength**, and **paperwidth** commands in the **DESC** file; it accepts the same arguments as the **papersize** command (see **groff_font**(5) for details).

–v Print the version number.

−w*n* Set the default line thickness to *n* thousandths of an em. If this option isn’t specified, the line thickness defaults to 0.04 em.

USAGE

There are styles called **R**, **I**, **B**, and **BI** mounted at font positions 1 to 4. The fonts are grouped into families **T** and **H** having members in each of these styles:

TR	CM Roman (cmr10)
TI	CM Text Italic (cmti10)
TB	CM Bold Extended Roman (cmbx10)
TBI	CM Bold Extended Text Italic (cmbxti10)
HR	CM Sans Serif (cmss10)
HI	CM Slanted Sans Serif (cmssi10)
HB	CM Sans Serif Bold Extended (cmssbx10)
HBI	CM Slanted Sans Serif Bold Extended (cmssbxo10)

There are also the following fonts which are not members of a family:

CW	CM Typewriter Text (cmtt10)
CWI	CM Italic Typewriter Text (cmitt10)

Special fonts are **MI** (cmmi10), **S** (cmsy10), **EX** (cmex10), **SC** (cmtex10, only for **CW**), and, perhaps surprisingly, **TR**, **TI**, and **CW**, due to the different font encodings of text fonts. For italic fonts, **CWI** is used instead of **CW**.

Finally, the symbol fonts of the American Mathematical Society are available as special fonts **SA** (msam10) and **SB** (msbm10). These two fonts are not mounted by default.

Using the option **−mec** (which loads the file **ec.tmac**) provides the EC and TC fonts. The design of the EC family is very similar to that of the CM fonts; additionally, they give a much better coverage of groff symbols. Note that **ec.tmac** must be called before any language-specific files; it doesn’t take care of hcode values.

ENVIRONMENT

GROFF_FONT_PATH
A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

FILES

c:/progra˜1/groff/share/groff/1.20/font/devdvi/DESC
Device description file.

c:/progra˜1/groff/share/groff/1.20/font/devdvi/*F*
Font description file for font *F*.

c:/progra˜1/groff/share/groff/1.20/tmac/dvi.tmac
Macros for use with **grodvi**.

c:/progra˜1/groff/share/groff/1.20/tmac/ec.tmac
Macros to switch to EC fonts.

BUGS

Dvi files produced by **grodvi** use a different resolution (57816 units per inch) to those produced by TEX. Incorrectly written drivers which assume the resolution used by TEX, rather than using the resolution specified in the dvi file will not work with **grodvi**.

When using the **−d** option with boxed tables, vertical and horizontal lines can sometimes protrude by one pixel. This is a consequence of the way TEX requires that the heights and widths of rules be rounded.

SEE ALSO

tfmtodit(1), **groff**(1), **troff**(1), **groff_out**(5), **groff_font**(5), **groff_char**(7), **groff_tmac**(5)

GROFF

NAME

groff – front-end for the groff document formatting system

SYNOPSIS

[**–abcegiklpstzCEGNRSUVXZ**] [**–d***cs*] [**–D***arg*] [**–f***fam*] [**–F***dir*] [**–I***dir*] [**–K***arg*] [**–L***arg*] [**–m***name*] [**–M***dir*] [**–n***num*] [**–o***list*] [**–P***arg*] [**–r***cn*] [**–T***dev*] [**–w***name*] [**–W***name*] [*file* ...] **–h** | **––help** **–v** | **––version** [*option* ...]

DESCRIPTION

This document describes the **groff** program, the main front-end for the *groff* document formatting system. The *groff* program and macro suite is the implementation of a **roff**(7) system within the free software collection GNU The *groff* system has all features of the classical *roff*, but adds many extensions.

The **groff** program allows to control the whole *groff* system by command line options. This is a great simplification in comparison to the classical case (which uses pipes only).

OPTIONS

The command line is parsed according to the usual GNU convention. The whitespace between a command line option and its argument is optional. Options can be grouped behind a single '–' (minus character). A filename of **–** (minus character) denotes the standard input.

As **groff** is a wrapper program for **troff** both programs share a set of options. But the **groff** program has some additional, native options and gives a new meaning to some **troff** options. On the other hand, not all **troff** options can be fed into **groff**.

Native groff Options

The following options either do not exist for **troff** or are differently interpreted by **groff**.

–D *arg* Set default input encoding used by **preconv** to *arg*. Implies **–k**.

–e Preprocess with **eqn**.

–g Preprocess with **grn**.

–G Preprocess with **grap**.

–h
––help Print a help message.

–I *dir* This option may be used to specify a directory to search for files (both those on the command line and those named in **.psbb** and **.so** requests, and **\X'ps: import'** and **\X'ps: file'** escapes). The current directory is always searched first. This option may be specified more than once; the directories are searched in the order specified. No directory search is performed for files specified using an absolute path. This option implies the **–s** option.

–k Preprocess with **preconv**. This is run before any other preprocessor. Please refer to **preconv**'s manual page for its behaviour if no **–K** (or **–D**) option is specified.

–K *arg* Set input encoding used by **preconv** to *arg*. Implies **–k**.

–l Send the output to a spooler program for printing. The command that should be used for this is specified by the **print** command in the device description file, see **groff_font**(5). If this command is not present, the output is piped into the **lpr**(1) program by default. See options **–L** and **–X**.

–L *arg* Pass *arg* to the spooler program. Several arguments should be passed with a separate -L option each. Note that **groff** does not prepend '-' (a minus sign) to *arg* before passing it to the spooler program.

–N Don't allow newlines within *eqn* delimiters. This is the same as the **–N** option in **eqn**.

–p Preprocess with **pic**.

–P *–option*
–P *–option* **–P** *arg*
Pass *–option* or *–option arg* to the postprocessor. The option must be specified with the necessary preceding minus sign(s) `'-'` or `'--'` because **groff** does not prepend any dashes before passing it to the postprocessor. For example, to pass a title to the **gxditview** postprocessor, the shell command

```
groff -X -P -title -P 'groff it' foo
```

is equivalent to

```
groff -X -Z foo | gxditview -title 'groff it' -
```

–R Preprocess with **refer**. No mechanism is provided for passing arguments to **refer** because most **refer** options have equivalent language elements that can be specified within the document. See **refer**(1) for more details.

–s Preprocess with **soelim**.

–S Safer mode. Pass the **–S** option to **pic** and disable the following **troff** requests: **.open**, **.opena**, **.pso**, **.sy**, and **.pi**. For security reasons, safer mode is enabled by default.

–t Preprocess with **tbl**.

–T *dev* Set output device to *dev*. For this device, **troff** generates the *intermediate output*; see **groff_out**(5). Then **groff** calls a postprocessor to convert **troff**'s *intermediate output* to its final format. Real devices in **groff** are

dvi	TeX DVI format (postprocessor is **grodvi**).
html xhtml	HTML and XHTML output (preprocessors are **soelim** and **pre-grohtml**, postprocessor is **post-grohtml**).
lbp	Canon CAPSL printers (LBP-4 and LBP-8 series laser printers; postprocessor is **grolbp**).
lj4	HP LaserJet4 compatible (or other PCL5 compatible) printers (postprocessor is **grolj4**).
ps	PostScript output (postprocessor is **grops**).

For the following TTY output devices (postprocessor is always **grotty**), **–T** selects the output encoding:

ascii	7bit ASCII.
cp1047	Latin-1 character set for EBCDIC hosts.
latin1	ISO 8859-1.
utf8	Unicode character set in UTF-8 encoding.

The following arguments select **gxditview** as the 'postprocessor' (it is rather a viewing program):

X75	75 dpi resolution, 10 pt document base font.
X75-12	75 dpi resolution, 12 pt document base font.
X100	100 dpi resolution, 10 pt document base font.
X100-12	100 dpi resolution, 12 pt document base font.

The default device is **ps**.

–U Unsafe mode. Reverts to the (old) unsafe behaviour; see option **–S**.

–v
––version
Output version information of **groff** and of all programs that are run by it; that is, the given command line is parsed in the usual way, passing **–v** to all subprograms.

–V Output the pipeline that would be run by **groff** (as a wrapper program) on the standard output, but do not execute it. If given more than once, the commands are both printed on the standard error and run.

–X Use **gxditview** instead of using the usual postprocessor to (pre)view a document. The printing spooler behavior as outlined with options **–l** and **–L** is carried over to **gxditview**(1) by determining an argument for the **–printCommand** option of **gxditview**(1). This sets the default **Print** action and the corresponding menu entry to that value. **–X** only produces good results with **−Tps**, **−TX75**, **−TX75-12**, **−TX100**, and **−TX100-12**. The default resolution for previewing **−Tps** output is 75 dpi; this can be changed by passing the **−resolution** option to **gxditview**, for example

```
groff -X -P-resolution -P100 -man foo.1
```

−z Suppress output generated by **troff**. Only error messages are printed.

−Z Do not automatically postprocess *groff intermediate output* in the usual manner. This will cause the **troff** *output* to appear on standard output, replacing the usual postprocessor output; see **groff_out**(5).

Transparent Options

The following options are transparently handed over to the formatter program **troff** that is called by **groff** subsequently. These options are described in more detail in **troff**(1).

−a ASCII approximation of output.

−b Backtrace on error or warning.

−c Disable color output. Please consult the **grotty**(1) man page for more details.

−C Enable compatibility mode.

−d *cs*
−d *name*=*s*
Define string.

−E Disable **troff** error messages.

−f *fam* Set default font family.

−F *dir* Set path for font DESC files.

−i Process standard input after the specified input files.

−m *name*
Include macro file *name*.**tmac** (or **tmac.***name*); see also **groff_tmac**(5).

−M *dir* Path for macro files.

−n *num*
Number the first page *num*.

−o *list* Output only pages in *list*.

−r *cn*
−r *name*=*n*
Set number register.

−w *name*
Enable warning *name*.

−W *name*
disable warning *name*.

USING GROFF

The *groff system* implements the infrastructure of classical roff; see **roff**(7) for a survey on how a *roff* system works in general. Due to the front-end programs available within the *groff* system, using *groff* is much easier than *classical roff*. This section gives an overview of the parts that constitute the *groff* system. It complements **roff**(7) with *groff*-specific features. This section can be regarded as a guide to the documentation around the *groff* system.

Paper Size

The *virtual* paper size used by **troff** to format the input is controlled globally with the requests **.po**, **.pl**, and **.ll**. See **groff_tmac**(5) for the ‘papersize’ macro package which provides a convenient interface.

The *physical* paper size, giving the actual dimensions of the paper sheets, is controlled by output devices like **grops** with the command line options **−p** and **−l**. See **groff_font**(5) and the man pages of the output devices for more details. **groff** uses the command line option **−P** to pass options to output devices; for example, the following selects A4 paper in landscape orientation for the PS device:

```
groff -Tps -P-pa4 -P-l ...
```

Front-ends

The **groff** program is a wrapper around the **troff**(1) program. It allows to specify the preprocessors by command line options and automatically runs the postprocessor that is appropriate for the selected device. Doing so, the sometimes tedious piping mechanism of classical **roff**(7) can be avoided.

The **grog**(1) program can be used for guessing the correct *groff* command line to format a file.

The **groffer**(1) program is an allround-viewer for *groff* files and man pages.

Preprocessors

The *groff* preprocessors are reimplementations of the classical preprocessors with moderate extensions. The standard preprocessors distributed with the *groff* package are

eqn(1)	for mathematical formulæ,
grn(1)	for including **gremlin**(1) pictures,
pic(1)	for drawing diagrams,
chem(1)	for chemical structure diagrams,
refer(1)	for bibliographic references,
soelim(1)	for including macro files from standard locations,

and

tbl(1)	for tables.

A new preprocessor not available in classical *troff* is **preconv**(1) which converts various input encodings to something **groff** can understand. It is always run first before any other preprocessor.

Besides these, there are some internal preprocessors that are automatically run with some devices. These aren't visible to the user.

Macro Packages

Macro packages can be included by option **−m**. The *groff* system implements and extends all classical macro packages in a compatible way and adds some packages of its own. Actually, the following macro packages come with *groff*:

man	The traditional man page format; see **groff_man**(7). It can be specified on the command line as **−man** or **−m man**.
mandoc	The general package for man pages; it automatically recognizes whether the documents uses the *man* or the *mdoc* format and branches to the corresponding macro package. It can be specified on the command line as **−mandoc** or **−m mandoc**.
mdoc	The BSD-style man page format; see **groff_mdoc**(7). It can be specified on the command line as **−mdoc** or **−m mdoc**.
me	The classical *me* document format; see **groff_me**(7). It can be specified on the command line as **−me** or **−m me**.
mm	The classical *mm* document format; see **groff_mm**(7). It can be specified on the command line as **−mm** or **−m mm**.
ms	The classical *ms* document format; see **groff_ms**(7). It can be specified on the command line as **−ms** or **−m ms**.
www	HTML-like macros for inclusion in arbitrary *groff* documents; see **groff_www**(7).

Details on the naming of macro files and their placement can be found in **groff_tmac**(5); this man page also documents some other, minor auxiliary macro packages not mentioned here.

Programming Language

General concepts common to all *roff* programming languages are described in **roff**(7).

The *groff* extensions to the classical *troff* language are documented in **groff_diff**(7).

The *groff* language as a whole is described in the (still incomplete) *groff info file*; a short (but complete) reference can be found in **groff**(7).

Formatters

The central *roff* formatter within the *groff* system is **troff**(1). It provides the features of both the classical *troff* and *nroff*, as well as the *groff* extensions. The command line option **−C** switches **troff** into *compatibility mode* which tries to emulate classical *roff* as much as possible.

There is a shell script **nroff**(1) that emulates the behavior of classical **nroff**. It tries to automatically select the proper output encoding, according to the current locale.

The formatter program generates *intermediate output*; see **groff_out**(7).

Devices

In *roff*, the output targets are called *devices*. A device can be a piece of hardware, e.g., a printer, or a software file format. A device is specified by the option **−T**. The *groff* devices are as follows.

ascii	Text output using the **ascii**(7) character set.
cp1047	Text output using the EBCDIC code page IBM cp1047 (e.g., OS/390 Unix).
dvi	TeX DVI format.
html	HTML output.
latin1	Text output using the ISO Latin-1 (ISO 8859-1) character set; see **iso_8859_1**(7).
lbp	Output for Canon CAPSL printers (LBP-4 and LBP-8 series laser printers).
lj4	HP LaserJet4-compatible (or other PCL5-compatible) printers.
ps	PostScript output; suitable for printers and previewers like **gv**(1).
utf8	Text output using the Unicode (ISO 10646) character set with UTF-8 encoding; see **unicode**(7).
xhtml	XHTML output.
X75	75dpi X Window System output suitable for the previewers **xditview**(1x) and **gxditview**(1). A variant for a 12 pt document base font is **X75-12**.
X100	100dpi X Window System output suitable for the previewers **xditview**(1x) and **gxditview**(1). A variant for a 12 pt document base font is **X100-12**.

The postprocessor to be used for a device is specified by the **postpro** command in the device description file; see **groff_font**(5). This can be overridden with the **-X** option.

The default device is **ps**.

Postprocessors

groff provides 3 hardware postprocessors:

grolbp(1)
for some Canon printers,

grolj4(1)
for printers compatible to the HP LaserJet 4 and PCL5,

grotty(1)
for text output using various encodings, e.g., on text-oriented terminals or line-printers.

Today, most printing or drawing hardware is handled by the operating system, by device drivers, or by software interfaces, usually accepting PostScript. Consequently, there isn't an urgent need for more hardware device postprocessors.

The *groff* software devices for conversion into other document file formats are

grodvi(1)
for the DVI format,

grohtml(1)
for HTML and XHTML formats,

grops(1)
: for PostScript.

Combined with the many existing free conversion tools this should be sufficient to convert a *troff* document into virtually any existing data format.

Utilities

The following utility programs around *groff* are available.

addftinfo(1)
: Add information to *troff* font description files for use with *groff*.

afmtodit(1)
: Create font description files for PostScript device.

eqn2graph(1)
: Convert an **eqn** image into a cropped image.

gdiffmk(1)
: Mark differences between *groff*, *nroff*, or *troff* files.

grap2graph(1)
: Convert a **grap** diagram into a cropped bitmap image.

groffer(1)
: General viewer program for *groff* files and man pages.

gxditview(1)
: The *groff* X viewer, the GNU version of **xditview**.

hpftodit(1)
: Create font description files for lj4 device.

indxbib(1)
: Make inverted index for bibliographic databases.

lkbib(1)
: Search bibliographic databases.

lookbib(1)
: Interactively search bibliographic databases.

pdfroff(1)
: Create PDF documents using **groff**.

pfbtops(1)
: Translate a PostScript font in .pfb format to ASCII.

pic2graph(1)
: Convert a **pic** diagram into a cropped image.

tfmtodit(1)
: Create font description files for TeX DVI device.

xditview(1x)
: *roff* viewer distributed with X window.

xtotroff(1)
: Convert X font metrics into GNU *troff* font metrics.

ENVIRONMENT

Normally, the path separator in the following environment variables is the colon; this may vary depending on the operating system. For example, DOS and Windows use a semicolon instead.

GROFF_BIN_PATH
: This search path, followed by **$PATH**, is used for commands that are executed by **groff**. If it is not set then the directory where the *groff* binaries were installed is prepended to **PATH**.

GROFF_COMMAND_PREFIX
: When there is a need to run different *roff* implementations at the same time *groff* provides the

facility to prepend a prefix to most of its programs that could provoke name clashings at run time (default is to have none). Historically, this prefix was the character **g**, but it can be anything. For example, **gtroff** stood for *groff*'s **troff**, **gtbl** for the *groff* version of **tbl**. By setting **GROFF_COMMAND_PREFIX** to different values, the different *roff* installations can be addressed. More exactly, if it is set to prefix *xxx* then **groff** as a wrapper program internally calls *xxx***troff** instead of **troff**. This also applies to the preprocessors **eqn**, **grn**, **pic**, **refer**, **tbl**, **soelim**, and to the utilities **indxbib** and **lookbib**. This feature does not apply to any programs different from the ones above (most notably **groff** itself) since they are unique to the *groff* package.

GROFF_ENCODING
The value of this environment value is passed to the **preconv** preprocessor to select the encoding of input files. Setting this option implies **groff**'s command line option **−k** (this is, **groff** actually always calls **preconv**). If set without a value, **groff** calls **preconv** without arguments. An explicit **−K** command line option overrides the value of **GROFF_ENCODING**. See **preconv**(1) for details.

GROFF_FONT_PATH
A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

GROFF_TMAC_PATH
A list of directories in which to search for macro files in addition to the default directories. See **troff**(1) and **groff_tmac**(5) for more details.

GROFF_TMPDIR
The directory in which temporary files are created. If this is not set but the environment variable **TMPDIR** instead, temporary files are created in the directory **$TMPDIR**. On MS-DOS and Windows 32 platforms, the environment variables **TMP** and **TEMP** (in that order) are searched also, after **GROFF_TMPDIR** and **TMPDIR**. Otherwise, temporary files are created in **/tmp**. The **refer**(1), **groffer**(1), **grohtml**(1), and **grops**(1) commands use temporary files.

GROFF_TYPESETTER
Preset the default device. If this is not set the **ps** device is used as default. This device name is overwritten by the option **−T**.

FILES

There are some directories in which *groff* installs all of its data files. Due to different installation habits on different operating systems, their locations are not absolutely fixed, but their function is clearly defined and coincides on all systems.

groff Macro Directory

This contains all information related to macro packages. Note that more than a single directory is searched for those files as documented in **groff_tmac**(5). For the *groff* installation corresponding to this document, it is located at *c:/progra~1/groff/share/groff/1.20/tmac*. The following files contained in the *groff macro directory* have a special meaning:

troffrc Initialization file for *troff*. This is interpreted by **troff** before reading the macro sets and any input.

troffrc-end
Final startup file for *troff*. It is parsed after all macro sets have been read.

*name***.tmac**
tmac.*name*
Macro file for macro package *name*.

groff Font Directory

This contains all information related to output devices. Note that more than a single directory is searched for those files; see **troff**(1). For the *groff* installation corresponding to this document, it is located at *c:/progra~1/groff/share/groff/1.20/font*. The following files contained in the *groff font directory* have a special meaning:

dev*name***/DESC**
Device description file for device *name*, see **groff_font**(5).

dev*name***/***F*
Font file for font *F* of device *name*.

EXAMPLES

The following example illustrates the power of the **groff** program as a wrapper around **troff**.

To process a *roff* file using the preprocessors **tbl** and **pic** and the **me** macro set, classical *troff* had to be called by

```
pic foo.me | tbl | troff -me -Tlatin1 | grotty
```

Using **groff**, this pipe can be shortened to the equivalent command

```
groff -p -t -me -T latin1 foo.me
```

An even easier way to call this is to use **grog**(1) to guess the preprocessor and macro options and execute the generated command (by using backquotes to specify shell command substitution)

```
`grog -Tlatin1 foo.me`
```

The simplest way is to view the contents in an automated way by calling

```
groffer foo.me
```

BUGS

On `EBCDIC` hosts (e.g., `OS/390 Unix`), output devices **ascii** and **latin1** aren't available. Similarly, output for `EBCDIC` code page **cp1047** is not available on `ASCII` based operating systems.

Report bugs to the groff maling list Include a complete, self-contained example that allows the bug to be reproduced, and say which version of *groff* you are using.

AVAILABILITY

Information on how to get *groff* and related information is available at the groff GNU website The most recent released version of *groff* is available at the groff development site

Three *groff* mailing lists are available:

for reporting bugs

for general discussion of *groff*,

the groff commit list a read-only list showing logs of commitments to the CVS repository.

Details on CVS access and much more can be found in the file **README** at the top directory of the *groff* source package.

There is a free implementation of the **grap** preprocessor, written by Ted Faber The actual version can be found at the grap website This is the only grap version supported by *groff*.

AUTHORS

This document is based on the original *groff* man page written by James Clark It was rewritten, enhanced, and put under the FDL license by Bernd Warken. It is maintained by Werner Lemberg

groff is a `GNU` free software project. All parts of the *groff package* are protected by `GNU copyleft licenses`. The software files are distributed under the terms of the `GNU General Public License` (`GPL`), while the documentation files mostly use the `GNU Free Documentation License` (`FDL`).

SEE ALSO

The *groff info file* contains all information on the *groff* system within a single document, providing many examples and background information. See **info**(1) on how to read it.

Due to its complex structure, the *groff* system has many man pages. They can be read with **man**(1) or **groffer**(1).

Introduction, history and further readings:
roff(7).

Viewer for groff files:
: **groffer**(1), **gxditview**(1), **xditview**(1x).

Wrapper programs for formatters:
: **groff**(1), **grog**(1).

Roff preprocessors:
: **eqn**(1), **grn**(1), **pic**(1), **chem**(1), **preconv**(1), **refer**(1), **soelim**(1), **tbl**(1), **grap**(1).

Roff language with the groff extensions:
: **groff**(7), **groff_char**(7), **groff_diff**(7), **groff_font**(5).

Roff formatter programs:
: **nroff**(1), **troff**(1), **ditroff**(7).

The intermediate output language:
: **groff_out**(7).

Postprocessors for the output devices:
: **grodvi**(1), **grohtml**(1), **grolbp**(1), **grolj4**(1), **lj4_font**(5), **grops**(1), **grotty**(1).

Groff macro packages and macro-specific utilities:
: **groff_tmac**(5), **groff_man**(7), **groff_mdoc**(7), **groff_me**(7), **groff_mm**(7), **groff_mmse**(7), **groff_mom**(7), **groff_ms**(7), **groff_www**(7), **groff_trace**(7), **mmroff**(7).

The following utilities are available:
: **addftinfo**(1), **afmtodit**(1), **eqn2graph**(1), **gdiffmk**(1), **grap2graph**(1), **groffer**(1), **gxditview**(1), **hpftodit**(1), **indxbib**(1), **lkbib**(1), **lookbib**(1), **pdfroff**(1), **pfbtops**(1), **pic2graph**(1), **tfmtodit**(1), **xtotroff**(1).

GROFFER

NAME

groffer – display groff files and man pages on X and tty

SYNOPSIS

[**option**. . .] [**--**] [**filespec**. . .] **-h|--help -v|--version**

DESCRIPTION

The **groffer** program is the easiest way to use **groff**(1). It can display arbitrary documents written in the *groff* language, see **groff**(7), or other *roff* languages, see **roff**(7), that are compatible to the original *troff* language. It finds and runs all necessary *groff* preprocessors, such as **chem**.

The **groffer** program also includes many of the features for finding and displaying the `Unix` manual pages (*man pages*), such that it can be used as a replacement for a **man**(1) program. Moreover, compressed files that can be handled by **gzip**(1) or **bzip2**(1) are decompressed on-the-fly.

The normal usage is quite simple by supplying a file name or name of a *man page* without further options. But the option handling has many possibilities for creating special behaviors. This can be done either in configuration files, with the shell environment variable **`$GROFFER_OPT`**, or on the command line.

The output can be generated and viewed in several different ways available for *groff*. This includes the *groff* native `X Window` viewer **gxditview**(1), each *Postcript*, *pdf*, or *dvi* display program, a web browser by generating *html* in *www mode*, or several *text modes* in text terminals.

Most of the options that must be named when running **groff** directly are determined automatically for **groffer**, due to the internal usage of the **grog**(1) program. But all parts can also be controlled manually by arguments.

Several file names can be specified on the command line arguments. They are transformed into a single document in the normal way of **groff**.

Option handling is done in `GNU` style. Options and file names can be mixed freely. The option '**––**' closes the option handling, all following arguments are treated as file names. Long options can be abbreviated in several ways.

OPTION OVERVIEW

breaking options

[**-h˜| --help**] [**-v˜| --version**]

groffer mode options

[**--auto**] [**--default**] [**--default–modes***mode1,mode2,. . .*] [**--dvi**] [**--dvi–viewer***prog*] [**--groff**] [**--html**] [**--html–viewer***prog*] [**--mode***display_mode*] [**--pdf**] [**--pdf–viewer-***prog*] [**--ps**] [**--ps–viewer***prog*] [**--source**] [**--text**] [**--to–stdout**] [**--tty**] [**--tty–viewer***prog*] [**--www**] [**--www–viewer***prog*] [**--x***--X*] [**--x–viewer***--X–viewer*]

options related to groff

[**-T˜| --device***device*] [**-Z˜| --intermediate–output˜| --ditroff**]

All further **groff** short options are accepted.

options for man pages

[**--apropos**] [**--apropos–data**] [**--apropos–devel**] [**--apropos–progs**] [**--man**] [**--no-man**] [**--no-special**] [**--whatis**]

long options taken over from GNU man

[**--all**] [**--ascii**] [**--ditroff**] [**--extension***suffix*] [**--locale***language*] [**--local-file**] [**--location˜| --where**] [**--manpath***dir1:dir2:. . .*] [**--no-location**] [**--pager***program*] [**--sections***sec1:sec2:. . .*] [**--systems***sys1,sys2,. . .*] [**--troff-device***device*]

Further long options of `GNU` **man** are accepted as well.

X Window Toolkit options

[**--bd˜| --bordercolor***pixels*] [**--bg˜| --background***color*] [**--bw˜| --borderwidth***pixels*] [**--display***X-display*] [**--fg˜| --foreground***color*] [**--fn˜| --ft˜| --font***font_name*] [**--geometry***size_pos*] [**--resolution***value*] [**--rv**] [**--title***string*] [**--xrm***X-resource*] *options for development*

[**--debug**] [**--debug–filenames**] [**--debug–grog**] [**--debug–keep**] [**--debug–params**] [**--debug–tmpdir**] [**--do–nothing**] [**--print***text*] [**-V**]

filespec arguments

The *filespec* parameters are all arguments that are neither an option nor an option argument. They usually mean a file name or a *man page* searching scheme.

In the following, the term *section_extension* is used. It means a word that consists of a *man section* that is optionally followed by an *extension*. The name of a *man section* is a single character from **[1-9on]**, the *extension* is some word. The *extension* is mostly lacking.

No *filespec* parameters means standard input.

-	stands for standard input (can occur several times).
filename	the path name of an existing file.
man:*name***(***section_extension***)** **man:***name***.***section_extension* *name***(***section_extension***)** *name***.***section_extension* *section_extension name*	search the man page *name* in the section with optional extension *section_extension*.
man:*name*	man page in the lowest *man section* that has *name*.
name	if *name* is not an existing file search for the man page *name* in the lowest man section.

OPTION DETAILS

The **groffer** program can usually be run with very few options. But for special purposes, it supports many options. These can be classified in 5 option classes.

All short options of **groffer** are compatible with the short options of **groff**(1). All long options of **groffer** are compatible with the long options of **man**(1).

Arguments for long option names can be abbreviated in several ways. First, the argument is checked whether it can be prolonged as is. Furthermore, each minus sign **-** is considered as a starting point for a new abbreviation. This leads to a set of multiple abbreviations for a single argument. For example, **--de–n–f** can be used as an abbreviation for **--debug–not–func**, but **--de–n** works as well. If the abbreviation of the argument leads to several resulting options an error is raised.

These abbreviations are only allowed in the environment variable **$GROFFER_OPT**, but not in the configuration files. In configuration, all long options must be exact.

groffer breaking Options

As soon as one of these options is found on the command line it is executed, printed to standard output, and the running **groffer** is terminated thereafter. All other arguments are ignored.

[**-h**|--help]
Print a helping information with a short explanation of option sto standard output.

[**-v--version**]
Print version information to standard output.

groffer Mode Options

The display mode and the viewer programs are determined by these options. If none of these mode and viewer options is specified **groffer** tries to find a suitable display mode automatically. The default modes are *mode pdf*, *mode ps*, *mode html*, *mode x*, and *mode dvi* in `X Window` with different viewers and *mode tty* with device *latin1* under **less** on a terminal; other modes are tested if the programs for the main default mode do not exist.

In `X Window`, many programs create their own window when called. **groffer** can run these viewers as an independent program in the background. As this does not work in text mode on a terminal (tty) there must be a way to know which viewers are `X Window` graphical programs. The **groffer** script has a small set of information on some viewer names. If a viewer argument of the command–line chooses an element that is kept as `X Window` program in this list it is treated as a viewer that can run in the background. All other, unknown viewer calls are not run in the background.

For each mode, you are free to choose whatever viewer you want. That need not be some graphical viewer suitable for this mode. There is a chance to view the output source; for example, the combination of the options **--mode=ps** and **--ps–viewer=less** shows the content of the *Postscript* output, the source code, with the pager **less**.

--auto Equivalent to **--mode=auto**.

--default
Reset all configuration from previously processed command line options to the default values. This is useful to wipe out all former options of the configuration, in **`$GROFFER_OPT`**, and restart option processing using only the rest of the command line.

--default–modes mode1,mode2,. . .
Set the sequence of modes for *auto mode* to the comma separated list given in the argument. See **--mode** for details on modes. Display in the default manner; actually, this means to try the modes *x*, *ps*, and *tty* in this sequence.

--dvi Equivalent to **--mode=dvi**.

--dvi–viewer prog
Choose a viewer program for *dvi mode*. This can be a file name or a program to be searched in **`$PATH`**. Known `X Window` *dvi* viewers include **xdvi**(1) and **dvilx**(1) In each case, arguments can be provided additionally.

--groff Equivalent to **--mode=groff**.

--html Equivalent to **--mode=html**.

--html–viewer
Choose a web browser program for viewing in *html mode*. It can be the path name of an executable file or a program in **`$PATH`**. In each case, arguments can be provided additionally.

--mode*value*
Set the display mode. The following mode values are recognized:

`auto` Select the automatic determination of the display mode. The sequence of modes that are tried can be set with the **--default–modes** option. Useful for restoring the *default mode* when a different mode was specified before.

`dvi` Display formatted input in a *dvi* viewer program. By default, the formatted input is displayed with the **xdvi**(1) program. **--dvi**.

`groff` After the file determination, switch **groffer** to process the input like **groff**(1) would do. This disables the *groffer* viewing features.

`html` Translate the input into html format and display the result in a web browser program. By default, the existence of a sequence of standard web browsers is tested, starting with **konqueror**(1) and **mozilla**(1). The text html viewer is **lynx**(1).

`pdf` Display formatted input in a *PDF* (Portable Document Format) viewer program. By default, the input is formatted by **groff** using the Postscript device, then it is transformed into the PDF file format using **gs**(1), or **ps2pdf**(1). If that's not possible, the *Postscript mode (ps)* is used instead. Finally it is displayed using different viewer programs. *pdf* has a big advantage because the text is displayed graphically and is searchable as well.

`ps` Display formatted input in a Postscript viewer program. By default, the formatted input is displayed in one of many viewer programs.

`text` Format in a *groff text mode* and write the result to standard output without a pager or viewer program. The text device, *latin1* by default, can be chosen with option **-T**.

`tty` Format in a *groff text mode* and write the result to standard output using a text pager program, even when in `X Window`.

`www` Equivalent to **--mode=html**.

x Display the formatted input in a native *roff* viewer. By default, the formatted input is displayed with the **gxditview**(1) program being distributed together with **groff**. But the standard `X Window` tool **xditview**(1) can also be chosen with the option **--x–viewer .** The default resolution is **75 dpi**, but **100 dpi** are also possible. The default *groff* device for the resolution of **75 dpi** is **X75–12**, for **100 dpi** it is **X100**. The corresponding *groff intermediate output* for the actual device is generated and the result is displayed. For a resolution of **100 dpi**, the default width of the geometry of the display program is chosen to **850 dpi**.

X Equivalent to **--mode=x**.

The following modes do not use the *groffer* viewing features. They are only interesting for advanced applications.

groff Generate device output with plain *groff* without using the special viewing features of *groffer*. If no device was specified by option **-T** the *groff* default **ps** is assumed.

source
Output the roff source code of the input files without further processing.

--pdf Equivalent to **--mode=pdf**.

--pdf–viewer prog
Choose a viewer program for *pdf mode*. This can be a file name or a program to be searched in **$PATH**; arguments can be provided additionally.

--ps Equivalent to **--mode=ps**.

--ps–viewer prog
Choose a viewer program for *ps mode*. This can be a file name or a program to be searched in **$PATH**. Common Postscript viewers inlude **gv**(1), **ghostview**(1), and **gs**(1), In each case, arguments can be provided additionally.

--source
Equivalent **--mode=source**.

--text Equivalent to **--mode=text**.

--to–stdout
The file for the chosen mode is generated and its content is printed to standard output. It will not be displayed in graphical mode.

--tty Equivalent to **--mode=tty**.

--tty–viewer prog
Choose a text pager for mode *tty*. The standard pager is **less**(1). This option is eqivalent to *man* option **--pager=prog**. The option argument can be a file name or a program to be searched in **$PATH**; arguments can be provided additionally.

--www Equivalent to **--mode=html**.

--www–viewer prog
Equivalent to **--html–viewer .**

--X˜| --x
Equivalent to **--mode=x**.

--X–viewer -- x–viewer prog
Choose a viewer program for *x mode*. Suitable viewer programs are **gxditview**(1) which is the default and **xditview**(1). The argument can be any executable file or a program in **$PATH**; arguments can be provided additionally.

-- Signals the end of option processing; all remaining arguments are interpreted as *filespec* parameters.

Besides these, **groffer** accepts all short options that are valid for the **groff**(1) program. All non-**groffer** options are sent unmodified via **grog** to **groff**. So postprocessors, macro packages, compatibility with *classical troff*, and much more can be manually specified.

Options related to groff

All short options of **groffer** are compatible with the short options of **groff**(1). The following of **groff** options have either an additional special meaning within **groffer** or make sense for normal usage.

Because of the special outputting behavior of the **groff** option **-Z groffer** was designed to be switched into *groff mode ;* the *groffer* viewing features are disabled there. The other **groff** options do not switch the mode, but allow to customize the formatting process.

--a This generates an ascii approximation of output in the *text modes*. That could be important when the text pager has problems with control sequences in *tty mode*.

--m*file* Add *file* as a *groff* macro file. This is useful in case it cannot be recognized automatically.

--P*opt_or_arg*
Send the argument *opt_or_arg* as an option or option argument to the actual **groff** postprocessor.

--T *devname* ˜| **--device** *devname*
This option determines **groff**'s output device. The most important devices are the text output devices for referring to the different character sets, such as **ascii**, **utf8**, **latin1**, and others. Each of these arguments switches **groffer** into a *text mode* using this device, to *mode tty* if the actual mode is not a *text mode*. The following *devname* arguments are mapped to the corresponding **groffer --mode=***devname* option: **dvi**, **html**, and **ps**. All **X**∗ arguments are mapped to *mode x*. Each other *devname* argument switches to *mode groff* using this device.

--X is equivalent to **groff −X**. It displays the *groff intermediate output* with **gxditview**. As the quality is relatively bad this option is deprecated; use **--X** instead because the *x mode* uses an *X*∗ device for a better display.

-Z˜| **--intermediate-output**˜| **--ditroff**
Switch into *groff mode* and format the input with the *groff intermediate output* without postprocessing; see **groff_out**(5). This is equivalent to option **--ditroff** of *man*, which can be used as well.

All other **groff** options are supported by **groffer**, but they are just transparently transferred to **groff** without any intervention. The options that are not explicitly handled by **groffer** are transparently passed to **groff**. Therefore these transparent options are not documented here, but in **groff**(1). Due to the automatism in **groffer**, none of these **groff** options should be needed, except for advanced usage.

Options for man pages

--apropos
Start the **apropos**(1) command or facility of **man**(1) for searching the *filespec* arguments within all *man page* descriptions. Each *filespec* argument is taken for search as it is; *section* specific parts are not handled, such that **7 groff** searches for the two arguments **7** and **groff**, with a large result; for the *filespec* **groff.7** nothing will be found. The *language* locale is handled only when the called programs do support this; the GNU **apropos** and **man −k** do not. The display differs from the **apropos** program by the following concepts:

- Construct a *groff* frame similar to a *man page* to the output of **apropos**,
- each *filespec* argument is searched on its own.
- The restriction by **--sections** is handled as well,
- wildcard characters are allowed and handled without a further option.

--apropos−data
Show only the **apropos** descriptions for data documents, these are the **man**(7) *sections 4*, *5*, and *7*. Direct *section* declarations are ignored, wildcards are accepted.

--apropos−devel
Show only the **apropos** descriptions for development documents, these are the **man**(7) *sections 2*, *3*, and *9*. Direct *section* declarations are ignored, wildcards are accepted.

--apropos−progs
Show only the **apropos** descriptions for documents on programs, these are the **man**(7) *sections 1*, *6*, and *8*. Direct *section* declarations are ignored, wildcards are accepted.

--whatis
For each *filespec* argument search all *man pages* and display their description — or say that it is not a *man page*. This is written from anew, so it differs from *man*'s **whatis** output by the following concepts

- each retrieved file name is added,
- local files are handled as well,
- the *language* and *system* locale is supported,
- the display is framed by a *groff* output format similar to a *man page*,
- wildcard characters are allowed without a further option.

The following options were added to **groffer** for choosing whether the file name arguments are interpreted as names for local files or as a search pattern for *man pages*. The default is looking up for local files.

--man Check the non-option command line arguments (*filespecs*) first on being *man pages*, then whether they represent an existing file. By default, a *filespec* is first tested whether it is an existing file.

--no-man˜| --local-file
Do not check for *man pages*. **--local-file** is the corresponding **man** option.

--no-special
Disable former calls of **--all , --apropos*** **,** and **--whatis .**

Long options taken over from GNU man

The long options of **groffer** were synchronized with the long options of GNU **man**. All long options of GNU **man** are recognized, but not all of these options are important to **groffer**, so most of them are just ignored. These ignored **man** options are **--catman , --troff ,** and **--update .**

In the following, the **man** options that have a special meaning for **groffer** are documented.

If your system has GNU **man** installed the full set of long and short options of the GNU **man** program can be passed via the environment variable **$MANOPT**; see **man**(1).

--all In searching *man pages*, retrieve all suitable documents instead of only one.

-7--ascii
In *text modes*, display ASCII translation of special characters for critical environment. This is equivalent to **groff -mtty_char**; see **groff_tmac**(5).

--ditroff
Produce *groff intermediate output*. This is equivalent to **groffer -Z .**

--extension*suffix*
Restrict *man page* search to file names that have *suffix* appended to their section element. For example, in the file name */usr/share/man/man3/terminfo.3ncurses.gz* the *man page* extension is *ncurses*.

--locale*language*
Set the language for *man pages*. This has the same effect, but overwrites **$LANG**

--location
Print the location of the retrieved files to standard error.

--no-location
Do not display the location of retrieved files; this resets a former call to **--location .** This was added by **groffer**.

--manpath*'dir1:dir2:...'*
Use the specified search path for retrieving *man pages* instead of the program defaults. If the argument is set to the empty string "" the search for *man page* is disabled.

--pager
Set the pager program in *tty mode*; default is **less**. This is equivalent to **--tty−viewer .**

--sections*'sec1:sec2:...'*
Restrict searching for *man pages* to the given *sections*, a colon-separated list.

--systems*'sys1,sys2,...'*
Search for *man pages* for the given operating systems; the argument *systems* is a comma-separated list.

--where
Eqivalent to **--location .**

X Window Toolkit Options

The following long options were adapted from the corresponding `X Window Toolkit` options. **groffer** will pass them to the actual viewer program if it is an `X Window` program. Otherwise these options are ignored.

Unfortunately these options use the old style of a single minus for long options. For **groffer** that was changed to the standard with using a double minus for long options, for example, **groffer** uses the option **--font** for the `X Window` option **-font .**

See **X**(7) and the documentation on the `X Window Toolkit` options for more details on these options and their arguments.

--background*color*
Set the background color of the viewer window.

--bd*pixels*
This is equivalent to **--bordercolor .**

--bg*color*
This is equivalent to **--background .**

--bw pixels
This is equivalent to **--borderwidth .**

--bordercolor*pixels*
Specifies the color of the border surrounding the viewer window.

--borderwidth*pixels*
Specifies the width in pixels of the border surrounding the viewer window.

--display*X-display*
Set the `X Window` display on which the viewer program shall be started, see the `X Window` documentation for the syntax of the argument.

--foreground*color*
Set the foreground color of the viewer window.

--fg*color*
This is equivalent to **-foreground .**

--fn font_name
This is equivalent to **--font .**

--font*font_name*
Set the font used by the viewer window. The argument is an `X Window` font name.

--ft*font_name*
This is equivalent to **--font .**

--geometry*size_pos*
Set the geometry of the display window, that means its size and its starting position. See **X**(7) for the syntax of the argument.

--resolution*value*
Set `X Window` resolution in dpi (dots per inch) in some viewer programs. The only supported dpi values are **75** and **100**. Actually, the default resolution for **groffer** is set to **75 dpi**. The resolution also sets the default device in *mode x*.

--rv Reverse foreground and background color of the viewer window.

--title'*some text*'
: Set the title for the viewer window.

--xrm'*resource*'
: Set X Window resource.

Options for Development

--debug
: Enable all debugging options **--debug–***type* **.** The temporary files are kept and not deleted, the **grog** output is printed, the name of the temporary directory is printed, the displayed file names are printed, and the parameters are printed.

--debug–filenames
: Print the names of the files and *man pages* that are displayed by **groffer**.

--debug–grog
: Print the output of all **grog** commands.

--debug–keep
: Enable two debugging informations. Print the name of the temporary directory and keep the temporary files, do not delete them during the run of groffer**.**

--debug–params
: Print the parameters, as obtained from the configuration files, from **GROFFER_OPT**, and the command line arguments.

--debug–tmpdir
: Print the name of the temporary directory.

--do-nothing
: This is like **--version ,** but without the output; no viewer is started. This makes only sense in development.

--print=*text*
: Just print the argument to standard error. This is good for parameter check.

-V This is an advanced option for debugging only. Instead of displaying the formatted input, a lot of *groffer* specific information is printed to standard output:

- the output file name in the temporary directory,
- the display mode of the actual **groffer** run,
- the display program for viewing the output with its arguments,
- the active parameters from the config files, the arguments in **$GROFFER_OPT**, and the arguments of the command line,
- the pipeline that would be run by the **groff** program, but without executing it.

Other useful debugging options are the **groff** option **-Z** and **--mode=groff**.

Filespec Arguments

A *filespec* parameter is an argument that is not an option or option argument. In **groffer**, *filespec* parameters are a file name or a template for searching *man pages*. These input sources are collected and composed into a single output file such as **groff** does.

The strange POSIX behavior to regard all arguments behind the first non-option argument as *filespec* arguments is ignored. The GNU behavior to recognize options even when mixed with *filespec* arguments is used througout. But, as usual, the double minus argument **--** ends the option handling and interprets all following arguments as *filespec* arguments; so the POSIX behavior can be easily adopted.

The options **--apropos*** have a special handling of *filespec* arguments. Each argument is taken as a search scheme of its own. Also a regexp (regular expression) can be used in the filespec. For example, **groffer --apropos '^gro.f$'** searches **groff** in the *man page* name, while **groffer --apropos groff** searches **groff** somewhere in the name or description of the *man pages*.

All other parts of *groffer*, such as the normal display or the output with **--whatis** have a different scheme for *filespecs*. No regular expressions are used for the arguments. The *filespec* arguments are handled by the following scheme.

It is necessary to know that on each system the *man pages* are sorted according to their content into several sections. The *classical man sections* have a single-character name, either a digit from **1** to **9** or one of the characters **n** or **o**.

This can optionally be followed by a string, the so-called *extension*. The *extension* allows to store several *man pages* with the same name in the same *section*. But the *extension* is only rarely used, usually it is omitted. Then the *extensions* are searched automatically by alphabet.

In the following, we use the name *section_extension* for a word that consists of a single character *section* name or a *section* character that is followed by an *extension*. Each *filespec* parameter can have one of the following forms in decreasing sequence.

- No *filespec* parameters means that **groffer** waits for standard input. The minus option **-** always stands for standard input; it can occur several times. If you want to look up a *man page* called **-** use the argument **man:−**.
- Next a *filespec* is tested whether it is the path name of an existing file. Otherwise it is assumed to be a searching pattern for a *man page*.
- **man:***name*(*section_extension*), **man:***name***.***section_extension,* *name*(*section_extension*), or *name***.***section_extension* search the man page *name* in man section and possibly extension of *section_extension*.
- Now **man:***name* searches for a *man page* in the lowest *man section* that has a document called *name*.
- *section_extension name* is a pattern of 2 arguments that originates from a strange argument parsing of the **man** program. Again, this searches the man page *name* with *section_extension*, a combination of a *section* character optionally followed by an *extension*.
- We are left with the argument *name* which is not an existing file. So this searches for the *man page* called *name* in the lowest *man section* that has a document for this name.

Several file name arguments can be supplied. They are mixed by **groff** into a single document. Note that the set of option arguments must fit to all of these file arguments. So they should have at least the same style of the *groff* language.

OUTPUT MODES

By default, the **groffer** program collects all input into a single file, formats it with the **groff** program for a certain device, and then chooses a suitable viewer program. The device and viewer process in **groffer** is called a *mode*. The mode and viewer of a running **groffer** program is selected automatically, but the user can also choose it with options. The modes are selected by option the arguments of **--mode=***anymode*. Additionally, each of this argument can be specified as an option of its own, such as **anymode**. Most of these modes have a viewer program, which can be chosen by an option that is constructed like **--***anymode*−viewer.

Several different modes are offered, graphical modes for `X Window`, *text modes*, and some direct *groff modes* for debugging and development.

By default, **groffer** first tries whether *x mode* is possible, then *ps mode*, and finally *tty mode*. This mode testing sequence for *auto mode* can be changed by specifying a comma separated list of modes with the option **--default−modes.**

The searching for *man pages* and the decompression of the input are active in every mode.

Graphical Display Modes

The graphical display modes work mostly in the `X Window` environment (or similar implementations within other windowing environments). The environment variable **`$DISPLAY`** and the option **--display** are used for specifying the `X Window` display to be used. If this environment variable is empty **groffer** assumes that no `X Window` is running and changes to a *text mode*. You can change this automatic behavior by the option **--default−modes**.

Known viewers for the graphical display modes and their standard `X Window` viewer progams are

- in a PDF viewer (*pdf mode*),

- in a web browser (*html* or *www mode*).

- in a Postscript viewer (*ps mode*),
- `X Window` *roff* viewers such as **gxditview**(1) or **xditview**(1) (in *x mode*),
- in a dvi viewer program (*dvi mode*),

The *pdf mode* has a major advantage — it is the only graphical diplay mode that allows to search for text within the viewer; this can be a really important feature. Unfortunately, it takes some time to transform the input into the PDF format, so it was not chosen as the major mode.

These graphical viewers can be customized by options of the `X Window Toolkit`. But the **groffer** options use a leading double minus instead of the single minus used by the `X Window Toolkit`.

Text modes

There are two modes for text output, *mode text* for plain output without a pager and *mode tty* for a text output on a text terminal using some pager program.

If the variable **`$DISPLAY`** is not set or empty, **groffer** assumes that it should use *tty mode*.

In the actual implementation, the *groff* output device *latin1* is chosen for *text modes*. This can be changed by specifying option **-T** or **--device**.

The pager to be used can be specified by one of the options **--pager** and **--tty–viewer**, or by the environment variable **`$PAGER`**. If all of this is not used the **less**(1) program with the option **-r** for correctly displaying control sequences is used as the default pager.

Special Modes for Debugging and Development

These modes use the *groffer* file determination and decompression. This is combined into a single input file that is fed directly into **groff** with different strategy without the *groffer* viewing facilities. These modes are regarded as advanced, they are useful for debugging and development purposes.

The *source mode* with option **--source** just displays the decompressed input.

Otion **--to–stdout** does not display in a graphical mode. It just generates the file for the chosen mode and then prints its content to standard output.

The *groff mode* passes the input to **groff** using only some suitable options provided to **groffer**. This enables the user to save the generated output into a file or pipe it into another program.

In *groff mode*, the option **-Z** disables post-processing, thus producing the *groff intermediate output*. In this mode, the input is formatted, but not postprocessed; see **groff_out**(5) for details.

All **groff** short options are supported by **groffer**.

MAN PAGE SEARCHING

The default behavior of **groffer** is to first test whether a file parameter represents a local file; if it is not an existing file name, it is assumed to represent the name of a *man page*. The following options can be used to determine whether the arguments should be handled as file name or *man page* arguments.

--man forces to interpret all file parameters as *filespecs* for searching *man pages*.

--no–man
--local–file
disable the *man* searching; so only local files are displayed.

If neither a local file nor a *man page* was retrieved for some file parameter a warning is issued on standard error, but processing is continued.

Search Algoritm

Let us now assume that a *man page* should be searched. The **groffer** program provides a search facility for *man pages*. All long options, all environment variables, and most of the functionality of the GNU **man**(1) program were implemented. The search algorithm shall determine which file is displayed for a given *man page*. The process can be modified by options and environment variables.

The only *man* action that is omitted in **groffer** are the preformatted *man pages*, also called *cat pages*. With the excellent performance of the actual computers, the preformatted *man pages* aren't necessary any longer. Additionally, **groffer** is a *roff* program; it wants to read *roff* source files and format them itself.

The algorithm for retrieving the file for a *man page* needs first a set of directories. This set starts with the so-called *man path* that is modified later on by adding names of *operating system* and *language*.

This arising set is used for adding the section directories which contain the *man page* files.

The *man path* is a list of directories that are separated by colon. It is generated by the following methods.

- The environment variable **$MANPATH** can be set.
- It can be read from the arguments of the environment variable **$MANOPT**.
- The *man path* can be manually specified by using the option **--manpath**. An empty argument disables the *man page* searching.
- When no *man path* was set the **manpath**(1) program is tried to determine one.
- If this does not work a reasonable default path from **$PATH** is determined.

We now have a starting set of directories. The first way to change this set is by adding names of *operating systems*. This assumes that *man pages* for several *operating systems* are installed. This is not always true. The names of such *operating systems* can be provided by 3 methods.

- The environment variable **$SYSTEM** has the lowest precedence.
- This can be overridden by an option in **$MANOPT**.
- This again is overridden by the command line option **--systems**.

Several names of *operating systems* can be given by appending their names, separated by a comma.

The *man path* is changed by appending each *system* name as subdirectory at the end of each directory of the set. No directory of the *man path* set is kept. But if no *system* name is specified the *man path* is left unchanged.

After this, the actual set of directories can be changed by *language* information. This assumes that there exist *man pages* in different languages. The wanted *language* can be chosen by several methods.

- Enviroment variable **$LANG**.
- This is overridden by **$LC_MESSAGES**.
- This is overridden by **$LC_ALL**.
- This can be overridden by providing an option in **$MANOPT**.
- All these environment variables are overridden by the command line option **--locale**.

The *default language* can be specified by specifying one of the pseudo-language parameters `C` or `POSIX`. This is like deleting a formerly given *language* information. The *man pages* in the *default language* are usually in English.

Of course, the *language* name is determined by **man**. In GNU **man**, it is specified in the `POSIX 1003.1` based format:

<*language*>[_<*territory*>[`.`<*character-set*>[`,`<*version*>]]],

but the two-letter code in <*language*> is sufficient for most purposes. If for a complicated *language* formulation no *man pages* are found **groffer** searches the country part consisting of these first two characters as well.

The actual directory set is copied thrice. The *language* name is appended as subdirectory to each directory in the first copy of the actual directory set (this is only done when a language information is given). Then the 2-letter abbreviation of the *language* name is appended as subdirectories to the second copy of the directory set (this is only done when the given language name has more than 2 letters). The third copy of the directory set is kept unchanged (if no *language* information is given this is the kept directory set). These maximally 3 copies are appended to get the new directory set.

We now have a complete set of directories to work with. In each of these directories, the *man* files are separated in *sections*. The name of a *section* is represented by a single character, a digit between *1* and *9*, or the character *o* or *n*, in this order.

For each available *section*, a subdirectory **`man`**<*section*> exists containing all *man* files for this *section*, where <*section*> is a single character as described before. Each *man* file in a *section* directory has the form **`man`**<*section*>**`/`**<*name*>`.`<*section*>*[*<*extension*>*][*`.`<*compression*>*]*, where <*extension*> and <*compression*> are optional. <*name*> is the name of the *man page* that is also specified as filespec argument on the command line.

The *extension* is an addition to the section. This postfix acts like a subsection. An *extension* occurs only in the file name, not in name of the *section* subdirectory. It can be specified on the command line.

On the other hand, the *compression* is just an information on how the file is compressed. This is not important for the user, such that it cannot be specified on the command line.

There are 4 methods to specify a *section* on the command line:

- Environment variable **`$MANSECT`**
- Command line option **--sections**
- Appendix to the *name* argument in the form <*name*>.<*section*>
- Preargument before the *name* argument in the form <*section*> <*name*>

It is also possible to specify several *sections* by appending the single characters separated by colons. One can imagine that this means to restrict the *man page* search to only some *sections*. The multiple *sections* are only possible for **`$MANSECT`** and **--sections**.

If no *section* is specified all *sections* are searched one after the other in the given order, starting with *section 1*, until a suitable file is found.

There are 4 methods to specify an *extension* on the command line. But it is not necessary to provide the whole extension name, some abbreviation is good enough in most cases.

- Environment variable **`$EXTENSION`**
- Command line option **--extension**
- Appendix to the <*name*>.<*section*> argument in the form <*name*>.<*section*><*extension*>
- Preargument before the *name* argument in the form <*section*><*extension*> <*name*>

For further details on *man page* searching, see **man**(1).

Examples of man files

`/usr/share/man/man1/groff.1`

This is an uncompressed file for the *man page* `groff` in *section 1*. It can be called by *sh#* groffer groff No *section* is specified here, so all *sections* should be searched, but as *section 1* is searched first this file will be found first. The file name is composed of the following components. **`/usr/share/man`** must be part of the *man path*; the subdirectory **`man1/`** and the part **`.1`** stand for the *section*; **`groff`** is the name of the *man page*.

`/usr/local/share/man/man7/groff.7.gz`

The file name is composed of the following components. **`/usr/local/share/man`** must be part of the *man path*; the subdirectory **`man7/`** and the part **`.7`** stand for the *section*; **`groff`** is the name of the *man page*; the final part **`.gz`** stands for a compression with **gzip**(1). As the *section* is not the first one it must be specified as well. This can be done by one of the following commands. *sh#* groffer groff.7 *sh#* groffer 7 groff *sh#* groffer −−sections=7 groff

`/usr/local/man/man1/ctags.1emacs21.bz2`

Here **`/usr/local/man`** must be in *man path*; the subdirectory **`man1/`** and the file name part **`.1`** stand for *section 1*; the name of the *man page* is **`ctags`**; the section has an extension **`emacs21`**; and the file is compressed as **`.bz2`** with **bzip2**(1). The file can be viewed with one of the following commands *sh#* groffer ctags.1e *sh#* groffer 1e ctags *sh#* groffer −−extension=e −−sections=1 ctags where `e` works as an abbreviation for the extension `emacs21`.

`/usr/man/linux/de/man7/man.7.Z`

The directory **`/usr/man`** is now part of the *man path*; then there is a subdirectory for an *operating system* name **`linux/`**; next comes a subdirectory **`de/`** for the German *language*; the *section* names **`man7`** and **`.7`** are known so far; **`man`** is the name of the *man page*; and **`.Z`** signifies the compression that can be handled by **gzip**(1). We want now show how to provide several values for some options. That is possible for *sections* and *operating system* names. So we use as *sections 5* and *7* and as *system* names *linux* and *aix*. The command is then

sh# groffer −−locale=de −−sections=5:7 −−systems=linux,aix man *sh#* LANG=de MANSECT=5:7 SYSTEM=linux,aix groffer man

DECOMPRESSION

The program has a decompression facility. If standard input or a file that was retrieved from the command line parameters is compressed with a format that is supported by either **gzip**(1) or **bzip2**(1) it is decompressed on-the-fly. This includes the GNU **.gz**, **.bz2**, and the traditional **.Z** compression. The program displays the concatenation of all decompressed input in the sequence that was specified on the command line.

ENVIRONMENT

The **groffer** program supports many system variables, most of them by courtesy of other programs. All environment variables of **groff**(1) and GNU **man**(1) and some standard system variables are honored.

Native groffer Variables

$GROFFER_OPT
Store options for a run of **groffer**. The options specified in this variable are overridden by the options given on the command line. The content of this variable is run through the shell builtin 'eval'; so arguments containing white-space or special shell characters should be quoted. Do not forget to export this variable, otherwise it does not exist during the run of **groffer**.

System Variables

The following variables have a special meaning for **groffer**.

$DISPLAY
If this variable is set this indicates that the X Window system is running. Testing this variable decides on whether graphical or text output is generated. This variable should not be changed by the user carelessly, but it can be used to start the graphical **groffer** on a remote X Window terminal. For example, depending on your system, **groffer** can be started on the second monitor by the command

sh# DISPLAY=:0.1 groffer what.ever &

$LC_ALL
$LC_MESSAGES
$LANG If one of these variables is set (in the above sequence), its content is interpreted as the locale, the language to be used, especially when retrieving *man pages*. A locale name is typically of the form *language*[_*territory*[.*codeset*[@*modifier*]]], where *language* is an ISO 639 language code, *territory* is an ISO 3166 country code, and *codeset* is a character set or encoding identifier like ISO-8859-1 or UTF-8; see **setlocale**(3). The locale values C and POSIX stand for the default, i.e. the *man page* directories without a language prefix. This is the same behavior as when all 3 variables are unset.

$PAGER
This variable can be used to set the pager for the tty output. For example, to disable the use of a pager completely set this variable to the **cat**(1) program

sh# PAGER=cat groffer anything

$PATH All programs within the **groffer** script are called without a fixed path. Thus this environment variable determines the set of programs used within the run of **groffer**.

Groff Variables

The **groffer** program internally calls **groff**, so all environment variables documented in **groff**(1) are internally used within **groffer** as well. The following variable has a direct meaning for the **groffer** program.

$GROFF_TMPDIR
If the value of this variable is an existing, writable directory, **groffer** uses it for storing its temporary files, just as **groff** does.

Man Variables

Parts of the functionality of the **man** program were implemented in **groffer**; support for all environment variables documented in **man**(1) was added to **groffer**, but the meaning was slightly modified due to the different approach in **groffer**; but the user interface is the same. The **man** environment variables can be overwritten by options provided with **$MANOPT**, which in turn is overwritten by the command line.

$EXTENSION
Restrict the search for *man pages* to files having this extension. This is overridden by option **--extension**; see there for details.

$MANOPT
This variable contains options as a preset for **man**(1). As not all of these are relevant for **groffer** only the essential parts of its value are extracted. The options specified in this variable overwrite the values of the other environment variables that are specific to *man*. All options specified in this variable are overridden by the options given on the command line.

$MANPATH
If set, this variable contains the directories in which the *man page* trees are stored. This is overridden by option **--manpath**.

$MANSECT
If this is a colon separated list of section names, the search for *man pages* is restricted to those manual sections in that order. This is overridden by option **--sections**.

$SYSTEM
If this is set to a comma separated list of names these are interpreted as *man page* trees for different operating systems. This variable can be overwritten by option **--systems**; see there for details.

The environment variable **$MANROFFSEQ** is ignored by **groffer** because the necessary preprocessors are determined automatically.

CONFIGURATION FILES

The **groffer** program can be preconfigured by two configuration files.

/etc/groff/groffer.conf
System-wide configuration file for **groffer**.

$HOME/.groff/groffer.conf
User-specific configuration file for **groffer**, where **$HOME** denotes the user's home directory. This file is called after the system-wide configuration file to enable overriding by the user.

Both files are handled for the configuration, but the configuration file in **/etc** comes first; it is overwritten by the configuration file in the home directory; both configuration files are overwritten by the environment variable **$GROFFER_OPT**; everything is overwritten by the command line arguments.

The configuration files contain options that should be called as default for every **groffer** run. These options are written in lines such that each contains either a long option, a short option, or a short option cluster; each with or without an argument. So each line with configuration information starts with a minus character '**–**'; a line with a long option starts with two minus characters '**––**', a line with a short option or short option cluster starts with a single minus '**–**'.

The option names in the configuration files may not be abbreviated, they must be exact.

The argument for a long option can be separated from the option name either by an equal sign '**=**' or by whitespace, i.e. one or several space or tab characters. An argument for a short option or short option cluster can be directly appended to the option name or separated by whitespace. The end of an argument is the end of the line. It is not allowed to use a shell environment variable in an option name or argument.

It is not necessary to use quotes in an option or argument, except for empty arguments. An empty argument can be provided by appending a pair of quotes to the separating equal sign or whitespace; with a short option, the separator can be omitted as well. For a long option with a separating equal sign '**=**', the pair of quotes can be omitted, thus ending the line with the separating equal sign. All other quote characters are cancelled internally.

In the configuration files, arbitrary whitespace is allowed at the beginning of each line, it is just ignored. Each whitespace within a line is replaced by a single space character ' ' internally.

All lines of the configuration lines that do not start with a minus character are ignored, such that comments starting with '**#**' are possible. So there are no shell commands in the configuration files.

As an example, consider the following configuration file that can be used either in **/etc/groff/groffer.conf** or **~/.groff/groffer.conf**.

```
# groffer configuration file
#
# groffer options that are used in each call of groffer
--foreground=DarkBlue
--resolution 100
--x-viewer=gxditview -geometry 900x1200
--pdf-viewer xpdf -z 150
```

The lines starting with **#** are just ignored, so they act as command lines. This configuration sets four **groffer** options (the lines starting with '**–**'). This has the following effects:

- Use a text color of **DarkBlue** in all viewers that support this, such as **gxditview**.
- Use a resolution of **100 dpi** in all viewers that support this, such as **gxditview**. By this, the default device in *x mode* is set to **X100**.
- Force **gxditview**(1) as the *x-mode* viewer using the geometry option for setting the width to **900 dpi** and the height to **1200 dpi**. This geometry is suitable for a resolution of **100 dpi**.
- Use **xpdf**(1) as the *pdf-mode* viewer with the argument **–Z 150**.

EXAMPLES

The usage of **groffer** is very easy. Usually, it is just called with a file name or *man page*. The following examples, however, show that **groffer** has much more fancy capabilities. *sh#* groffer /usr/local/share/doc/groff/meintro.ms.gz Decompress, format and display the compressed file **`meintro.ms.gz`** in the directory **`/usr/local/share/doc/groff`**, using the standard viewer **gxditview** as graphical viewer when in `X Window`, or the **less**(1) pager program when not in `X Window`.

sh# groffer groff

If the file **`./groff`** exists use it as input. Otherwise interpret the argument as a search for the *man page* named **groff** in the smallest possible *man section*, being section 1 in this case.

sh# groffer man:groff

search for the *man page* of **groff** even when the file **`./groff`** exists.

sh# groffer groff.7 *sh#* groffer 7 groff

search the *man page* of **groff** in *man section* **7**. This section search works only for a digit or a single character from a small set.

sh# groffer fb.modes

If the file **`./fb.modes`** does not exist interpret this as a search for the *man page* of **fb.modes**. As the extension *modes* is not a single character in classical section style the argument is not split to a search for **fb**.

sh# groffer groff 'troff(1)' man:roff

The arguments that are not existing files are looked-up as the following *man pages*: **groff** (automatic search, should be found in *man* section 1), **troff** (in section 1), and **roff** (in the section with the lowest number, being 7 in this case). The quotes around *'troff(1)'* are necessary because the paranthesis are special shell characters; escaping them with a backslash character \(and \) would be possible, too. The formatted files are concatenated and displayed in one piece.

sh# LANG=de groffer --man --www --www-viever=galeon ls

Retrieve the German *man page* (language *de*) for the **ls** program, decompress it, format it to *html* format (*www mode*) and view the result in the web browser **galeon**. The option **--man** guarantees that the *man page* is retrieved, even when a local file **`ls`** exists in the actual directory.

sh# groffer --source 'man:roff(7)'

Get the *man page* called *roff* in *man* section 7, decompress it, and print its unformatted content, its source code.

sh# groffer --de-p --in --ap

This is a set of abbreviated arguments, it is determined as

sh# groffer --debug-params --intermediate-output --apropos

sh# cat file.gz | groffer -Z -mfoo"

The file **`file.gz`** is sent to standard input, this is decompressed, and then this is transported to the *groff intermediate output mode* without post-processing (**groff** option **-Z),** using macro package *foo* (**groff** option **-m) .**

sh# echo '\f[CB]WOW!' | > groffer --x --bg red --fg yellow --geometry 200x100 -

Display the word **`WOW!`** in a small window in constant-width bold font, using color yellow on red background.

COMPATIBILITY

The **groffer** program is written in Perl, the Perl version during writing was v5.8.8.

groffer provides its own parser for command line arguments that is compatible to both `POSIX` **getopts**(1) and `GNU` **getopt**(1). It can handle option arguments and file names containing white space and a large set of special characters. The following standard types of options are supported.

- The option consisting of a single minus **-** refers to standard input.
- A single minus followed by characters refers to a single character option or a combination thereof; for example, the **groffer** short option combination **-Qmfoo** is equivalent to **-Q –m foo .**
- Long options are options with names longer than one character; they are always preceded by a double minus. An option argument can either go to the next command line argument or be appended with an equal sign to the argument; for example, **--long=arg** is equivalent to **--long arg**.
- An argument of **--** ends option parsing; all further command line arguments are interpreted as *filespec* parameters, i.e. file names or constructs for searching *man pages*).
- All command line arguments that are neither options nor option arguments are interpreted as *filespec* parameters and stored until option parsing has finished. For example, the command line

  sh# groffer file1 -a -o arg file2

  is equivalent to

  sh# groffer -a -o arg -- file1 file2

The free mixing of options and *filespec* parameters follows the GNU principle. That does not fulfill the strange option behavior of `POSIX` that ends option processing as soon as the first non-option argument has been reached. The end of option processing can be forced by the option '**––**' anyway.

BUGS

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

You can also use the groff mailing list but you must first subscribe to this list. You can do that by visiting the groff mailing list web page

See **groff**(1) for information on availability.

SEE ALSO

groff(1), **troff**(1)
: Details on the options and environment variables available in **groff**; all of them can be used with **groffer**.

groff(7)
: Documentation of the *groff* language.

grog(1) Internally, **groffer** tries to guess the **groff** command line options from the input using this program.

chem(1)
: Preprocessor of *groff* that is run automatically.

groff_out(5)
: Documentation on the *groff intermediate output* (*ditroff* output).

groff_tmac(5)
: Documentation on the *groff* macro files.

man(1) The standard program to display *man pages*. The information there is only useful if it is the *man page* for GNU **man**. Then it documents the options and environment variables that are supported by **groffer**.

gxditview(1), **xditview**(1x)
: Viewers for **groffer**'s *x mode*.

kpdf(1), **kghostview**(1), **evince**(1), **ggv**(1), **gv**(1), **ghostview**(1), **gs**(1)
: Viewers for **groffer**'s *ps mode*.

kpdf(1), **acroread**(1), **evince**(1), **xpdf**(1), **gpdf**(1), **kghostview**(1), **ggv**(1)
: Viewers for **groffer**'s *pdf mode*.

kdvi(1), **xdvi**(1), **dvilx**(1)
: Viewers for **groffer**'s *dvi mode*.

konqueror(1), **epiphany**(1), **firefox**(1), **mozilla**(1), **netscape**(1), **lynx**(1)
: Web-browsers for **groffer**'s *html* or *www mode*.

less(1) Standard pager program for the *tty mode* .

gzip(1), **bzip2**(1)
: The decompression programs supported by **groffer**.

AUTHOR

This file was written by Bernd Warken.

COPYING

GROG

NAME

grog – guess options for groff command

SYNOPSIS

[**–C**] [*groff–option* ...] [**––**] [*filespec* ...] **–h | ––help –v | ––version**

DESCRIPTION

grog reads the input (file names or standard input) and guesses which of the **groff**(1) options are needed to perform the input with the **groff** program. The corresponding **groff** command is output.

OPTIONS

The only **grog** options recognized are **–C** (which is also passed on) to enable compatibility mode; **–v** and **––version** print information on the version number; and **–h** and **––help** print usage information. **–v**, **––version**, **–h**, and **––help** stop the program directly without printing a **groff** command to standard output.

All other specified short options (words starting with one minus character **–**) are interpreted as **groff** options or option clusters with or without argument. No space is allowed between options and their argument. Except from the **–m***arg* options, all options will be passed on, i.e. they are included unchanged in the command for the output without effecting the work of **grog**.

A *filespec* argument can either be the name of an existing file or a single minus **–** to mean standard input. If no *filespec* is specified standard input is read automatically.

DETAILS

grog reads all *filespec* parameters as a whole. It tries to guess which of the following **groff** options are required for running the input under **groff**: **–e**, **–man**, **–me**, **–mm**, **–mom**, **–ms**, **–mdoc, –mdoc-old, –p**, **–R**, **–g**, **–G**, **–s**, and **–t**. The guessed **groff** command including those options and the found *filespec* parameters is put on the standard output.

It is possible to specify arbitrary **groff** options on the command line. These are passed on the output without change, except for the **–m***arg* options.

The **groff** program has trouble when the wrong **–m***arg* option or several of these options are specified. In these cases, **grog** will print an error message and exit with an error code. It is better to specify no **–m***arg* option. Because such an option is only accepted and passed when **grog** does not find any of these options or the same option is found.

If several different **–m***arg* options are found by **grog** an error message is produced and the program is terminated with an error code. But the output is written with the wrong options nevertheless.

Remember that it is not necessary to determine a macro package. A *roff* file can also be written in the *groff* language without any macro package. **grog** will produce an output without an **–m***arg* option.

As **groff** also works with pure text files without any *roff* requests, **grog** cannot be used to identify a file to be a *roff* file.

The **groffer**(1) program heavily depends on a working **grog**.

The **grog** source contains two files written in different programming languages: **`grog.pl`** is the *Perl* version, while **`grog.sh`** is a shell script using BR awk (1). During the run of **make**(1), it is determined whether the system contains a suitable version of **perl**(1). If so, **`grog.pl`** is transformed into **grog**; otherwise **`grog.sh`** is used instead.

EXAMPLES

- Calling

  ```
  grog meintro.me
  ```

  results in

  ```
  groff –me meintro.me
  ```

  So **grog** recognized that the file *meintro.me* is written with the **–me** macro package.

- On the other hand,

  ```
  grog pic.ms
  ```

outputs

```
groff −pte −ms pic.ms
```

Besides determining the macro package **−ms**, **grog** recognized that the file *pic.ms* additionally needs **−pte**, the combination of **−p** for *pic*, **−t** for *tbl*, and **−e** for *eqn*.

- If both files are combined by the command

  ```
  grog meintro.me pic.ms
  ```

  an error message is sent to standard error because **groff** cannot work with two different macro packages:

  ```
  grog: error: there are several macro packages: -me -ms
  ```

  Additionally the corresponding output with the wrong options is printed to standard output:

  ```
  groff -pte -me -ms meintro.me pic.ms
  ```

  But the program is terminated with an error code.

- The call of

  ```
  grog −ksS −Tdvi grnexmpl.g
  ```

  contains several **groff** options that are just passed on the output without any interface to **grog**. These are the option cluster **−ksS** consisting of **−k**, **−s**, and **−S**; and the option **−T** with argument **dvi**. The output is

  ```
  groff −ksS −Tdvi grnexmpl.g
  ```

  so no additional option was added by **grog**. As no option **−m***arg* was found by **grog** this file does not use a macro package.

- **grog** can also handle files using the *chem* language. The example

  ```
  grog chAh_brackets.chem
  ```

  outputs

  ```
  chem chAh_brackets.chem | groff −pe
  ```

  So **chem** is run first and **groff** is appended. The option **−p** for **pic** is implied automatically by **chem**. Additionally, the file uses *eqn* with **−e**.

SEE ALSO

groff(1), **troff**(1), **tbl**(1), **pic**(1), **eqn**(1), **refer**(1), **grn**(1), **grap**(1), **soelim**(1), **groff_me**(7), **groff_ms**(7), **groff_mm**(7), **groff_mom**(7), **groff_man**(7), **groffer**(1)

COPYING

GROHTML

NAME

grohtml – html driver for groff

SYNOPSIS

grohtml [**–bhlnprv**] [**–a***aa-text-bits*] [**–D***dir*] [**–F***dir*] [**–g***aa-graphic-bits*] [**–i***resolution*]
[**–I***image-stem*] [**–j***filename*] [**–o***image-vertical-offset*] [**–s***size*] [**–S***level*]
[**–x***html-dialect*] [*files*...]

DESCRIPTION

The **grohtml** front end (which consists of a preprocessor, **pre-grohtml**, and a device driver, **post-grohtml**) translates the output of GNU **troff** to html. Users should always invoke **grohtml** via the groff command with a **–Thtml** option. If no files are given, **grohtml** will read the standard input. A filename of **–** will also cause **grohtml** to read the standard input. Html output is written to the standard output. When **grohtml** is run by **groff** options can be passed to **grohtml** using **groff**'s **–P** option.

OPTIONS

–a*aa-text-bits*
Number of bits of antialiasing information to be used by *text* when generating png images. The default is 4 but valid values are 0, 1, 2, and 4. Note your version of **gs** needs to support the **–dTextAlphaBits** and **–dGraphicAlphaBits** options in order to exploit antialiasing. A value of 0 stops **grohtml** from issuing antialiasing commands to **gs**.

–b Initialize the background color to white.

–D*dir* Inform **grohtml** to place all image files into directory *dir*.

–e This option should not be directly invoked by the user as it is an internal option utilized by **groff** when **–Thtml** or **–Txhtml** is specified. It is used by the **grohtml** preprocessor to determine whether **eqn** should attempt to produce MathML (if **–Txhtml** is specified).

–F*dir* Prepend directory *dir***/dev***name* to the search path for font and device description files; *name* is the name of the device, usually **html**.

–g*aa-graphic-bits*
Number of bits of antialiasing information to be used by *graphics* when generating png images. The default is 4 but valid values are 0, 1, 2, and 4. Note your version of **gs** needs to support the **–dTextAlphaBits** and **–dGraphicAlphaBits** options in order to exploit antialiasing. A value of 0 stops **grohtml** from issuing antialiasing commands to **gs**.

–h Generate section and number headings by using **<B>**...**</B>** and increasing the font size, rather than using the **<H***n***>**...**</H***n***>** tags.

–i*resolution*
Select the resolution for all images. By default this is 100 pixels per inch. Example: **–i200** indicates 200 pixels per inch.

–I*stem* Determine the image stem name. If omitted grohtml uses **grohtml-***XXX* (*XXX* is the process ID).

–j*filename*
Inform **grohtml** to split the html output into multiple files. The *filename* is the stem and specified section headings (default is level one) start a new file, named *filename-n.html*.

–l Turn off the production of automatic section links at the top of the document.

–n Generate simple heading anchors whenever a section/number heading is found. Without the option the anchor value is the textual heading. This can cause problems when a heading contains a '?' on older versions of some browsers (Netscape). This flag is automatically turned on if a heading contains an image.

–o*vertical-offset*
Specify the vertical offset of images in points.

–p Display page rendering progress to stderr. **grohtml** only displays a page number when an image is required.

–r Turn off the automatic header and footer line (html rule).

−s size Set the base point size of the source file. Thereafter when this point size is used in the source it will correspond to the html base size. Every increase of two points in the source will yield a **<big>** tag, and conversely when a decrease of two points is seen a **<small>** tag is emitted.

−S*level* When splitting html output, split at the heading level (or higher) defined by *level*.

−v Print the version number.

−V Create an XHTML or HTML validator button at the bottom of each page of the document.

−x*dialect*
Select HTML dialect. Currently, *dialect* should be either the digit **4** or the letter **x** which indicates whether **grohtml** should generate HTML 4 or XHTML, respectively. This option should not be directly invoked by the user as it is an internal option utilized by **groff** when **−Thtml** or **−Txhtml** is specified.

−y Produce a right-justified groff signature at the end of the document. This is only generated if the **−V** flag is also specified.

USAGE

There are styles called **R**, **I**, **B**, and **BI** mounted at font positions 1 to 4.

DEPENDENCIES

grohtml is dependent upon the png utilities (**pnmcut**, **pnmcrop**, **pnmtopng**) and GhostScript (**gs**). **pnmtopng** (version 2.37.6 or greater) and **pnmcut** from the netpbm package (version 9.16 or greater) will work also. It is also dependent upon **psselect** from the **PSUtils** package. Images are generated whenever a table, picture, equation or line is encountered.

ENVIRONMENT

GROFF_FONT_PATH
A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

BUGS

Grohtml has been completely redesigned and rewritten. It is still beta code.

SEE ALSO

afmtodit(1), **groff**(1), **troff**(1), **psbb**(1), **groff_out**(5), **groff_font**(5), **groff_char**(7)

GROLBP

NAME

grolbp − groff driver for Canon CAPSL printers (LBP-4 and LBP-8 series laser printers).

SYNOPSIS

grolpb [**−l**] [**−−landscape**] [**−v**] [**−−version**] [**−c***n*] [**−−copies=***numcopies*] [**−p***paper_size*] [**−−papersize=***paper_size*] [**−o***orientation*] [**−−orientation=***orientation*] [**−w***width*] [**−−linewidth=***width*] [**−F***dir*] [**−−fontdir=***dir*] [**−h**] [**−−help**] [*files*. . .]

DESCRIPTION

grolbp is a driver for **groff** that produces output in CAPSL and VDM format suitable for Canon LBP−4 and LBP−8 printers.

For compatibility with grolj4 there is an additional drawing command available:

\D'R *dh dv***'**
Draw a rule (i.e. a solid black rectangle), with one corner at the current position, and the diagonally opposite corner at the current position +(*dh*,*dv*).

OPTIONS

Note that there can be whitespace between a one-letter option and its argument; on the other hand, there must be whitespace and/or an equal sign ('=') between a long-name option and its argument.

−c*numcopies*
−−copies=*numcopies*
Print *numcopies* copies of each page.

−l
−−landscape
Print the document with a landscape orientation.

−p*paper_size*
−−papersize=*paper_size*
Set the paper size to *paper_size*, which must be a valid paper size description as indicated in the section **PAPER SIZES**.

−o*orientation*
−−orientation=*orientation*
Print the document with *orientation* orientation, which must be 'portrait' or 'landscape'.

−w*width*
−−linewidth=*width*
Set the default line thickness to *width* thousandths of an em. If this option isn't specified, the line thickness defaults to 0.04 em.

−v
−−version
Print the version number.

−F*dir*
−−fontdir=*dir*
Prepend directory *dir***/dev***name* to the search path for font and device description files; *name* is the name of the device, usually **lbp**.

−h
−−help Print a short help text.

TYPEFACES

The driver supports the Dutch, Swiss and Swiss-Narrow scalable typefaces, each one in the Regular, Bold, Italic and Bold-Italic styles. Additionally, the Courier and Elite monospaced typefaces at the sizes 8 and 12 points (for Courier) resp. 8 and 10 points (for Elite) are supported, each one in the Regular, Bold and Italic styles.

The following chart summarizes the font names you can use to access these fonts:

PAPER SIZES

The paper size can be set in the **DESC** file or with command line options to **grolbp**. If the paper size is specified both ways, the command line options take precedence over the contents of the **DESC** file (this applies to the page orientation too).

See **groff_font**(1) how to set the paper dimensions in the **DESC** file.

To set the paper size in the command line, add

−p *paper-size*

or

−−papersize=*paper-size*

to the other **grolbp** options, where *paper-size* is in the same format as in the **DESC** file.

If no paper size is specified in the **DESC** file or the command line, a default size of A4 is used.

PAGE ORIENTATION

As with the page size, the orientation of the printed page (**portrait** or **landscape**) can be set in the **DESC** file or with command line options. It is also case insensitive.

To set the orientation in the **DESC** file, insert a line with the following content:

orientation [**portrait**|**landscape**]

Only the first valid orientation command in the **DESC** file is used.

To set the page orientation with command line options you can use the **−o** or **−−orientation** option with the same parameters (**portrait** or **landscape**) as in the **DESC** file. Or you can use the **−l** option to force the pages to be printed in landscape.

FONT FILE FORMAT

In addition to the usual commands described in **groff_font**(5), **grolbp** provides the command *lbpname* which sets the font name sent to the printer when requesting this font. The syntax of this command is:

lbpname *printer_font_name*

- For bitmapped fonts, *printer_font_name* has the form

  N⟨*base_fontname*⟩⟨*font_style*⟩

  base_fontname is the font name as it appears in the printers font listings without the first letter, up to (but not including) the font size. *font_style* can be one of the letters **R**, **I**, or **B**, indicating the font styles Roman, Italic and Bold respectively.

  For instance, if the printer's *font listing A* shows font ‘Nelite12I.ISO_USA’, the corresponding entry in the font description file is

  lbpname NeliteI

  Note that you may need to modify **grolbp** to add support for new bitmapped fonts, since the available font names and font sizes of bitmapped fonts (as documented above) are hard-coded into the program.

- For scalable fonts, *printer_font_name* is identical to the font name as it appears in the printer's *font listing A*.

  For instance, to select the ‘Swiss’ font in bold style, which appears in the printer's *font listing A* as ‘Swiss-Bold’, the required **lbpname** command line is

  lbpname Swiss-Bold

The argument of **lbpname** is case sensitive.

ENVIRONMENT

GROFF_FONT_PATH
A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

FILES

c:/progra~1/groff/share/groff/1.20/font/devlbp/DESC
Device description file.

c:/progra~1/groff/share/groff/1.20/font/devlbp/*F*
Font description file for font *F*.

c:/progra~1/groff/share/groff/1.20/tmac/lbp.tmac
Macros for use with **grolbp**.

SEE ALSO

groff(1), **troff**(1), **groff_out**(5), **groff_font**(5), **groff_char**(7)

GROLJ4

NAME

grolj4 – groff driver for HP Laserjet 4 family

SYNOPSIS

grolj4 [**–lv**] [**–d**[*n*]] [**–c***n*] [**–p***paper_size*] [**–w***n*] [**–F***dir*] [*files*...]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

grolj4 is a driver for **groff** that produces output in PCL5 format suitable for an HP Laserjet 4 printer.

There is an additional drawing command available:

\D'R *dh dv***'**
: Draw a rule (solid black rectangle), with one corner at the current position, and the diagonally opposite corner at the current position +(*dh*,*dv*). Afterwards the current position will be at the opposite corner. This generates a PCL fill rectangle command, and so will work on printers that do not support HPGL/2 unlike the other **\D** commands.

OPTIONS

–c*n* Print *n* copies of each page.

–l Print the document with a landscape orientation.

–d [*n***]** Use duplex mode *n*: 1 is long-side binding; 2 is short-side binding; default is 1.

–p*size* Set the paper size to *size*, which must be one of letter, legal, executive, a4, com10, monarch, c5, b5, dl.

–v Print the version number.

–w*n* Set the default line thickness to *n* thousandths of an em. If this option isn't specified, the line thickness defaults to 0.04 em.

–F*dir* Prepend directory *dir***/dev***name* to the search path for font and device description files; *name* is the name of the device, usually **lj4**.

The following four commands are available additionally in the font description files:

pclweight *N*
: The integer value *N* must be in the range -7 to +7; default is 0.

pclstyle *N*
: The integer value *N* must be in the range 0 to 32767; default is 0.

pclproportional *N*
: A boolean flag which can be either 0 or 1; default is 0.

pcltypeface *N*
: The integer value *N* must be in the range 0 to 65535; default is 0.

ENVIRONMENT

GROFF_FONT_PATH
: A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

FILES

c:/progra˜1/groff/share/groff/1.20/font/devlj4/DESC
: Device description file.

c:/progra˜1/groff/share/groff/1.20/font/devlj4/*F*
: Font description file for font *F*.

c:/progra˜1/groff/share/groff/1.20/tmac/lj4.tmac
: Macros for use with **grolj4**.

BUGS

Small dots.

SEE ALSO

lj4_font(5), **groff**(1), **troff**(1), **groff_out**(5), **groff_font**(5), **groff_char**(7)

GROPS

NAME

grops – PostScript driver for groff

SYNOPSIS

[**–glmv**] [**–b***n*] [**–c***n*] [**–F***dir*] [**–I***dir*] [**–p***papersize*] [**–P***prologue*] [**–w***n*] [*files* ...]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

grops translates the output of GNU **troff** to PostScript. Normally **grops** should be invoked by using the groff command with a **–Tps** option. (Actually, this is the default for groff.) If no files are given, **grops** reads the standard input. A filename of **–** also causes **grops** to read the standard input. PostScript output is written to the standard output. When **grops** is run by **groff** options can be passed to **grops** using **groff**'s **–P** option.

Note that **grops** doesn't produce a valid document structure (conforming to the Document Structuring Convention) if called with multiple file arguments. To print such concatenated output it is necessary to deactivate DSC handling in the printing program or previewer. See section **FONT INSTALLATION** below for a guide how to install fonts for **grops**.

OPTIONS

–b*n* Provide workarounds for older printers, broken spoolers, and previewers. Normally **grops** produces output at PostScript LanguageLevel 2 that conforms to the Document Structuring Conventions version 3.0. Some older printers, spoolers, and previewers can't handle such output. The value of *n* controls what **grops** does to make its output acceptable to such programs. A value of 0 causes grops not to employ any workarounds.

Add 1 if no **%%BeginDocumentSetup** and **%%EndDocumentSetup** comments should be generated; this is needed for early versions of TranScript that get confused by anything between the **%%EndProlog** comment and the first **%%Page** comment.

Add 2 if lines in included files beginning with **%!** should be stripped out; this is needed for Sun's pageview previewer.

Add 4 if **%%Page**, **%%Trailer** and **%%EndProlog** comments should be stripped out of included files; this is needed for spoolers that don't understand the **%%BeginDocument** and **%%EndDocument** comments.

Add 8 if the first line of the PostScript output should be **%!PS-Adobe-2.0** rather than **%!PS-Adobe-3.0**; this is needed when using Sun's Newsprint with a printer that requires page reversal.

Add 16 if no media size information should be included in the document (this is, neither use **%%DocumentMedia** nor the **setpagedevice** PostScript command). This was the behaviour of groff version 1.18.1 and earlier; it is needed for older printers which don't understand PostScript LanguageLevel 2. It is also necessary if the output is further processed to get an encapsulated PS (EPS) file – see below.

The default value can be specified by a

broken *n*

command in the DESC file. Otherwise the default value is 0.

–c*n* Print *n* copies of each page.

–F*dir* Prepend directory *dir***/dev***name* to the search path for prologue, font, and device description files; *name* is the name of the device, usually **ps**.

–g Guess the page length. This generates PostScript code that guesses the page length. The guess is correct only if the imageable area is vertically centered on the page. This option allows you to generate documents that can be printed both on letter (8.5×11) paper and on A4 paper without change.

–I*dir* This option may be used to add a directory to the search path for files on the command line and files named in **\X'ps: import'** and **\X'ps: file'** escapes. The search path is initialized with the current directory. This option may be specified more than once; the directories are then searched in the order specified (but before the current directory). If you want to make the current directory be read before other directories, add **–I.** at the appropriate place.

No directory search is performed for files with an absolute file name.

–l Print the document in landscape format.

–m Turn manual feed on for the document.

–p*paper-size*
Set physical dimension of output medium. This overrides the **papersize**, **paperlength**, and **paperwidth** commands in the **DESC** file; it accepts the same arguments as the **papersize** command. See **groff_font (5)** for details.

–P*prologue-file*
Use the file *prologue-file* (in the font path) as the prologue instead of the default prologue file **prologue**. This option overrides the environment variable GROPS_PROLOGUE.

–w*n* Lines should be drawn using a thickness of *n* thousandths of an em. If this option is not given, the line thickness defaults to 0.04 em.

–v Print the version number.

USAGE

The input to **grops** must be in the format output by **troff**(1). This is described in **groff_out**(5).

In addition, the device and font description files for the device used must meet certain requirements: The resolution must be an integer multiple of 72 times the **sizescale**. The **ps** device uses a resolution of 72000 and a sizescale of 1000.

The device description file must contain a valid paper size; see **groff_font**(5) for more information.

Each font description file must contain a command

internalname *psname*

which says that the PostScript name of the font is *psname*. It may also contain a command

encoding *enc_file*

which says that the PostScript font should be reencoded using the encoding described in *enc_file*; this file should consist of a sequence of lines of the form:

pschar code

where *pschar* is the PostScript name of the character, and *code* is its position in the encoding expressed as a decimal integer; valid values are in the range 0 to 255. Lines starting with # and blank lines are ignored. The code for each character given in the font file must correspond to the code for the character in encoding file, or to the code in the default encoding for the font if the PostScript font is not to be reencoded. This code can be used with the **\N** escape sequence in **troff** to select the character, even if the character does not have a groff name. Every character in the font file must exist in the PostScript font, and the widths given in the font file must match the widths used in the PostScript font. **grops** assumes that a character with a groff name of **space** is blank (makes no marks on the page); it can make use of such a character to generate more efficient and compact PostScript output.

Note that **grops** is able to display all glyphs in a PostScript font, not only 256. *enc_file* (or the default encoding if no encoding file specified) just defines the order of glyphs for the first 256 characters; all other glyphs are accessed with additional encoding vectors which **grops** produces on the fly.

grops can automatically include the downloadable fonts necessary to print the document. Such fonts must be in PFA format. Use **pfbtops**(1) to convert a Type 1 font in PFB format. Any downloadable fonts which should, when required, be included by **grops** must be listed in the file **c:/progra~1/groff/share/groff/1.20/font/devps/download**; this should consist of lines of the form

font filename

where *font* is the PostScript name of the font, and *filename* is the name of the file containing the font; lines beginning with # and blank lines are ignored; fields may be separated by tabs or spaces; *filename* is searched for using the same mechanism that is used for groff font metric files. The **download** file itself is also searched for using this mechanism; currently, only the first found file in the font path is used.

If the file containing a downloadable font or imported document conforms to the Adobe Document Structuring Conventions, then **grops** interprets any comments in the files sufficiently to ensure that its own output is conforming. It also supplies any needed font resources that are listed in the **download** file as well as any needed file resources. It is also able to handle inter-resource dependencies. For example, suppose that you have a downloadable font called Garamond, and also a downloadable font called Garamond-Outline which depends on Garamond (typically it would be defined to copy Garamond's font dictionary, and change the PaintType), then it is necessary for Garamond to appear before Garamond-Outline in the PostScript document. **grops** handles this automatically provided that the downloadable font file for Garamond-Outline indicates its dependence on Garamond by means of the Document Structuring Conventions, for example by beginning with the following lines

```
%!PS-Adobe-3.0 Resource-Font
%%DocumentNeededResources: font Garamond
%%EndComments
%%IncludeResource: font Garamond
```

In this case both Garamond and Garamond-Outline would need to be listed in the **download** file. A downloadable font should not include its own name in a **%%DocumentSuppliedResources** comment.

grops does not interpret **%%DocumentFonts** comments. The **%%DocumentNeededResources**, **%%DocumentSuppliedResources**, **%%IncludeResource**, **%%BeginResource**, and **%%EndResource** comments (or possibly the old **%%DocumentNeededFonts**, **%%DocumentSuppliedFonts**, **%%IncludeFont**, **%%BeginFont**, and **%%EndFont** comments) should be used.

In the default setup there are styles called **R**, **I**, **B**, and **BI** mounted at font positions 1 to 4. The fonts are grouped into families **A**, **BM**, **C**, **H**, **HN**, **N**, **P**, and **T** having members in each of these styles:

AR	AvantGarde-Book
AI	*AvantGarde-BookOblique*
AB	**AvantGarde-Demi**
ABI	***AvantGarde-DemiOblique***
BMR	Bookman-Light
BMI	*Bookman-LightItalic*
BMB	**Bookman-Demi**
BMBI	***Bookman-DemiItalic***
CR	Courier
CI	*Courier-Oblique*
CB	**Courier-Bold**
CBI	***Courier-BoldOblique***
HR	Helvetica
HI	*Helvetica-Oblique*
HB	**Helvetica-Bold**
HBI	***Helvetica-BoldOblique***
HNR	Helvetica-Narrow
HNI	*Helvetica-Narrow-Oblique*
HNB	**Helvetica-Narrow-Bold**
HNBI	***Helvetica-Narrow-BoldOblique***
NR	NewCenturySchlbk-Roman
NI	*NewCenturySchlbk-Italic*
NB	**NewCenturySchlbk-Bold**
NBI	***NewCenturySchlbk-BoldItalic***
PR	Palatino-Roman
PI	*Palatino-Italic*
PB	**Palatino-Bold**
PBI	***Palatino-BoldItalic***
TR	Times-Roman
TI	*Times-Italic*
TB	**Times-Bold**
TBI	***Times-BoldItalic***

There is also the following font which is not a member of a family:

ZCMI *ZapfChancery-MediumItalic*

There are also some special fonts called **S** for the PS Symbol font, and **SS**, containing slanted lowercase Greek letters taken from PS Symbol. Zapf Dingbats is available as **ZD**, and a reversed version of ZapfDingbats (with symbols pointing in the opposite direction) is available as **ZDR**; most characters in these fonts are unnamed and must be accessed using **\N**.

The default color for **\m** and **\M** is black; for colors defined in the 'rgb' color space **setrgbcolor** is used, for 'cmy' and 'cmyk' **setcmykcolor**, and for 'gray' **setgray**. Note that **setcmykcolor** is a PostScript LanguageLevel 2 command and thus not available on some older printers.

grops understands various X commands produced using the **\X** escape sequence; **grops** only interprets commands that begin with a **ps:** tag.

\X'ps: exec *code***'**
: This executes the arbitrary PostScript commands in *code*. The PostScript currentpoint is set to the position of the **\X** command before executing *code*. The origin is at the top left corner of the page, and y coordinates increase down the page. A procedure **u** is defined that converts groff units to the coordinate system in effect (provided the user doesn't change the scale). For example,

```
.nr x 1i
\X'ps: exec \nx u 0 rlineto stroke'
```

draws a horizontal line one inch long. *code* may make changes to the graphics state, but any changes persist only to the end of the page. A dictionary containing the definitions specified by the **def** and **mdef** is on top of the dictionary stack. If your code adds definitions to this dictionary, you should allocate space for them using **\X'ps mdef** *n***'**. Any definitions persist only until the end of the page. If you use the **\Y** escape sequence with an argument that names a macro, *code* can extend over multiple lines. For example,

```
.nr x 1i
.de y
ps: exec
\nx u 0 rlineto
stroke
..
\Yy
```

is another way to draw a horizontal line one inch long. Note the single backslash before 'nx' – the only reason to use a number register while defining the macro 'y' is to convert a user-specified dimension '1i' to internal groff units which are in turn converted to PS units with the **u** procedure.

grops wraps user-specified PostScript code into a dictionary, nothing more. In particular, it doesn't start and end the inserted code with **save** and **restore**, respectively. This must be supplied by the user, if necessary.

\X'ps: file *name***'**
: This is the same as the **exec** command except that the PostScript code is read from file *name*.

\X'ps: def *code***'**
: Place a PostScript definition contained in *code* in the prologue. There should be at most one definition per **\X** command. Long definitions can be split over several **\X** commands; all the *code* arguments are simply joined together separated by newlines. The definitions are placed in a dictionary which is automatically pushed on the dictionary stack when an **exec** command is executed. If you use the **\Y** escape sequence with an argument that names a macro, *code* can extend over multiple lines.

\X'ps: mdef *n code***'**
: Like **def**, except that *code* may contain up to *n* definitions. **grops** needs to know how many definitions *code* contains so that it can create an appropriately sized PostScript dictionary to contain them.

\X'ps: import *file llx lly urx ury width* [*height*]**'**
: Import a PostScript graphic from *file*. The arguments *llx*, *lly*, *urx*, and *ury* give the bounding

box of the graphic in the default PostScript coordinate system; they should all be integers; *llx* and *lly* are the x and y coordinates of the lower left corner of the graphic; *urx* and *ury* are the x and y coordinates of the upper right corner of the graphic; *width* and *height* are integers that give the desired width and height in groff units of the graphic.

The graphic is scaled so that it has this width and height and translated so that the lower left corner of the graphic is located at the position associated with **\X** command. If the height argument is omitted it is scaled uniformly in the x and y directions so that it has the specified width.

Note that the contents of the **\X** command are not interpreted by **troff**; so vertical space for the graphic is not automatically added, and the *width* and *height* arguments are not allowed to have attached scaling indicators.

If the PostScript file complies with the Adobe Document Structuring Conventions and contains a **%%BoundingBox** comment, then the bounding box can be automatically extracted from within groff by using the **psbb** request.

See **groff_tmac**(5) for a description of the **PSPIC** macro which provides a convenient high-level interface for inclusion of PostScript graphics.

\X'ps: invis'
\X'ps: endinvis'

No output is generated for text and drawing commands that are bracketed with these **\X** commands. These commands are intended for use when output from **troff** is previewed before being processed with **grops**; if the previewer is unable to display certain characters or other constructs, then other substitute characters or constructs can be used for previewing by bracketing them with these **\X** commands.

For example, **gxditview** is not able to display a proper **\(em** character because the standard X11 fonts do not provide it; this problem can be overcome by executing the following request

```
.char \(em \X'ps: invis'\
\Z'\v'-.25m'\h'.05m'\D'l .9m 0'\h'.05m''\
\X'ps: endinvis'\(em
```

In this case, **gxditview** is unable to display the **\(em** character and draws the line, whereas **grops** prints the **\(em** character and ignores the line (this code is already in file **Xps.tmac** which is loaded if a document intended for **grops** is previewed with **gxditview**).

If a PostScript procedure **BPhook** has been defined via a '**ps: def**' or '**ps: mdef**' device command, it is executed at the beginning of every page (before anything is drawn or written by groff). For example, to underlay the page contents with the word 'DRAFT' in light gray, you might use

```
.de XX
ps: def
/BPhook
{ gsave .9 setgray clippath pathbbox exch 2 copy
  .5 mul exch .5 mul translate atan rotate pop pop
  /NewCenturySchlbk-Roman findfont 200 scalefont setfont
  (DRAFT) dup stringwidth pop −.5 mul −70 moveto show
  grestore }
def
..
.devicem XX
```

Or, to cause lines and polygons to be drawn with square linecaps and mitered linejoins instead of the round linecaps and linejoins normally used by **grops**, use

```
.de XX
ps: def
/BPhook { 2 setlinecap 0 setlinejoin } def
..
.devicem XX
```

(square linecaps, as opposed to butt linecaps (0 setlinecap), give true corners in boxed tables even though the lines are drawn unconnected).

Encapsulated PostScript

grops itself doesn't emit bounding box information. With the help of Ghostscript the following simple script, **groff2eps**, produces an encapsulated PS file.

```
#! /bin/sh
groff −P−b16 $1 >$1.ps
gs −dNOPAUSE −sDEVICE=bbox −− $1.ps 2>$1.bbox
cat $1.ps \
| sed −e "/^%%Orientation/r$1.bbox" \
    −e "/^%!PS-Adobe-3.0/s/$/ EPSF-3.0/" >$1.eps
rm $1.ps $1.bbox
```

Just say

```
groff2eps foo
```

to convert file **foo** to **foo.eps**.

TrueType and other font formats

TrueType fonts can be used with **grops** if converted first to **Type 42** format, a special PostScript wrapper equivalent to the PFA format mentioned in **pfbtops**(1). There are several different methods to generate a type42 wrapper and most of them involve the use of a PostScript interpreter such as Ghostscript – see **gs**(1).

Yet, the easiest method involves the use of the application **ttftot42**(1). This program uses **freetype**(3) (version 1.3.1) to generate type42 font wrappers and well-formed AFM files that can be fed to the **afmtodit**(1) script to create appropriate metric files. The resulting font wrappers should be added to the **download** file. **ttftot42** source code can be downloaded from ftp://www.giga.or.at/pub/nih/ttftot42/

Another solution for creating type42 wrappers is to use FontForge, available from http://fontforge.sf.net This font editor can convert most outline font formats.

FONT INSTALLATION

This section gives a summary of the above explanations; it can serve as a step-by-step font installation guide for **grops**.

- Convert your font to something groff understands. This is either a PostScript Type 1 font in PFA format or a PostScript Type 42 font, together with an AFM file.

  The very first characters in a PFA file look like this:

  ```
  %!PS-AdobeFont-1.0:
  ```

  A PFB file has this also in the first line, but the string is preceded with some binary bytes.

  The very first characters in a Type 42 font file look like this:

  ```
  %!PS-TrueTypeFont
  ```

  This is a wrapper format for TrueType fonts. Old PS printers might not support it (this is, they don't have a built-in TrueType font interpreter).

  If your font is in PFB format (such fonts normally have '.pfb' as the file extension), you might use groff's **pfbtops**(1) program to convert it to PFA. For TrueType fonts, try **ttftot42** or **fontforge**. For all other font formats use **fontforge** which can convert most outline font formats.

- Convert the AFM file to a groff font description file with the **afmtodit**(1) program. An example call is

  ```
  afmtodit Foo-Bar-Bold.afm textmap FBB
  ```

  which converts the metric file 'Foo-Bar-Bold.afm' to the groff font 'FBB'. If you have a font family which comes with normal, bold, italic, and bold italic faces, it is recommended to use the letters **R**, **B**, **I**, and **BI**, respectively, as postfixes in the groff font names to make groff's '.fam' request work. An example is groff's built-in Times-Roman font: The font family name is **T**, and the groff font names are **TR**, **TB**, **TI**, and **TBI**.

- Install both the groff font description files and the fonts in a 'devps' subdirectory of the font path which groff finds. See the **ENVIRONMENT** section in the **troff**(1) man page which lists the actual value of the font path. Note that groff doesn't use the AFM files (but it is a good idea to store them anyway).

- Register all fonts which must be downloaded to the printer in the 'devps/download' file. Only the first occurrence of this file in the font path is read. This means that you should copy the default 'download' file to the first directory in your font path and add your fonts there. To continue the above example we assume that the PS font name for Foo-Bar-Bold.pfa is 'XY-Foo-Bar-Bold' (the PS font name is stored in the **internalname** field in the 'FBB' file), thus the following line should be added to 'download'.

  XY-Foo-Bar-Bold Foo-Bar-Bold.pfa

OLD FONTS

groff versions 1.19.2 and earlier contain a slightly different set of the 35 Adobe core fonts; the difference is mainly the lack of the 'Euro' glyph and a reduced set of kerning pairs. For backwards compatibility, these old fonts are installed also in the

c:/progra˜1/groff/share/groff/1.20/oldfont/devps

directory.

To use them, make sure that **grops** finds the fonts before the default system fonts (with the same names): Either add command line option **−F** to **grops**

groff −Tps −P−F −Pc:/progra˜1/groff/share/groff/1.20/oldfont . . .

or add the directory to groff's font path environment variable

GROFF_FONT_PATH=c:/progra˜1/groff/share/groff/1.20/oldfont

ENVIRONMENT

GROPS_PROLOGUE
If this is set to *foo*, then **grops** uses the file *foo* (in the font path) instead of the default prologue file **prologue**. The option **−P** overrides this environment variable.

GROFF_FONT_PATH
A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

FILES

c:/progra˜1/groff/share/groff/1.20/font/devps/DESC
Device description file.

c:/progra˜1/groff/share/groff/1.20/font/devps/*F*
Font description file for font *F*.

c:/progra˜1/groff/share/groff/1.20/font/devps/download
List of downloadable fonts.

c:/progra˜1/groff/share/groff/1.20/font/devps/text.enc
Encoding used for text fonts.

c:/progra˜1/groff/share/groff/1.20/tmac/ps.tmac
Macros for use with **grops**; automatically loaded by **troffrc**

c:/progra˜1/groff/share/groff/1.20/tmac/pspic.tmac
Definition of **PSPIC** macro, automatically loaded by **ps.tmac**.

c:/progra˜1/groff/share/groff/1.20/tmac/psold.tmac
Macros to disable use of characters not present in older PostScript printers (e.g., 'eth' or 'thorn').

/tmp/grops*XXXXXX*
Temporary file.

SEE ALSO

afmtodit(1), **groff**(1), **troff**(1), **pfbtops**(1), **groff_out**(5), **groff_font**(5), **groff_char**(7), **groff_tmac**(5)

PostScript Language Document Structuring Conventions Specification

GROTTY

NAME

grotty – groff driver for typewriter-like devices

SYNOPSIS

grotty [**–bBcdfhioruUv**] [**–F***dir*] [*files* . . .]

It is possible to have whitespace between the **–F** option and its parameter.

DESCRIPTION

grotty translates the output of GNU **troff** into a form suitable for typewriter-like devices. Normally **grotty** should be invoked by using the **groff** command with a **–Tascii**, **–Tlatin1** or **–Tutf8** option on ASCII based systems, and with **–Tcp1047** and **–Tutf8** on EBCDIC based hosts. If no files are given, **grotty** reads the standard input. A filename of **–** also causes **grotty** to read the standard input. Output is written to the standard output.

By default, **grotty** emits SGR escape sequences (from ISO 6429, also called ANSI color escapes) to change text attributes (bold, italic, colors). This makes it possible to have eight different background and foreground colors; additionally, bold and italic attributes can be used ***at the same time*** (by using the BI font).

The following colors are defined in **tty.tmac**: black, white, red, green, blue, yellow, magenta, cyan. Unknown colors are mapped to the default color (which is dependent on the settings of the terminal; in most cases, this is black for the foreground and white for the background).

Use the **–c** switch to revert to the old behaviour, printing a bold character *c* with the sequence '*c* BACKSPACE *c*' and an italic character *c* by the sequence '_ BACKSPACE *c*'. At the same time, color output is disabled. The same effect can be achieved by setting either the **GROFF_NO_SGR** environment variable or using the 'sgr' X command (see below).

For SGR support, it is necessary to use the **–R** option of **less**(1) to disable the interpretation of **grotty**'s old output format. Consequently, all programs which use **less** as the pager program have to pass this option to it. For **man**(1) in particular, either add **–R** to the **$PAGER** environment variable, e.g.

```
PAGER="/usr/bin/less -R"
export PAGER
```

or use the **–P** option of **man** to set the pager executable and its options, or modify the configuration file of **man** in a similar fashion. Note that with some **man**(1) versions, you have to use the **$MANPAGER** environment variable instead.

grotty's old output format can be displayed on a terminal by piping through **ul**(1). Pagers such as **more**(1) or **less**(1) are also able to display these sequences. Use either **–B** or **–U** when piping into **less**(1); use **–b** when piping into **more**(1). There is no need to filter the output through **col**(1) since **grotty** never outputs reverse line feeds.

The font description file may contain a command

internalname *n*

where *n* is a decimal integer. If the 01 bit in *n* is set, then the font is treated as an italic font; if the 02 bit is set, then it is treated as a bold font. The code field in the font description field gives the code which is used to output the character. This code can also be used in the **\N** escape sequence in **troff**.

OPTIONS

–b Suppress the use of overstriking for bold characters. Ignored if **–c** isn't used.

–B Use only overstriking for bold-italic characters. Ignored if **–c** isn't used.

–c Use **grotty**'s old output format (see above). This also disables color output.

–d Ignore all **\D** commands. Without this **grotty** renders **\D'l . . .'** commands that have at least one zero argument (and so are either horizontal or vertical) using **–**, **|**, and **+** characters. In a similar way, **grotty** handles **\D'p . . .'** commands which consist entirely of horizontal and vertical lines.

–f Use form feeds in the output. A form feed is output at the end of each page that has no output on its last line.

−F*dir* Prepend directory *dir***/dev***name* to the search path for font and device description files; *name* is the name of the device, usually **ascii**, **latin1**, **utf8**, or **cp1047**.

−h Use horizontal tabs in the output. Tabs are assumed to be set every 8 columns.

−i Use escape sequences to set the italic text attribute instead of the underline attribute for italic fonts ('I' and 'BI'). Note that most terminals (including xterm) don't support this. Ignored if **−c** is active.

−o Suppress overstriking (other than for bold or underlined characters in case the old output format has been activated with **−c**).

−r Use escape sequences to set the reverse text attribute instead of the underline attribute for italic fonts ('I' and 'BI'). Ignored if **−c** is active.

−u Suppress the use of underlining for italic characters. Ignored if **−c** isn't used.

−U Use only underlining for bold-italic characters. Ignored if **−c** isn't used.

−v Print the version number.

USAGE

grotty understands a single X command produced using the **\X** escape sequence.

\X'tty: sgr *n***'**
If *n* is non-zero or missing, enable SGR output (this is the default), otherwise use the old drawing scheme for bold and underline.

ENVIRONMENT

GROFF_NO_SGR
If set, the old drawing scheme for bold and underline (using the backspace character) is active. Colors are disabled.

GROFF_FONT_PATH
A list of directories in which to search for the **dev***name* directory in addition to the default ones. See **troff**(1) and **groff_font**(5) for more details.

FILES

c:/progra˜1/groff/share/groff/1.20/font/devascii/DESC
Device description file for **ascii** device.

c:/progra˜1/groff/share/groff/1.20/font/devascii/*F*
Font description file for font *F* of **ascii** device.

c:/progra˜1/groff/share/groff/1.20/font/devlatin1/DESC
Device description file for **latin1** device.

c:/progra˜1/groff/share/groff/1.20/font/devlatin1/*F*
Font description file for font *F* of **latin1** device.

c:/progra˜1/groff/share/groff/1.20/font/devutf8/DESC
Device description file for **utf8** device.

c:/progra˜1/groff/share/groff/1.20/font/devutf8/*F*
Font description file for font *F* of **utf8** device.

c:/progra˜1/groff/share/groff/1.20/font/devcp1047/DESC
Device description file for **cp1047** device.

c:/progra˜1/groff/share/groff/1.20/font/devcp1047/*F*
Font description file for font *F* of **cp1047** device.

c:/progra˜1/groff/share/groff/1.20/tmac/tty.tmac
Macros for use with **grotty**.

c:/progra˜1/groff/share/groff/1.20/tmac/tty-char.tmac
Additional klugdey character definitions for use with **grotty**.

Note that on EBCDIC hosts, only files for the **cp1047** device is installed.

BUGS

grotty is intended only for simple documents.

There is no support for fractional horizontal or vertical motions.

There is no support for **\D** commands other than horizontal and vertical lines.

Characters above the first line (ie with a vertical position of 0) cannot be printed.

Color handling is different compared to **grops**(1). **\M** doesn't set the fill color for closed graphic objects (which **grotty** doesn't support anyway) but changes the background color of the character cell, affecting all subsequent operations.

SEE ALSO

groff(1), **troff**(1), **groff_out**(5), **groff_font**(5), **groff_char**(7), **ul**(1), **more**(1), **man**(1), **less**(1)

HPFTODIT

NAME

hpftodit − create font description files for use with groff −Tlj4

SYNOPSIS

hpftodit [**−adqsv**] [**−i***n*] *tfm_file map_file font*

It is possible to have whitespace between the **−i** option and its parameter.

DESCRIPTION

hpftodit creates a font file for use with a Hewlett-Packard LaserJet 4–series (or newer) printer with **groff −Tlj4**, using data from an HP tagged font metric (TFM) file. *tfm_file* is the name of the TFM file for the font; Intellifont and TrueType TFM files are supported, but symbol set TFM files are not. *map_file* is a file giving the groff names for characters in the font; this file should consist of a sequence of lines of the form:

m u c1 c2 . . . [# *comment*]

where *m* is a decimal integer giving the MSL number of the character, *u* is a hexadecimal integer giving the Unicode value of the character, and *c1*, *c2*, . . . are the groff names of the character. The values can be separated by any whitespace; the Unicode value must use uppercase digits A−F, and must be without a leading '0x', 'u', or 'U+'. Unicode values corresponding to composite glyphs are decomposed; e.g., 'u00C0' becomes 'u0041_0300'. The name for a glyph without a groff name may be given as u*XXXX* if the glyph corresponds to a Unicode value, or as an unnamed glyph '---'. If the given Unicode value is in the Private Use Area (0xE000−0xF8FF), the glyph is included as an unnamed glyph. Refer to **groff_diff**(1) for additional information about unnamed glyphs and how to access them.

Blank lines and lines beginning with '#' are ignored. A '#' following one or more groff names begins a comment. Because '#' is a valid groff name, it must appear first in a list of groff names if a comment is included, e.g.,

```
3   0023   #   # number sign
```

or

```
3   0023   # sh   # number sign
```

rather than

```
3   0023   sh #   # number sign
```

which will treat the first '#' as the beginning of the comment.

font is the name of the groff font file. The groff font file is written to *font*; if *font* is specified as '−', the output is written to the standard output.

The **−s** option should be given if the font is special (a font is *special* if **troff** should search it whenever a character is not found in the current font). If the font is special, it should be listed in the **fonts** command in the DESC file; if it is not special, there is no need to list it, since **troff** can automatically mount it when it's first used.

If the **−i** option is used, **hpftodit** automatically will generate an italic correction, a left italic correction and a subscript correction for each character (the significance of these parameters is explained in **groff_font**(5)).

OPTIONS

−a Include characters in the TFM file that are not included in the map file. A glyph with corresponding Unicode value is given the name u*XXXX*; a glyph without a Unicode value is included as an unnamed glyph '---'. A glyph with a Unicode value in the Private Use Area (0xE000−0xF8FF) also is included as an unnamed glyph.

This option provides a simple means of adding Unicode-named and unnamed glyphs to a font without including them in the map file, but it affords little control over which glyphs are placed in a regular font and which are placed in a special font. The presence or absence of the **−s** option has some effect on which glyphs are included: without the **−s** option, only the "text" symbol sets are searched for matching glyphs; with the **−s** option, only the "mathematical" symbol sets are searched. Nonetheless, restricting the symbol sets searched isn't very selective—many glyphs are placed in both regular and special fonts. Normally, the **−a** option should be used only as a last resort.

−d Dump information about the TFM file to the standard output; this option can be useful for ensuring that a TFM file is a proper match for a font, and that the contents of the TFM file are suitable. The information includes the values of important TFM tags, and a listing (by MSL number for Intellifont TFM files or by Unicode value for TrueType TFM files) of the glyphs included in the TFM file. The unit of measure 'DU' for some tags indicates design units; there are 8782 design units per em for Intellifont fonts, and 2048 design units per em for TrueType fonts. Note that the accessibility of a glyph depends on its inclusion in a symbol set; some TFM files list many glyphs but only a few symbol sets.

The glyph listing includes the glyph index within the TFM file, the MSL or Unicode value, and the symbol set and character code that will be used to print the glyph. If *map_file* is given, groff names are given for matching glyphs. If only the glyph index and MSL or Unicode value are given, the glyph does not appear in any supported symbol set and cannot be printed.

With the **−d** option, *map_file* is optional, and *font* is ignored if given.

−q Suppress warnings about characters in the map file that were not found in the TFM file. Warnings never are given for unnamed glyphs or by glyphs named by their Unicode values. This option is useful when sending the output of **hpftodit** to the standard output.

−v Print the **hpftodit** version number.

−s The font is special. This option adds the **special** command to the font file, and affects the order in which HP symbol sets are searched for each glyph. Without the **−s** option, the "text" sets are searched before the "mathematical" symbol sets. With the **−s** option, the search order is reversed.

−i*n* Generate an italic correction for each character so that the character's width plus the character's italic correction is equal to *n* thousandths of an em plus the amount by which the right edge of the character's bounding is to the right of the character's origin. If this would result in a negative italic correction, use a zero italic correction instead.

Also generate a subscript correction equal to the product of the tangent of the slant of the font and four fifths of the x-height of the font. If this would result in a subscript correction greater than the italic correction, use a subscript correction equal to the italic correction instead.

Also generate a left italic correction for each character equal to *n* thousandths of an em plus the amount by which the left edge of the character's bounding box is to the left of the character's origin. The left italic correction may be negative.

This option normally is needed only with italic or oblique fonts; a value of 50 (0.05 em) usually is a reasonable choice.

FILES

c:/progra 1/groff/share/groff/1.20/font/devlj4/DESC	Device description file.
c:/progra 1/groff/share/groff/1.20/font/devlj4/*F*	Font description file for font *F*.
c:/progra 1/groff/share/groff/1.20/font/devlj4/generate/*.map	Symbol mapping files

SEE ALSO

groff(1), **groff_diff**(1), **grolj4**(1), **groff_font**(5), **lj4_font**(5)

INDXBIB

NAME

indxbib − make inverted index for bibliographic databases

SYNOPSIS

indxbib [**−vw**] [**−c***file*] [**−d***dir*] [**−f***file*] [**−h***n*] [**−i***string*] [**−k***n*] [**−l***n*] [**−n***n*] [**−o***file*] [**−t***n*] [*filename* ...]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

indxbib makes an inverted index for the bibliographic databases in *filename* ... for use with **refer**(1), **lookbib**(1), and **lkbib**(1). The index will be named *filename***.i**; the index is written to a temporary file which is then renamed to this. If no filenames are given on the command line because the **−f** option has been used, and no **−o** option is given, the index will be named **Ind.i**.

Bibliographic databases are divided into records by blank lines. Within a record, each fields starts with a **%** character at the beginning of a line. Fields have a one letter name which follows the **%** character.

The values set by the **−c**, **−n**, **−l** and **−t** options are stored in the index; when the index is searched, keys will be discarded and truncated in a manner appropriate to these options; the original keys will be used for verifying that any record found using the index actually contains the keys. This means that a user of an index need not know whether these options were used in the creation of the index, provided that not all the keys to be searched for would have been discarded during indexing and that the user supplies at least the part of each key that would have remained after being truncated during indexing. The value set by the **−i** option is also stored in the index and will be used in verifying records found using the index.

OPTIONS

−v Print the version number.

−w Index whole files. Each file is a separate record.

−c*file* Read the list of common words from *file* instead of **c:/progra 1/groff/share/groff/1.20/eign**.

−d*dir* Use *dir* as the pathname of the current working directory to store in the index, instead of the path printed by **pwd**(1). Usually *dir* will be a symbolic link that points to the directory printed by **pwd**(1).

−f*file* Read the files to be indexed from *file*. If *file* is **−**, files will be read from the standard input. The **−f** option can be given at most once.

−i*string*
Don’t index the contents of fields whose names are in *string*. Initially *string* is **XYZ**.

−h*n* Use the first prime greater than or equal to *n* for the size of the hash table. Larger values of *n* will usually make searching faster, but will make the index larger and **indxbib** use more memory. Initially *n* is 997.

−k*n* Use at most *n* keys per input record. Initially *n* is 100.

−l*n* Discard keys that are shorter than *n*. Initially *n* is 3.

−n*n* Discard the *n* most common words. Initially *n* is 100.

−o*basename*
The index should be named *basename***.i**.

−t*n* Truncate keys to *n*. Initially *n* is 6.

FILES

*filename***.i**	Index.
Ind.i	Default index name.
c:/progra 1/groff/share/groff/1.20/eign	List of common words.
indxbib*XXXXXX*	Temporary file.

SEE ALSO

refer(1), **lkbib**(1), **lookbib**(1)

LKBIB

NAME

lkbib − search bibliographic databases

SYNOPSIS

lkbib [**−v**] [**−i***fields*] [**−p***filename*] [**−t***n*] *key* . . .

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

lkbib searches bibliographic databases for references that contain the keys *key* . . . and prints any references found on the standard output. **lkbib** will search any databases given by **−p** options, and then a default database. The default database is taken from the **REFER** environment variable if it is set, otherwise it is **/usr/dict/papers/Ind**. For each database *filename* to be searched, if an index *filename***.i** created by **indxbib**(1) exists, then it will be searched instead; each index can cover multiple databases.

OPTIONS

−v Print the version number.

−p*filename*
Search *filename*. Multiple **−p** options can be used.

−i*string*
When searching files for which no index exists, ignore the contents of fields whose names are in *string*.

−t*n* Only require the first *n* characters of keys to be given. Initially *n* is 6.

ENVIRONMENT

REFER Default database.

FILES

/usr/dict/papers/Ind	Default database to be used if the **REFER** environment variable is not set.
*filename***.i**	Index files.

SEE ALSO

refer(1), **lookbib**(1), **indxbib**(1)

LOOKBIB

NAME

lookbib − search bibliographic databases

SYNOPSIS

lookbib [**−v**] [**−i***string*] [**−t***n*] *filename* . . .

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

lookbib prints a prompt on the standard error (unless the standard input is not a terminal), reads from the standard input a line containing a set of keywords, searches the bibliographic databases *filename* . . . for references containing those keywords, prints any references found on the standard output, and repeats this process until the end of input. For each database *filename* to be searched, if an index *filename***.i** created by **indxbib**(1) exists, then it will be searched instead; each index can cover multiple databases.

OPTIONS

−v Print the version number.

−i*string*
When searching files for which no index exists, ignore the contents of fields whose names are in *string*.

−t*n* Only require the first *n* characters of keys to be given. Initially *n* is 6.

FILES

*filename***.i** Index files.

SEE ALSO

refer(1), **lkbib**(1), **indxbib**(1)

MMROFF

NAME

mmroff – reference preprocessor

SYNOPSIS

mmroff [*-x*] *groff_arguments*

DESCRIPTION

mmroff is a simple preprocessor for **groff**, it is used for expanding references in **mm**, see **groff_mm(7)**. **groff** is executed twice, first with **-z** and **-rRef=1** to collect all references and then to do the real processing when the reference file is up to date.

−x Just create the reference file. This can be used to refresh the reference file, it isn't always needed to have accurate references and by using this option **groff** will only be run once.

AUTHOR

Jörgen Hägg, Lund, Sweden <jh@axis.se>.

FILES

c:/progra 1/groff/share/groff/1.20/tmac/m.tmac

c:/progra 1/groff/share/groff/1.20/tmac/mm/*.cov

c:/progra 1/groff/share/groff/1.20/tmac/mm/*.MT

c:/progra 1/groff/share/groff/1.20/tmac/mm/locale

SEE ALSO

groff_mm(7), groff_mmse(7), groff(1), **troff**(1), **tbl**(1), **pic**(1), **eqn**(1)

NEQN

NAME

neqn − format equations for ascii output

SYNOPSIS

neqn [eqn options]

DESCRIPTION

The **neqn** program is actually just a shell script which invokes the **eqn**(1) command with the **ascii** output device.

Note that **eqn** does not support low-resolution, typewriter-like devices (although it may work adequately for very simple input).

SEE ALSO

eqn(1)

NROFF

NAME

nroff – emulate nroff command with groff

SYNOPSIS

[**−CchipStUvwW**] [**−d***cs*] [**−M***dir*] [**−m***name*] [**−n***num*] [**−o***list*] [**−r***cn*] [**−T***name*] [*file* ...] **−−help**
−v | **−−version**

DESCRIPTION

The **nroff** script emulates the **nroff** command using groff. Only **ascii**, **latin1**, **utf8**, and **cp1047** are devices accepted by **nroff** to select the output encoding emitted by **grotty**, groff's TTY output device. If neither the **GROFF_TYPESETTER** environment variable nor the **−T** command line option (which overrides the environment variable) specifies a (valid) device, **nroff** checks the current locale to select a default output device. It first tries the **locale** program, then the environment variables **LC_ALL**, **LC_CTYPE**, and **LANG**, and finally the **LESSCHARSET** environment variable.

The **−h** and **−c** options are equivalent to **grotty**'s options **−h** (using tabs in the output) and **−c** (using the old output scheme instead of SGR escape sequences). The **−d**, **−C**, **−i**, **−M**, **−m**, **−n**, **−o**, **−r**, **−w**, and **−W** options have the effect described in **troff**(1). In addition, **nroff** silently ignores the options **−e**, **−q**, and **−s** (which are not implemented in **troff**). Options **−p** (pic), **−t** (tbl), **−S** (safer), and **−U** (unsafe) are passed to **groff**. **−v** and **−−version** show the version number, **−−help** prints a help message.

ENVIRONMENT

GROFF_TYPESETTER
The default device for **groff**. If not set (which is the normal case), it defaults to 'ps'.

GROFF_BIN_PATH
A colon separated list of directories in which to search for the **groff** executable before searching in PATH. If unset, 'c:/progra 1/groff/bin' is used.

NOTES

This shell script is basically intended for use with **man**(1). nroff-style character definitions (in the file tty-char.tmac) are also loaded to emulate unrepresentable glyphs.

SEE ALSO

groff(1), **troff**(1), **grotty**(1)

PDFROFF

NAME

pdfroff – create PDF documents using groff

SYNOPSIS

[**–abcegilpstzCEGNRSUVXZ**] [**–d***cs*] [**–f***fam*] [**–F***dir*] [**–I***dir*] [**–L***arg*] [**–m***name*] [**–M***dir*] [**–n***num*] [**–o***list*] [**–P***arg*] [**–r***cn*] [**–T***dev*] [**–w***name*] [**–W***name*] [**––emit–ps**] [**––no–toc–relocation**] [**––no-kill–null–pages**] [**––stylesheet=***name*] [**––no–pdf–output**] [**––pdf–output=***name*] [**––no–reference–dictionary**] [**––reference–dictionary=***name*] [**––report–progress**] [**––keep–temporary–files**] *file* . . . **–h** | **––help** **–v** | **––version** [*option* . . .]

DESCRIPTION

pdfroff is a wrapper program for the GNU text processing system, **groff**. It transparently handles the mechanics of multiple pass **groff** processing, when applied to suitably marked up **groff** source files, such that tables of contents and body text are formatted separately, and are subsequently combined in the correct order, for final publication as a single PDF document. A further optional "style sheet" capability is provided; this allows for the definition of content which is required to precede the table of contents, in the published document.

For each invocation of **pdfroff**, the ultimate **groff** output stream is post-processed by the GhostScript interpreter, to produce a finished PDF document.

pdfroff makes no assumptions about, and imposes no restrictions on, the use of any **groff** macro packages which the user may choose to employ, in order to achieve a desired document format; however, it *does* include specific built in support for the **pdfmark** macro package, should the user choose to employ it. Specifically, if the *pdfhref* macro, defined in the **pdfmark.tmac** package, is used to define public reference marks, or dynamic links to such reference marks, then **pdfroff** performs as many preformatting **groff** passes as required, up to a maximum limit of *four*, in order to compile a document reference dictionary, to resolve references, and to expand the dynamically defined content of links.

USAGE

The command line is parsed in accordance with normal GNU conventions, but with one exception — when specifying any short form option (i.e., a single character option introduced by a single hyphen), and if that option expects an argument, then it *must* be specified independently (i.e., it may *not* be appended to any group of other single character short form options).

Long form option names (i.e., those introduced by a double hyphen) may be abbreviated to their minimum length unambiguous initial substring.

Otherwise, **pdfroff** usage closely mirrors that of **groff** itself. Indeed, with the exception of the **–h**, **–v**, and **–T** *dev* short form options, and all long form options, which are parsed internally by **pdfroff**, all options and file name arguments specified on the command line are passed on to **groff**, to control the formatting of the PDF document. Consequently, **pdfroff** accepts all options and arguments, as specified in **groff**(1), which may also be considered as the definitive reference for all standard **pdfroff** options and argument usage.

OPTIONS

pdfroff accepts all of the short form options (i.e., those introduced by a single hyphen), which are available with **groff** itself. In most cases, these are simply passed transparently to **groff**; the following, however, are handled specially by **pdfroff**.

–h Same as **––help**; see below.

–i Process standard input, after all other specified input files. This is passed transparently to **groff**, but, if grouped with other options, it *must* be the first in the group. Hiding it within a group breaks standard input processing, in the multiple pass **groff** processing context of **pdfroff**.

–T *dev* Only **–T ps** is supported by **pdfroff**. Attempting to specify any other device causes **pdfroff** to abort.

–v Same as **––version**; see below.

See **groff**(1) for a description of all other short form options, which are transparently passed through **pdfroff** to **groff**.

All long form options (i.e., those introduced by a double hyphen) are interpreted locally by **pdfroff**;

they are **not** passed on to **groff**, unless otherwise stated below.

−−help Causes **pdfroff** to display a summary of the its usage syntax, and supported options, and then exit.

−−emit−ps
Suppresses the final output conversion step, causing **pdfroff** to emit PostScript output instead of PDF. This may be useful, to capture intermediate PostScript output, when using a specialised postprocessor, such as *gpresent* for example, in place of the default *GhostScript* PDF writer.

−−keep−temporary−files
Suppresses the deletion of temporary files, which normally occurs after **pdfroff** has completed PDF document formatting; this may be useful, when debugging formatting problems.

See section **FILES**, for a description of the temporary files used by **pdfroff**.

−−no−pdf−output
May be used with the **−−reference−dictionary=***name* option (described below) to eliminate the overhead of PDF formatting, when running **pdfroff** to create a reference dictionary, for use in a different document.

−−no−reference−dictionary
May be used to eliminate the overhead of creating a reference dictionary, when it is known that the target PDF document contains no public references, created by the *pdfhref* macro.

−−no−toc−relocation
May be used to eliminate the extra **groff** processing pass, which is required to generate a table of contents, and relocate it to the start of the PDF document, when processing any document which lacks an automatically generated table of contents.

−−no−kill−null−pages
While preparing for simulation of the manual collation step, which is traditionally required to relocate of a *table of contents* to the start of a document, **pdfroff** accumulates a number of empty page descriptions into the intermediate *PostScript* output stream. During the final collation step, these empty pages are normally discarded from the finished document; this option forces **pdfroff** to leave them in place.

−−pdf−output=*name*
Specifies the name to be used for the resultant PDF document; if unspecified, the PDF output is written to standard output. A future version of **pdfroff** may use this option, to encode the document name in a generated reference dictionary.

−−reference−dictionary=*name*
Specifies the name to be used for the generated reference dictionary file; if unspecified, the reference dictionary is created in a temporary file, which is deleted when **pdfroff** completes processing of the current document. This option *must* be specified, if it is desired to save the reference dictionary, for use in references placed in other PDF documents.

−−report−progress
Causes **pdfroff** to display an informational message on standard error, at the start of each **groff** processing pass.

−−stylesheet=*name*
Specifies the name of an *input file*, to be used as a style sheet for formatting of content, which is to be placed *before* the table of contents, in the formatted PDF document.

−−version
Causes **pdfroff** to display a version identification message. The entire command line is then passed transparently to **groff**, in a *one* pass operation *only*, in order to display the associated **groff** version information, before exiting.

ENVIRONMENT

The following environment variables may be set, and exported, to modify the behaviour of **pdfroff**.

PDFROFF_COLLATE
Specifies the program to be used for collation of the finshed PDF document.

This collation step may be required to move *tables of contents* to the start of the finished PDF document, when formatting with traditional macro packages, which print them at the end. However, users should not normally need to specify **PDFROFF_COLLATE**, (and indeed, are not encouraged to do so). If unspecified, **pdfroff** uses **sed**(1) by default, which normally suffices.

If **PDFROFF_COLLATE** *is* specified, then it must act as a filter, accepting a list of file name arguments, and write its output to the *stdout* stream, whence it is piped to the **PDFROFF_POSTPROCESSOR_COMMAND**, to produce the finished PDF output.

When specifying **PDFROFF_COLLATE**, it is normally necessary to also specify **PDFROFF_KILL_NULL_PAGES**.

PDFROFF_COLLATE is ignored, if **pdfroff** is invoked with the *−−no−kill−null−pages* option.

PDFROFF_KILL_NULL_PAGES

Specifies options to be passed to the **PDFROFF_COLLATE** program.

It should not normally be necessary to specify **PDFROFF_KILL_NULL_PAGES**. The internal default is a **sed**(1) script, which is intended to remove completely blank pages from the collated output stream, and which should be appropriate in most applications of **pdfroff**. However, if any alternative to **sed**(1) is specified for **PDFROFF_COLLATE**, then it is likely that a corresponding alternative specification for **PDFROFF_KILL_NULL_PAGES** is required.

As in the case of **PDFROFF_COLLATE**, **PDFROFF_KILL_NULL_PAGES** is ignored, if **pdfroff** is invoked with the *−−no−kill−null−pages* option.

PDFROFF_POSTPROCESSOR_COMMAND

Specifies the command to be used for the final document conversion from PostScript intermediate output to PDF. It must behave as a filter, writing its output to the *stdout* stream, and must accept an arbitrary number of *files* . . . arguments, with the special case of − representing the *stdin* stream.

If unspecified, **PDFROFF_POSTPROCESSOR_COMMAND** defaults to

gs −dBATCH −dQUIET −dNOPAUSE −sDEVICE=pdfwrite −sOutputFile=−

GROFF_TMPDIR

Identifies the directory in which **pdfroff** should create temporary files. If **GROFF_TMPDIR** is *not* specified, then the variables **TMPDIR**, **TMP** and **TEMP** are considered in turn, as possible temporary file repositories. If none of these are set, then temporary files are created in the current directory.

GROFF_GHOSTSCRIPT_INTERPRETER

Specifies the program to be invoked, when **pdfroff** converts **groff** PostScript output to PDF. If **PDFROFF_POSTPROCESSOR_COMMAND** is specified, then the command name it specifies is *implicitly* assigned to **GROFF_GHOSTSCRIPT_INTERPRETER**, overriding any explicit setting specified in the environment. If **GROFF_GHOSTSCRIPT_INTERPRETER** is not specified, then **pdfroff** searches the process **PATH**, looking for a program with any of the well known names for the GhostScript interpreter; if no GhostScript interpreter can be found, **pdfroff** aborts.

GROFF_AWK_INTERPRETER

Specifies the program to be invoked, when **pdfroff** is extracting reference dictionary entries from a **groff** intermediate message stream. If **GROFF_AWK_INTERPRETER** is not specified, then **pdfroff** searches the process **PATH**, looking for any of the preferred programs, ‘gawk’, ‘mawk’, ‘nawk’ and ‘awk’, in this order; if none of these are found, **pdfroff** issues a warning message, and continue processing; however, in this case, no reference dictionary is created.

OSTYPE

Typically defined automatically by the operating system, **OSTYPE** is used on Microsoft Win32/MS-DOS platforms *only*, to infer the default **PATH_SEPARATOR** character, which is used when parsing the process **PATH** to search for external helper programs.

PATH_SEPARATOR
If set, **PATH_SEPARATOR** overrides the default separator character, (':' on POSIX/UNIX systems, inferred from **OSTYPE** on Microsoft Win32/MS-DOS), which is used when parsing the process **PATH** to search for external helper programs.

SHOW_PROGRESS
If this is set to a non-empty value, then **pdfroff** always behaves as if the **−−report−progress** option is specified, on the command line.

FILES

Input and output files for **pdfroff** may be named according to any convention of the user's choice. Typically, input files may be named according to the choice of the principal formatting macro package, e.g., *file***.ms** might be an input file for formatting using the **ms** macros (**s.tmac**); normally, the final output file should be named *file***.pdf**.

Temporary files, created by **pdfroff**, are placed in the directory specified by environment variables (see section **ENVIRONMENT**), and named according to the convention **pdf***$$***.***∗*, where *$$* is the standard shell variable representing the process ID of the **pdfroff** process itself, and ∗ represents any of the extensions used by **pdfroff** to identify the following temporary and intermediate files.

pdf*$$***.tmp**
A scratch pad file, used to capture reference data emitted by **groff**, during the *reference dictionary* compilation phase.

pdf*$$***.ref**
The *reference dictionary*, as compiled in the last but one pass of the *reference dictionary* compilation phase; (at the start of the first pass, this file is created empty; in successive passes, it contains the *reference dictionary* entries, as collected in the preceding pass).

If the **−−reference−dictionary**=*name* option is specified, this intermediate file becomes permanent, and is named *name*, rather than **pdf***$$***.ref**.

pdf*$$***.cmp**
Used to collect *reference dictionary* entries during the active pass of the *reference dictionary* compilation phase. At the end of any pass, when the content of **pdf***$$***.cmp** compares as identical to **pdf***$$***.ref**, (or the corresponding file named by the **−−reference−dictionary**=*name* option), then *reference dictionary* compilation is terminated, and the *document reference map* is appended to this intermediate file, for inclusion in the final formatting passes.

pdf*$$***.tc**
An intermediate *PostScript* file, in which "Table of Contents" entries are collected, to facilitate relocation before the body text, on ultimate output to the *GhostScript* postprocessor.

pdf*$$***.ps**
An intermediate *PostScript* file, in which the body text is collected prior to ultimate output to the *GhostScript* postprocessor, in the proper sequence, *after* **pdf***$$***.tc**.

SEE ALSO

See **groff**(1) for the definitive reference to document formatting with **groff**. Since **pdfroff** provides a superset of all **groff** capabilities, **groff**(1) may also be considered to be the definitive reference to all *standard* capabilities of **pdfroff**, with this document providing the reference to **pdfroff**'s extended features.

While **pdfroff** imposes neither any restriction on, nor any requirement for, the use of any specific **groff** macro package, a number of supplied macro packages, and in particular those associated with the package **pdfmark.tmac**, are best suited for use with **pdfroff** as the preferred formatter. Detailed documentation on the use of these packages may be found, in PDF format, in the reference guide **"Portable Document Format Publishing with GNU Troff"**, included in the installed documentation set as **c:/progra 1/groff/share/doc/groff/1.20/pdf/pdfmark.pdf**.

AUTHOR

PFBTOPS

NAME

pfbtops − translate a PostScript font in .pfb format to ASCII

SYNOPSIS

pfbtops [**−v**] [*pfb_file*]

DESCRIPTION

pfbtops translates a PostScript font in **.pfb** format to ASCII, splitting overlong lines in text packets into smaller chunks. If *pfb_file* is omitted the pfb file will be read from the standard input. The ASCII format PostScript font will be written on the standard output. PostScript fonts for MS-DOS are normally supplied in **.pfb** format.

The resulting ASCII format PostScript font can be used with groff. It must first be listed in **c:/progra 1/groff/share/groff/1.20/font/devps/download**.

OPTIONS

−v Print the version number.

SEE ALSO

grops(1)

PIC

NAME

pic – compile pictures for troff or TeX

SYNOPSIS

pic [**–nvCSU**] [*filename* . . .]
pic –t [**–cvzCSU**] [*filename* . . .]

DESCRIPTION

This manual page describes the GNU version of **pic**, which is part of the groff document formatting system. **pic** compiles descriptions of pictures embedded within **troff** or TeX input files into commands that are understood by TeX or **troff**. Each picture starts with a line beginning with **.PS** and ends with a line beginning with **.PE**. Anything outside of **.PS** and **.PE** is passed through without change.

It is the user's responsibility to provide appropriate definitions of the **PS** and **PE** macros. When the macro package being used does not supply such definitions (for example, old versions of –ms), appropriate definitions can be obtained with **–mpic**: These will center each picture.

OPTIONS

Options that do not take arguments may be grouped behind a single **–**. The special option **––** can be used to mark the end of the options. A filename of **–** refers to the standard input.

–C Recognize **.PS** and **.PE** even when followed by a character other than space or newline.

–S Safer mode; do not execute **sh** commands. This can be useful when operating on untrustworthy input. (enabled by default)

–U Unsafe mode; revert the default option **–S**.

–n Don't use the groff extensions to the troff drawing commands. You should use this if you are using a postprocessor that doesn't support these extensions. The extensions are described in **groff_out**(5). The **–n** option also causes **pic** not to use zero-length lines to draw dots in troff mode.

–t TeX mode.

–c Be more compatible with **tpic**. Implies **–t**. Lines beginning with \ are not passed through transparently. Lines beginning with **.** are passed through with the initial **.** changed to \. A line beginning with **.ps** is given special treatment: it takes an optional integer argument specifying the line thickness (pen size) in milliinches; a missing argument restores the previous line thickness; the default line thickness is 8 milliinches. The line thickness thus specified takes effect only when a non-negative line thickness has not been specified by use of the **thickness** attribute or by setting the **linethick** variable.

–v Print the version number.

–z In TeX mode draw dots using zero-length lines.

The following options supported by other versions of **pic** are ignored:

–D Draw all lines using the \D escape sequence. **pic** always does this.

–T *dev* Generate output for the **troff** device *dev*. This is unnecessary because the **troff** output generated by **pic** is device-independent.

USAGE

This section describes only the differences between GNU **pic** and the original version of **pic**. Many of these differences also apply to newer versions of Unix **pic**. A complete documentation is available in the file

c:/progra 1/groff/share/doc/groff/1.20/pic.ms

TeX mode

TeX mode is enabled by the **–t** option. In TeX mode, **pic** will define a vbox called **\graph** for each picture. Use the **figname** command to change the name of the vbox. You must yourself print that vbox using, for example, the command

\centerline{\box\graph}

Actually, since the vbox has a height of zero (it is defined with \vtop) this will produce slightly more vertical space above the picture than below it;

\centerline{\raise 1em\box\graph}

would avoid this.

To make the vbox having a positive height and a depth of zero (as used e.g. by LaTeX's **graphics.sty**), define the following macro in your document:

```
\def\gpicbox#1{%
  \vbox{\unvbox\csname #1\endcsname\kern 0pt}}
```

Now you can simply say **\gpicbox{graph}** instead of \box\graph.

You must use a TeX driver that supports the **tpic** specials, version 2.

Lines beginning with **** are passed through transparently; a **%** is added to the end of the line to avoid unwanted spaces. You can safely use this feature to change fonts or to change the value of **\baselineskip**. Anything else may well produce undesirable results; use at your own risk. Lines beginning with a period are not given any special treatment.

Commands

for *variable* **=** *expr1* **to** *expr2* [**by** [*]*expr3*] **do** *X body X*
: Set *variable* to *expr1*. While the value of *variable* is less than or equal to *expr2*, do *body* and increment *variable* by *expr3*; if **by** is not given, increment *variable* by 1. If *expr3* is prefixed by * then *variable* will instead be multiplied by *expr3*. The value of *expr3* can be negative for the additive case; *variable* is then tested whether it is greater than or equal to *expr2*. For the multiplicative case, *expr3* must be greater than zero. If the constraints aren't met, the loop isn't executed. *X* can be any character not occurring in *body*.

if *expr* **then** *X if-true X* [**else** *Y if-false Y*]
: Evaluate *expr*; if it is non-zero then do *if-true*, otherwise do *if-false*. *X* can be any character not occurring in *if-true*. *Y* can be any character not occurring in *if-false*.

print *arg*...
: Concatenate the arguments and print as a line on stderr. Each *arg* must be an expression, a position, or text. This is useful for debugging.

command *arg*...
: Concatenate the arguments and pass them through as a line to troff or TeX. Each *arg* must be an expression, a position, or text. This has a similar effect to a line beginning with **.** or ****, but allows the values of variables to be passed through. For example,

```
.PS
x = 14
command ".ds string x is " x "."
.PE
\*[string]
```

prints

```
x is 14.
```

sh *X command X*
: Pass *command* to a shell. *X* can be any character not occurring in *command*.

copy "*filename***"**
: Include *filename* at this point in the file.

copy [**"***filename***"**] **thru** *X body X* [**until "***word***"**]
copy [**"***filename***"**] **thru** *macro* [**until "***word***"**]
: This construct does *body* once for each line of *filename*; the line is split into blank-delimited words, and occurrences of **$***i* in *body*, for *i* between 1 and 9, are replaced by the *i*-th word of the line. If *filename* is not given, lines are taken from the current input up to **.PE**. If an **until** clause is specified, lines will be read only until a line the first word of which is *word*; that line will then be discarded. *X* can be any character not occurring in *body*. For example,

```
.PS
copy thru % circle at ($1,$2) % until "END"
```

```
1 2
3 4
5 6
END
box
.PE
```

is equivalent to

```
.PS
circle at (1,2)
circle at (3,4)
circle at (5,6)
box
.PE
```

The commands to be performed for each line can also be taken from a macro defined earlier by giving the name of the macro as the argument to **thru**.

reset
reset *variable1*[**,**] *variable2* …
Reset pre-defined variables *variable1*, *variable2* … to their default values. If no arguments are given, reset all pre-defined variables to their default values. Note that assigning a value to **scale** also causes all pre-defined variables that control dimensions to be reset to their default values times the new value of scale.

plot *expr* [**"***text***"**]
This is a text object which is constructed by using *text* as a format string for sprintf with an argument of *expr*. If *text* is omitted a format string of **"%g"** is used. Attributes can be specified in the same way as for a normal text object. Be very careful that you specify an appropriate format string; **pic** does only very limited checking of the string. This is deprecated in favour of **sprintf**.

variable **:=** *expr*
This is similar to **=** except *variable* must already be defined, and *expr* will be assigned to *variable* without creating a variable local to the current block. (By contrast, **=** defines the variable in the current block if it is not already defined there, and then changes the value in the current block only.) For example, the following:

```
.PS
x = 3
y = 3
[
  x := 5
  y = 5
]
print x " " y
.PE
```

prints

```
5 3
```

Arguments of the form

X anything X

are also allowed to be of the form

{ *anything* **}**

In this case *anything* can contain balanced occurrences of **{** and **}**. Strings may contain *X* or imbalanced occurrences of **{** and **}**.

Expressions

The syntax for expressions has been significantly extended:

x ^ *y* (exponentiation)
sin(*x***)**
cos(*x***)**
atan2(*y***,** *x***)**
log(*x***)** (base 10)
exp(*x***)** (base 10, ie 10^x)
sqrt(*x***)**
int(*x***)**
rand() (return a random number between 0 and 1)
rand(*x***)** (return a random number between 1 and *x*; deprecated)
srand(*x***)** (set the random number seed)
max(*e1***,** *e2***)**
min(*e1***,** *e2***)**
!*e*
e1 **&&** *e2*
e1 **||** *e2*
e1 **==** *e2*
e1 **!=** *e2*
e1 **>=** *e2*
e1 **>** *e2*
e1 **<=** *e2*
e1 **<** *e2*
"*str1***" == "***str2***"**
"*str1***" != "***str2***"**

String comparison expressions must be parenthesised in some contexts to avoid ambiguity.

Other Changes

A bare expression, *expr*, is acceptable as an attribute; it is equivalent to *dir expr*, where *dir* is the current direction. For example

line 2i

means draw a line 2 inches long in the current direction. The 'i' (or 'I') character is ignored; to use another measurement unit, set the *scale* variable to an appropriate value.

The maximum width and height of the picture are taken from the variables **maxpswid** and **maxpsht**. Initially these have values 8.5 and 11.

Scientific notation is allowed for numbers. For example

x = 5e−2

Text attributes can be compounded. For example,

"foo" above ljust

is valid.

There is no limit to the depth to which blocks can be examined. For example,

[A: [B: [C: box]]] with .A.B.C.sw at 1,2
circle at last [ ].A.B.C

is acceptable.

Arcs now have compass points determined by the circle of which the arc is a part.

Circles, ellipses, and arcs can be dotted or dashed. In TEX mode splines can be dotted or dashed also.

Boxes can have rounded corners. The **rad** attribute specifies the radius of the quarter-circles at each corner. If no **rad** or **diam** attribute is given, a radius of **boxrad** is used. Initially, **boxrad** has a value of 0. A box with rounded corners can be dotted or dashed.

Boxes can have slanted sides. This effectively changes the shape of a box from a rectangle to an arbitrary parallelogram. The **xslanted** and **yslanted** attributes specify the x and y offset of the box's upper right corner from its default position.

The **.PS** line can have a second argument specifying a maximum height for the picture. If the width of zero is specified the width will be ignored in computing the scaling factor for the picture. Note that GNU **pic** will always scale a picture by the same amount vertically as well as horizontally. This is different from the DWB 2.0 **pic** which may scale a picture by a different amount vertically than horizontally if a height is specified.

Each text object has an invisible box associated with it. The compass points of a text object are determined by this box. The implicit motion associated with the object is also determined by this box. The dimensions of this box are taken from the width and height attributes; if the width attribute is not supplied then the width will be taken to be **textwid**; if the height attribute is not supplied then the height will be taken to be the number of text strings associated with the object times **textht**. Initially **textwid** and **textht** have a value of 0.

In (almost all) places where a quoted text string can be used, an expression of the form

> **sprintf("***format***",** *arg***,**. . .**)**

can also be used; this will produce the arguments formatted according to *format*, which should be a string as described in **printf**(3) appropriate for the number of arguments supplied.

The thickness of the lines used to draw objects is controlled by the **linethick** variable. This gives the thickness of lines in points. A negative value means use the default thickness: in TEX output mode, this means use a thickness of 8 milliinches; in TEX output mode with the **-c** option, this means use the line thickness specified by **.ps** lines; in troff output mode, this means use a thickness proportional to the pointsize. A zero value means draw the thinnest possible line supported by the output device. Initially it has a value of -1. There is also a **thick**[**ness**] attribute. For example,

> **circle thickness 1.5**

would draw a circle using a line with a thickness of 1.5 points. The thickness of lines is not affected by the value of the **scale** variable, nor by the width or height given in the **.PS** line.

Boxes (including boxes with rounded corners or slanted sides), circles and ellipses can be filled by giving them an attribute of **fill**[**ed**]. This takes an optional argument of an expression with a value between 0 and 1; 0 will fill it with white, 1 with black, values in between with a proportionally gray shade. A value greater than 1 can also be used: this means fill with the shade of gray that is currently being used for text and lines. Normally this will be black, but output devices may provide a mechanism for changing this. Without an argument, then the value of the variable **fillval** will be used. Initially this has a value of 0.5. The invisible attribute does not affect the filling of objects. Any text associated with a filled object will be added after the object has been filled, so that the text will not be obscured by the filling.

Three additional modifiers are available to specify colored objects: **outline**[**d**] sets the color of the outline, **shaded** the fill color, and **colo**[**u**]**r**[**ed**] sets both. All three keywords expect a suffix specifying the color, for example

> **circle shaded "green" outline "black"**

Currently, color support isn't available in TEX mode. Predefined color names for **groff** are in the device macro files, for example **ps.tmac**; additional colors can be defined with the **.defcolor** request (see the manual page of **troff**(1) for more details).

To change the name of the vbox in TEX mode, set the pseudo-variable **figname** (which is actually a specially parsed command) within a picture. Example:

```
.PS
figname = foobar;
...
.PE
```

The picture is then available in the box **\foobar**.

pic assumes that at the beginning of a picture both glyph and fill color are set to the default value.

Arrow heads will be drawn as solid triangles if the variable **arrowhead** is non-zero and either TEX mode is enabled or the **−n** option has not been given. Initially **arrowhead** has a value of 1. Note that solid arrow heads are always filled with the current outline color.

The troff output of **pic** is device-independent. The **−T** option is therefore redundant. All numbers are taken to be in inches; numbers are never interpreted to be in troff machine units.

Objects can have an **aligned** attribute. This will only work if the postprocessor is **grops**. Any text associated with an object having the **aligned** attribute will be rotated about the center of the object so that it is aligned in the direction from the start point to the end point of the object. Note that this attribute will have no effect for objects whose start and end points are coincident.

In places where *n***th** is allowed **'***expr***'th** is also allowed. Note that **'th** is a single token: no space is allowed between the **'** and the **th**. For example,

```
for i = 1 to 4 do {
  line from 'i'th box.nw to 'i+1'th box.se
}
```

CONVERSION

To obtain a stand-alone picture from a **pic** file, enclose your **pic** code with **.PS** and **.PE** requests; **roff** configuration commands may be added at the beginning of the file, but no **roff** text.

It is necessary to feed this file into **groff** without adding any page information, so you must check which **.PS** and **.PE** requests are actually called. For example, the mm macro package adds a page number, which is very annoying. At the moment, calling standard **groff** without any macro package works. Alternatively, you can define your own requests, e.g. to do nothing:

```
.de PS
..
.de PE
..
```

groff itself does not provide direct conversion into other graphics file formats. But there are lots of possibilities if you first transform your picture into PostScript® format using the **groff** option **-Tps**. Since this *ps*-file lacks BoundingBox information it is not very useful by itself, but it may be fed into other conversion programs, usually named **ps2***other* or **psto***other* or the like. Moreover, the PostScript interpreter **ghostscript** (**gs**) has built-in graphics conversion devices that are called with the option

```
gs -sDEVICE=<devname>
```

Call

```
gs --help
```

for a list of the available devices.

As the Encapsulated PostScript File Format **EPS** is getting more and more important, and the conversion wasn't regarded trivial in the past you might be interested to know that there is a conversion tool named **ps2eps** which does the right job. It is much better than the tool **ps2epsi** packaged with **gs**.

For bitmapped graphic formats, you should use **pstopnm**; the resulting (intermediate) **PNM** file can be then converted to virtually any graphics format using the tools of the **netpbm** package .

FILES

c:/progra 1/groff/share/groff/1.20/tmac/pic.tmac
Example definitions of the **PS** and **PE** macros.

SEE ALSO

troff(1), **groff_out**(5), **tex**(1), **gs**(1), **ps2eps**(1), **pstopnm**(1), **ps2epsi**(1), **pnm**(5)

Tpic: Pic for TEX

Brian W. Kernighan, PIC — A Graphics Language for Typesetting (User Manual). AT&T Bell Laboratories, Computing Science Technical Report No. 116 <http://cm.bell-labs.com/cm/cs/cstr/116.ps.gz> (revised May, 1991).

ps2eps is available from CTAN mirrors, e.g.
<ftp://ftp.dante.de/tex-archive/support/ps2eps/>

W. Richard Stevens - Turning PIC Into HTML
<http://www.kohala.com/start/troff/pic2html.html>

W. Richard Stevens - Examples of picMacros
<http://www.kohala.com/start/troff/pic.examples.ps>

BUGS

Input characters that are invalid for **groff** (i.e., those with ASCII code 0, or 013 octal, or between 015 and 037 octal, or between 0200 and 0237 octal) are rejected even in TEX mode.

The interpretation of **fillval** is incompatible with the pic in 10th edition Unix, which interprets 0 as black and 1 as white.

PostScript® is a registered trademark of Adobe Systems Incorporation.

PIC2GRAPH

NAME

pic2graph – convert a PIC diagram into a cropped image

SYNOPSIS

pic2graph [**–unsafe**] [**–format** *fmt*] [**–eqn** *delim*]

DESCRIPTION

Reads a PIC program as input; produces an image file (by default in Portable Network Graphics format) suitable for the Web as output. Also translates **eqn**(1) constructs, so it can be used for generating images of mathematical formulae.

PIC is a rather expressive graphics minilanguage suitable for producing box-and-arrow diagrams of the kind frequently used in technical papers and textbooks. The language is sufficiently flexible to be quite useful for state charts, Petri-net diagrams, flow charts, simple circuit schematics, jumper layouts, and other kinds of illustration involving repetitive uses of simple geometric forms and splines. Because PIC descriptions are procedural and object-based, they are both compact and easy to modify.

The PIC language is fully documented in *Making Pictures With GNU PIC*, a document which is part of the **groff**(1) distribution.

Your input PIC code should *not* be wrapped with the .PS and .PE macros that normally guard it within **groff**(1) macros.

The output image will be clipped to the smallest possible bounding box that contains all the black pixels. Older versions of **convert**(1) will produce a black-on-white graphic; newer ones may produce a black-on-transparent graphic. By specifying command-line options to be passed to **convert**(1) you can give it a border, force the background transparent, set the image's pixel density, or perform other useful transformations.

This program uses **pic**(1), **eqn**(1), **groff**(1), **gs**(1), and the ImageMagick **convert**(1) program. These programs must be installed on your system and accessible on your $PATH for **pic2graph** to work.

OPTIONS

–unsafe
: Run **pic**(1) and **groff**(1) in the 'unsafe' mode enabling the PIC macro **sh** to execute arbitrary commands. The default is to forbid this.

–format *fmt*
: Specify an output format; the default is PNG (Portable Network Graphics). Any format that **convert**(1) can emit is supported.

–eqn *delim*
: Change the fencepost characters that delimit **eqn**(1) directives (**$** and **$**, by default). This option requires an argument, but an empty string is accepted as a directive to disable **eqn**(1) processing.

Command-line switches and arguments not listed above are passed to **convert**(1).

FILES

c:/progra 1/groff/share/groff/1.20/tmac/eqnrc The **eqn**(1) initialization file.

ENVIRONMENT

GROFF_TMPDIR
: The directory in which temporary files will be created. If this is not set **pic2graph** searches the environment variables **TMPDIR**, **TMP**, and **TEMP** (in that order). Otherwise, temporary files will be created in **/tmp**.

BUGS

Due to changes in the behavior of ImageMagick **convert**(1) that are both forward and backward-incompatible, mismatches between your **pic2graph** and **convert**(1) versions may produce zero-sized or untrimmed output images. For this version of **pic2graph** you will need a version of **convert**(1) that supports the **–trim** option; older versions of **pic2graph** used **–crop 0x0**, which no longer has trimming behavior.

SEE ALSO

eqn2graph(1), **grap2graph**(1), **pic**(1), **eqn**(1), **groff**(1), **gs**(1), **convert**(1).

AUTHOR

Eric S. Raymond <esr@thyrsus.com>, based on a recipe by W. Richard Stevens.

PRAG

NAME

prag – compile diagrams for **pic**

SYNOPSIS

`prag` [*filename* ...]

DESCRIPTION

`.G1` [*width* [*height*]]
`.G2`

prag recognizes its input only between **.G1** and **.G2**. All other lines are copied literal to standard output. Two optional arguments to **.G1** specify the width and the height.

`#` *comment*

Lines starting with a hash mark '#' are treated as comments.

`draw` *line-style*

The `draw` statement sets the line style for drawing the graphs. Valid arguments are the line styles as known from **pic**(1) or the pseudo-style `marked`. `marked` will yield a solid graph with the data points marked. Up to five default marking characters (\(bu •, \(*D Δ, \(pl +, \(sq □, \(mu ×) are used for the first five graphs. If you don't like these characters, you can override them by giving an additional drawing character to each data point (as second or third argument). Use line style `invis` to draw unconnected data points.

`new` [*line-style*] *name* [*label-string*]]]

This statement switches a new graph within a diagram. The line style of the graph is set to *line-style*. The graph gets the name *name*, which can be referred in subsequent **pic** statements and is labeled with the string *label-string*. *label-string* can be any valid **pic** string.

`label` [`left` | `right` | `top` | `bot`] *label-string*

label-string is placed at the specified side of the whole diagram.

`ticks` [`left` | `right` | `top` | `bot`] [*where*] *position* ...

Tick marks for the specified side are placed at the given positions. *where* says to place the ticks inside (`in`) or outside (`out`) the diagram.

`ticks` [`left` | `right` | `top` | `bot`] [*where*] `from` *begin* `to` *end* [`by` *step*]

This second variant of the `tick`-statement allows the automatic generation of ticks in the range from *begin* to *end* with an optional distance *step*.

`frame` *frame-attributes* ...

The *frame-attributes* are given to the box, that builds the frame of the diagram. Valid **pic** box attributes can be used. The internal name of the diagram frame is `FRAME` and of the surrounding box `GRAPH`. Both can be used in **pic** statements to get special effects. The boxes for the top, left, right and bottom margin are `TMARG`, `LMARG`, `RMARG` and `BMARG`.

`spline`
`nospline`

`spline` forces prag to use splines for drawing graphs, which is the default. `nospline` will use **pic**'s `line` directives respectively.

`range` *xmin ymin xmax ymax*

Graphs are drawn within the range specified.

`ht` *number*

Sets the height of the diagram to *number*.

`wid` *number*

Sets the width of the diagram to *number*.

`grid` [*line-style*]

Draws a grid of *line-style*, `dotted` per default.

`pic` { *anything* }

Anything is passed literally through **pic**.

`next` [*graph-name*] [`at`] *position*

Sets a new data point for graph *graph-name* or the current graph at position *position*.

position

Position is described by an y coordinate and an optional x coordinate. If the x coordinate is missing, then x is assumed to be 0, 1, 2, 3, etc.

position drawing-char

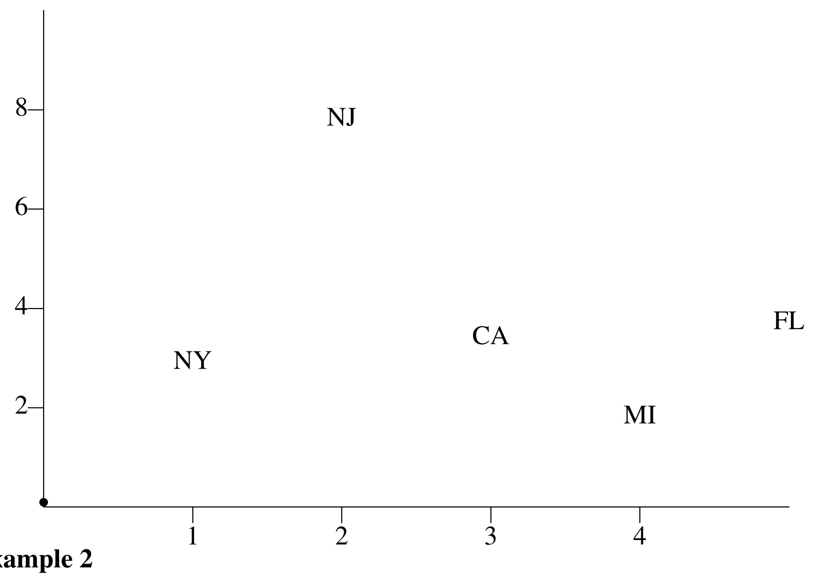
For each data point an optional (as second or third argument respectively) drawing character can be declared. Any **troff** and **eqn** character or special character or sequence of characters can be used.

EXAMPLES

Example 1

```
frame invis
pic { line from FRAME.sw to FRAME.se }
pic { line from FRAME.sw to FRAME.nw }
draw invis
0 \&
2.9 NY
7.8 NJ
3.4 CA
1.8 MI
3.7 FL
```

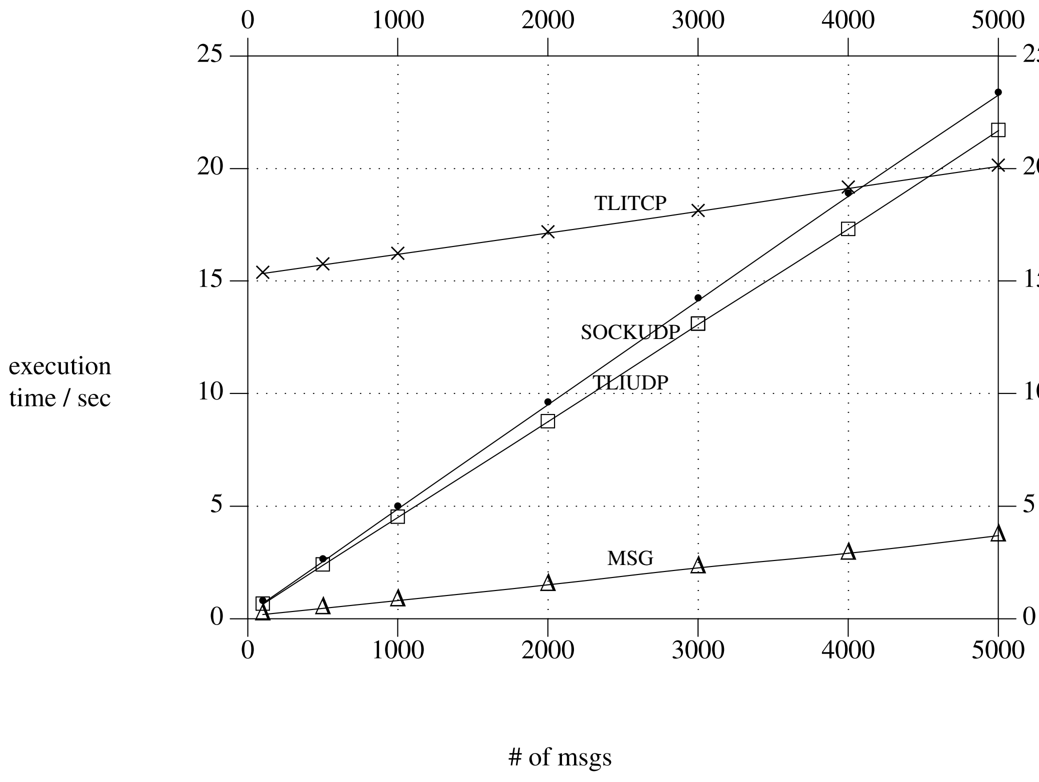
will produce:



Example 2

```
wid 4
ht 3
grid dotted
label left "execution" "time / sec"
label bot "# of msgs"
range 0 0 5000 25
ticks left 0 5 10 15 20 25
ticks right 0 5 10 15 20 25
ticks top 0 1000 2000 3000 4000 5000
ticks bot 0 1000 2000 3000 4000 5000
new marked MSG "\s-2MSG\s+2" above
100 0.19 \(*D
. . .
next TLITCP at 5000 20.09 \(mu
3000 18.08 \(mu
4000 19.12 \(mu
```

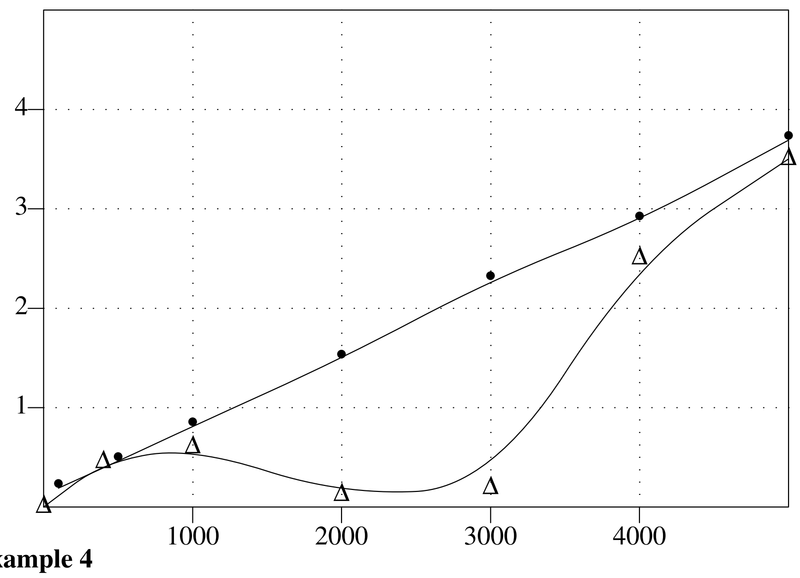
will produce:



Example 3

```
grid dotted
draw marked
100 0.19
. . .
5000 3.69
new marked
0 0
400 0.45
. . .
5000 3.5
```

will produce:

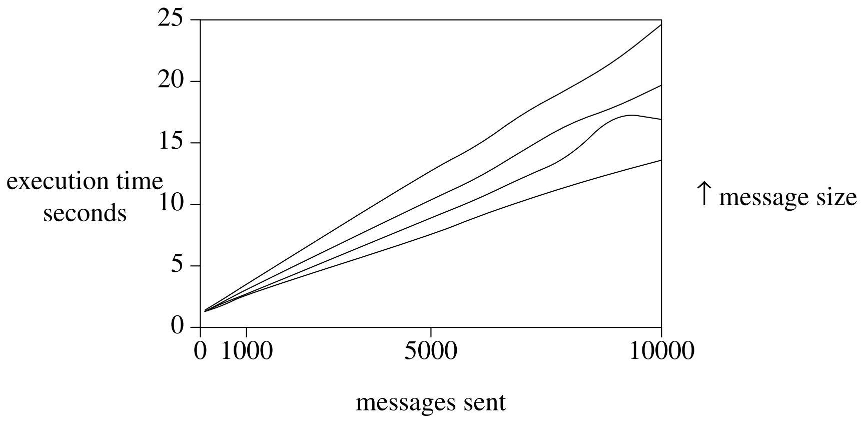


Example 4

```
label left "execution time" "seconds"
label right "\(ua message size"
label bot "messages sent"
```

```
range 0 0 10000 25
ticks left 0 5 10 15 20 25
ticks bot 0 1000 5000 10000
100    1.3
. . .
10000  24.6
```

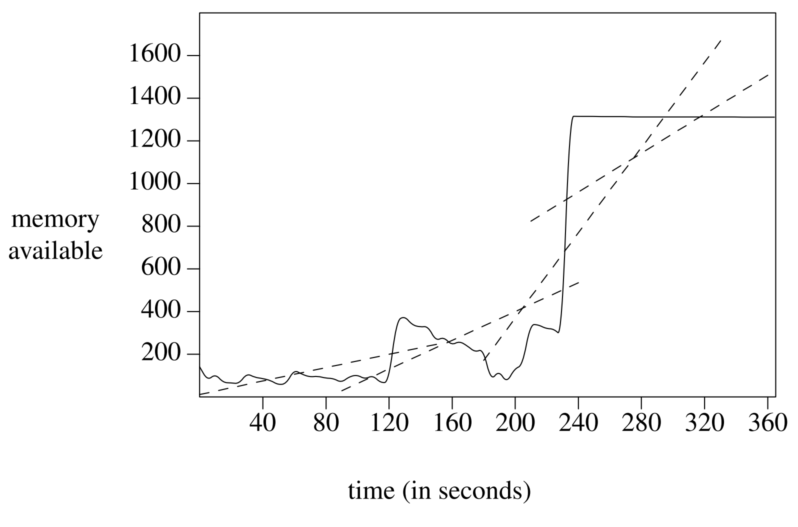
will produce:



Example 5

```
label bot "time (in seconds)"
label left "memory" "available"
range 0 0 365 1800
ticks left 200 400 600 800 1000 1200 1400 1600
ticks bot 40 80 120 160 200 240 280 320 360
new
0 141
. . .
365 1309
new dashed
nospline
0 12
150 247
. . .
new dashed
nospline
210 824
360 1508
```

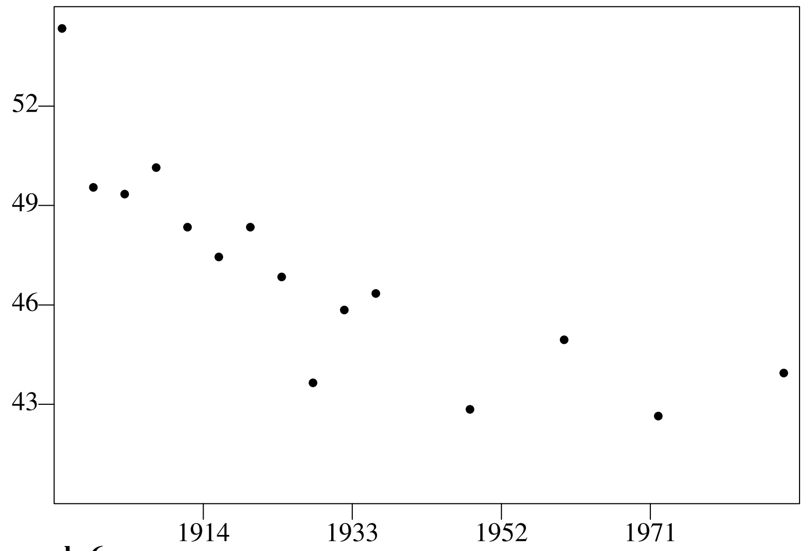
will produce:



Example 6

```
draw invis
1896 54.2
1900 49.4
. . .
1988 43.8
```

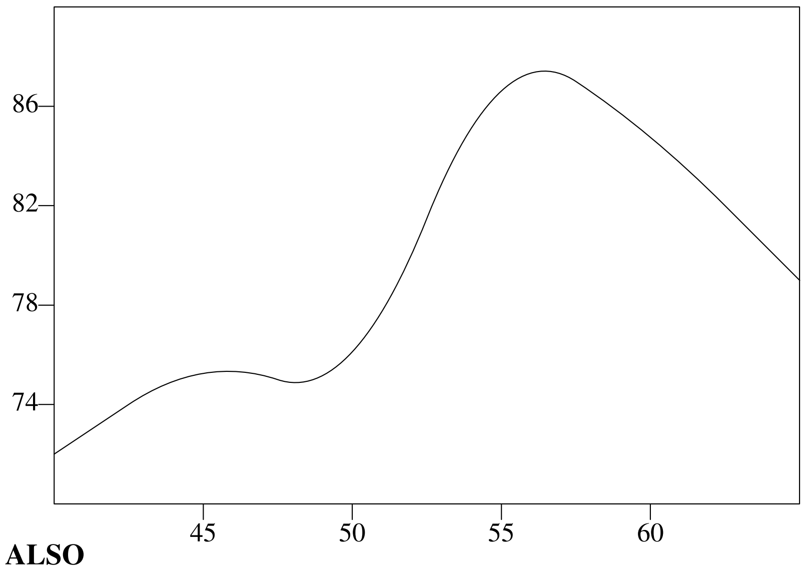
will produce:



Example 6

```
40 72
45 76
. . .
65 79
```

will produce:



SEE ALSO

AT&T Bell Laboratories, Computing Science Technical Report No. 116, GRAP − A Graphics Language for Typesetting. This can be obtained by sending a mail message to netlib@research.att.com with a body of ‘send 116 from research/cstr’.

BUGS

Bug reports or suggested improvements should go to hm@GUUG.de.

AUTHOR

This program was contributed by Holger Meyer at University of Rostock, Germany. It is available via anonymous ftp from ftp.informatik.uni-rostock.de in the directory /pub/local/software.

PRECONV

NAME

preconv – convert encoding of input files to something GNU troff understands

SYNOPSIS

[**–dr**] [**–e***encoding*] [*files . . .*] **–h | ––help –v | ––version**

It is possible to have whitespace between the **–e** command line option and its parameter.

DESCRIPTION

preconv reads *files* and converts its encoding(s) to a form GNU **troff**(1) can process, sending the data to standard output. Currently, this means ASCII characters and '\[uXXXX]' entities, where 'XXXX' is a hexadecimal number with four to six digits, representing a Unicode input code. Normally, **preconv** should be invoked with the **–k** and **–K** options of **groff**.

OPTIONS

–d Emit debugging messages to standard error (mainly the used encoding).

–D*encoding*
Specify default encoding if everything fails (see below).

–e*encoding*
Specify input encoding explicitly, overriding all other methods. This corresponds to **groff**'s **–K***encoding* option. Without this switch, **preconv** uses the algorithm described below to select the input encoding.

––help
–h Print help message.

–r Do not add .lf requests.

––version
–v Print version number.

USAGE

preconv tries to find the input encoding with the following algorithm.

1. If the input encoding has been explicitly specified with option **–e**, use it.
2. Otherwise, check whether the input starts with a *Byte Order Mark* (BOM, see below). If found, use it.
3. Finally, check whether there is a known *coding tag* (see below) in either the first or second input line. If found, use it.
4. If everything fails, use a default encoding as given with option **–D**, by the current locale, or 'latin1' if the locale is set to 'C', 'POSIX', or empty (in that order).

Note that the **groff** program supports a **GROFF_ENCODING** environment variable which is eventually expanded to option **–k**.

Byte Order Mark

The Unicode Standard defines character U+FEFF as the Byte Order Mark (BOM). On the other hand, value U+FFFE is guaranteed not be a Unicode character at all. This allows to detect the byte order within the data stream (either big-endian or lower-endian), and the MIME encodings 'UTF-16' and 'UTF-32' mandate that the data stream starts with U+FEFF. Similarly, the data stream encoded as 'UTF-8' might start with a BOM (to ease the conversion from and to UTF-16 and UTF-32). In all cases, the byte order mark is *not* part of the data but part of the encoding protocol; in other words, **preconv**'s output doesn't contain it.

Note that U+FEFF not at the start of the input data actually is emitted; it has then the meaning of a 'zero width no-break space' character – something not needed normally in **groff**.

Coding Tags

Editors which support more than a single character encoding need tags within the input files to mark the file's encoding. While it is possible to guess the right input encoding with the help of heuristic algorithms for data which represents a greater amount of a natural language, it is still just a guess. Additionally, all algorithms fail easily for input which is either too short or doesn't represent a natural language.

For these reasons, **preconv** supports the coding tag convention (with some restrictions) as used by **GNU Emacs** and **XEmacs** (and probably other programs too).

Coding tags in **GNU Emacs** and **XEmacs** are stored in so-called *File Variables*. **preconv** recognizes the following syntax form which must be put into a troff comment in the first or second line.

> −*− *tag1*: *value1*; *tag2*: *value2*; . . . −*−

The only relevant tag for **preconv** is 'coding' which can take the values listed below. Here an example line which tells **Emacs** to edit a file in troff mode, and to use latin2 as its encoding.

> .\" −*− mode: troff; coding: latin-2 −*−

The following list gives all MIME coding tags (either lowercase or uppercase) supported by **preconv**; this list is hard-coded in the source.

> big5, cp1047, euc-jp, euc-kr, gb2312, iso-8859-1, iso-8859-2, iso-8859-5, iso-8859-7, iso-8859-9, iso-8859-13, iso-8859-15, koi8-r, us-ascii, utf-8, utf-16, utf-16be, utf-16le

In addition, the following hard-coded list of other tags is recognized which eventually map to values from the list above.

> ascii, chinese-big5, chinese-euc, chinese-iso-8bit, cn-big5, cn-gb, cn-gb-2312, cp878, csascii, csisolatin1, cyrillic-iso-8bit, cyrillic-koi8, euc-china, euc-cn, euc-japan, euc-japan-1990, euc-korea, greek-iso-8bit, iso-10646/utf8, iso-10646/utf-8, iso-latin-1, iso-latin-2, iso-latin-5, iso-latin-7, iso-latin-9, japanese-euc, japanese-iso-8bit, jis8, koi8, korean-euc, korean-iso-8bit, latin-0, latin1, latin-1, latin-2, latin-5, latin-7, latin-9, mule-utf-8, mule-utf-16, mule-utf-16be, mule-utf-16-be, mule-utf-16be-with-signature, mule-utf-16le, mule-utf-16-le, mule-utf-16le-with-signature, utf8, utf-16-be, utf-16-be-with-signature, utf-16be-with-signature, utf-16-le, utf-16-le-with-signature, utf-16le-with-signature

Those tags are taken from **GNU Emacs** and **XEmacs**, together with some aliases. Trailing '-dos', '-unix', and '-mac' suffixes of coding tags (which give the end-of-line convention used in the file) are stripped off before the comparison with the above tags happens.

Iconv Issues

preconv by itself only supports three encodings: latin-1, cp1047, and UTF-8; all other encodings are passed to the **iconv** library functions. At compile time it is searched and checked for a valid **iconv** implementation; a call to 'preconv −−version' shows whether **iconv** is used.

BUGS

preconv doesn't support *local variable lists* yet. This is a different syntax form to specify local variables at the end of a file.

SEE ALSO

groff(1)
the **GNU Emacs** and **XEmacs** info pages

REFER

NAME

refer – preprocess bibliographic references for groff

SYNOPSIS

refer [**–benvCPRS**] [**–a***n*] [**–c***fields*] [**–f***n*] [**–i***fields*] [**–k***field*] [**–l***m,n*] [**–p***filename*]
[**–s***fields*] [**–t***n*] [**–B***field.macro*] [*filename . . .*]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

This file documents the GNU version of **refer**, which is part of the groff document formatting system. **refer** copies the contents of *filename . . .* to the standard output, except that lines between **.[** and **.]** are interpreted as citations, and lines between **.R1** and **.R2** are interpreted as commands about how citations are to be processed.

Each citation specifies a reference. The citation can specify a reference that is contained in a bibliographic database by giving a set of keywords that only that reference contains. Alternatively it can specify a reference by supplying a database record in the citation. A combination of these alternatives is also possible.

For each citation, **refer** can produce a mark in the text. This mark consists of some label which can be separated from the text and from other labels in various ways. For each reference it also outputs **groff** commands that can be used by a macro package to produce a formatted reference for each citation. The output of **refer** must therefore be processed using a suitable macro package. The **–ms** and **–me** macros are both suitable. The commands to format a citation's reference can be output immediately after the citation, or the references may be accumulated, and the commands output at some later point. If the references are accumulated, then multiple citations of the same reference will produce a single formatted reference.

The interpretation of lines between **.R1** and **.R2** as commands is a new feature of GNU **refer**. Documents making use of this feature can still be processed by Unix refer just by adding the lines

```
.de R1
.ig R2
..
```

to the beginning of the document. This will cause **troff** to ignore everything between **.R1** and **.R2**. The effect of some commands can also be achieved by options. These options are supported mainly for compatibility with Unix refer. It is usually more convenient to use commands.

refer generates **.lf** lines so that filenames and line numbers in messages produced by commands that read **refer** output will be correct; it also interprets lines beginning with **.lf** so that filenames and line numbers in the messages and **.lf** lines that it produces will be accurate even if the input has been preprocessed by a command such as **soelim**(1).

OPTIONS

Most options are equivalent to commands (for a description of these commands see the **Commands** subsection):

–b	**no-label-in-text; no-label-in-reference**
–e	**accumulate**
–n	**no-default-database**
–C	**compatible**
–P	**move-punctuation**
–S	**label "(A.n|Q) ', ' (D.y|D)"; bracket-label " (") "; "**
–a*n*	**reverse A***n*
–c*fields*	**capitalize** *fields*
–f*n*	**label** %*n*

−i*fields*	**search-ignore** *fields*
−k	**label L~%a**
−k*field*	**label** *field***~%a**
−l	**label A.nD.y%a**
−l*m*	**label A.n+***m***D.y%a**
−l,*n*	**label A.nD.y−***n***%a**
−l*m***,***n*	**label A.n+***m***D.y−***n***%a**
−p*filename*	**database** *filename*
−s*spec*	**sort** *spec*
−t*n*	**search-truncate** *n*

These options are equivalent to the following commands with the addition that the filenames specified on the command line are processed as if they were arguments to the **bibliography** command instead of in the normal way:

−B	**annotate X AP; no-label-in-reference**
−B*field***.***macro*	**annotate** *field macro***; no-label-in-reference**

The following options have no equivalent commands:

−v Print the version number.

−R Don't recognize lines beginning with **.R1**/**.R2**.

USAGE

Bibliographic databases

The bibliographic database is a text file consisting of records separated by one or more blank lines. Within each record fields start with a **%** at the beginning of a line. Each field has a one character name that immediately follows the **%**. It is best to use only upper and lower case letters for the names of fields. The name of the field should be followed by exactly one space, and then by the contents of the field. Empty fields are ignored. The conventional meaning of each field is as follows:

A The name of an author. If the name contains a title such as **Jr.** at the end, it should be separated from the last name by a comma. There can be multiple occurrences of the **A** field. The order is significant. It is a good idea always to supply an **A** field or a **Q** field.

B For an article that is part of a book, the title of the book.

C The place (city) of publication.

D The date of publication. The year should be specified in full. If the month is specified, the name rather than the number of the month should be used, but only the first three letters are required. It is a good idea always to supply a **D** field; if the date is unknown, a value such as **in press** or **unknown** can be used.

E For an article that is part of a book, the name of an editor of the book. Where the work has editors and no authors, the names of the editors should be given as **A** fields and **, (ed)** or **, (eds)** should be appended to the last author.

G US Government ordering number.

I The publisher (issuer).

J For an article in a journal, the name of the journal.

K Keywords to be used for searching.

L Label.

N Journal issue number.

O Other information. This is usually printed at the end of the reference.

P Page number. A range of pages can be specified as *m*−*n*.

Q The name of the author, if the author is not a person. This will only be used if there are no **A** fields. There can only be one **Q** field.

R Technical report number.

S Series name.

T Title. For an article in a book or journal, this should be the title of the article.

V Volume number of the journal or book.

X Annotation.

For all fields except **A** and **E**, if there is more than one occurrence of a particular field in a record, only the last such field will be used.

If accent strings are used, they should follow the character to be accented. This means that the **AM** macro must be used with the **−ms** macros. Accent strings should not be quoted: use one \ rather than two.

Citations

The format of a citation is

```
.[opening-text
flags keywords
fields
.]closing-text
```

The *opening-text*, *closing-text* and *flags* components are optional. Only one of the *keywords* and *fields* components need be specified.

The *keywords* component says to search the bibliographic databases for a reference that contains all the words in *keywords*. It is an error if more than one reference if found.

The *fields* components specifies additional fields to replace or supplement those specified in the reference. When references are being accumulated and the *keywords* component is non-empty, then additional fields should be specified only on the first occasion that a particular reference is cited, and will apply to all citations of that reference.

The *opening-text* and *closing-text* component specifies strings to be used to bracket the label instead of the strings specified in the **bracket-label** command. If either of these components is non-empty, the strings specified in the **bracket-label** command will not be used; this behaviour can be altered using the **[** and **]** flags. Note that leading and trailing spaces are significant for these components.

The *flags* component is a list of non-alphanumeric characters each of which modifies the treatment of this particular citation. Unix refer will treat these flags as part of the keywords and so will ignore them since they are non-alphanumeric. The following flags are currently recognized:

This says to use the label specified by the **short-label** command, instead of that specified by the **label** command. If no short label has been specified, the normal label will be used. Typically the short label is used with author-date labels and consists of only the date and possibly a disambiguating letter; the **#** is supposed to be suggestive of a numeric type of label.

[Precede *opening-text* with the first string specified in the **bracket-label** command.

] Follow *closing-text* with the second string specified in the **bracket-label** command.

One advantages of using the **[** and **]** flags rather than including the brackets in *opening-text* and *closing-text* is that you can change the style of bracket used in the document just by changing the **bracket-label** command. Another advantage is that sorting and merging of citations will not necessarily be inhibited if the flags are used.

If a label is to be inserted into the text, it will be attached to the line preceding the **.[** line. If there is no such line, then an extra line will be inserted before the **.[** line and a warning will be given.

There is no special notation for making a citation to multiple references. Just use a sequence of citations, one for each reference. Don't put anything between the citations. The labels for all the citations will be attached to the line preceding the first citation. The labels may also be sorted or merged. See the description of the **<>** label expression, and of the **sort-adjacent-labels** and **abbreviate-label-ranges** command. A label will not be merged if its citation has a non-empty *opening-text* or *closing-text*. However, the labels for a citation using the **]** flag and without any *closing-text* immediately followed by a citation using the **[** flag and without any *opening-text* may be sorted and merged even though the first citation's *opening-text* or the second citation's *closing-text* is non-empty. (If you wish to prevent this just make the first citation's *closing-text* **\&**.)

Commands

Commands are contained between lines starting with **.R1** and **.R2**. Recognition of these lines can be prevented by the **−R** option. When a **.R1** line is recognized any accumulated references are flushed out. Neither **.R1** nor **.R2** lines, nor anything between them is output.

Commands are separated by newlines or **;**s. **#** introduces a comment that extends to the end of the line (but does not conceal the newline). Each command is broken up into words. Words are separated by spaces or tabs. A word that begins with **"** extends to the next **"** that is not followed by another **"**. If there is no such **"** the word extends to the end of the line. Pairs of **"** in a word beginning with **"** collapse to a single **"**. Neither **#** nor **;** are recognized inside **"**s. A line can be continued by ending it with ****; this works everywhere except after a **#**.

Each command *name* that is marked with ∗ has an associated negative command **no-***name* that undoes the effect of *name*. For example, the **no-sort** command specifies that references should not be sorted. The negative commands take no arguments.

In the following description each argument must be a single word; *field* is used for a single upper or lower case letter naming a field; *fields* is used for a sequence of such letters; *m* and *n* are used for a non-negative numbers; *string* is used for an arbitrary string; *filename* is used for the name of a file.

abbreviate∗ *fields string1 string2 string3 string4*
: Abbreviate the first names of *fields*. An initial letter will be separated from another initial letter by *string1*, from the last name by *string2*, and from anything else (such as a **von** or **de**) by *string3*. These default to a period followed by a space. In a hyphenated first name, the initial of the first part of the name will be separated from the hyphen by *string4*; this defaults to a period. No attempt is made to handle any ambiguities that might result from abbreviation. Names are abbreviated before sorting and before label construction.

abbreviate-label-ranges∗ *string*
: Three or more adjacent labels that refer to consecutive references will be abbreviated to a label consisting of the first label, followed by *string* followed by the last label. This is mainly useful with numeric labels. If *string* is omitted it defaults to **−**.

accumulate∗
: Accumulate references instead of writing out each reference as it is encountered. Accumulated references will be written out whenever a reference of the form

```
.[
$LIST$
.]
```

is encountered, after all input files have been processed, and whenever **.R1** line is recognized.

annotate∗ *field string*
: *field* is an annotation; print it at the end of the reference as a paragraph preceded by the line

.*string*

If *string* is omitted it will default to **AP**; if *field* is also omitted it will default to **X**. Only one field can be an annotation.

articles *string* . . .
: *string* . . . are definite or indefinite articles, and should be ignored at the beginning of **T** fields when sorting. Initially, **the**, **a** and **an** are recognized as articles.

bibliography *filename* . . .
: Write out all the references contained in the bibliographic databases *filename* . . . This command should come last in a **.R1/.R2** block.

bracket-label *string1 string2 string3*
: In the text, bracket each label with *string1* and *string2*. An occurrence of *string2* immediately followed by *string1* will be turned into *string3*. The default behaviour is

    bracket-label *([. *(.] ", "

capitalize *fields* Convert *fields* to caps and small caps.

compatible∗ Recognize **.R1** and **.R2** even when followed by a character other than space or newline.

database *filename* . . . Search the bibliographic databases *filename* . . . For each *filename* if an index *filename***.i** created by **indxbib**(1) exists, then it will be searched instead; each index can cover multiple databases.

date-as-label∗ *string* *string* is a label expression that specifies a string with which to replace the **D** field after constructing the label. See the **Label expressions** subsection for a description of label expressions. This command is useful if you do not want explicit labels in the reference list, but instead want to handle any necessary disambiguation by qualifying the date in some way. The label used in the text would typically be some combination of the author and date. In most cases you should also use the **no-label-in-reference** command. For example,

    date-as-label D.+yD.y%a*D.-y

would attach a disambiguating letter to the year part of the **D** field in the reference.

default-database∗ The default database should be searched. This is the default behaviour, so the negative version of this command is more useful. **refer** determines whether the default database should be searched on the first occasion that it needs to do a search. Thus a **no-default-database** command must be given before then, in order to be effective.

discard∗ *fields* When the reference is read, *fields* should be discarded; no string definitions for *fields* will be output. Initially, *fields* are **XYZ**.

et-al∗ *string m n* Control use of **et al** in the evaluation of @ expressions in label expressions. If the number of authors needed to make the author sequence unambiguous is *u* and the total number of authors is *t* then the last $t-u$ authors will be replaced by *string* provided that $t-u$ is not less than *m* and *t* is not less than *n*. The default behaviour is

    et-al " et al" 2 3

include *filename* Include *filename* and interpret the contents as commands.

join-authors *string1 string2 string3*
: This says how authors should be joined together. When there are exactly two authors, they will be joined with *string1*. When there are more than two authors, all but the last two will be joined with *string2*, and the last two authors will be joined with *string3*. If *string3* is omitted, it will default to *string1*; if *string2* is also omitted it will also default to *string1*. For example,

    join-authors " and " ", " ", and "

will restore the default method for joining authors.

label-in-reference∗ When outputting the reference, define the string **[F** to be the reference's label. This is the default behaviour; so the negative version of this command is more useful.

label-in-text* For each reference output a label in the text. The label will be separated from the surrounding text as described in the **bracket-label** command. This is the default behaviour; so the negative version of this command is more useful.

label *string* *string* is a label expression describing how to label each reference.

separate-label-second-parts *string*
When merging two-part labels, separate the second part of the second label from the first label with *string*. See the description of the **<>** label expression.

move-punctuation* In the text, move any punctuation at the end of line past the label. It is usually a good idea to give this command unless you are using superscripted numbers as labels.

reverse* *string* Reverse the fields whose names are in *string*. Each field name can be followed by a number which says how many such fields should be reversed. If no number is given for a field, all such fields will be reversed.

search-ignore* *fields* While searching for keys in databases for which no index exists, ignore the contents of *fields*. Initially, fields **XYZ** are ignored.

search-truncate* *n* Only require the first *n* characters of keys to be given. In effect when searching for a given key words in the database are truncated to the maximum of *n* and the length of the key. Initially *n* is 6.

short-label* *string* *string* is a label expression that specifies an alternative (usually shorter) style of label. This is used when the **#** flag is given in the citation. When using author-date style labels, the identity of the author or authors is sometimes clear from the context, and so it may be desirable to omit the author or authors from the label. The **short-label** command will typically be used to specify a label containing just a date and possibly a disambiguating letter.

sort* *string* Sort references according to **string**. References will automatically be accumulated. *string* should be a list of field names, each followed by a number, indicating how many fields with the name should be used for sorting. **+** can be used to indicate that all the fields with the name should be used. Also **.** can be used to indicate the references should be sorted using the (tentative) label. (The **Label expressions** subsection describes the concept of a tentative label.)

sort-adjacent-labels* Sort labels that are adjacent in the text according to their position in the reference list. This command should usually be given if the **abbreviate-label-ranges** command has been given, or if the label expression contains a **<>** expression. This will have no effect unless references are being accumulated.

Label expressions

Label expressions can be evaluated both normally and tentatively. The result of normal evaluation is used for output. The result of tentative evaluation, called the *tentative label*, is used to gather the information that normal evaluation needs to disambiguate the label. Label expressions specified by the **date-as-label** and **short-label** commands are not evaluated tentatively. Normal and tentative evaluation are the same for all types of expression other than @, *, and % expressions. The description below applies to normal evaluation, except where otherwise specified.

field
field n The *n*-th part of *field*. If *n* is omitted, it defaults to 1.

'*string*' The characters in *string* literally.

@ All the authors joined as specified by the **join-authors** command. The whole of each author's name will be used. However, if the references are sorted by author (that is the sort specification starts with **A+**), then authors' last names will be used instead, provided that this does not introduce ambiguity, and also an initial subsequence of the authors may be used instead of all the authors, again provided that this does not introduce ambiguity. The use of only the last name for the *i*-th author of some reference is considered to be ambiguous if there is some other reference, such that the first *i* - 1 authors of the references are the same, the *i*-th authors are not the same, but the *i*-th authors' last names are the same. A proper initial subsequence of the sequence of authors for some reference is considered to be ambiguous if there is a reference with some other sequence of authors which also has that subsequence as a proper initial subsequence. When an initial subsequence of authors is used, the remaining authors are replaced by the string specified by the **et-al** command; this command may also specify additional requirements that must be met before an initial subsequence can be used. @ tentatively evaluates to a canonical representation of the authors, such that authors that compare equally for sorting purpose will have the same representation.

%*n*
%a
%A
%i
%I The serial number of the reference formatted according to the character following the **%**. The serial number of a reference is 1 plus the number of earlier references with same tentative label as this reference. These expressions tentatively evaluate to an empty string.

*expr** If there is another reference with the same tentative label as this reference, then *expr*, otherwise an empty string. It tentatively evaluates to an empty string.

*expr***+***n*
*expr***−***n* The first (**+**) or last (**−**) *n* upper or lower case letters or digits of *expr*. Troff special characters (such as **\('a**) count as a single letter. Accent strings are retained but do not count towards the total.

*expr***.l** *expr* converted to lowercase.

*expr***.u** *expr* converted to uppercase.

*expr***.c** *expr* converted to caps and small caps.

*expr***.r** *expr* reversed so that the last name is first.

*expr***.a** *expr* with first names abbreviated. Note that fields specified in the **abbreviate** command are abbreviated before any labels are evaluated. Thus **.a** is useful only when you want a field to be abbreviated in a label but not in a reference.

*expr***.y** The year part of *expr*.

*expr***.+y**
The part of *expr* before the year, or the whole of *expr* if it does not contain a year.

*expr***.−y**
The part of *expr* after the year, or an empty string if *expr* does not contain a year.

*expr***.n** The last name part of *expr*.

*expr1***~***expr2*
expr1 except that if the last character of *expr1* is **−** then it will be replaced by *expr2*.

expr1 expr2
The concatenation of *expr1* and *expr2*.

*expr1***|***expr2*
If *expr1* is non-empty then *expr1* otherwise *expr2*.

*expr1***&***expr2*
If *expr1* is non-empty then *expr2* otherwise an empty string.

*expr1***?***expr2***:***expr3*
If *expr1* is non-empty then *expr2* otherwise *expr3*.

<*expr***>** The label is in two parts, which are separated by *expr*. Two adjacent two-part labels which have the same first part will be merged by appending the second part of the second label onto the first label separated by the string specified in the **separate-label-second-parts** command (initially, a comma followed by a space); the resulting label will also be a two-part label with the same first part as before merging, and so additional labels can be merged into it. Note that it is permissible for the first part to be empty; this maybe desirable for expressions used in the **short-label** command.

(*expr*) The same as *expr*. Used for grouping.

The above expressions are listed in order of precedence (highest first); **&** and **|** have the same precedence.

Macro interface

Each reference starts with a call to the macro **]-**. The string **[F** will be defined to be the label for this reference, unless the **no-label-in-reference** command has been given. There then follows a series of string definitions, one for each field: string **[***X* corresponds to field *X*. The number register **[P** is set to 1 if the **P** field contains a range of pages. The **[T**, **[A** and **[O** number registers are set to 1 according as the **T**, **A** and **O** fields end with one of the characters **.?!**. The **[E** number register will be set to 1 if the **[E** string contains more than one name. The reference is followed by a call to the **][** macro. The first argument to this macro gives a number representing the type of the reference. If a reference contains a **J** field, it will be classified as type 1, otherwise if it contains a **B** field, it will type 3, otherwise if it contains a **G** or **R** field it will be type 4, otherwise if contains a **I** field it will be type 2, otherwise it will be type 0. The second argument is a symbolic name for the type: **other**, **journal-article**, **book**, **article-in-book** or **tech-report**. Groups of references that have been accumulated or are produced by the **bibliography** command are preceded by a call to the **]<** macro and followed by a call to the **]>** macro.

FILES

/usr/dict/papers/Ind	Default database.
*file***.i**	Index files.

ENVIRONMENT

REFER If set, overrides the default database.

SEE ALSO

indxbib(1), **lookbib**(1), **lkbib**(1)

BUGS

In label expressions, **<>** expressions are ignored inside **.***char* expressions.

roff2dvi

NAME

roff2dvi – transform roff code into dvi mode

SYNOPSIS

[*groffer_option*...] [**––**] [*filespec*...] **–h | ––help –v | ––version**

The options **–v** and **––version** print the version information of the program to standard output and exit. The options **–h** and **––help** print a usage information of the program to standard output and stop the program instantly.

All other options are assumed to be **groffer** options. They are internally passed to **groffer**. They override the behavior of the program. The options are optional, they can be omitted.

The *filespec* arguments correspond to the *filespec* arguments of **groffer**. So they are either the names of existing, readable files or **–** for standard input, or the name of a man page or a **groffer**(1) man page search pattern. If no *filespec* is specified standard input is assumed automatically.

DESCRIPTION

roff2dvi transforms *roff* code into *dvi* mode. Print the result to standard output.

There are more of these programs for generating other formats of *roff* input.

roff2html(1)
: generates *html* output.

roff2pdf(1)
: outputs *pdf* mode.

roff2ps(1)
: prints *PostScript* format to standard output.

roff2text(1)
: generates text output in the **groff** device *latin1*.

roff2x(1)
: prints the output in the **groff** device **X** that is suitable for programs like **gxditview**(1) or **xditview**(1).

SEE ALSO

groff(1), **groffer**(1), **roff2html**(1), **roff2pdf**(1), **roff2ps**(1), **roff2text**(1), **roff2x**(1), **gxditview**(1).

AUTHOR

This file was written by Bernd Warken.

COPYING

roff2html

NAME

roff2html – transform roff code into html mode

SYNOPSIS

[*groffer_option*...] [**––**] [*filespec*...] **–h | ––help –v | ––version**

The options **–v** and **––version** print the version information of the program to standard output and exit. The options **–h** and **––help** print a usage information of the program to standard output and stop the program instantly.

All other options are assumed to be **groffer** options. They are internally passed to **groffer**. They override the behavior of the program. The options are optional, they can be omitted.

The *filespec* arguments correspond to the *filespec* arguments of **groffer**. So they are either the names of existing, readable files or **–** for standard input, or the name of a man page or a **groffer**(1) man page search pattern. If no *filespec* is specified standard input is assumed automatically.

DESCRIPTION

roff2html transforms *roff* code into *html* mode. Print the result to standard output.

There are more of these programs for generating other formats of *roff* input.

roff2dvi(1)
: is for *dvi* mode.

roff2pdf(1)
: outputs *pdf* mode.

roff2ps(1)
: prints *PostScript* format to standard output.

roff2text(1)
: generates text output in the **groff** device *latin1*.

roff2x(1)
: prints the output in the **groff** device **X** that is suitable for programs like **gxditview**(1) or **xditview**(1).

SEE ALSO

groff(1), **groffer**(1), **roff2dvi**(1), **roff2pdf**(1), **roff2ps**(1), **roff2text**(1), **roff2x**(1), **gxditview**(1).

AUTHOR

This file was written by Bernd Warken.

COPYING

roff2pdf

NAME

roff2pdf – transform roff code into pdf mode

SYNOPSIS

[*groffer_option*. . .] [**––**] [*filespec*. . .] **–h | ––help –v | ––version**

The options **–v** and **––version** print the version information of the program to standard output and exit. The options **–h** and **––help** print a usage information of the program to standard output and stop the program instantly.

All other options are assumed to be **groffer** options. They are internally passed to **groffer**. They override the behavior of the program. The options are optional, they can be omitted.

The *filespec* arguments correspond to the *filespec* arguments of **groffer**. So they are either the names of existing, readable files or **–** for standard input, or the name of a man page or a **groffer**(1) man page search pattern. If no *filespec* is specified standard input is assumed automatically.

DESCRIPTION

roff2pdf transforms *roff* code into *pdf* mode. Print the result to standard output.

There are more of these programs for generating other formats of *roff* input.

roff2dvi(1)
is for *dvi* mode.

roff2html(1)
generates *html* output.

roff2ps(1)
prints *PostScript* format to standard output.

roff2text(1)
generates text output in the **groff** device *latin1*.

roff2x(1)
prints the output in the **groff** device **X** that is suitable for programs like **gxditview**(1) or **xditview**(1).

SEE ALSO

groff(1), **groffer**(1), **roff2dvi**(1), **roff2html**(1), **roff2ps**(1), **roff2text**(1), **roff2x**(1), **gxditview**(1).

AUTHOR

This file was written by Bernd Warken.

COPYING

roff2ps

NAME

roff2ps – transform roff code into ps mode

SYNOPSIS

[*groffer_option*...] [**––**] [*filespec*...] **–h** | **––help** **–v** | **––version**

The options **–v** and **––version** print the version information of the program to standard output and exit. The options **–h** and **––help** print a usage information of the program to standard output and stop the program instantly.

All other options are assumed to be **groffer** options. They are internally passed to **groffer**. They override the behavior of the program. The options are optional, they can be omitted.

The *filespec* arguments correspond to the *filespec* arguments of **groffer**. So they are either the names of existing, readable files or **–** for standard input, or the name of a man page or a **groffer**(1) man page search pattern. If no *filespec* is specified standard input is assumed automatically.

DESCRIPTION

roff2ps transforms *roff* code into *ps* mode. Print the result to standard output.

There are more of these programs for generating other formats of *roff* input.

roff2dvi(1)
  is for *dvi* mode.

roff2html(1)
  generates *html* output.

roff2pdf(1)
  outputs *pdf* mode.

roff2text(1)
  generates text output in the **groff** device *latin1*.

roff2x(1)
  prints the output in the **groff** device **X** that is suitable for programs like **gxditview**(1) or **xditview**(1).

SEE ALSO

groff(1), **groffer**(1), **roff2dvi**(1), **roff2html**(1), **roff2pdf**(1), **roff2text**(1), **roff2x**(1), **gxditview**(1).

AUTHOR

This file was written by Bernd Warken.

COPYING

roff2text

NAME

roff2text – transform roff code into text mode

SYNOPSIS

[*groffer_option*...] [**––**] [*filespec*...] **–h | ––help –v | ––version**

The options **–v** and **––version** print the version information of the program to standard output and exit. The options **–h** and **––help** print a usage information of the program to standard output and stop the program instantly.

All other options are assumed to be **groffer** options. They are internally passed to **groffer**. They override the behavior of the program. The options are optional, they can be omitted.

The *filespec* arguments correspond to the *filespec* arguments of **groffer**. So they are either the names of existing, readable files or **–** for standard input, or the name of a man page or a **groffer**(1) man page search pattern. If no *filespec* is specified standard input is assumed automatically.

DESCRIPTION

roff2text transforms *roff* code into *text* mode. Print the result to standard output.

There are more of these programs for generating other formats of *roff* input.

roff2dvi(1)
is for *dvi* mode.

roff2html(1)
generates *html* output.

roff2pdf(1)
outputs *pdf* mode.

roff2ps(1)
prints *PostScript* format to standard output.

roff2x(1)
prints the output in the **groff** device **X** that is suitable for programs like **gxditview**(1) or **xditview**(1).

SEE ALSO

groff(1), **groffer**(1), **roff2dvi**(1), **roff2html**(1), **roff2pdf**(1), **roff2ps**(1), **roff2x**(1), **gxditview**(1).

AUTHOR

This file was written by Bernd Warken.

COPYING

roff2x

NAME

roff2x – transform roff code into x mode

SYNOPSIS

[*groffer_option*...] [**––**] [*filespec*...] **–h | ––help –v | ––version**

The options **–v** and **––version** print the version information of the program to standard output and exit. The options **–h** and **––help** print a usage information of the program to standard output and stop the program instantly.

All other options are assumed to be **groffer** options. They are internally passed to **groffer**. They override the behavior of the program. The options are optional, they can be omitted.

The *filespec* arguments correspond to the *filespec* arguments of **groffer**. So they are either the names of existing, readable files or **–** for standard input, or the name of a man page or a **groffer**(1) man page search pattern. If no *filespec* is specified standard input is assumed automatically.

DESCRIPTION

roff2x transforms *roff* code into *X* mode corresponding to the *groff* devices **X***; this mode is suitable for **gxditview**(1). Print the result to standard output.

There are more of these programs for generating other formats of *roff* input.

roff2dvi(1)
is for *dvi* mode.

roff2html(1)
generates *html* output.

roff2pdf(1)
outputs *pdf* mode.

roff2ps(1)
prints *PostScript* format to standard output.

roff2text(1)
generates text output in the **groff** device *latin1*.

SEE ALSO

groff(1), **groffer**(1), **roff2dvi**(1), **roff2html**(1), **roff2pdf**(1), **roff2ps**(1), **roff2text**(1), **gxditview**(1).

AUTHOR

This file was written by Bernd Warken.

COPYING

SOELIM

NAME

soelim – interpret .so requests in groff input

SYNOPSIS

[**–Crtv**] [**–I***dir*] [*files* ...]

It is possible to have whitespace between the **–I** command line option and its parameter.

DESCRIPTION

soelim reads *files* and replaces lines of the form

.so *file*

by the contents of *file*. It is useful if files included with **.so** need to be preprocessed. Normally, **soelim** should be invoked with the **–s** option of **groff**.

To embed '\' in the file name, write '\\' or '\e'. To embed a space, write '\ '. Any other escape sequence in *file* makes **soelim** ignore the whole line.

Note that there must be no whitespace between the leading dot and the two characters 's' and 'o'. Otherwise, only **groff** interprets the **.so** request (and **soelim** ignores it).

OPTIONS

–C Recognize **.so** even when followed by a character other than space or newline.

–I*dir* This option may be used to add a directory to the search path for files (both those on the command line and those named in **.so** requests). The search path is initialized with the current directory. This option may be specified more than once; the directories are then searched in the order specified (but before the current directory). If you want to make the current directory be read before other directories, add **–I.** at the appropriate place.

No directory search is performed for files with an absolute file name.

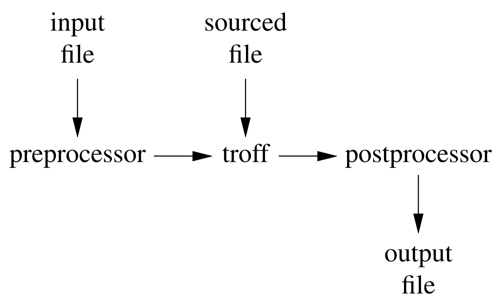
–r Do not add **.lf** requests (for general use, with non-groff files).

–t Don't emit **.lf** requests but TeX comment lines (starting with '%') giving the current file and line number.

–v Print the version number.

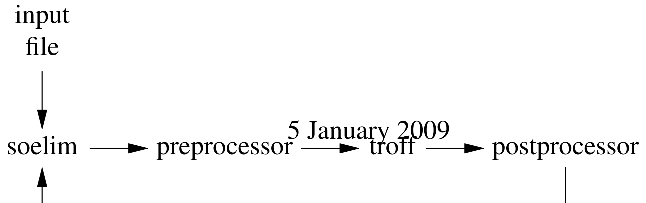
USAGE

The normal processing sequence of groff is this:



That is, files sourced with **.so** are normally read *only* by **troff** (the actual formatter). **soelim** is *not* required for **troff** to source files.

If a file to be sourced should also be preprocessed, it must already be read *before* the input file passes through the preprocessor. This is handled by **soelim**:



SEE ALSO

groff(1)

TBL

NAME

tbl – format tables for troff

SYNOPSIS

[**–Cv**] [*files* ...]

DESCRIPTION

This manual page describes the GNU version of **tbl**, which is part of the groff document formatting system. **tbl** compiles descriptions of tables embedded within **troff** input files into commands that are understood by **troff**. Normally, it should be invoked using the **–t** option of **groff.** It is highly compatible with Unix **tbl**. The output generated by GNU **tbl** cannot be processed with Unix **troff**; it must be processed with GNU **troff**. If no files are given on the command line or a filename of **–** is given, the standard input is read.

OPTIONS

–C Enable compatibility mode to recognize **.TS** and **.TE** even when followed by a character other than space or newline. Leader characters (\a) are handled as interpreted.

–v Print the version number.

USAGE

tbl expects to find table descriptions wrapped in the **.TS** (table start) and **.TE** (table end) macros.

Global options

The line immediately following the **.TS** macro may contain any of the following global options (ignoring the case of characters – Unix tbl only accepts options with all characters lowercase or all characters uppercase), separated by spaces, tabs, or commas:

allbox Enclose each item of the table in a box.

box Enclose the table in a box.

center Center the table (default is left-justified). The alternative keyword name **centre** is also recognized (this is a GNU tbl extension).

decimalpoint(*c***)**
Set the character to be recognized as the decimal point in numeric columns (GNU tbl only).

delim(*xy***)**
Use *x* and *y* as start and end delimiters for **eqn**(1).

doublebox
Enclose the table in a double box.

doubleframe
Same as doublebox (GNU tbl only).

expand
Make the table as wide as the current line length (providing a column separation factor). Ignored if one or more ‘x’ column specifiers are used (see below).

In case the sum of the column widths is larger than the current line length, the column separation factor is set to zero; such tables extend into the right margin, and there is no column separation at all.

frame Same as box (GNU tbl only).

linesize(*n***)**
Set lines or rules (e.g. from **box**) in *n*-point type.

nokeep Don’t use diversions to prevent page breaks (GNU tbl only). Normally **tbl** attempts to prevent undesirable breaks in boxed tables by using diversions. This can sometimes interact badly with macro packages’ own use of diversions, when footnotes, for example, are used.

nospaces
Ignore leading and trailing spaces in data items (GNU tbl only).

tab(*x***)** Use the character *x* instead of a tab to separate items in a line of input data.

The global options must end with a semicolon. There might be whitespace between an option and its argument in parentheses.

Table format specification

After global options come lines describing the format of each line of the table. Each such format line describes one line of the table itself, except that the last format line (which you must end with a period) describes all remaining lines of the table. A single-key character describes each column of each line of the table. Key characters can be separated by spaces or tabs. You may run format specifications for multiple lines together on the same line by separating them with commas.

You may follow each key character with specifiers that determine the font and point size of the corresponding item, that determine column width, inter-column spacing, etc.

The longest format line defines the number of columns in the table; missing format descriptors at the end of format lines are assumed to be **L**. Extra columns in the data (which have no corresponding format entry) are ignored.

The available key characters are:

a,**A** Center longest line in this column and then left-justifies all other lines in this column with respect to that centered line. The idea is to use such alphabetic subcolumns (hence the name of the key character) in combination with **L**; they are called subcolumns because **A** items are indented by 1n relative to **L** entries. Example:

.TS tab(;); ln,an. item one;1 subitem two;2 subitem three;3 .T& ln,an. item eleven;11 subitem twentytwo;22 subitem thirtythree;33 .TE

Result:

```
item one              1
  subitem two         2
  subitem three       3
item eleven          11
  subitem twentytwo  22
  subitem thirtythree 33
```

c,**C** Center item within the column.

l,**L** Left-justify item within the column.

n,**N** Numerically justify item in the column: Units positions of numbers are aligned vertically. If there is one or more dots adjacent to a digit, use the rightmost one for vertical alignment. If there is no dot, use the rightmost digit for vertical alignment; otherwise, center the item within the column. Alignment can be forced to a certain position using '\&'; if there is one or more instances of this special (non-printing) character present within the data, use the leftmost one for alignment. Example:

.TS n. 1 1.5 1.5.3 abcde a\&bcde .TE

Result:

```
  1
  1.5
1.5.3
 abcde
  abcde
```

If numerical entries are combined with **L** or **R** entries – this can happen if the table format is changed with **.T&** –, center the widest *number* (of the data entered under the **N** specifier regime) relative to the widest **L** or **R** entry, preserving the alignment of all numerical entries. Contrary to **A** type entries, there is no extra indentation.

Using equations (to be processed with **eqn**) within columns which use the **N** specifier is problematic in most cases due to **tbl**'s algorithm for finding the vertical alignment, as described above. Using the global **delim** option, however, it is possible to make **tbl** ignore the data within **eqn** delimiters for that purpose.

r,**R** Right-justify item within the column.

s,**S** Span previous item on the left into this column. Not allowed for the first column.

^ Span down entry from previous row in this column. Not allowed for the first row.

_,**-** Replace this entry with a horizontal line.

= Replace this entry with a double horizontal line.

| The corresponding column becomes a vertical rule (if two of these are adjacent, a double vertical rule).

A vertical bar to the left of the first key letter or to the right of the last one produces a line at the edge of the table.

To change the data format within a table, use the **.T&** command (at the start of a line). It is followed by format and data lines (but no global options) similar to the **.TS** request.

Column specifiers

Here are the specifiers that can appear in suffixes to column key letters (in any order):

b,**B** Short form of **fB** (make affected entries bold).

d,**D** Start an item vertically spanning rows at the bottom of its range rather than vertically centering it (GNU tbl only).

e,**E** Make equally-spaced columns. All columns marked with this specifier get the same width; this happens after the affected column widths have been computed (this means that the largest width value rules).

f,**F** Either of these specifiers may be followed by a font name (either one or two characters long), font number (a single digit), or long name in parentheses (the last form is a GNU tbl extension). A one-letter font name must be separated by one or more blanks from whatever follows.

i,**I** Short form of **fI** (make affected entries italic).

m,**M** This is a GNU tbl extension. Either of these specifiers may be followed by a macro name (either one or two characters long), or long name in parentheses. A one-letter macro name must be separated by one or more blanks from whatever follows. The macro which name can be specified here must be defined before creating the table. It is called just before the table's cell text is output. As implemented currently, this macro is only called if block input is used, that is, text between 'T{' and 'T}'. The macro should contain only simple **troff** requests to change the text block formatting, like text adjustment, hyphenation, size, or font. The macro is called *after* other cell modifications like **b**, **f** or **v** are output. Thus the macro can overwrite other modification specifiers.

p,**P** Followed by a number, this does a point size change for the affected fields. If signed, the current point size is incremented or decremented (using a signed number instead of a signed digit is a GNU tbl extension). A point size specifier followed by a column separation number must be separated by one or more blanks.

t,**T** Start an item vertically spanning rows at the top of its range rather than vertically centering it.

u,**U** Move the corresponding column up one half-line.

v,**V** Followed by a number, this indicates the vertical line spacing to be used in a multi-line table entry. If signed, the current vertical line spacing is incremented or decremented (using a signed number instead of a signed digit is a GNU tbl extension). A vertical line spacing specifier followed by a column separation number must be separated by one or more blanks. No effect if the corresponding table entry isn't a text block.

w,**W** Minimal column width value. Must be followed either by a **troff**(1) width expression in parentheses or a unitless integer. If no unit is given, en units are used. Also used as the default line length for included text blocks. If used multiple times to specify the width for a particular column, the last entry takes effect.

x,**X** An expanded column. After computing all column widths without an **x** specifier, use the remaining line width for this column. If there is more than one expanded column, distribute the remaining horizontal space evenly among the affected columns (this is a GNU extension). This feature has the same effect as specifying a minimum column width.

z,**Z** Ignore the corresponding column for width-calculation purposes, this is, don't use the fields but only the specifiers of this column to compute its width.

A number suffix on a key character is interpreted as a column separation in en units (multiplied in proportion if the **expand** option is on – in case of overfull tables this might be zero). Default separation is 3n.

The column specifier **x** is mutually exclusive with **e** and **w** (but **e** is not mutually exclusive with **w**); if specified multiple times for a particular column, the last entry takes effect: **x** unsets both **e** and **w**, while either **e** or **w** overrides **x**.

Table data

The format lines are followed by lines containing the actual data for the table, followed finally by **.TE**. Within such data lines, items are normally separated by tab characters (or the character specified with the **tab** option). Long input lines can be broken across multiple lines if the last character on the line is '\' (which vanishes after concatenation).

Note that **tbl** computes the column widths line by line, applying \w on each entry which isn't a text block. As a consequence, constructions like

```
.TS c,l.  \s[20]MM MMMM .TE
```

fail; you must either say

```
.TS cp20,lp20.  MM MMMM .TE
```

or

```
.TS c,l.  \s[20]MM \s[20]MMMM .TE
```

A dot starting a line, followed by anything but a digit is handled as a troff command, passed through without changes. The table position is unchanged in this case.

If a data line consists of only '_' or '=', a single or double line, respectively, is drawn across the table at that point; if a single item in a data line consists of only '_' or '=', then that item is replaced by a single or double line, joining its neighbours. If a data item consists only of '_' or '\=', a single or double line, respectively, is drawn across the field at that point which does not join its neighbours.

A data item consisting only of '\Rx' ('x' any character) is replaced by repetitions of character 'x' as wide as the column (not joining its neighbours).

A data item consisting only of '\^' indicates that the field immediately above spans downward over this row.

Text blocks

A text block can be used to enter data as a single entry which would be too long as a simple string between tabs. It is started with 'T{' and closed with 'T}'. The former must end a line, and the latter must start a line, probably followed by other data columns (separated with tabs or the character given with the **tab** global option).

By default, the text block is formatted with the settings which were active before entering the table, possibly overridden by the **m**, **v**, and **w** tbl specifiers. For example, to make all text blocks ragged-right, insert **.na** right before the starting **.TS** (and **.ad** after the table).

If either 'w' or 'x' specifiers are not given for *all* columns of a text block span, the default length of the text block (to be more precise, the line length used to process the text block diversion) is computed as L×C/(N+1), where 'L' is the current line length, 'C' the number of columns spanned by the text block, and 'N' the total number of columns in the table. Note, however, that the actual diversion width as returned in register **\n[dl]** is used eventually as the text block width. If necessary, you can also control the text block width with a direct insertion of a **.ll** request right after 'T{'.

Miscellaneous

The number register **\n[TW]** holds the table width; it can't be used within the table itself but is defined right before calling **.TE** so that this macro can make use of it.

tbl also defines a macro **.T#** which produces the bottom and side lines of a boxed table. While **tbl** does call this macro itself at the end of the table, it can be used by macro packages to create boxes for multi-page tables by calling it within the page footer. An example of this is shown by the **–ms** macros which provide this functionality if a table starts with **.TS H** instead of the standard call to the **.TS** macro.

INTERACTION WITH EQN

tbl(1) should always be called before **eqn**(1) (**groff**(1) automatically takes care of the correct order of preprocessors).

GNU TBL ENHANCEMENTS

There is no limit on the number of columns in a table, nor any limit on the number of text blocks. All the lines of a table are considered in deciding column widths, not just the first 200. Table continuation (**.T&**) lines are not restricted to the first 200 lines.

Numeric and alphabetic items may appear in the same column.

Numeric and alphabetic items may span horizontally.

tbl uses register, string, macro and diversion names beginning with the digit **3**. When using **tbl** you should avoid using any names beginning with a **3**.

GNU TBL WITHIN MACROS

Since **tbl** defines its own macros (right before each table) it is necessary to use an 'end-of-macro' macro. Additionally, the escape character has to be switched off. Here an example.

```
.eo .de ATABLE .. .TS allbox tab(;); cl. \$1;\$2 .TE ... .ec .ATABLE A table .ATABLE
Another table .ATABLE And "another one"
```

Note, however, that not all features of **tbl** can be wrapped into a macro because **tbl** sees the input earlier than **troff**. For example, number formatting with vertically aligned decimal points fails if those numbers are passed on as macro parameters because decimal point alignment is handled by **tbl** itself: It only sees '\$1', '\$2', etc., and therefore can't recognize the decimal point.

BUGS

You should use **.TS H/.TH** in conjunction with a supporting macro package for *all* multi-page boxed tables. If there is no header that you wish to appear at the top of each page of the table, place the **.TH** line immediately after the format section. Do not enclose a multi-page table within keep/release macros, or divert it in any other way.

A text block within a table must be able to fit on one page.

The **bp** request cannot be used to force a page-break in a multi-page table. Instead, define **BP** as follows

```
.de BP . ie '\\n(.z'' .bp \\$1 . el \!.BP \\$1 ..
```

and use **BP** instead of **bp**.

Using \a directly in a table to get leaders does not work (except in compatibility mode). This is correct behaviour: \a is an **uninterpreted** leader. To get leaders use a real leader, either by using a control A or like this:

```
.ds a \a .TS tab(;); lw(1i) l. A\*a;B .TE
```

REFERENCE

Lesk, M.E.: "TBL – A Program to Format Tables". For copyright reasons it cannot be included in the groff distribution, but copies can be found with a title search on the World Wide Web.

SEE ALSO

groff(1), **troff**(1)

TFMTODIT

NAME

tfmtodit – create font files for use with groff −Tdvi

SYNOPSIS

tfmtodit [**−sv**] [**−g***gf_file*] [**−k***skewchar*] *tfm_file map_file font*

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

tfmtodit creates a font file for use with **groff −Tdvi**. *tfm_file* is the name of the TEX font metric file for the font. *map_file* is a file giving the groff names for characters in the font; this file should consist of a sequence of lines of the form:

n c1 c2 . . .

where *n* is a decimal integer giving the position of the character in the font, and *c1*, *c2*,. . . are the groff names of the character. If a character has no groff names but exists in the tfm file, then it will be put in the groff font file as an unnamed character. *font* is the name of the groff font file. The groff font file is written to *font*.

The **−s** option should be given if the font is special (a font is *special* if **troff** should search it whenever a character is not found in the current font.) If the font is special, it should be listed in the **fonts** command in the DESC file; if it is not special, there is no need to list it, since **troff** can automatically mount it when it's first used.

To do a good job of math typesetting, groff requires font metric information not present in the tfm file. The reason for this is that TEX has separate math italic fonts whereas groff uses normal italic fonts for math. The additional information required by groff is given by the two arguments to the **math_fit** macro in the Metafont programs for the Computer Modern fonts. In a text font (a font for which **math_fitting** is false), Metafont normally ignores these two arguments. Metafont can be made to put this information in the gf file by loading the following definition after **cmbase** when creating **cm.base**:

```
def ignore_math_fit(expr left_adjustment,right_adjustment) =
  special "adjustment";
  numspecial left_adjustment*16/designsize;
  numspecial right_adjustment*16/designsize;
  enddef;
```

For the EC font family, load the following definition after **exbase** (it is probably easiest to patch **exbase.mf** locally):

```
def ignore_math_fit(expr left_adjustment,right_adjustment) =
  ori_special "adjustment";
  ori_numspecial left_adjustment*16/designsize;
  ori_numspecial right_adjustment*16/designsize;
  enddef;
```

The gf file created using this modified **cm.base** or **exbase** should be specified with the **−g** option. The **−g** option should not be given for a font for which **math_fitting** is true.

OPTIONS

−v Print the version number.

−s The font is special. The effect of this option is to add the **special** command to the font file.

−k*n* The skewchar of this font is at position *n*. *n* should be an integer; it may be given in decimal, or with a leading **0** in octal, or with a leading **0x** in hexadecimal. The effect of this option is to ignore any kerns whose second component is the specified character.

−g*gf_file*
gf_file is a gf file produced by Metafont containing special and numspecial commands giving additional font metric information.

FILES

c:/progra 1/groff/share/groff/1.20/font/devdvi/DESC
Device description file.

c:/progra 1/groff/share/groff/1.20/font/devdvi/*F*
Font description file for font *F*.

SEE ALSO

groff(1), **grodvi**(1), **groff_font**(5)

TR2TEX

NAME

tr2latex – convert a document from troff to LaTeX

SYNOPSIS

tr2latex [**–m**] [**–t**] [*–n*] [**–s** *style*] [**–o** *outfile*] *filename...*

DESCRIPTION

Tr2latex converts a document typeset in **troff** to a **LaTeX** format. It is intended to do the first pass of the conversion. The user should then finish up the rest of the conversion and customize the converted manuscript to his/her liking. It can also serve as a tutor for those who want to convert from troff to LaTeX.

Most of the converted document will be in LaTeX but some of it may be in plain **TeX.** It will also use some macros in **troffms.sty** or **troffman.sty** which are included in the package and must be available to the document when processed with LaTeX.

If there is more than one input file, they will all be converted into one LaTeX document.

Tr2latex understands most of the **-ms** and **-man** macros and **eqn** preprocessor symbols. It also understands several plain **troff** commands. Few **tbl** preprocessor commands are understood to help convert very simple tables.

When converting manuals, use the **-m** flag. Otherwise **tr2latex** assumes to translate a **-me** input file.

If a troff command cannot be converted, the line that contain that command will be commented out.

NOTE: if you have **eqn** symbols, you must have the in-line mathematics delimiter defined by **delim** in the file you are converting. If it is defined in another setup file, that setup file has to be concatenated with the file to be converted, otherwise **tr2latex** will regard the in-line math as ordinary text.

OPTIONS

–m	Convert manual pages. This makes **tr2latex** understand most of the **-man** macros. It uses the style file **troffman.sty**.
–t	Produce twoside page style.
–n	Use a font size of *n*pt. The default font size is 12pt for **–man** and 11pt otherwise.
–s *style*	Use the style file *style* instead of the default **article.sty**.
–o *outfile*	Write output to file *outfile*.

BUGS

Many of these bugs are harmless. Most of them cause local errors that can be fixed in the converted manuscript.

– Some macros and macro arguments are not recognized.

– Commands that are not separated from their argument by a space are not properly parsed (e.g .sp3i).

– When some operators (notably over, sub and sup) are renamed (via define), then they are encountered in the text, **tr2latex** will treat them as ordinary macros and will not apply their rules.

– rpile, lpile and cpile are treated the same as cpile.

– rcol, lcol are treated the same as ccol.

– Math-mode size, gsize, fat, and gfont are ignored.

– lineup and mark are ignored. The rules are so different.

– Some troff commands are translated to commands that require delimiters that have to be explicitly put. Since they are sometimes not put in troff, they can create problems. Example: .nf not closed by .fi.

– When local motions are converted to \raise or \lower, an \hbox is needed, which has to be put manually after the conversion.

– 'a sub i sub j' is converted to 'a_i_j' which TeX parses as 'a_i{}_j}' with a complaint that it is vague. 'a sub {i sub j}' is parsed correctly and converted to 'a_{i_j}'.

– Line spacing is not changed within a paragraph in TeX (which is a bad practice anyway). TeX uses the last line spacing in effect in that paragraph.

TODO

Access registers via **.nr** command.

FILES

$(TEXLIB)/macros/troffman.sty
$(TEXLIB)/macros/troffms.sty

AUTHOR

Kamal Al-Yahya, Stanford University
Christian Engel, Aachen University of Technology'

TROFF

NAME

troff – the troff processor of the groff text formatting system

SYNOPSIS

[**–abcivzCERU**] [**–d***cs*] [**–f***fam*] [**–F***dir*] [**–I***dir*] [**–m***name*] [**–M***dir*] [**–n***num*] [**–o***list*] [**–r***cn*] [**–T***name*] [**–w***name*] [**–W***name*] [*file* ...]

DESCRIPTION

This manual page describes the GNU version of **troff**. It is part of the groff document formatting system. It is functionally compatible with UNIX troff, but has many extensions, see **groff_diff**(7). Usually it should be invoked using the **groff**(1) command which will also run preprocessors and postprocessors in the appropriate order and with the appropriate options.

OPTIONS

It is possible to have whitespace between a command line option and its parameter.

–a Generate an ASCII approximation of the typeset output.

–b Print a backtrace with each warning or error message. This backtrace should help track down the cause of the error. The line numbers given in the backtrace may not always be correct, for **troff**'s idea of line numbers gets confused by **as** or **am** requests.

–c Disable color output (always disabled in compatibility mode).

–C Enable compatibility mode.

–d*cs*
–d*name***=***s* Define *c* or *name* to be a string *s*; *c* must be a one letter name.

–E Inhibit all error messages of **troff**. Note that this doesn't affect messages output to standard error by macro packages using the **tm** or **tm1** requests.

–f*fam* Use *fam* as the default font family.

–F*dir* Search in directory (or directory path) *dir* for subdirectories **dev***name* (*name* is the name of the device) and there for the **DESC** file and font files. *dir* is scanned before all other font directories.

–i Read the standard input after all the named input files have been processed.

–I*dir* This option may be used to add a directory to the search path for files (both those on the command line and those named in **.psbb** requests). The search path is initialized with the current directory. This option may be specified more than once; the directories are then searched in the order specified (but before the current directory). If you want to make the current directory be read before other directories, add **–I.** at the appropriate place.

No directory search is performed for files with an absolute file name.

–m*name* Read in the file *name***.tmac**. If it isn't found, try **tmac.***name* instead. It will be first searched for in directories given with the **–M** command line option, then in directories given in the **GROFF_TMAC_PATH** environment variable, then in the current directory (only if in unsafe mode), the home directory, c:/progra 1/groff/lib/groff/site-tmac, c:/progra 1/groff/share/groff/site-tmac, and c:/progra 1/groff/share/groff/1.20/tmac.

–M*dir* Search directory (or directory path) *dir* for macro files. This is scanned before all other macro directories.

–n*num* Number the first page *num*.

–o*list* Output only pages in *list*, which is a comma-separated list of page ranges; *n* means print page *n*, *m*−*n* means print every page between *m* and *n*, −*n* means print every page up to *n*, *n*− means print every page from *n*. **troff** will exit after printing the last page in the list.

–r*cn*
–r*name***=***n* Set number register *c* or *name* to *n*; *c* must be a one character name; *n* can be any troff numeric expression.

–R Don't load **troffrc** and **troffrc-end**.

−T*name*	Prepare output for device *name*, rather than the default **ps**; see **groff**(1) for a more detailed description.
−U	Unsafe mode. This will enable the following requests: **open**, **opena**, **pso**, **sy**, and **pi**. For security reasons, these potentially dangerous requests are disabled otherwise. It will also add the current directory to the macro search path.
−v	Print the version number.
−w*name*	Enable warning *name*. Available warnings are described in the section *WARNINGS* below. For example, to enable all warnings, use **−w all**. Multiple **−w** options are allowed.
−W*name*	Inhibit warning *name*. Multiple **−W** options are allowed.
−z	Suppress formatted output.

WARNINGS

The warnings that can be given by **troff** are divided into the following categories. The name associated with each warning is used by the **−w** and **−W** options; the number is used by the **warn** request, and by the **.warn** register; it is always a power of 2 to allow bitwise composition.

Bit	Code	Warning	Bit	Code	Warning
0	*1*	**char**	10	*1024*	**reg**
1	*2*	**number**	11	*2048*	**tab**
2	*4*	**break**	12	*4096*	**right-brace**
3	*8*	**delim**	13	*8192*	**missing**
4	*16*	**el**	14	*16384*	**input**
5	*32*	**scale**	15	*32768*	**escape**
6	*64*	**range**	16	*65536*	**space**
7	*128*	**syntax**	17	*131072*	**font**
8	*256*	**di**	18	*262144*	**ig**
9	*512*	**mac**	19	*524288*	**color**

break	4	In fill mode, lines which could not be broken so that their length was less than the line length. This is enabled by default.
char	1	Non-existent characters. This is enabled by default.
color	524288	Color related warnings.
delim	8	Missing or mismatched closing delimiters.
di	256	Use of **di** or **da** without an argument when there is no current diversion.
el	16	Use of the **el** request with no matching **ie** request.
escape	32768	Unrecognized escape sequences. When an unrecognized escape sequence is encountered, the escape character is ignored.
font	131072	Non-existent fonts. This is enabled by default.
ig	262144	Invalid escapes in text ignored with the **ig** request. These are conditions that are errors when they do not occur in ignored text.
input	16384	Invalid input characters.
mac	512	Use of undefined strings, macros and diversions. When an undefined string, macro or diversion is used, that string is automatically defined as empty. So, in most cases, at most one warning will be given for each name.
missing	8192	Requests that are missing non-optional arguments.
number	2	Invalid numeric expressions. This is enabled by default.
range	64	Out of range arguments.
reg	1024	Use of undefined number registers. When an undefined number register is used, that register is automatically defined to have a value of 0. So, in most cases, at most one warning will be given for use of a particular name.
right-brace	4096	Use of **\}** where a number was expected.

scale	32	Meaningless scaling indicators.
space	65536	Missing space between a request or macro and its argument. This warning will be given when an undefined name longer than two characters is encountered, and the first two characters of the name make a defined name. The request or macro will not be invoked. When this warning is given, no macro is automatically defined. This is enabled by default. This warning will never occur in compatibility mode.
syntax	128	Dubious syntax in numeric expressions.
tab	2048	Inappropriate use of a tab character. Either use of a tab character where a number was expected, or use of tab character in an unquoted macro argument.

There are also names that can be used to refer to groups of warnings:

all All warnings except **di**, **mac**, and **reg**. It is intended that this covers all warnings that are useful with traditional macro packages.

w All warnings.

ENVIRONMENT

GROFF_TMAC_PATH
A colon separated list of directories in which to search for macro files. **troff** will scan directories given in the **−M** option before these, and in standard directories (current directory if in unsafe mode, home directory, **c:/progra 1/groff/lib/groff/site-tmac**, **c:/progra 1/groff/share/groff/site-tmac**, **c:/progra 1/groff/share/groff/1.20/tmac**) after these.

GROFF_TYPESETTER
Default device.

GROFF_FONT_PATH
A colon separated list of directories in which to search for the **dev***name* directory. **troff** will scan directories given in the **−F** option before these, and in standard directories (**c:/progra 1/groff/share/groff/site-font**, **c:/progra 1/groff/share/groff/1.20/font**, **/usr/lib/font**) after these.

FILES

c:/progra 1/groff/share/groff/1.20/tmac/troffrc
Initialization file (called before any other macro package).

c:/progra 1/groff/share/groff/1.20/tmac/troffrc-end
Initialization file (called after any other macro package).

c:/progra 1/groff/share/groff/1.20/tmac/*name***.tmac**
c:/progra 1/groff/share/groff/1.20/tmac/tmac.*name*
Macro files

c:/progra 1/groff/share/groff/1.20/font/dev*name***/DESC**
Device description file for device *name*.

c:/progra 1/groff/share/groff/1.20/font/dev*name***/***F*
Font file for font *F* of device *name*.

Note that **troffrc** and **troffrc-end** are neither searched in the current nor in the home directory by default for security reasons (even if the **−U** option is given). Use the **−M** command line option or the **GROFF_TMAC_PATH** environment variable to add these directories to the search path if necessary.

AUTHOR

This document was written by James Clark, with modifications from Werner Lemberg and Bernd Warken

This document is part of *groff*, the GNU roff distribution.

SEE ALSO

groff(1)
The main program of the *groff* system, a wrapper around *troff*.

groff(7)
A description of the *groff* language, including a short but complete reference of all predefined requests, registers, and escapes of plain *groff*. From the command line, this is called by

man 7 groff

groff_diff(7)
The differences of the *groff* language and the *classical troff* language. Currently, this is the most actual document of the *groff* system.

roff(7) An overview over *groff* and other *roff* systems, including pointers to further related documentation.

The *groff info file*, cf. **info**(1), presents all groff documentation within a single document.

GROFF_FONT

NAME

groff_font – format of groff device and font description files

DESCRIPTION

The groff font format is roughly a superset of the ditroff font format. The font files for device *name* are stored in a directory **dev***name*. There are two types of file: a device description file called **DESC** and for each font *F* a font file called *F*. These are text files; unlike the ditroff font format, there is no associated binary format.

DESC file format

The DESC file can contain the following types of line as shown below. Later entries in the file override previous values.

Empty lines are ignored.

charset
: This line and everything following in the file are ignored. It is allowed for the sake of backwards compatibility.

family *fam*
: The default font family is *fam*.

fonts *n F1 F2 F3 … Fn*
: Fonts *F1*, …, *Fn* are mounted in the font positions $m+1, \ldots, m+n$ where *m* is the number of styles. This command may extend over more than one line. A font name of **0** causes no font to be mounted on the corresponding font position.

hor *n*
: The horizontal resolution is *n* machine units.

image_generator *string*
: Needed for **grohtml** only. It specifies the program to generate PNG images from PostScript input. Under GNU/Linux this is usually *gs* but under other systems (notably cygwin) it might be set to another name.

paperlength *n*
: The physical vertical dimension of the output medium in machine units. This isn't used by **troff** itself but by output devices. Deprecated. Use **papersize** instead.

papersize *string*
: Select a paper size. Valid values for *string* are the ISO paper types A0-A7, B0-B7, C0-C7, D0-D7, DL, and the US paper types letter, legal, tabloid, ledger, statement, executive, com10, and monarch. Case is not significant for *string* if it holds predefined paper types. Alternatively, *string* can be a file name (e.g. '/etc/papersize'); if the file can be opened, **groff** reads the first line and tests for the above paper sizes. Finally, *string* can be a custom paper size in the format *length***,***width* (no spaces before and after the comma). Both *length* and *width* must have a unit appended; valid values are 'i' for inches, 'c' for centimeters, 'p' for points, and 'P' for picas. Example: **12c,235p**. An argument which starts with a digit is always treated as a custom paper format. **papersize** sets both the vertical and horizontal dimension of the output medium.

  More than one argument can be specified; **groff** scans from left to right and uses the first valid paper specification.

paperwidth *n*
: The physical horizontal dimension of the output medium in machine units. Deprecated. Use **papersize** instead. This isn't used by **troff** itself but by output devices.

pass_filenames
: Make troff tell the driver the source file name being processed. This is achieved by another tcommand: **F** *filename*.

postpro *program*
: Use *program* as the postprocessor.

prepro *program*
: Call *program* as a preprocessor.

print *program*
: Use *program* as the spooler program for printing. If omitted, the **−l** and **−L** options of **groff** are ignored.

res *n* There are *n* machine units per inch.

sizes *s1 s2 … sn* **0**
: This means that the device has fonts at *s1*, *s2*, …, *sn* scaled points. The list of sizes must be terminated by a **0**. Each *si* can also be a range of sizes *m*−*n*. The list can extend over more than one line.

sizescale *n*
: The scale factor for point sizes. By default this has a value of 1. One *scaled point* is equal to one point / *n*. The arguments to the **unitwidth** and **sizes** commands are given in scaled points.

styles *S1 S2 … Sm*
: The first *m* font positions are associated with styles *S1*, …, *Sm*.

tcommand
: This means that the postprocessor can handle the **t** and **u** output commands.

unicode
: Indicate that the output device supports the complete Unicode repertoire. Useful only for devices which produce *character entities* instead of glyphs.

: If **unicode** is present, no **charset** section is required in the font description files since the Unicode handling built into **groff** is used. However, if there are entries in a **charset** section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

: This is used for **−Tutf8**, **−Thtml**, and **−Txhtml**.

unitwidth *n*
: Quantities in the font files are given in machine units for fonts whose point size is *n* scaled points.

unscaled_charwidths
: Make the font handling module always return unscaled glyph widths. Needed for the **grohtml** device.

use_charnames_in_special
: This command indicates that troff should encode named glyphs inside special commands.

vert *n* The vertical resolution is *n* machine units.

The **res**, **unitwidth**, **fonts**, and **sizes** lines are compulsory. Not all commands in the DESC file are used by **troff** itself; some of the keywords (or even additional ones) are used by postprocessors to store arbitrary information about the device.

Here a list of obsolete keywords which are recognized by **groff** but completely ignored: **spare1**, **spare2**, **biggestfont**.

Font file format

A font file has two sections; empty lines are ignored in both of them.

The first section is a sequence of lines each containing a sequence of blank delimited words; the first word in the line is a key, and subsequent words give a value for that key.

ligatures *lig1 lig2 … lign* [**0**]
: Glyphs *lig1*, *lig2*, …, *lign* are ligatures; possible ligatures are **ff**, **fi**, **fl**, **ffi**, and **ffl**. For backwards compatibility, the list of ligatures may be terminated with a **0.** The list of ligatures may not extend over more than one line.

name *F*
: The name of the font is *F*.

slant *n* The glyphs of the font have a slant of *n* degrees. (Positive means forward.)

spacewidth *n*
: The normal width of a space is *n*.

special The font is *special*; this means that when a glyph is requested that is not present in the current font, it is searched for in any special fonts that are mounted.

Other commands are ignored by **troff** but may be used by postprocessors to store arbitrary information about the font in the font file.

The first section can contain comments which start with the **#** character and extend to the end of a line.

The second section contains one or two subsections. It must contain a *charset* subsection and it may also contain a *kernpairs* subsection. These subsections can appear in any order. Each subsection starts with a word on a line by itself.

The word **charset** starts the charset subsection. The **charset** line is followed by a sequence of lines. Each line gives information for one glyph. A line comprises a number of fields separated by blanks or tabs. The format is

> *name metrics type code* [*entity_name*] [**--** *comment*]

name identifies the glyph: if *name* is a single glyph *c* then it corresponds to the groff input character *c*; if it is of the form \c where c is a single character, then it corresponds to the special character **\\[***c***]**; otherwise it corresponds to the groff input character **\\[***name***]**. If it is exactly two characters *xx* it can be entered as **\\(***xx*. Note that single-letter special characters can't be accessed as \c; the only exception is '\\-' which is identical to '\\[-]'. The name **---** is special and indicates that the glyph is unnamed; such glyphs can only be used by means of the **\\N** escape sequence in **troff**.

The *type* field gives the glyph type:

1	means the glyph has a descender, for example, 'p';
2	means the glyph has an ascender, for example, 'b';
3	means the glyph has both an ascender and a descender, for example, '('.

The *code* field gives the code which the postprocessor uses to print the glyph. The glyph can also be input to groff using this code by means of the **\\N** escape sequence. The code can be any integer. If it starts with a **0** it is interpreted as octal; if it starts with **0x** or **0X** it is intepreted as hexadecimal. Note, however, that the **\\N** escape sequence only accepts a decimal integer.

The *entity_name* field gives an ASCII string identifying the glyph which the postprocessor uses to print that glyph. This field is optional and is currently used by **grops** to build sub-encoding arrays for PS fonts containing more than 256 glyphs. (It has also been used for **grohtml**'s entity names but for efficiency reasons this data is now compiled directly into **grohtml**.)

Anything on the line after the encoding field or '--' are ignored.

The *metrics* field has the form (in one line; it is broken here for the sake of readability):

> *width*[**,***height*[**,***depth*[**,***italic-correction*
> [**,***left-italic-correction*[**,***subscript-correction*]]]]]

There must not be any spaces between these subfields. Missing subfields are assumed to be 0. The subfields are all decimal integers. Since there is no associated binary format, these values are not required to fit into a variable of type **char** as they are in ditroff. The *width* subfields gives the width of the glyph. The *height* subfield gives the height of the glyph (upwards is positive); if a glyph does not extend above the baseline, it should be given a zero height, rather than a negative height. The *depth* subfield gives the depth of the glyph, that is, the distance below the lowest point below the baseline to which the glyph extends (downwards is positive); if a glyph does not extend below above the baseline, it should be given a zero depth, rather than a negative depth. The *italic-correction* subfield gives the amount of space that should be added after the glyph when it is immediately to be followed by a glyph from a roman font. The *left-italic-correction* subfield gives the amount of space that should be added before the glyph when it is immediately to be preceded by a glyph from a roman font. The *subscript-correction* gives the amount of space that should be added after a glyph before adding a subscript. This should be less than the italic correction.

A line in the charset section can also have the format

> *name* **"**

This indicates that *name* is just another name for the glyph mentioned in the preceding line.

The word **kernpairs** starts the kernpairs section. This contains a sequence of lines of the form:

c1 c2 n

This means that when glyph *c1* appears next to glyph *c2* the space between them should be increased by *n*. Most entries in kernpairs section have a negative value for *n*.

FILES

c:/progra 1/groff/share/groff/1.20/font/dev*name***/DESC**
Device description file for device *name*.

c:/progra 1/groff/share/groff/1.20/font/dev*name***/***F*
Font file for font *F* of device *name*.

SEE ALSO

groff_out(5), **troff**(1).

GROFF_OUT

NAME

groff_out – groff intermediate output format

DESCRIPTION

This manual page describes the *intermediate output* format of the GNU **roff**(7) text processing system **groff**(1). This output is produced by a run of the GNU **troff**(1) program. It contains already all device-specific information, but it is not yet fed into a device postprocessor program.

As the GNU *roff* processor **groff**(1) is a wrapper program around **troff** that automatically calls a postprocessor, this output does not show up normally. This is why it is called *intermediate* within the *groff system*. The **groff** program provides the option **-Z** to inhibit postprocessing, such that the produced *intermediate output* is sent to standard output just like calling **troff** manually.

In this document, the term *troff output* describes what is output by the GNU **troff** program, while *intermediate output* refers to the language that is accepted by the parser that prepares this output for the postprocessors. This parser is smarter on whitespace and implements obsolete elements for compatibility, otherwise both formats are the same. Both formats can be viewed directly with **gxditview**(1).

The main purpose of the *intermediate output* concept is to facilitate the development of postprocessors by providing a common programming interface for all devices. It has a language of its own that is completely different from the **groff**(7) language. While the *groff* language is a high-level programming language for text processing, the *intermediate output* language is a kind of low-level assembler language by specifying all positions on the page for writing and drawing.

The pre-*groff roff* versions are denoted as *classical troff*. The *intermediate output* produced by **groff** is fairly readable, while *classical troff* output was hard to understand because of strange habits that are still supported, but not used any longer by *GNU troff*.

LANGUAGE CONCEPTS

During the run of **troff**, the *roff* input is cracked down to the information on what has to be printed at what position on the intended device. So the language of the *intermediate output* format can be quite small. Its only elements are commands with or without arguments. In this document, the term "command" always refers to the *intermediate output* language, never to the *roff* language used for document formatting. There are commands for positioning and text writing, for drawing, and for device controlling.

Separation

Classical troff output had strange requirements on whitespace. The **groff** output parser, however, is smart about whitespace by making it maximally optional. The whitespace characters, i.e., the *tab*, *space*, and *newline* characters, always have a syntactical meaning. They are never printable because spacing within the output is always done by positioning commands.

Any sequence of *space* or *tab* characters is treated as a single *syntactical space*. It separates commands and arguments, but is only required when there would occur a clashing between the command code and the arguments without the space. Most often, this happens when variable length command names, arguments, argument lists, or command clusters meet. Commands and arguments with a known, fixed length need not be separated by *syntactical space*.

A line break is a syntactical element, too. Every command argument can be followed by whitespace, a comment, or a newline character. Thus a *syntactical line break* is defined to consist of optional *syntactical space* that is optionally followed by a comment, and a newline character.

The normal commands, those for positioning and text, consist of a single letter taking a fixed number of arguments. For historical reasons, the parser allows to stack such commands on the same line, but fortunately, in *groff intermediate output*, every command with at least one argument is followed by a line break, thus providing excellent readability.

The other commands — those for drawing and device controlling — have a more complicated structure; some recognize long command names, and some take a variable number of arguments. So all **D** and **x** commands were designed to request a *syntactical line break* after their last argument. Only one command, '**x X**' has an argument that can stretch over several lines, all other commands must have all of their arguments on the same line as the command, i.e., the arguments may not be split by a line break.

Empty lines, i.e., lines containing only space and/or a comment, can occur everywhere. They are just ignored.

Argument Units

Some commands take integer arguments that are assumed to represent values in a measurement unit, but the letter for the corresponding *scale indicator* is not written with the output command arguments; see **groff**(7) and the *groff info file* for more on this topic. Most commands assume the scale indicator **u**, the basic unit of the device, some use **z**, the *scaled point unit* of the device, while others, such as the color commands expect plain integers. Note that these scale indicators are relative to the chosen device. They are defined by the parameters specified in the device's *DESC* file; see **groff_font**(5).

Note that single characters can have the eighth bit set, as can the names of fonts and special characters (this is, glyphs). The names of glyphs and fonts can be of arbitrary length. A glyph that is to be printed will always be in the current font.

A string argument is always terminated by the next whitespace character (space, tab, or newline); an embedded **#** character is regarded as part of the argument, not as the beginning of a comment command. An integer argument is already terminated by the next non-digit character, which then is regarded as the first character of the next argument or command.

Document Parts

A correct *intermediate output* document consists of two parts, the *prologue* and the *body*.

The task of the *prologue* is to set the general device parameters using three exactly specified commands. The *groff prologue* is guaranteed to consist of the following three lines (in that order):

x T *device*
x res *n h v*
x init

with the arguments set as outlined in the section **Device Control Commands**. However, the parser for the *intermediate output* format is able to swallow additional whitespace and comments as well.

The *body* is the main section for processing the document data. Syntactically, it is a sequence of any commands different from the ones used in the *prologue*. Processing is terminated as soon as the first **x stop** command is encountered; the last line of any *groff intermediate output* always contains such a command.

Semantically, the *body* is page oriented. A new page is started by a **p** command. Positioning, writing, and drawing commands are always done within the current page, so they cannot occur before the first **p** command. Absolute positioning (by the **H** and **V** commands) is done relative to the current page, all other positioning is done relative to the current location within this page.

COMMAND REFERENCE

This section describes all *intermediate output* commands, the classical commands as well as the *groff* extensions.

Comment Command

#*anything*⟨**end-of-line**⟩
A comment. Ignore any characters from the **#** character up to the next newline character.

This command is the only possibility for commenting in the *intermediate output*. Each comment can be preceded by arbitrary *syntactical space*; every command can be terminated by a comment.

Simple Commands

The commands in this subsection have a command code consisting of a single character, taking a fixed number of arguments. Most of them are commands for positioning and text writing. These commands are smart about whitespace. Optionally, *syntactical space* can be inserted before, after, and between the command letter and its arguments. All of these commands are stackable, i.e., they can be preceded by other simple commands or followed by arbitrary other commands on the same line. A separating *syntactical space* is only necessary when two integer arguments would clash or if the preceding argument ends with a string argument.

C *xxx*⟨white-space⟩
Print a glyph (special character) named *xxx*. The trailing *syntactical space* or *line break* is necessary to allow glyph names of arbitrary length. The glyph is printed at the current print position; the glyph's size is read from the font file. The print position is not changed.

c *c* Print glyph with single-letter name *c* at the current print position; the glyph's size is read from the font file. The print position is not changed.

f *n* Set font to font number *n* (a non-negative integer).

H *n* Move right to the absolute vertical position *n* (a non-negative integer in basic units **u**) relative to left edge of current page.

h *n* Move *n* (a non-negative integer) basic units **u** horizontally to the right. *[CSTR #54]* allows negative values for *n* also, but *groff* doesn't use this.

m *color-scheme* [*component . . .*]
Set the color for text (glyphs), line drawing, and the outline of graphic objects using different color schemes; the analoguous command for the filling color of graphic objects is **DF**. The color components are specified as integer arguments between 0 and 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by the *groff* escape sequence **\m**. No position changing. These commands are a *groff* extension.

mc *cyan magenta yellow*
Set color using the CMY color scheme, having the 3 color components cyan, magenta, and yellow.

md Set color to the default color value (black in most cases). No component arguments.

mg *gray*
Set color to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

mk *cyan magenta yellow black*
Set color using the CMYK color scheme, having the 4 color components cyan, magenta, yellow, and black.

mr *red green blue*
Set color using the RGB color scheme, having the 3 color components red, green, and blue.

N *n* Print glyph with index *n* (an integer, normally non-negative) of the current font. The print position is not changed. If **−T html** or **−T xhtml** is used, negative values are emitted also to indicate an unbreakable space with given width. For example, **N −193** represents an unbreakable space which has a width of 193 u. This command is a *groff* extension.

n *b a* Inform the device about a line break, but no positioning is done by this command. In *classical troff*, the integer arguments *b* and *a* informed about the space before and after the current line to make the *intermediate output* more human readable without performing any action. In *groff*, they are just ignored, but they must be provided for compatibility reasons.

p *n* Begin a new page in the outprint. The page number is set to *n*. This page is completely independent of pages formerly processed even if those have the same page number. The vertical position on the outprint is automatically set to 0. All positioning, writing, and drawing is always done relative to a page, so a **p** command must be issued before any of these commands.

s *n* Set point size to *n* scaled points (this is unit **z** in GNU **troff**). *Classical troff* used the unit *points* (**p**) instead; see section **COMPATIBILITY**.

t *xyz . . .*⟨white-space⟩
t *xyz . . . dummy-arg*⟨white-space⟩
Print a word, i.e., a sequence of glyphs with single-letter names *x*, *y*, *z*, etc., terminated by a space character or a line break; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). The first glyph should be printed at the current position, the current horizontal position should then be increased by the width of the first glyph, and so on for each glyph. The widths of the glyph are read from the font file, scaled for the current point size, and rounded to a multiple of the horizontal resolution. Special characters (glyphs with names longer than a single letter) cannot be printed using this command; use the **C** command for those glyphs. This command is a *groff* extension; it is only used for devices whose *DESC* file contains the **tcommand** keyword; see **groff_font**(5).

u *n xyz*...⟨white-space⟩
: Print word with track kerning. This is the same as the **t** command except that after printing each glyph, the current horizontal position is increased by the sum of the width of that glyph and *n* (an integer in basic units **u**). This command is a *groff* extension; it is only used for devices whose *DESC* file contains the **tcommand** keyword; see **groff_font**(5).

V *n*
: Move down to the absolute vertical position *n* (a non-negative integer in basic units **u**) relative to upper edge of current page.

v *n*
: Move *n* basic units **u** down (*n* is a non-negative integer). *[CSTR #54]* allows negative values for *n* also, but *groff* doesn't use this.

w
: Informs about a paddable whitespace to increase readability. The spacing itself must be performed explicitly by a move command.

Graphics Commands

Each graphics or drawing command in the *intermediate output* starts with the letter **D** followed by one or two characters that specify a subcommand; this is followed by a fixed or variable number of integer arguments that are separated by a single space character. A **D** command may not be followed by another command on the same line (apart from a comment), so each **D** command is terminated by a *syntactical line break*.

troff output follows the classical spacing rules (no space between command and subcommand, all arguments are preceded by a single space character), but the parser allows optional space between the command letters and makes the space before the first argument optional. As usual, each space can be any sequence of tab and space characters.

Some graphics commands can take a variable number of arguments. In this case, they are integers representing a size measured in basic units **u**. The *h* arguments stand for horizontal distances where positive means right, negative left. The *v* arguments stand for vertical distances where positive means down, negative up. All these distances are offsets relative to the current location.

Unless indicated otherwise, each graphics command directly corresponds to a similar *groff* **\D** escape sequence; see **groff**(7).

Unknown **D** commands are assumed to be device-specific. Its arguments are parsed as strings; the whole information is then sent to the postprocessor.

In the following command reference, the syntax element ⟨*line-break*⟩ means a *syntactical line break* as defined in section **Separation**.

D h_1 v_1 h_2 v_2 ... h_n v_n ⟨line-break⟩
: Draw B-spline from current position to offset (h_1, v_1), then to offset (h_2, v_2) if given, etc., up to (h_n, v_n). This command takes a variable number of argument pairs; the current position is moved to the terminal point of the drawn curve.

Da h_1 v_1 h_2 v_2 ⟨line-break⟩
: Draw arc from current position to $(h_1, v_1) + (h_2, v_2)$ with center at (h_1, v_1); then move the current position to the final point of the arc.

DC *d* ⟨line-break⟩
DC *d dummy-arg* ⟨line-break⟩
: Draw a solid circle using the current fill color with diameter *d* (integer in basic units **u**) with leftmost point at the current position; then move the current position to the rightmost point of the circle. An optional second integer argument is ignored (this allows to the formatter to generate an even number of arguments). This command is a *groff* extension.

Dc *d* ⟨line-break⟩
: Draw circle line with diameter *d* (integer in basic units **u**) with leftmost point at the current position; then move the current position to the rightmost point of the circle.

DE *h v* ⟨line-break⟩
: Draw a solid ellipse in the current fill color with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units **u**) with the leftmost point at the current position; then move to the rightmost point of the ellipse. This command is a *groff* extension.

De *h v* ⟨line-break⟩
: Draw an outlined ellipse with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units **u**) with the leftmost point at current position; then move to the rightmost point of the ellipse.

DF *color-scheme* [*component*...] ⟨line-break⟩
: Set fill color for solid drawing objects using different color schemes; the analoguous command for setting the color of text, line graphics, and the outline of graphic objects is **m**. The color components are specified as integer arguments between 0 and 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by the *groff* escape sequences **\D'F** ...**'** and **\M** (with no other corresponding graphics commands). No position changing. This command is a *groff* extension.

  DFc *cyan magenta yellow* ⟨line-break⟩
  : Set fill color for solid drawing objects using the CMY color scheme, having the 3 color components cyan, magenta, and yellow.

  DFd ⟨line-break⟩
  : Set fill color for solid drawing objects to the default fill color value (black in most cases). No component arguments.

  DFg *gray* ⟨line-break⟩
  : Set fill color for solid drawing objects to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

  DFk *cyan magenta yellow black* ⟨line-break⟩
  : Set fill color for solid drawing objects using the CMYK color scheme, having the 4 color components cyan, magenta, yellow, and black.

  DFr *red green blue* ⟨line-break⟩
  : Set fill color for solid drawing objects using the RGB color scheme, having the 3 color components red, green, and blue.

Df *n* ⟨line-break⟩
: The argument *n* must be an integer in the range -32767 to 32767.

  $0 \le n \le 1000$
  : Set the color for filling solid drawing objects to a shade of gray, where 0 corresponds to solid white, 1000 (the default) to solid black, and values inbetween to intermediate shades of gray; this is obsoleted by command **DFg**.

  $n < 0$ or $n > 1000$
  : Set the filling color to the color that is currently being used for the text and the outline, see command **m**. For example, the command sequence

  ```
  mg 0 0 65536 Df -1
  ```

  sets all colors to blue.

  No position changing. This command is a *groff* extension.

Dl *h v* ⟨line-break⟩
: Draw line from current position to offset (*h*, *v*) (integers in basic units **u**); then set current position to the end of the drawn line.

Dp $h_1\ v_1\ h_2\ v_2 \ldots h_n\ v_n$ ⟨line-break⟩
: Draw a polygon line from current position to offset (h_1, v_1), from there to offset (h_2, v_2), etc., up to offset (h_n, v_n), and from there back to the starting position. For historical reasons, the position is changed by adding the sum of all arguments with odd index to the actual horizontal position and the even ones to the vertical position. Although this doesn't make sense it is kept for compatibility. This command is a *groff* extension.

DP $h_1\ v_1\ h_2\ v_2 \ldots h_n\ v_n$ ⟨line-break⟩
: The same macro as the corresponding **Dp** command with the same arguments, but draws a solid polygon in the current fill color rather than an outlined polygon. The position is changed in the same way as with **Dp**. This command is a *groff* extension.

Dt *n* ⟨line-break⟩

Set the current line thickness to *n* (an integer in basic units **u**) if $n > 0$; if $n = 0$ select the smallest available line thickness; if $n < 0$ set the line thickness proportional to the point size (this is the default before the first **Dt** command was specified). For historical reasons, the horizontal position is changed by adding the argument to the actual horizontal position, while the vertical position is not changed. Although this doesn't make sense it is kept for compatibility. This command is a *groff* extension.

Device Control Commands

Each device control command starts with the letter **x** followed by a space character (optional or arbitrary space/tab in *groff*) and a subcommand letter or word; each argument (if any) must be preceded by a *syntactical space*. All **x** commands are terminated by a *syntactical line break*; no device control command can be followed by another command on the same line (except a comment).

The subcommand is basically a single letter, but to increase readability, it can be written as a word, i.e., an arbitrary sequence of characters terminated by the next tab, space, or newline character. All characters of the subcommand word but the first are simply ignored. For example, **troff** outputs the initialization command **x i** as **x init** and the resolution command **x r** as **x res**. But writings like **x i_like_groff** and **x roff_is_groff** are accepted as well to mean the same commands.

In the following, the syntax element ⟨*line-break*⟩ means a *syntactical line break* as defined in section **Separation**.

xF *name* ⟨line-break⟩

(*Filename* control command)
Use *name* as the intended name for the current file in error reports. This is useful for remembering the original file name when **groff** uses an internal piping mechanism. The input file is not changed by this command. This command is a *groff* extension.

xf *n s* ⟨line-break⟩

(*font* control command)
Mount font position *n* (a non-negative integer) with font named *s* (a text word), cf. **groff_font**(5).

xH *n* ⟨line-break⟩

(*Height* control command)
Set character height to *n* (a positive integer in scaled points **z**). *Classical troff* used the unit points (**p**) instead; see section **COMPATIBILITY**.

xi ⟨line-break⟩

(*init* control command)
Initialize device. This is the third command of the *prologue*.

xp ⟨line-break⟩

(*pause* control command)
Parsed but ignored. The classical documentation reads *pause device, can be restarted*.

xr *n h v* ⟨line-break⟩

(*resolution* control command)
Resolution is *n*, while *h* is the minimal horizontal motion, and *v* the minimal vertical motion possible with this device; all arguments are positive integers in basic units **u** per inch. This is the second command of the *prologue*.

xS *n* ⟨line-break⟩

(*Slant* control command)
Set slant to *n* degrees (an integer in basic units **u**).

xs ⟨line-break⟩

(*stop* control command)
Terminates the processing of the current file; issued as the last command of any *intermediate troff output*.

xt ⟨line-break⟩

(*trailer* control command)

Generate trailer information, if any. In **groff**, this is actually just ignored.

xT *xxx* ⟨line-break⟩
: (*Typesetter* control command)
Set name of device to word *xxx*, a sequence of characters ended by the next whitespace character. The possible device names coincide with those from the groff **−T** option. This is the first command of the *prologue*.

xu *n* ⟨line-break⟩
: (*underline* control command)
Configure underlining of spaces. If *n* is 1, start underlining of spaces; if *n* is 0, stop underlining of spaces. This is needed for the **cu** request in **nroff** mode and is ignored otherwise. This command is a *groff* extension.

xX *anything* ⟨line-break⟩
: (*X-escape* control command)
Send string *anything* uninterpreted to the device. If the line following this command starts with a **+** character this line is interpreted as a continuation line in the following sense. The **+** is ignored, but a newline character is sent instead to the device, the rest of the line is sent uninterpreted. The same applies to all following lines until the first character of a line is not a **+** character. This command is generated by the *groff* escape sequence **\X**. The line-continuing feature is a *groff* extension.

Obsolete Command

In *classical troff* output, emitting a single glyph was mostly done by a very strange command that combined a horizontal move and the printing of a glyph. It didn't have a command code, but is represented by a 3-character argument consisting of exactly 2 digits and a character.

ddc Move right *dd* (exactly two decimal digits) basic units **u**, then print glyph with single-letter name *c*.

In *groff*, arbitrary *syntactical space* around and within this command is allowed to be added. Only when a preceding command on the same line ends with an argument of variable length a separating space is obligatory. In *classical troff*, large clusters of these and other commands were used, mostly without spaces; this made such output almost unreadable.

For modern high-resolution devices, this command does not make sense because the width of the glyphs can become much larger than two decimal digits. In **groff**, this is only used for the devices **X75**, **X75-12**, **X100**, and **X100-12**. For other devices, the commands **t** and **u** provide a better functionality.

POSTPROCESSING

The *roff* postprocessors are programs that have the task to translate the *intermediate output* into actions that are sent to a device. A device can be some piece of hardware such as a printer, or a software file format suitable for graphical or text processing. The *groff* system provides powerful means that make the programming of such postprocessors an easy task.

There is a library function that parses the *intermediate output* and sends the information obtained to the device via methods of a class with a common interface for each device. So a *groff* postprocessor must only redefine the methods of this class. For details, see the reference in section **FILES**.

EXAMPLES

This section presents the *intermediate output* generated from the same input for three different devices. The input is the sentence *hell world* fed into **groff** on the command line.

• High-resolution device *ps*

shell> echo "hell world" | groff -Z -T ps

```
x T ps
x res 72000 1 1
x init
p1
x font 5 TR
f5
s10000
```

```
V12000
H72000
thell
wh2500
tw
H96620
torld
n12000 0
x trailer
V792000
x stop
```

This output can be fed into the postprocessor **grops**(1) to get its representation as a PostScript file.

- Low-resolution device *latin1*

  This is similar to the high-resolution device except that the positioning is done at a minor scale. Some comments (lines starting with #) were added for clarification; they were not generated by the formatter.

  shell> "hell world" | groff -Z -T latin1

  ```
  # prologue
  x T latin1
  x res 240 24 40
  x init
  # begin a new page
  p1
  # font setup
  x font 1 R
  f1
  s10
  # initial positioning on the page
  V40
  H0
  # write text ‘hell’
  thell
  # inform about a space, and do it by a horizontal jump
  wh24
  # write text ‘world’
  tworld
  # announce line break, but do nothing because ...
  n40 0
  # ... the end of the document has been reached
  x trailer
  V2640
  x stop
  ```

This output can be fed into the postprocessor **grotty**(1) to get a formatted text document.

- Classical style output

  As a computer monitor has a very low resolution compared to modern printers the *intermediate output* for the X devices can use the jump-and-write command with its 2-digit displacements.

  shell> "hell world" | groff -Z -T X100

  ```
  x T X100
  x res 100 1 1
  x init
  p1
  x font 5 TR
  f5
  s10
  V16
  ```

```
H100
# write text with old-style jump-and-write command
ch07e07l03lw06w11o07r05l03dh7
n16 0
x trailer
V1100
x stop
```

This output can be fed into the postprocessor **xditview**(1x) or **gxditview**(1) for displaying in X.

Due to the obsolete jump-and-write command, the text clusters in the classical output are almost unreadable.

COMPATIBILITY

The *intermediate output* language of the *classical troff* was first documented in *[CSTR #97]*. The *groff intermediate output* format is compatible with this specification except for the following features.

- The classical quasi device independence is not yet implemented.
- The old hardware was very different from what we use today. So the *groff* devices are also fundamentally different from the ones in *classical troff*. For example, the classical PostScript device was called *post* and had a resolution of 720 units per inch, while *groff*'s *ps* device has a resolution of 72000 units per inch. Maybe, by implementing some rescaling mechanism similar to the classical quasi device independence, these could be integrated into modern *groff*.
- The B-spline command **D** is correctly handled by the *intermediate output* parser, but the drawing routines aren't implemented in some of the postprocessor programs.
- The argument of the commands **s** and **x H** has the implicit unit scaled point **z** in *groff*, while *classical troff* had point (**p**). This isn't an incompatibility, but a compatible extension, for both units coincide for all devices without a *sizescale* parameter, including all classical and the *groff* text devices. The few *groff* devices with a sizescale parameter either did not exist, had a different name, or seem to have had a different resolution. So conflicts with classical devices are very unlikely.
- The position changing after the commands **Dp**, **DP**, and **Dt** is illogical, but as old versions of groff used this feature it is kept for compatibility reasons.

The differences between *groff* and *classical troff* are documented in **groff_diff**(7).

FILES

c:/progra 1/groff/share/groff/1.20/font/dev*name***/DESC**
: Device description file for device *name*.

⟨*groff-source-dir*⟩**/src/libs/libdriver/input.cpp**
: Defines the parser and postprocessor for the *intermediate output*. It is located relative to the top directory of the *groff* source tree. This parser is the definitive specification of the *groff intermediate output* format.

SEE ALSO

A reference like **groff**(7) refers to a manual page; here **groff** in section *7* of the man-page documentation system. To read the example, look up section 7 in your desktop help system or call from the shell prompt

shell> man 7 groff

For more details, see **man**(1).

groff(1)
: option **-Z** and further readings on groff.

groff(7)
: for details of the *groff* language such as numerical units and escape sequences.

groff_font(5)
: for details on the device scaling parameters of the **DESC** file.

troff(1) generates the device-independent intermediate output.

roff(7) for historical aspects and the general structure of roff systems.

groff_diff(7)
The differences between the intermediate output in groff and classical troff.

gxditview(1)
Viewer for the *intermediate output*.

grodvi(1), **grohtml**(1), **grolbp**(1), **grolj4**(1), **grops**(1), **grotty**(1)
the groff postprocessor programs.

For a treatment of all aspects of the groff system within a single document, see the *groff info file*. It can be read within the integrated help systems, within **emacs**(1) or from the shell prompt by
shell> info groff

The *classical troff output language* is described in two AT&T Bell Labs CSTR documents available on-line at Bell Labs CSTR site

[CSTR #97]
A Typesetter-independent TROFF by *Brian Kernighan* is the original and most comprehensive documentation on the output language; see CSTR #97

[CSTR #54]
The 1992 revision of the *Nroff/Troff User's Manual* by *J. F. Ossanna* and *Brian Kernighan* isn't as comprehensive as *[CSTR #97]* regarding the output language; see CSTR #54

AUTHORS

This document is part of *groff*, the GNU *roff* distribution. It is based on a former version − published under the GPL − that described only parts of the *groff* extensions of the output language. It was rewritten in 2002 by Bernd Warken and is maintained by Werner Lemberg

GROFF_TMAC

NAME

groff_tmac – macro files in the roff typesetting system

DESCRIPTION

The **roff**(7) type-setting system provides a set of macro packages suitable for special kinds of documents. Each macro package stores its macros and definitions in a file called the package's **tmac file**. The name is deduced from '**T***roff* **MAC***ros*'.

The tmac files are normal roff source documents, except that they usually contain only definitions and setup commands, but no text. All tmac files are kept in a single or a small number of directories, the **tmac** directories.

GROFF MACRO PACKAGES

groff provides all classical macro packages, some more full packages, and some secondary packages for special purposes. Note that it is not possible to use multiple primary macro packages at the same time; saying e.g.

```
sh# groff −m man −m ms foo
```

or

```
sh# groff −m man foo −m ms bar
```

fails. Exception to this is the use of man pages written with either the **mdoc** or the **man** macro package. See below the description of the **andoc.tmac** file.

Man Pages

man This is the classical macro package for UNIX manual pages (man pages); it is quite handy and easy to use; see **groff_man**(7).

doc
mdoc An alternative macro package for man pages mainly used in BSD systems; it provides many new features, but it is not the standard for man pages; see **groff_mdoc**(7).

andoc
mandoc
Use this file in case you don't know whether the **man** macros or the **mdoc** package should be used. Multiple man pages (in either format) can be handled.

Full Packages

The packages in this section provide a complete set of macros for writing documents of any kind, up to whole books. They are similar in functionality; it is a matter of taste which one to use.

me The classical *me* macro package; see **groff_me**(7).

mm The semi-classical *mm* macro package; see **groff_mm**(7).

mom The new *mom* macro package, only available in groff. As this is not based on other packages, it can be freely designed. So it is expected to become quite a nice, modern macro package. See **groff_mom**(7).

ms The classical *ms* macro package; see **groff_ms**(7).

Language-specific Packages

cs This file adds support for Czech localization, including the main macro packages (me, mom, mm, and ms).

Note that **cs.tmac** sets the input encoding to latin-2.

de
den German localization support, including the main macro packages (me, mom, mm, and ms).

de.tmac selects hyphenation patterns for traditional orthography, and **den.tmac** does the same for the new orthography ('Rechtschreibreform'). It should be used as the last macro package on the command line.

fr This file adds support for French localization, including the main macro packages (me, mom, mm, and ms). Example:

```
sh# groff -ms -mfr foo.ms > foo.ps
```

Note that **fr.tmac** sets the input encoding to latin-9 to get proper support of the 'oe' ligature.

sv Swedish localization support, including the me, mom, and ms macro packages. Note that Swedish for the mm macros is handled separately; see **groff_mmse**(7). It should be used as the last macro package on the command line.

Input Encodings

latin1
latin2
latin5
latin9 Various input encodings supported directly by groff. Normally, this macro is loaded at the very beginning of a document or specified as the first macro argument on the command line. **roff** loads latin1 by default at start-up. Note that these macro packages don't work on EBCDIC hosts.

cp1047 Encoding support for EBCDIC. On those platforms it is loaded automatically at start-up. Due to different character ranges used in **roff** it doesn't work on architectures which are based on ASCII.

Note that it can happen that some input encoding characters are not available for a particular output device. For example, saying

```
groff -Tlatin1 -mlatin9 ...
```

fails if you use the Euro character in the input. Usually, this limitation is present only for devices which have a limited set of output glyphs (**−Tascii**, **−Tlatin1**); for other devices it is usually sufficient to install proper fonts which contain the necessary glyphs.

Special Packages

The macro packages in this section are not intended for stand-alone usage, but can be used to add special functionality to any other macro package or to plain groff.

60bit Provide some macros for addition, multiplication, and division of 60bit integers (allowing safe multiplication of 30bit integers, for example).

ec Switch to the EC and TC font families. To be used with **grodvi**(1) – this man page also gives more details of how to use it.

papersize

This macro file is already loaded at start-up by **troff** so it isn't necessary to call it explicitly. It provides an interface to set the paper size on the command line with the option **−dpaper=***size*. Possible values for *size* are the same as the predefined **papersize** values in the DESC file (only lowercase; see **groff_font**(5) for more) except **a7**-**d7**. An appended **l** (ell) character denotes landscape orientation. Examples: **a4**, **c3l**, **letterl**.

Most output drivers need additional command line switches **−p** and **−l** to override the default paper length and orientation as set in the driver specific DESC file. For example, use the following for PS output on A4 paper in landscape orientation:

```
sh# groff −Tps −dpaper=a4l −P−pa4 −P−l −ms foo.ms > foo.ps
```

pic This file provides proper definitions for the macros **PS** and **PE**, needed for the **pic**(1) preprocessor. They center each picture. Use it only if your macro package doesn't provide proper definitions for those two macros (actually, most of them already do).

pspic A single macro is provided in this file, **PSPIC**, to include a PostScript graphic in a document. The following output devices support inclusion of PS images: **−Tps**, **−Tdvi**, **−Thtml**, and **−Txhtml**; for all other devices the image is replaced with a hollow rectangle of the same size. This macro file is already loaded at start-up by **troff** so it isn't necessary to call it explicitly.

Syntax:

.PSPIC [**−L** | **−R** | **−C** | **−I** *n*] *file* [*width* [*height*]]

file is the name of the PostScript file; *width* and *height* give the desired width and height of the image. If neither a *width* nor a *height* argument is specified, the image's natural width (as given in the file's bounding box) or the current line length is used as the width, whatever is smaller. The *width* and *height* arguments may have scaling indicators attached; the default scaling indicator is **i**. This macro scales the graphic uniformly in the x and y directions so that it is no more than *width* wide and *height* high. Option **−C** centers the graphic horizontally, which is the default. The **−L** and **−R** options cause the graphic to be left-aligned and right-aligned, respectively. The **−I** option causes the graphic to be indented by *n* (default scaling indicator is **m**).

For use of **.PSPIC** within a diversion it is recommended to extend it with the following code, assuring that the diversion's width completely covers the image's width.

```
.am PSPIC .  vpt 0 \h'(\\n[ps-offset]u + \\n[ps-deswid]u)' .  sp -1 .  vpt 1 ..
```

ptx A single macro is provided in this file, **xx**, for formatting permuted index entries as produces by the GNU **ptx**(1) program. In case you need a different formatting, copy the macro into your document and adapt it to your needs.

trace Use this for tracing macro calls. It is only useful for debugging. See **groff_trace**(7)**.**

tty-char
Overrides the definition of standard troff characters and some groff characters for TTY devices. The optical appearance is intentionally inferior compared to that of normal TTY formatting to allow processing with critical equipment.

www Additions of elements known from the HTML format, as used in the internet (World Wide Web) pages; this includes URL links and mail addresses; see **groff_www**(7).

NAMING

Classical roff systems were designed before the conventions of the modern C **getopt**(3) call evolved, and used a naming scheme for macro packages that looks odd to modern eyes. Macro packages were always included with the option **−m**; when this option was directly followed by its argument without an intervening space, this looked like a long option preceded by a single minus — a sensation in the computer stone age. To make this invocation form work, classical troff macro packages used names that started with the letter 'm', which was omitted in the naming of the macro file.

For example, the macro package for the man pages was called *man*, while its macro file *tmac.an*. So it could be activated by the argument *an* to option **−m**, or **−man** for short.

For similar reasons, macro packages that did not start with an 'm' had a leading 'm' added in the documentation and in speech; for example, the package corresponding to *tmac.doc* was called *mdoc* in the documentation, although a more suitable name would be *doc*. For, when omitting the space between the option and its argument, the command line option for activating this package reads **−mdoc**.

To cope with all situations, actual versions of **groff**(1) are smart about both naming schemes by providing two macro files for the inflicted macro packages; one with a leading 'm' the other one without it. So in *groff*, the *man* macro package may be specified as on of the following four methods:

sh# groff −m man *sh#* groff −man *sh#* groff −mman *sh#* groff −m an

Recent packages that do not start with 'm' do not use an additional 'm' in the documentation. For example, the *www* macro package may be specified only as one of the two methods:

sh# groff −m www *sh#* groff −mwww

Obviously, variants like *−mmwww* would not make much sense.

A second strange feature of classical troff was to name macro files in the form **tmac.***name*. In modern operating systems, the type of a file is specified as a postfix, the file name extension. Again, groff copes with this situation by searching both *anything***.tmac** and **tmac.***anything* if only *anything* is specified.

The easiest way to find out which macro packages are available on a system is to check the man page **groff**(1), or the contents of the *tmac* directories.

In *groff*, most macro packages are described in man pages called **groff_***name*(7), with a leading 'm' for the classical packages.

INCLUSION

There are several ways to use a macro package in a document. The classical way is to specify the troff/groff option **−m** *name* at run-time; this makes the contents of the macro package *name* available. In groff, the file *name***.tmac** is searched within the tmac path; if not found, **tmac.***name* is searched for instead.

Alternatively, it is also possible to include a macro file by adding the request **.so** *filename* into the document; the argument must be the full file name of an existing file, possibly with the directory where it is kept. In groff, this was improved by the similar request **.mso** *package*, which added searching in the tmac path, just like option **−m** does.

Note that in order to resolve the **.so** and **.mso** requests, the roff preprocessor **soelim**(1) must be called if the files to be included need preprocessing. This can be done either directly by a pipeline on the command line or by using the troff/groff option **−s**. *man* calls soelim automatically.

For example, suppose a macro file is stored as

c:/progra 1/groff/share/groff/1.20/tmac/macros.tmac

and is used in some document called *docu.roff*.

At run-time, the formatter call for this is

sh# groff −m macros docu.roff

To include the macro file directly in the document either

```
.mso macros.tmac
```

is used or

```
.so c:/progra 1/groff/share/groff/1.20/tmac/macros.tmac
```

In both cases, the formatter should be called with option **−s** to invoke **soelim**.

sh# groff −s docu.roff

If you want to write your own groff macro file, call it *whatever***.tmac** and put it in some directory of the tmac path, see section **FILES**. Then documents can include it with the **.mso** request or the option **−m**.

WRITING MACROS

A **roff**(7) document is a text file that is enriched by predefined formatting constructs, such as requests, escape sequences, strings, numeric registers, and macros from a macro package. These elements are described in **roff**(7).

To give a document a personal style, it is most useful to extend the existing elements by defining some macros for repeating tasks; the best place for this is near the beginning of the document or in a separate file.

Macros without arguments are just like strings. But the full power of macros reveals when arguments are passed with a macro call. Within the macro definition, the arguments are available as the escape sequences **\$1**, . . ., **\$9**, **\$[**. . .**]**, **\$***, and **\$@**, the name under which the macro was called is in **\$0**, and the number of arguments is in register **\n[.$]**; see **groff**(7).

Copy-in Mode

The phase when groff reads a macro is called *copy-in mode* or *copy mode* in roff-talk. This is comparable to the C preprocessing phase during the development of a program written in the C language.

In this phase, groff interprets all backslashes; that means that all escape sequences in the macro body are interpreted and replaced by their value. For constant expressions, this is wanted, but strings and registers that might change between calls of the macro must be protected from being evaluated. This is most easily done by doubling the backslash that introduces the escape sequence. This doubling is most important for the positional parameters. For example, to print information on the arguments that were passed to the macro to the terminal, define a macro named '.print_args', say.

```
.ds midpart was called with .de print_args . tm \f[I]\\$0\f[] \*[midpart] \\n[.$] arguments:
. tm \\$* ..
```

When calling this macro by

```
.print_args arg1 arg2
```

the following text is printed to the terminal:

`print_args` was called with the following 2 arguments: arg1 arg2

Let's analyze each backslash in the macro definition. As the positional parameters and the number of arguments change with each call of the macro their leading backslash must be doubled, which results in \\*$** and \\*[.$]*. The same applies to the macro name because it could be called with an alias name, so \\*$0*.

On the other hand, *midpart* is a constant string, it does not change, so no doubling for **[midpart]*. The *f* escape sequences are predefined groff elements for setting the font within the text. Of course, this behavior does not change, so no doubling with *f[I]* and *f[]*.

Draft Mode

Writing groff macros is easy when the escaping mechanism is temporarily disabled. In groff, this is done by enclosing the macro definition(s) into a pair of **.eo** and **.ec** requests. Then the body in the macro definition is just like a normal part of the document — text enhanced by calls of requests, macros, strings, registers, etc. For example, the code above can be written in a simpler way by

.eo .ds midpart was called with .de print_args . tm \f[I]\$0\f[] *[midpart] \n[.$] arguments: . tm \$* .. .ec

Unfortunately, draft mode cannot be used universally. Although it is good enough for defining normal macros, draft mode fails with advanced applications, such as indirectly defined strings, registers, etc. An optimal way is to define and test all macros in draft mode and then do the backslash doubling as a final step; do not forget to remove the *.eo* request.

Tips for Macro Definitions

- Start every line with a dot, for example, by using the groff request **.nop** for text lines, or write your own macro that handles also text lines with a leading dot.

  .de Text . if (\\n[.$] == 0) \ . return . nop \)\\$*\) ..

- Write a comment macro that works both for copy-in and draft mode; for as escaping is off in draft mode, trouble might occur when normal comments are used. For example, the following macro just ignores its arguments, so it acts like a comment line:

  .de c .. .c This is like a comment line.

- In long macro definitions, make ample use of comment lines or almost-empty lines (this is, lines which have a leading dot and nothing else) for a better structuring.
- To increase readability, use groff's indentation facility for requests and macro calls (arbitrary whitespace after the leading dot).

Diversions

Diversions can be used to implement quite advanced programming constructs. They are comparable to pointers to large data structures in the C programming language, but their usage is quite different.

In their simplest form, diversions are multi-line strings, but they get their power when diversions are used dynamically within macros. The (formatted) information stored in a diversion can be retrieved by calling the diversion just like a macro.

Most of the problems arising with diversions can be avoided if you remain aware of the fact that diversions always store complete lines. If diversions are used when the line buffer has not been flushed, strange results are produced; not knowing this, many people get desperate about diversions. To ensure that a diversion works, line breaks should be added at the right places. To be on the secure side, enclose everything that has to do with diversions into a pair of line breaks; for example, by explicitly using **.br** requests. This rule should be applied to diversion definition, both inside and outside, and to all calls of diversions. This is a bit of overkill, but it works nicely.

[If you really need diversions which should ignore the current partial line, use environments to save the current partial line and/or use the **.box** request.]

The most powerful feature using diversions is to start a diversion within a macro definition and end it within another macro. Then everything between each call of this macro pair is stored within the diversion and can be manipulated from within the macros.

FILES

All macro names must be named *name***.tmac** to fully use the tmac mechanism. **tmac.***name* as with classical packages is possible as well, but deprecated.

The macro files are kept in the *tmac directories*; a colon separated list of these constitutes the *tmac path*.

The search sequence for macro files is (in that order):

- the directories specified with troff/groff's **−M** command line option
- the directories given in the **$GROFF_TMAC_PATH** environment variable
- the current directory (only if in unsafe mode, which is enabled by the **−U** command line switch)
- the home directory
- a platform-specific directory, being

  c:/progra 1/groff/lib/groff/site-tmac

  in this installation
- a site-specific (platform-independent) directory, being

  c:/progra 1/groff/share/groff/site-tmac

  in this installation
- the main tmac directory, being

  c:/progra 1/groff/share/groff/1.20/tmac

  in this installation

ENVIRONMENT

$GROFF_TMAC_PATH
A colon separated list of additional tmac directories in which to search for macro files. See the previous section for a detailed description.

AUTHOR

This document is part of *groff*, the GNU roff distribution. It was written by Bernd Warken it is maintained by Werner Lemberg

SEE ALSO

A complete reference for all parts of the groff system is found in the groff **info**(1) file.

groff(1)
an overview of the groff system.

groff_man(7),
groff_mdoc(7),
groff_me(7),
groff_mm(7),
groff_mom(7),
groff_ms(7),
groff_trace(7),
groff_www(7).
the groff tmac macro packages.

groff(7)
the groff language.

The Filesystem Hierarchy Standard is available at the FHS web site

LJ4_FONT

NAME

lj4_font − groff fonts for use with devlj4

DESCRIPTION

Nominally, all Hewlett-Packard LaserJet 4–series and newer printers have the same internal fonts: 45 scalable fonts and one bitmapped Lineprinter font. The scalable fonts are available in sizes between 0.25 point and 999.75 points, in 0.25-point increments; the Lineprinter font is available only in 8.5-point size.

The LaserJet font files included with **groff** assume that all printers since the LaserJet 4 are identical. There are some differences between fonts in the earlier and more recent printers, however. The LaserJet 4 printer used Agfa Intellifont technology for 35 of the internal scalable fonts; the remaining 10 scalable fonts were TrueType. Beginning with the LaserJet 4000–series printers introduced in 1997, all scalable internal fonts have been TrueType. The number of printable glyphs differs slightly between Intellifont and TrueType fonts (generally, the TrueType fonts include more glyphs), and there are some minor differences in glyph metrics. Differences among printer models are described in the *PCL 5 Comparison Guide* and the *PCL 5 Comparison Guide Addendum* (for printers introduced since approximately 2001).

LaserJet printers reference a glyph by a combination of a 256-glyph symbol set and an index within that symbol set. Many glyphs appear in more than one symbol set; all combinations of symbol set and index that reference the same glyph are equivalent. For each glyph, **hpftodit**(1) searches a list of symbol sets, and selects the first set that contains the glyph. The printing code generated by **hpftodit**(1) is an integer that encodes a numerical value for the symbol set in the high byte(s), and the index in the low byte. See **groff_font**(5) for a complete description of the font file format; symbol sets are described in greater detail in the *PCL 5 Printer Language Technical Reference Manual*.

Two of the scalable fonts, Symbol and Wingdings, are bound to 256-glyph symbol sets; the remaining scalable fonts, as well as the Lineprinter font, support numerous symbol sets, sufficient to enable printing of more than 600 glyphs.

The metrics generated by **hpftodit**(1) assume that the DESC file contains values of 1200 for res and 6350 for unitwidth (or any combination (e.g., 2400 and 3175) for which res × unitwidth = 7 620 000). Although HP PCL 5 LaserJet printers support an internal resolution of 7200 units per inch, they use a 16-bit signed integer for cursor positioning; if **devlj4** is to support U.S. ledger paper (11″ × 17″), the maximum usable resolution is 32 767 / 17, or 1927, units per inch, which rounds down to 1200 units per inch. If the largest required paper size is less (e.g., 8.5″ × 11″ or A5), a greater resolution (and lesser unitwidth) can be specified.

LIMITATIONS

Font metrics for Intellifont fonts were provided by Tagged Font Metric (TFM) files originally developed by Agfa/Compugraphic. The TFM files provided for these fonts supported 600+ glyphs and contained extensive lists of kern pairs.

To accommodate developers who had become accustomed to TFM files, HP also provided TFM files for the 10 TrueType fonts included in the LaserJet 4. The TFM files for TrueType fonts generally included less information than the Intellifont TFMs, supporting fewer glyphs, and in most cases, providing no kerning information. By the time the LaserJet 4000 printer was introduced, most developers had migrated to other means of obtaining font metrics, and support for new TFM files was very limited. The TFM files provided for the TrueType fonts in the LaserJet 4000 support only the Latin 2 (ISO 8859-2) symbol set, and include no kerning information; consequently, they are of little value for any but the most rudimentary documents.

Because the Intellifont TFM files contain considerably more information, they generally are preferable to the TrueType TFM files even for use with the TrueType fonts in the newer printers. The metrics for the TrueType fonts are very close, though not identical, to those for the earlier Intellifont fonts of the same names. Although most output using the Intellifont metrics with the newer printers is quite acceptable, a few glyphs may fail to print as expected. The differences in glyph metrics may be particularly noticeable with composite parentheses, brackets, and braces used by **eqn**(1). A script, located in **c:/progra 1/groff/share/groff/1.20/font/devlj4/generate**, can be used to adjust the metrics for these glyphs in the special font S for use with printers that have all TrueType fonts.

At the time HP last supported TFM files, only Version 1 of the Unicode standard was available.

Consequently, many glyphs lacking assigned code points were assigned by HP to the Private Use Area (PUA). Later versions of the Unicode standard included code points outside the PUA for many of these glyphs. The HP-supplied TrueType TFM files use the PUA assignments; TFM files generated from more recent TrueType font files require the later Unicode values to access the same glyphs. Consequently, two different mapping files may be required: one for the HP-supplied TFM files, and one for more recent TFM files.

FILES

c:/progra 1/groff/share/groff/1.20/font/devlj4/DESC
Device description file.

c:/progra 1/groff/share/groff/1.20/font/devlj4/*F*
Font description file for font *F*.

SEE ALSO

groff(1), **groff_diff**(1), **hpftodit**(1), **grolj4**(1), **groff_font**(5)

DITROFF

NAME

ditroff – classical device independent roff

DESCRIPTION

The name *ditroff* once marked a development level of the *troff* text processing system. In actual **roff**(7) systems, the name *troff* is used as a synonym for *ditroff*.

The first roff system was written by Joe Ossanna around 1973. It supported only two output devices, the **nroff** program produced text oriented tty output, while the **troff** program generated graphical output for exactly one output device, the Wang *Graphic Systems CAT* typesetter.

In 1979, Brian Kernighan rewrote troff to support more devices by creating an intermediate output format for troff that can be fed into postprocessor programs which actually do the printout on the device. Kernighan's version marks what is known as *classical troff* today. In order to distinguish it from Ossanna's original mono-device version, it was called *ditroff* (*d*evice *i*ndependent *troff) on some systems, though this naming isn't mentioned in the classical documentation.*

Today, any existing roff system is based on Kernighan's multi-device troff. The distinction between *troff* and *ditroff* isn't necessary any longer, for each modern *troff* provides already the complete functionality of *ditroff*. On most systems, the name *troff* is used to denote *ditroff*.

The easiest way to use ditroff is the GNU roff system, *groff*. The **groff**(1) program is a wrapper around *(di)troff* that automatically handles postprocessing.

SEE ALSO

[CSTR #54]
: The 1992 revision of the *Nroff/Troff User's Manual* by *J. F. Ossanna* and *Brian Kernighan*, see Bell Labs CSTR #54

[CSTR #97]
: *A Typesetter-independent TROFF* by *Brian Kernighan* is the original documentation of the first multi-device troff (*ditroff*), see Bell Labs CSTR #97

roff(7) This document gives details on the history and concepts of roff.

troff(1) The actual implementation of *ditroff*.

groff(1)
: The GNU roff program and pointers to all documentation around groff.

groff_out(5)
: The groff version of the intermediate output language, the basis for multi-devicing.

AUTHORS

This document is part of *groff*, the GNU roff distribution. It was written by Bernd Warken and is maintained by Werner Lemberg

GROFF

NAME

groff − a short reference for the GNU roff language

DESCRIPTION

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

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

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

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

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

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

Control lines represent commands, optionally with arguments. They have the following syntax. The leading control character can be followed by a command name; arguments, if any, are separated by spaces (but not tab characters) from the command name and among themselves, for example,

```
.command_name arg1 arg2
```

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

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

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

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

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

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

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

GROFF ELEMENTS

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

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

The user can define her own formatting commands using the **de** request. These commands are called **macros**, but they are used exactly like requests. Macro packages are pre-defined sets of macros written in the groff language. A user's possibilities to create escape sequences herself is very limited, only special characters can be mapped.

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

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

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

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

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

Glyphs are visual representation forms of **characters**. In groff, the distinction between those two elements is not always obvious (and a full discussion is beyond the scope of this man page). A first approximation is that glyphs have a specific size and colour and are taken from a specific font; they can't be modified any more – characters are the input, and glyphs are the output. As soon as an output line has been generated, it no longer contains characters but glyphs. In this man page, we use either 'glyph' or 'character', whatever is more appropriate.

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

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

CONTROL CHARACTERS

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

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

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

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

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

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

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

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

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

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

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

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

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

NUMERICAL EXPRESSIONS

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

The *roff* language defines the following scaling indicators.

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

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

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

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

e1>?*e2*	The maximum of *e1* and *e2*.
e1<?*e2*	The minimum of *e1* and *e2*.
(*c*;*e*)	Evaluate *e* using *c* as the default scaling indicator.

For details see the groff info file.

CONDITIONS

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

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

REQUESTS

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

Most requests take one or more arguments. The arguments are separated by space characters (no tabs!); there is no inherent limit for their length or number.

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

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

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

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

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

Request Short Reference

`.` Empty line, ignored. Useful for structuring documents.

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

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

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

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

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

`.aln` *alias register*
  Create alias name for *register*.

`.als` *alias object*
  Create alias name for request, string, macro, or diversion *object*.

`.am` *macro*
  Append to *macro* until **..** is encountered.

`.am` *macro end*
  Append to *macro* until **.***end* is called.

.aml *macro*
: Same as **.am** but with compatibility mode switched off during macro expansion.

.aml *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 register *macro* until **..** is encountered.

.ami *macro end*
: Append to a macro indirectly. *macro* and *end* are string registers whose contents are interpolated for the macro name and the end macro, respectively.

.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 *stringvar anything*
: Append *anything* to *stringvar*.

.as1 *stringvar anything*
: Same as **.as** but with compatibility mode switched off during string expansion.

.asciify *diversion*
: Unformat ASCII characters, spaces, and some escape sequences in *diversion*.

.backtrace
: Print a backtrace of the input on stderr.

.bd *font N*
: Embolden *font* by *N*-1 units.

.bd *S font N*
: Embolden Special Font *S* when current font is *font*.

.blm Unset the blank line macro.

.blm *macro*
: Set the blank line macro to *macro*.

.box End current diversion.

.box *macro*
: Divert to *macro*, omitting a partially filled line.

.boxa End current diversion.

.boxa *macro*
: Divert and append to *macro*, omitting a partially filled line.

.bp Eject current page and begin new page.

.bp ±*N* Eject current page; next page number ±*N*.

.br Line break.

.brp Break and spread output line. Same as **\p**.

.break Break out of a while loop.

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

.c2 *c* Set no-break control character to *c*.

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

.cc *c* Set control character to *c*.

.ce Center the next input line.

.ce *N* Center following *N* input lines.

.cf *filename*
: Copy contents of file *filename* unprocessed to stdout or to the diversion.

.cflags *mode c1 c2 . . .*
: Treat characters *c1*, *c2*, . . . according to *mode* number.

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

.char *c anything*
: Define entity *c* as string *anything*.

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

.close *stream*
: Close the *stream*.

`.color` Enable colors.

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

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

`.da` End current diversion.

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

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

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

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

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

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

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

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

`.dei1` *macro*
: Same as **.dei** but with compatibility mode switched off during macro expansion.

`.dei1` *macro end*
: Same as **.dei** but with compatibility mode switched off during macro expansion.

`.device` *anything*
: Write *anything* to the intermediate output as a device control function.

`.devicem` *name*
: Write contents of macro or string *name* uninterpreted to the intermediate output as a device control function.

`.di` End current diversion.

`.di` *macro*
: Divert to *macro*.

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

`.ds` *stringvar anything*
: Set *stringvar* to *anything*.

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

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

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

`.ec` *c* Set escape character to *c*.

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

.ecs Save current escape character.

.el *anything*
: Else part for if-else (**ie**) request.

.em *macro*
The *macro* is run after the end of input.

.eo Turn off escape character mechanism.

.ev Switch to previous environment and pop it off the stack.

.ev *env* Push down environment number or name *env* to the stack and switch to it.

.evc *env* Copy the contents of environment *env* to the current environment. No pushing or popping.

.ex Exit from roff processing.

.fam Return to previous font family.

.fam *name*
: Set the current font family to *name*.

.fc Disable field mechanism.

.fc *a* Set field delimiter to *a* and pad glyph to space.

.fc *a b* Set field delimiter to *a* and pad glyph to *b*.

.fchar *c anything*
: Define fallback character (or glyph) *c* as string *anything*.

.fcolor Set fill color to previous fill color.

.fcolor *c*
: Set fill color to *c*.

.fi Fill output lines.

.fl Flush output buffer.

.fp *n font*
: Mount *font* on position *n*.

.fp *n internal external*
: Mount font with long *external* name to short *internal* name on position *n*.

.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 Return to previous font. Same as **\f[]** or **\fP**.

.ft *font* Change to font name or number *font*; same as **\f[***font***]** escape sequence.

.ftr *font1 font2*
: Translate *font1* to *font2*.

.fzoom *font*
: Don't magnify *font*.

.fzoom *font zoom*
: Set zoom factor for *font* (in multiples of 1/1000th).

.gcolor Set glyph color to previous glyph color.

.gcolor *c*
: Set glyph color to *c*.

.hc Remove additional hyphenation indicator character.

.hc *c* Set up additional hyphenation indicator character *c*.

.hcode *c1 code1 c2 code2 . . .*
: Set the hyphenation code of character *c1* to *code1*, that of *c2* to *code2*, etc.

.hla *lang*
: Set the current hyphenation language to *lang*.

.hlm *n* Set the maximum number of consecutive hyphenated lines to *n*.

.hpf *file* Read hyphenation patterns from *file*.

.hpfa *file*
: Append hyphenation patterns from *file*.

.hpfcode *a b c d . . .*
: Set input mapping for **.hpf**.

.hw *words*	List of *words* with exceptional hyphenation.
.hy *N*	Switch to hyphenation mode *N*.
.hym *n*	Set the hyphenation margin to *n* (default scaling indicator **m**).
.hys *n*	Set the hyphenation space to *n*.
.ie *cond anything*	If *cond* then *anything* else goto **.el**.
.if *cond anything*	If *cond* then *anything*; otherwise do nothing.
.ig	Ignore text until **..** is encountered.
.ig *end*	Ignore text until **.***end* is called.
.in	Change to previous indentation value.
.in ±*N*	Change indentation according to ±*N* (default scaling indicator **m**).
.it *N trap*	Set an input-line count trap for the next *N* lines.
.itc *N trap*	Same as **.it** but count lines interrupted with **\c** as one line.
.kern	Enable pairwise kerning.
.kern *n*	If *n* is zero, disable pairwise kerning, otherwise enable it.
.lc	Remove leader repetition glyph.
.lc *c*	Set leader repetition glyph to *c*.
.length *register anything*	Write the length of the string *anything* to *register*.
.linetabs	Enable line-tabs mode (i.e., calculate tab positions relative to output line).
.linetabs *n*	If *n* is zero, disable line-tabs mode, otherwise enable it.
.lf *N*	Set input line number to *N*.
.lf *N file*	Set input line number to *N* and filename to *file*.
.lg *N*	Ligature mode on if *N*>0.
.ll	Change to previous line length.
.ll ±*N*	Set line length according to ±*N* (default length 6.5**i**, default scaling indicator **m**).
.ls	Change to the previous value of additional intra-line skip.
.ls *N*	Set additional intra-line skip value to *N*, i.e., *N*-1 blank lines are inserted after each text output line.
.lt ±*N*	Length of title (default scaling indicator **m**).
.mc	Margin glyph off.
.mc *c*	Print glyph *c* after each text line at actual distance from right margin.
.mc *c N*	Set margin glyph to *c* and distance to *N* from right margin (default scaling indicator **m**).
.mk *register*	Mark current vertical position in *register*.
.mso *file*	The same as **.so** except that *file* is searched in the tmac directories.
.na	No output-line adjusting.
.ne	Need a one-line vertical space.
.ne *N*	Need *N* vertical space (default scaling indicator **v**).
.nf	No filling or adjusting of output-lines.
.nh	No hyphenation.
.nm	Number mode off.
.nm ±*N* [*M* [*S* [*I*]]]	In line number mode, set number, multiple, spacing, and indentation.
.nn	Do not number next line.
.nn *N*	Do not number next *N* lines.
.nop *anything*	Always process *anything*.
.nr *register* ±*N* [*M*]	Define or modify *register* using ±*N* with auto-increment *M*.

`.nroff`	Make the built-in conditions **n** true and **t** false.
`.ns`	Turn on no-space mode.
`.nx`	Immediately jump to end of current file.
`.nx` *filename*	Immediately continue processing with file *file*.
`.open` *stream filename*	Open *filename* for writing and associate the stream named *stream* with it.
`.opena` *stream filename*	Like **.open** but append to it.
`.os`	Output vertical distance that was saved by the **sv** request.
`.output` *string*	Emit *string* directly to intermediate output, allowing leading whitespace if *string* starts with `"` (which is stripped off).
`.pc`	Reset page number character to '`%`'.
`.pc` *c*	Page number character.
`.pev`	Print the current environment and each defined environment state to stderr.
`.pi` *program*	Pipe output to *program* (nroff only).
`.pl`	Set page length to default 11`i`. The current page length is stored in register **.p**.
`.pl` ±*N*	Change page length to ±*N* (default scaling indicator `v`).
`.pm`	Print macro names and sizes (number of blocks of 128 bytes).
`.pm` *t*	Print only total of sizes of macros (number of 128 bytes blocks).
`.pn` ±*N*	Next page number *N*.
`.pnr`	Print the names and contents of all currently defined number registers on stderr.
`.po`	Change to previous page offset. The current page offset is available in register **.o**.
`.po` ±*N*	Page offset *N*.
`.ps`	Return to previous point size.
`.ps` ±*N*	Point size; same as **\s[**±*N***]**.
`.psbb` *filename*	Get the bounding box of a PostScript image *filename*.
`.pso` *command*	This behaves like the **so** request except that input comes from the standard output of *command*.
`.ptr`	Print the names and positions of all traps (not including input line traps and diversion traps) on stderr.
`.pvs`	Change to previous post-vertical line spacing.
`.pvs` ±*N*	Change post-vertical line spacing according to ±*N* (default scaling indicator `p`).
`.rchar` *c1 c2* . . .	Remove the definitions of entities *c1*, *c2*, . . .
`.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 definitions of entities *c1*, *c2*, . . . for font *f*.
`.rj` *n*	Right justify the next *n* input lines.
`.rm` *name*	Remove request, macro, or string *name*.
`.rn` *old new*	Rename request, macro, or string *old* to *new*.
`.rnn` *reg1 reg2*	Rename register *reg1* to *reg2*.
`.rr` *register*	Remove *register*.
`.rs`	Restore spacing; turn no-space mode off.

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

`.schar` *c anything*
Define global fallback character (or glyph) *c* as string *anything*.

`.shc` Reset soft hyphen glyph to **\(hy**.

`.shc` *c* Set the soft hyphen glyph to *c*.

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

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

`.so` *filename*
Include source file.

`.sp` Skip one line vertically.

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

`.special`
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 without changing its value.

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

`.ss` *N* Set space glyph size to *N*/12 of the space width in the current font.

`.ss` *N M* Set space glyph size to *N*/12 and sentence space size set to *M*/12 of the space width in the current font.

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

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

`.sv` Save 1 v of vertical space.

`.sv` *N* Save the vertical distance *N* for later output with **os** request (default scaling indicator **v**).

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

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

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

`.tc` Remove tab repetition glyph.

`.tc` *c* Set tab repetition glyph to *c*.

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

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

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

`.tm` *anything*
Print *anything* on stdout.

`.tm1` *anything*
Print *anything* on stdout, allowing leading whitespace if *anything* starts with `"` (which is stripped off).

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

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

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

`.trin` *abcd…*
This is the same as the **tr** request except that the **asciify** request uses the character code (if

any) before the character translation.

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

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

.uf *font* Set underline font to *font* (to be switched to by **.ul**).

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

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

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

.vs Change to previous vertical base line spacing.

.vs ±*N* Set vertical base line spacing to ±*N* (default scaling indicator **p**).

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

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

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

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

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

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

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

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

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

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

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

ESCAPE SEQUENCES

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

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

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

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

While one-character escape sequences are mainly used for in-line functions and system related tasks, the two-letter names following the \(construct are glyphs predefined by the roff system; these are called 'Special Characters' in the classical documentation. Escapes sequences of the form **\[***name***]** denote glyphs too.

Single-Character Escapes

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

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

*****_s_ The string stored in the string variable with one-character name _s_.

***(**_st_ The string stored in the string variable with two-character name _st_.

***[**_string_**]**
The string stored in the string variable with name _string_ (with arbitrary length).

***[**_stringvar arg1 arg2 . . ._**]**
The string stored in the string variable with arbitrarily long name _stringvar_, taking _arg1_, _arg2_, . . . as arguments.

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

\$_x_ Macro or string argument with one-digit number _x_ in the range 1 to 9.

\$(_xy_ Macro or string argument with two-digit number _xy_ (larger than zero).

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

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

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

\$^ In a macro, the representation of all parameters as if they were an argument to the **ds** request.

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

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

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

\- The − (minus) sign in the current font.

_ The same as **\(ul**, the underline character.

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

\% Default optional hyphenation character.

\! Transparent line indicator.

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

****_space_ Unpaddable space size space glyph (no line break).

\0 Digit-width space.

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

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

\& Non-printable, zero-width glyph.

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

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

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

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

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

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

\{ Begin conditional input.

\} End conditional input.

\(_sc_ A glyph with two-character name _sc_; see section **Special Characters**.

\[_name_**]**
A glyph with name _name_ (of arbitrary length).

\[_comp1 comp2 . . ._**]**
A composite glyph with components _comp1_, _comp2_, . . .

`\a` Non-interpreted leader character.

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

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

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

`\c` Interrupt text processing.

`\C'`*glyph*`'`
: The glyph called *glyph*; same as **\[***glyph***]**, but compatible to other roff versions.

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

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

`\e` Printable version of the current escape character.

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

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

`\fP` Switch back to previous font.

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

`\f[`*font*`]`
: Change to font with arbitrarily long name or number expression *font*.

`\f[]` Switch back to previous font.

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

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

`\F[`*fam*`]`
: Change to font family with arbitrarily long name *fam*.

`\F[]` Switch back to previous font family.

`\g`*r* Return format of register with one-character name *r* suitable for **af** request.

`\g(`*rg* Return format of register with two-character name *rg* suitable for **af** request.

`\g[`*reg*`]`
: Return format of register with arbitrarily long name *reg* suitable for **af** request.

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

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

`\k`*r* Mark horizontal input place in one-character register *r*.

`\k(`*rg* Mark horizontal input place in two-character register *rg*.

`\k[`*reg*`]`
: Mark horizontal input place in register with arbitrarily long name *reg*.

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

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

`\m`*c* Change to color with one-character name *c*.

`\m(`*cl* Change to color with two-character name *cl*.

`\m[`*color*`]`
: Change to color with arbitrarily long name *color*.

`\m[]` Switch back to previous color.

`\M`*c* Change filling color for closed drawn objects to color with one-character name *c*.

`\M(`*cl* Change filling color for closed drawn objects to color with two-character name *cl*.

`\M[`*color*`]`
: Change filling color for closed drawn objects to color with arbitrarily long name *color*.

`\M[]` Switch to previous fill color.

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

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

`\n[`*reg*`]`
: The numerical value stored in the register variable with arbitrarily long name *reg*.

`\N'`*n*`'` Typeset the glyph with index *n* in the current font. No special fonts are searched. Useful for adding (named) entities to a document using the **char** request and friends.

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

\O0 Disable glyph output. Mainly for internal use.

\O1 Enable glyph output. Mainly for internal use.

\p Break and spread output line.

\r Reverse 1 em vertical motion (reverse line in nroff).

\R'*name* ±*n***'**
: The same as **.nr** *name* ±*n*.

\s±*N* Set/increase/decrease the point size to/by *N* scaled points; *N* is a one-digit number in the range 1 to 9. Same as **ps** request.

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

\s[±*N***]**
\s±**[***N***]**
\s'±*N***'**
\s±**'***N***'**
: Set/increase/decrease the point size to/by *N* scaled points. Same as **ps** request.

\S'*N***'** Slant output by *N* degrees.

\t Non-interpreted horizontal tab.

\u Reverse (up) 1/2 em vertical motion (1/2 line in nroff).

\v'*N***'** Local vertical motion; move down *N* (up if negative).

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

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

\V[*env***]**
: The contents of the environment variable with arbitrarily long name *env*.

\w'*string***'**
: The width of the glyph sequence *string*.

\x'*N***'** Extra line-space function (negative before, positive after).

\X'*string***'**
: Output *string* as device control function.

\Y*n* Output string variable or macro with one-character name *n* uninterpreted as device control function.

\Y(*nm* Output string variable or macro with two-character name *nm* uninterpreted as device control function.

\Y[*name***]**
: Output string variable or macro with arbitrarily long name *name* uninterpreted as device control function.

\z*c* Print *c* with zero width (without spacing).

\Z'*anything***'**
: Print *anything* and then restore the horizontal and vertical position; *anything* may not contain tabs or leaders.

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

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

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

Special Characters

[Note: 'Special Characters' is a misnomer; those entities are (output) glyphs, not (input) characters.]

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

\(bu Bullet sign

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

Strings

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

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

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

REGISTERS

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

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

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

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

Read-only Registers

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

\n[`.$`]	Number of arguments in the current macro or string.
\n[`.a`]	Post-line extra line-space most recently utilized using **\x**.
\n[`.A`]	Set to 1 in **troff** if option **`-A`** is used; always 1 in **nroff**.
\n[`.br`]	Within a macro, set to 1 if macro called with the 'normal' control character, and to 0 otherwise.
\n[`.c`]	Current input line number.
\n[`.C`]	1 if compatibility mode is in effect, 0 otherwise.
\n[`.cdp`]	The depth of the last glyph added to the current environment. It is positive if the glyph extends below the baseline.
\n[`.ce`]	The number of lines remaining to be centered, as set by the **ce** request.
\n[`.cht`]	The height of the last glyph added to the current environment. It is positive if the glyph extends above the baseline.
\n[`.color`]	1 if colors are enabled, 0 otherwise.
\n[`.csk`]	The skew of the last glyph added to the current environment. The skew of a glyph 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**]	Current vertical place in current diversion; equal to register **nl**.
\n[**.ev**]	The name or number of the current environment (string-valued).
\n[**.f**]	Current font number.
\n[**.fam**]	The current font family (string-valued).
\n[**.fn**]	The current (internal) real font name (string-valued).
\n[**.fp**]	The number of the next free font position.
\n[**.g**]	Always 1 in GNU troff. Macros should use it to test if running under groff.
\n[**.h**]	Text base-line high-water mark on current page or diversion.
\n[**.H**]	Available horizontal resolution in basic units.
\n[**.height**]	The current font height as set with **\H**.
\n[**.hla**]	The current hyphenation language as set by the **hla** request.
\n[**.hlc**]	The number of immediately preceding consecutive hyphenated lines.
\n[**.hlm**]	The maximum allowed number of consecutive hyphenated lines, as set by the **hlm** request.
\n[**.hy**]	The current hyphenation flags (as set by the **hy** request).
\n[**.hym**]	The current hyphenation margin (as set by the **hym** request).
\n[**.hys**]	The current hyphenation space (as set by the **hys** request).
\n[**.i**]	Current indentation.
\n[**.in**]	The indentation that applies to the current output line.
\n[**.int**]	Positive if last output line contains **\c**.
\n[**.kern**]	1 if pairwise kerning is enabled, 0 otherwise.
\n[**.l**]	Current line length.
\n[**.lg**]	The current ligature mode (as set by the **lg** request).
\n[**.linetabs**]	The current line-tabs mode (as set by the **linetabs** request).
\n[**.ll**]	The line length that applies to the current output line.
\n[**.lt**]	The title length (as set by the **lt** request).
\n[**.m**]	The current drawing color (string-valued).
\n[**.M**]	The current background color (string-valued).
\n[**.n**]	Length of text portion on previous output line.
\n[**.ne**]	The amount of space that was needed in the last **ne** request that caused a trap to be sprung. Useful in conjunction with register **.trunc**.
\n[**.ns**]	1 if in no-space mode, 0 otherwise.
\n[**.o**]	Current page offset.
\n[**.p**]	Current page length.
\n[**.pe**]	1 during page ejection, 0 otherwise.
\n[**.pn**]	The number of the next page: either the value set by a **pn** request, or the number of the current page plus 1.
\n[**.ps**]	The current point size in scaled points.
\n[**.psr**]	The last-requested point size in scaled points.
\n[**.pvs**]	The current post-vertical line spacing.
\n[**.rj**]	The number of lines to be right-justified as set by the **rj** request.
\n[**.s**]	Current point size as a decimal fraction.
\n[**.slant**]	The slant of the current font as set with **\S**.
\n[**.sr**]	The last requested point size in points as a decimal fraction (string-valued).
\n[**.ss**]	The value of the parameters set by the first argument of the **ss** request.
\n[**.sss**]	The value of the parameters set by the second argument of the **ss** request.
\n[**.sty**]	The current font style (string-valued).
\n[**.t**]	Vertical distance to the next trap.
\n[**.T**]	Set to 1 if option **-T** is used.
\n[**.tabs**]	A string representation of the current tab settings suitable for use as an argument to the **ta** request.
\n[**.trunc**]	The amount of vertical space truncated by the most recently sprung vertical position trap, or, if the trap was sprung by a **ne** request, minus the amount of vertical motion produced by **.ne**. Useful in conjunction with the register **.ne**.
\n[**.u**]	Equal to 1 in fill mode and 0 in no-fill mode.

\n[**.U**]	Equal to 1 in safer mode and 0 in unsafe mode.
\n[**.v**]	Current vertical line spacing.
\n[**.V**]	Available vertical resolution in basic units.
\n[**.vpt**]	1 if vertical position traps are enabled, 0 otherwise.
\n[**.w**]	Width of previous glyph.
\n[**.warn**]	The sum of the number codes of the currently enabled warnings.
\n[**.x**]	The major version number.
\n[**.y**]	The minor version number.
\n[**.Y**]	The revision number of groff.
\n[**.z**]	Name of current diversion.
\n[**.zoom**]	Zoom factor for current font (in multiples of 1/1000th; zero if no magnification).

Writable Registers

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

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

COMPATIBILITY

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

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

BUGS

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

AUTHORS

This document is part of *groff*, the GNU roff distribution. It was written by Bernd Warken it is maintained by Werner Lemberg

SEE ALSO

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

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

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

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

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

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

[CSTR #54]
: Nroff/Troff User’s Manual by Ossanna & Kernighan — the bible for classical troff.

GROFF_CHAR

NAME

groff_char – groff glyph names

DESCRIPTION

This manual page lists the standard **groff** glyph names and the default input mapping, latin1. The glyphs in this document look different depending on which output device was chosen (with option **–T** for the **man**(1) program or the roff formatter). Glyphs not available for the device that is being used to print or view this manual page are marked with '(N/A)'; the device currently used is 'ps'.

In the actual version, **groff** provides only 8-bit characters for direct input and named entities for further glyphs. On ASCII platforms, input character codes in the range 0 to 127 (decimal) represent the usual 7-bit ASCII characters, while codes between 127 and 255 are interpreted as the corresponding characters in the *latin1* (*ISO-8859-1*) code set by default. This mapping is contained in the file `latin1.tmac` and can be changed by loading a different input encoding. Note that some of the input characters are reserved by **groff**, either for internal use or for special input purposes. On EBCDIC platforms, only code page *cp1047* is supported (which contains the same characters as latin1; the input encoding file is called `cp1047.tmac`). Again, some input characters are reserved for internal and special purposes.

All roff systems provide the concept of named glyphs. In traditional roff systems, only names of length 2 were used, while groff also provides support for longer names. It is strongly suggested that only named glyphs are used for all character representations outside of the printable 7-bit ASCII range.

Some of the predefined groff escape sequences (with names of length 1) also produce single glyphs; these exist for historical reasons or are printable versions of syntactical characters. They include '`\\`', '`\´`', '`\‘`', '`\-`', '`\.`', and '`\e`'; see **groff**(7).

In groff, all of these different types of characters and glyphs can be tested positively with the '`.if c`' conditional.

REFERENCE

In this section, the glyphs in groff are specified in tabular form. The meaning of the columns is as follows.

Output shows how the glyph is printed for the current device; although this can have quite a different shape on other devices, it always represents the same glyph.

Input name
specifies how the glyph is input either directly by a key on the keyboard, or by a groff escape sequence.

Input code
applies to glyphs which can be input with a single character, and gives the ISO latin1 decimal code of that input character. Note that this code is equivalent to the lowest 256 Unicode characters, including 7-bit ASCII in the range 0 to 127.

PostScript name
gives the usual PostScript name of the glyph.

Unicode decomposed
is the glyph name used in composite glyph names.

7-bit Character Codes 32-126

These are the basic glyphs having 7-bit ASCII code values assigned. They are identical to the printable characters of the character standards ISO-8859-1 (latin1) and Unicode (range *Basic Latin*). The glyph names used in composite glyph names are 'u0020' up to 'u007E'.

Note that input characters in the range 0–31 and character 127 are *not* printable characters. Most of them are invalid input characters for **groff** anyway, and the valid ones have special meaning. For EBCDIC, the printable characters are in the range 66–255.

48−57 Decimal digits 0 to 9 (print as themselves).

65−90 Upper case letters A−Z (print as themselves).

97−122 Lower case letters a−z (print as themselves).

Most of the remaining characters not in the just described ranges print as themselves; the only exceptions are the following characters:

` the ISO latin1 ‘Grave Accent’ (code 96) prints as ‘, a left single quotation mark; the original character can be obtained with ‘\`’.

' the ISO latin1 ‘Apostrophe’ (code 39) prints as ’, a right single quotation mark; the original character can be obtained with ‘\(aq’.

- the ISO latin1 ‘Hyphen, Minus Sign’ (code 45) prints as a hyphen; a minus sign can be obtained with ‘\-’.

the ISO latin1 ‘Tilde’ (code 126) is reduced in size to be usable as a diacritic; a larger glyph can be obtained with ‘\(ti’.

ˆ the ISO latin1 ‘Circumflex Accent’ (code 94) is reduced in size to be usable as a diacritic; a larger glyph can be obtained with ‘\(ha’.

Output	*Input name*	*Input code*	*PostScript name*	*Unicode decomposed*	*Notes*
!	!	33	exclam	u0021	
"	"	34	quotedbl	u0022	
#	#	35	numbersign	u0023	
$	$	36	dollar	u0024	
%	%	37	percent	u0025	
&	&	38	ampersand	u0026	
’	'	39	quoteright	u0027	
(	(	40	parenleft	u0028	
)	)	41	parenright	u0029	
*	*	42	asterisk	u002A	
+	+	43	plus	u002B	
,	,	44	comma	u002C	
-	−	45	hyphen	u2010	
.	.	46	period	u002E	
/	/	47	slash	u002F	
:	:	58	colon	u003A	
;	;	59	semicolon	u003B	
<	<	60	less	u003C	
=	=	61	equal	u003D	
>	>	62	greater	u003E	
?	?	63	question	u003F	
@	@	64	at	u0040	
[	[	91	bracketleft	u005B	
\	\	92	backslash	u005C	
]	]	93	bracketright	u005D	
ˆ	ˆ	94	circumflex	u005E	circumflex accent
_	_	95	underscore	u005F	
‘	`	96	quoteleft	u0060	
{	{	123	braceleft	u007B	
|	|	124	bar	u007C	
}	}	125	braceright	u007D	
		126	tilde	u007E	tilde accent

8-bit Character Codes 160 to 255

They are interpreted as printable characters according to the *latin1* (*ISO-8859-1*) code set, being identical to the Unicode range *Latin-1 Supplement*.

Input characters in range 128-159 (on non-EBCDIC hosts) are not printable characters.

160 the ISO latin1 *no-break space* is mapped to ‘\ ’, the stretchable space character.

173 the soft hyphen control character. **groff** never uses this character for output (thus it is omitted in the table below); the input character 173 is mapped onto ‘\%’.

The remaining ranges (161–172, 174–255) are printable characters that print as themselves. Although they can be specified directly with the keyboard on systems with a latin1 code page, it is better to use their glyph names; see next section.

Output	*Input name*	*Input code*	*PostScript name*	*Unicode decomposed*	*Notes*
¡	¡	161	exclamdown	u00A1	inverted exclamation mark
¢	¢	162	cent	u00A2	
£	£	163	sterling	u00A3	
¤	¤	164	currency	u00A4	
¥	¥	165	yen	u00A5	
¦	¦	166	brokenbar	u00A6	
§	§	167	section	u00A7	
¨	¨	168	dieresis	u00A8	
©	©	169	copyright	u00A9	
ª	ª	170	ordfeminine	u00AA	
«	«	171	guillemotleft	u00AB	
¬	¬	172	logicalnot	u00AC	
®	®	174	registered	u00AE	
¯	¯	175	macron	u00AF	
°	°	176	degree	u00B0	
±	±	177	plusminus	u00B1	
²	²	178	twosuperior	u00B2	
³	³	179	threesuperior	u00B3	
´	´	180	acute	u00B4	acute accent
µ	µ	181	mu	u00B5	micro sign
¶	¶	182	paragraph	u00B6	
·	·	183	periodcentered	u00B7	
¸	¸	184	cedilla	u00B8	
¹	¹	185	onesuperior	u00B9	
º	º	186	ordmasculine	u00BA	
»	»	187	guillemotright	u00BB	
¼	¼	188	onequarter	u00BC	
½	½	189	onehalf	u00BD	
¾	¾	190	threequarters	u00BE	
¿	¿	191	questiondown	u00BF	
À	À	192	Agrave	u0041_0300	
Á	Á	193	Aacute	u0041_0301	
Â	Â	194	Acircumflex	u0041_0302	
Ã	Ã	195	Atilde	u0041_0303	
Ä	Ä	196	Adieresis	u0041_0308	
Å	Å	197	Aring	u0041_030A	
Æ	Æ	198	AE	u00C6	
Ç	Ç	199	Ccedilla	u0043_0327	
È	È	200	Egrave	u0045_0300	
É	É	201	Eacute	u0045_0301	
Ê	Ê	202	Ecircumflex	u0045_0302	
Ë	Ë	203	Edieresis	u0045_0308	
Ì	Ì	204	Igrave	u0049_0300	
Í	Í	205	Iacute	u0049_0301	
Î	Î	206	Icircumflex	u0049_0302	
Ï	Ï	207	Idieresis	u0049_0308	
Ð	Ð	208	Eth	u00D0	
Ñ	Ñ	209	Ntilde	u004E_0303	
Ò	Ò	210	Ograve	u004F_0300	
Ó	Ó	211	Oacute	u004F_0301	

Output	*Input name*	*Input code*	*PostScript name*	*Unicode decomposed*	*Notes*
Ô	Ô	212	Ocircumflex	u004F_0302	
Õ	Õ	213	Otilde	u004F_0303	
Ö	Ö	214	Odieresis	u004F_0308	
×	×	215	multiply	u00D7	
Ø	Ø	216	Oslash	u00D8	
Ù	Ù	217	Ugrave	u0055_0300	
Ú	Ú	218	Uacute	u0055_0301	
Û	Û	219	Ucircumflex	u0055_0302	
Ü	Ü	220	Udieresis	u0055_0308	
Ý	Ý	221	Yacute	u0059_0301	
Þ	Þ	222	Thorn	u00DE	
ß	ß	223	germandbls	u00DF	
à	à	224	agrave	u0061_0300	
á	á	225	aacute	u0061_0301	
â	â	226	acircumflex	u0061_0302	
ã	ã	227	atilde	u0061_0303	
ä	ä	228	adieresis	u0061_0308	
å	å	229	aring	u0061_030A	
æ	æ	230	ae	u00E6	
ç	ç	231	ccedilla	u0063_0327	
è	è	232	egrave	u0065_0300	
é	é	233	eacute	u0065_0301	
ê	ê	234	ecircumflex	u0065_0302	
ë	ë	235	edieresis	u0065_0308	
ì	ì	236	igrave	u0069_0300	
í	í	237	iacute	u0069_0301	
î	î	238	icircumflex	u0069_0302	
ï	ï	239	idieresis	u0069_0308	
ð	ð	240	eth	u00F0	
ñ	ñ	241	ntilde	u006E_0303	
ò	ò	242	ograve	u006F_0300	
ó	ó	243	oacute	u006F_0301	
ô	ô	244	ocircumflex	u006F_0302	
õ	õ	245	otilde	u006F_0303	
ö	ö	246	odieresis	u006F_0308	
÷	÷	247	divide	u00F7	
ø	ø	248	oslash	u00F8	
ù	ù	249	ugrave	u0075_0300	
ú	ú	250	uacute	u0075_0301	
û	û	251	ucircumflex	u0075_0302	
ü	ü	252	udieresis	u0075_0308	
ý	ý	253	yacute	u0079_0301	
þ	þ	254	thorn	u00FE	
ÿ	ÿ	255	ydieresis	u0079_0308	

Named Glyphs

Glyph names can be embedded into the document text by using escape sequences. **groff**(7) describes how these escape sequences look. Glyph names can consist of quite arbitrary characters from the ASCII or latin1 code set, not only alphanumeric characters. Here some examples:

`\(`*ch* A glyph having the 2-character name *ch*.

`\[`*char_name*`]`
A glyph having the name *char_name* (having length 1, 2, 3, ...). Note that '*c*' is not the same as '`\[`*c*`]`' (*c* a single character): The latter is internally mapped to glyph name '\c'. By default, groff defines a single glyph name starting with a backslash, namely '\\-', which can be either accessed as '`\-`' or '`\[-]`'.

\[*base_glyph composite_1 composite_2* ...]
A composite glyph; see below for a more detailed description.

In groff, each 8-bit input character can also referred to by the construct '\[char*n*]' where *n* is the decimal code of the character, a number between 0 and 255 without leading zeros (those entities are *not* glyph names). They are normally mapped onto glyphs using the `.trin` request. Another special convention is the handling of glyphs with names directly derived from a Unicode code point; this is discussed below. Moreover, new glyph names can be created by the `.char` request; see **groff**(7).

In the following, a plus sign in the 'Notes' column indicates that this particular glyph name appears in the PS version of the original troff documentation, CSTR 54.

Entries marked with '***' denote glyphs for mathematical purposes (mainly used for DVI output). Normally, such glyphs have metrics which make them unusable in normal text.

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*
Ð	`\[-D]`	Eth	u00D0	uppercase eth
ð	`\[Sd]`	eth	u00F0	lowercase eth
Þ	`\[TP]`	Thorn	u00DE	uppercase thorn
þ	`\[Tp]`	thorn	u00FE	lowercase thorn
ß	`\[ss]`	germandbls	u00DF	German sharp s

Ligatures and Other Latin Glyphs

ff	`\[ff]`	ff	u0066_0066	ff ligature +
fi	`\[fi]`	fi	u0066_0069	fi ligature +
fl	`\[fl]`	fl	u0066_006C	fl ligature +
ffi	`\[Fi]`	ffi	u0066_0066_0069	ffi ligature +
ffl	`\[Fl]`	ffl	u0066_0066_006C	ffl ligature +
Ł	`\[/L]`	Lslash	u0141	(Polish)
ł	`\[/l]`	lslash	u0142	(Polish)
Ø	`\[/O]`	Oslash	u00D8	(Scandinavic)
ø	`\[/o]`	oslash	u00F8	(Scandinavic)
Æ	`\[AE]`	AE	u00C6	
æ	`\[ae]`	ae	u00E6	
Œ	`\[OE]`	OE	u0152	
œ	`\[oe]`	oe	u0153	
IJ	`\[IJ]`	IJ	u0132	(Dutch)
ij	`\[ij]`	ij	u0133	(Dutch)
ı	`\[.i]`	dotlessi	u0131	(Turkish)
(N/A)	`\[.j]`	dotlessj	---	j without a dot

Accented Characters

Á	`\['A]`	Aacute	u0041_0301	
Ć	`\['C]`	Cacute	u0043_0301	
É	`\['E]`	Eacute	u0045_0301	
Í	`\['I]`	Iacute	u0049_0301	
Ó	`\['O]`	Oacute	u004F_0301	
Ú	`\['U]`	Uacute	u0055_0301	
Ý	`\['Y]`	Yacute	u0059_0301	
á	`\['a]`	aacute	u0061_0301	
ć	`\['c]`	cacute	u0063_0301	
é	`\['e]`	eacute	u0065_0301	
í	`\['i]`	iacute	u0069_0301	
ó	`\['o]`	oacute	u006F_0301	
ú	`\['u]`	uacute	u0075_0301	
ý	`\['y]`	yacute	u0079_0301	
Ä	`\[:A]`	Adieresis	u0041_0308	A with umlaut

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*
Ë	`\[:E]`	Edieresis	u0045_0308	
Ï	`\[:I]`	Idieresis	u0049_0308	
Ö	`\[:O]`	Odieresis	u004F_0308	
Ü	`\[:U]`	Udieresis	u0055_0308	
Ÿ	`\[:Y]`	Ydieresis	u0059_0308	
ä	`\[:a]`	adieresis	u0061_0308	
ë	`\[:e]`	edieresis	u0065_0308	
ï	`\[:i]`	idieresis	u0069_0308	
ö	`\[:o]`	odieresis	u006F_0308	
ü	`\[:u]`	udieresis	u0075_0308	
ÿ	`\[:y]`	ydieresis	u0079_0308	
Â	`\[^A]`	Acircumflex	u0041_0302	
Ê	`\[^E]`	Ecircumflex	u0045_0302	
Î	`\[^I]`	Icircumflex	u0049_0302	
Ô	`\[^O]`	Ocircumflex	u004F_0302	
Û	`\[^U]`	Ucircumflex	u0055_0302	
â	`\[^a]`	acircumflex	u0061_0302	
ê	`\[^e]`	ecircumflex	u0065_0302	
î	`\[^i]`	icircumflex	u0069_0302	
ô	`\[^o]`	ocircumflex	u006F_0302	
û	`\[^u]`	ucircumflex	u0075_0302	
À	``\[`A]``	Agrave	u0041_0300	
È	``\[`E]``	Egrave	u0045_0300	
Ì	``\[`I]``	Igrave	u0049_0300	
Ò	``\[`O]``	Ograve	u004F_0300	
Ù	``\[`U]``	Ugrave	u0055_0300	
à	``\[`a]``	agrave	u0061_0300	
è	``\[`e]``	egrave	u0065_0300	
ì	``\[`i]``	igrave	u0069_0300	
ò	``\[`o]``	ograve	u006F_0300	
ù	``\[`u]``	ugrave	u0075_0300	
Ã	`\[ A]`	Atilde	u0041_0303	
Ñ	`\[ N]`	Ntilde	u004E_0303	
Õ	`\[ O]`	Otilde	u004F_0303	
ã	`\[ a]`	atilde	u0061_0303	
ñ	`\[ n]`	ntilde	u006E_0303	
õ	`\[ o]`	otilde	u006F_0303	
Š	`\[vS]`	Scaron	u0053_030C	
š	`\[vs]`	scaron	u0073_030C	
Ž	`\[vZ]`	Zcaron	u005A_030C	
ž	`\[vz]`	zcaron	u007A_030C	
Ç	`\[,C]`	Ccedilla	u0043_0327	
ç	`\[,c]`	ccedilla	u0063_0327	
Å	`\[oA]`	Aring	u0041_030A	
å	`\[oa]`	aring	u0061_030A	

Accents

The **composite** request is used to map most of the accents to non-spacing glyph names; the values given in parentheses are the original (spacing) ones.

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*

˝	\[a"]	hungarumlaut	u030B (u02DD)	(Hungarian)
¯	\[a-]	macron	u0304 (u00AF)	
˙	\[a.]	dotaccent	u0307 (u02D9)	
ˆ	\[a^]	circumflex	u0302 (u005E)	
´	\[aa]	acute	u0301 (u00B4)	+
`	\[ga]	grave	u0300 (u0060)	+
˘	\[ab]	breve	u0306 (u02D8)	
¸	\[ac]	cedilla	u0327 (u00B8)	
¨	\[ad]	dieresis	u0308 (u00A8)	umlaut
ˇ	\[ah]	caron	u030C (u02C7)	háček
˚	\[ao]	ring	u030A (u02DA)	circle
˜	\[a]	tilde	u0303 (u007E)	
˛	\[ho]	ogonek	u0328 (u02DB)	hook
^	\[ha]	asciicircum	u005E	(spacing)
~	\[ti]	asciitilde	u007E	(spacing)

Quotes

„	\[Bq]	quotedblbase	u201E	low double comma quote
‚	\[bq]	quotesinglbase	u201A	low single comma quote
“	\[lq]	quotedblleft	u201C	
”	\[rq]	quotedblright	u201D	
‘	\[oq]	quoteleft	u2018	single open quote
’	\[cq]	quoteright	u2019	single closing quote
'	\[aq]	quotesingle	u0027	apostrophe quote (ASCII 39)
"	\[dq]	quotedbl	u0022	double quote (ASCII 34)
«	\[Fo]	guillemotleft	u00AB	
»	\[Fc]	guillemotright	u00BB	
‹	\[fo]	guilsinglleft	u2039	
›	\[fc]	guilsinglright	u203A	

Punctuation

¡	\[r!]	exclamdown	u00A1	
¿	\[r?]	questiondown	u00BF	
—	\[em]	emdash	u2014	+
–	\[en]	endash	u2013	
‐	\[hy]	hyphen	u2010	+

Brackets

The extensible bracket pieces are font-invariant glyphs. In classical troff only one glyph was available to vertically extend brackets, braces, and parentheses: ‘bv’. We map it rather arbitrarily to u23AA.

Note that not all devices contain extensible bracket pieces which can be piled up with ‘\b’ due to the restrictions of the escape’s piling algorithm. A general solution to build brackets out of pieces is the following macro:

```
.\" Make a pile centered vertically 0.5em
.\" above the baseline.
.\" The first argument is placed at the top.
.\" The pile is returned in string ‘pile’
.eo
.de pile-make
.  nr pile-wd 0
.  nr pile-ht 0
.  ds pile-args
.
.  nr pile-# \n[.$]
.  while \n[pile-#] \{\
.    nr pile-wd (\n[pile-wd] >? \w’\$[\n[pile-#]]’)
.    nr pile-ht +(\n[rst] - \n[rsb])
.    as pile-args \v’\n[rsb]u’\"
.    as pile-args \Z'\$[\n[pile-#]]'\"
.    as pile-args \v'-\n[rst]u'\"
.    nr pile-# -1
.  \}
.
.  ds pile \v'(-0.5m + (\n[pile-ht]u / 2u))'\"
.  as pile \*[pile-args]\"
.  as pile \v'((\n[pile-ht]u / 2u) + 0.5m)'\"
.  as pile \h'\n[pile-wd]u'\"
..
.ec
```

Another complication is the fact that some glyphs which represent bracket pieces in original troff can be used for other mathematical symbols also, for example ‘lf’ and ‘rf’ which provide the ‘floor’ operator. Other devices (most notably for DVI output) don’t unify such glyphs. For this reason, the four glyphs ‘lf’, ‘rf’, ‘lc’, and ‘rc’ are not unified with similarly looking bracket pieces. In **groff**, only glyphs with long names are guaranteed to pile up correctly for all devices (provided those glyphs exist).

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*
[	`\[lB]`	bracketleft	u005B	
]	`\[rB]`	bracketright	u005D	
{	`\[lC]`	braceleft	u007B	
}	`\[rC]`	braceright	u007D	
⟨	`\[la]`	angleleft	u27E8	left angle bracket
⟩	`\[ra]`	angleright	u27E9	right angle bracket
⎪	`\[bv]`	braceex	u23AA	vertical extension *** +
⎪	`\[braceex]`	braceex	u23AA	
⎡	`\[bracketlefttp]`	bracketlefttp	u23A1	
⎣	`\[bracketleftbt]`	bracketleftbt	u23A3	
⎢	`\[bracketleftex]`	bracketleftex	u23A2	
⎤	`\[bracketrighttp]`	bracketrighttp	u23A4	
⎦	`\[bracketrightbt]`	bracketrightbt	u23A6	
⎥	`\[bracketrightex]`	bracketrightex	u23A5	
⎧	`\[lt]`	bracelefttp	u23A7	+
⎧	`\[bracelefttp]`	bracelefttp	u23A7	
⎨	`\[lk]`	braceleftmid	u23A8	+
⎨	`\[braceleftmid]`	braceleftmid	u23A8	
⎩	`\[lb]`	braceleftbt	u23A9	+
⎩	`\[braceleftbt]`	braceleftbt	u23A9	
⎪	`\[braceleftex]`	braceleftex	u23AA	
⎫	`\[rt]`	bracerighttp	u23AB	+
⎫	`\[bracerighttp]`	bracerighttp	u23AB	
⎬	`\[rk]`	bracerightmid	u23AC	+
⎬	`\[bracerightmid]`	bracerightmid	u23AC	
⎭	`\[rb]`	bracerightbt	u23AD	+
⎭	`\[bracerightbt]`	bracerightbt	u23AD	
⎪	`\[bracerightex]`	bracerightex	u23AA	
⎛	`\[parenlefttp]`	parenlefttp	u239B	
⎝	`\[parenleftbt]`	parenleftbt	u239D	
⎜	`\[parenleftex]`	parenleftex	u239C	
⎞	`\[parenrighttp]`	parenrighttp	u239E	
⎠	`\[parenrightbt]`	parenrightbt	u23A0	
⎟	`\[parenrightex]`	parenrightex	u239F	

Arrows

←	`\[<-]`	arrowleft	u2190	+
→	`\[->]`	arrowright	u2192	+
↔	`\[<>]`	arrowboth	u2194	(horizontal)
↓	`\[da]`	arrowdown	u2193	+
↑	`\[ua]`	arrowup	u2191	+
↕	`\[va]`	arrowupdn	u2195	
⇐	`\[lA]`	arrowdblleft	u21D0	
⇒	`\[rA]`	arrowdblright	u21D2	
⇔	`\[hA]`	arrowdblboth	u21D4	(horizontal)
⇓	`\[dA]`	arrowdbldown	u21D3	
⇑	`\[uA]`	arrowdblup	u21D1	
(N/A)	`\[vA]`	uni21D5	u21D5	vertical double-headed double arrow
—	`\[an]`	arrowhorizex	u23AF	horizontal arrow extension

Lines

The font-invariant glyphs 'br', 'ul', and 'rn' form corners; they can be used to build boxes. Note that both the PostScript and the Unicode-derived names of these three glyphs are just rough approximations.

'rn' also serves in classical troff as the horizontal extension of the square root sign.

'ru' is a font-invariant glyph, namely a rule of length 0.5m.

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*
|	`\[ba]`	bar	u007C	
|	`\[br]`	SF110000	u2502	box rule +
_	`\[ul]`	underscore	u005F	+
‾	`\[rn]`	overline	u203E	use '`\[radicalex]`' for continuation of square root +
_	`\[ru]`	---	---	baseline rule +
¦	`\[bb]`	brokenbar	u00A6	
/	`\[sl]`	slash	u002F	+
\	`\[rs]`	backslash	u005C	reverse solidus

Text markers

○	`\[ci]`	circle	u25CB	+
•	`\[bu]`	bullet	u2022	+
‡	`\[dd]`	daggerdbl	u2021	double dagger sign +
†	`\[dg]`	dagger	u2020	+
◊	`\[lz]`	lozenge	u25CA	
□	`\[sq]`	uni25A1	u25A1	white square +
¶	`\[ps]`	paragraph	u00B6	
§	`\[sc]`	section	u00A7	+
☜	`\[lh]`	uni261C	u261C	hand pointing left +
☞	`\[rh]`	a14	u261E	hand pointing right +
@	`\[at]`	at	u0040	
#	`\[sh]`	numbersign	u0023	
↵	`\[CR]`	carriagereturn	u21B5	
✓	`\[OK]`	a19	u2713	check mark, tick

Legal Symbols

©	`\[co]`	copyright	u00A9	+
®	`\[rg]`	registered	u00AE	+
™	`\[tm]`	trademark	u2122	
(N/A)	`\[bs]`	---	---	AT&T Bell Labs logo (not used in groff) +

Currency symbols

$	\[Do]	dollar	u0024	
¢	\[ct]	cent	u00A2	+
€	\[eu]	---	u20AC	official Euro symbol
€	\[Eu]	Euro	u20AC	font-specific Euro glyph variant
¥	\[Ye]	yen	u00A5	
£	\[Po]	sterling	u00A3	British currency sign
¤	\[Cs]	currency	u00A4	Scandinavian currency sign
ƒ	\[Fn]	florin	u0192	Dutch currency sign

Units

°	\[de]	degree	u00B0	+
‰	\[%0]	perthousand	u2030	per thousand, per mille sign
′	\[fm]	minute	u2032	footmark, prime +
″	\[sd]	second	u2033	
µ	\[mc]	mu	u00B5	micro sign
ª	\[Of]	ordfeminine	u00AA	
º	\[Om]	ordmasculine	u00BA	

Logical Symbols

∧	\[AN]	logicaland	u2227	
∨	\[OR]	logicalor	u2228	
¬	\[no]	logicalnot	u00AC	+
¬	\[tno]	logicalnot	u00AC	text variant of ‘no’
∃	\[te]	existential	u2203	there exists, existential quantifier
∀	\[fa]	universal	u2200	for all, universal quantifier
∋	\[st]	suchthat	u220B	
∴	\[3d]	therefore	u2234	
∴	\[tf]	therefore	u2234	
|	\[or]	bar	u007C	bitwise OR operator (as used in C) +

Mathematical Symbols

½	\[12]	onehalf	u00BD	+
¼	\[14]	onequarter	u00BC	+
¾	\[34]	threequarters	u00BE	+
⅛	\[18]	oneeighth	u215B	
⅜	\[38]	threeeighths	u215C	
⅝	\[58]	fiveeighths	u215D	
⅞	\[78]	seveneighths	u215E	
¹	\[S1]	onesuperior	u00B9	
²	\[S2]	twosuperior	u00B2	
³	\[S3]	threesuperior	u00B3	
+	\[pl]	plus	u002B	plus sign in special font +
−	\[mi]	minus	u2212	minus sign in special font +
(N/A)	\[-+]	uni2213	u2213	
±	\[+-]	plusminus	u00B1	+
±	\[t+-]	plusminus	u00B1	text variant of ‘+−’
·	\[pc]	periodcentered	u00B7	
·	\[md]	dotmath	u22C5	multiplication dot
×	\[mu]	multiply	u00D7	+
×	\[tmu]	multiply	u00D7	text variant of ‘mu’
⊗	\[c*]	circlemultiply	u2297	multiply sign in a circle
⊕	\[c+]	circleplus	u2295	plus sign in a circle
÷	\[di]	divide	u00F7	division sign +
÷	\[tdi]	divide	u00F7	text variant of ‘di’
⁄	\[f/]	fraction	u2044	bar for fractions
∗	\[**]	asteriskmath	u2217	+
≤	\[<=]	lessequal	u2264	+

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*
≥	`\[>=]`	greaterequal	u2265	+
≪	`\[<<]`	uni226A	u226A	much less
≫	`\[>>]`	uni226B	u226B	much greater
=	`\[eq]`	equal	u003D	equals sign in special font +
≠	`\[!=]`	notequal	u003D_0338	+
≡	`\[==]`	equivalence	u2261	+
≢	`\[ne]`	uni2262	u2261_0338	
≅	`\[= ]`	congruent	u2245	approx. equal
≃	`\[|=]`	uni2243	u2243	asymptot. equal to +
~	`\[ap]`	similar	u223C	+
≈	`\[  ]`	approxequal	u2248	almost equal to
≈	`\[ =]`	approxequal	u2248	
∝	`\[pt]`	proportional	u221D	+
∅	`\[es]`	emptyset	u2205	+
∈	`\[mo]`	element	u2208	+
∉	`\[nm]`	notelement	u2208_0338	
⊂	`\[sb]`	propersubset	u2282	+
⊄	`\[nb]`	notsubset	u2282_0338	
⊃	`\[sp]`	propersuperset	u2283	+
⊅	`\[nc]`	uni2285	u2283_0338	not superset
⊆	`\[ib]`	reflexsubset	u2286	+
⊇	`\[ip]`	reflexsuperset	u2287	+
∩	`\[ca]`	intersection	u2229	intersection, cap +
∪	`\[cu]`	union	u222A	union, cup +
∠	`\[/_]`	angle	u2220	
⊥	`\[pp]`	perpendicular	u22A5	
∫	`\[is]`	integral	u222B	+
∫	`\[integral]`	integral	u222B	***
∑	`\[sum]`	summation	u2211	***
∏	`\[product]`	product	u220F	***
(N/A)	`\[coproduct]`	uni2210	u2210	***
∇	`\[gr]`	gradient	u2207	+
√	`\[sr]`	radical	u221A	square root +
√	`\[sqrt]`	radical	u221A	***
‾	`\[radicalex]`	radicalex	---	continuation of square root
‾	`\[sqrtex]`	radicalex	---	***
⌈	`\[lc]`	uni2308	u2308	left ceiling +
⌉	`\[rc]`	uni2309	u2309	right ceiling +
⌊	`\[lf]`	uni230A	u230A	left floor +
⌋	`\[rf]`	uni230B	u230B	right floor +
∞	`\[if]`	infinity	u221E	+
ℵ	`\[Ah]`	aleph	u2135	
ℑ	`\[Im]`	Ifraktur	u2111	Gothic I, imaginary
ℜ	`\[Re]`	Rfraktur	u211C	Gothic R, real
℘	`\[wp]`	weierstrass	u2118	Weierstrass p
∂	`\[pd]`	partialdiff	u2202	partial differentiation sign +
ℏ	`\[-h]`	uni210F	u210F	Planck constant over two pi
ℏ	`\[hbar]`	uni210F	u210F	

Greek glyphs

These glyphs are intended for technical use, not for real Greek; normally, the uppercase letters have upright shape, and the lowercase ones are slanted. There is a problem with the mapping of letter phi to Unicode. Prior to Unicode version 3.0, the difference between U+03C6, GREEK SMALL LETTER PHI, and U+03D5, GREEK PHI SYMBOL, was not clearly described; only the glyph shapes in the Unicode book could be used as a reference. Starting with Unicode 3.0, the reference glyphs have been exchanged and described verbally also: In mathematical context, U+03D5 is the stroked variant and

U+03C6 the curly glyph. Unfortunately, most font vendors didn't update their fonts to this (incompatible) change in Unicode. At the time of this writing (January 2006), it is not clear yet whether the Adobe Glyph Names 'phi' and 'phi1' also change its meaning if used for mathematics, thus compatibility problems are likely to happen – being conservative, groff currently assumes that 'phi' in a PostScript symbol font is the stroked version.

In groff, symbol '`\[*f]`' always denotes the stroked version of phi, and '`\[+f]`' the curly variant.

Α	`\[*A]`	Alpha	u0391	+
Β	`\[*B]`	Beta	u0392	+
Γ	`\[*G]`	Gamma	u0393	+
Δ	`\[*D]`	Delta	u0394	+
Ε	`\[*E]`	Epsilon	u0395	+
Ζ	`\[*Z]`	Zeta	u0396	+
Η	`\[*Y]`	Eta	u0397	+
Θ	`\[*H]`	Theta	u0398	+
Ι	`\[*I]`	Iota	u0399	+
Κ	`\[*K]`	Kappa	u039A	+
Λ	`\[*L]`	Lambda	u039B	+
Μ	`\[*M]`	Mu	u039C	+
Ν	`\[*N]`	Nu	u039D	+
Ξ	`\[*C]`	Xi	u039E	+
Ο	`\[*O]`	Omicron	u039F	+
Π	`\[*P]`	Pi	u03A0	+
Ρ	`\[*R]`	Rho	u03A1	+
Σ	`\[*S]`	Sigma	u03A3	+
Τ	`\[*T]`	Tau	u03A4	+
ϒ	`\[*U]`	Upsilon	u03A5	+
Φ	`\[*F]`	Phi	u03A6	+
Χ	`\[*X]`	Chi	u03A7	+
Ψ	`\[*Q]`	Psi	u03A8	+
Ω	`\[*W]`	Omega	u03A9	+
α	`\[*a]`	alpha	u03B1	+
β	`\[*b]`	beta	u03B2	+
γ	`\[*g]`	gamma	u03B3	+
δ	`\[*d]`	delta	u03B4	+
ε	`\[*e]`	epsilon	u03B5	+
ζ	`\[*z]`	zeta	u03B6	+
η	`\[*y]`	eta	u03B7	+
θ	`\[*h]`	theta	u03B8	+
ι	`\[*i]`	iota	u03B9	+
κ	`\[*k]`	kappa	u03BA	+
λ	`\[*l]`	lambda	u03BB	+
μ	`\[*m]`	mu	u03BC	+
ν	`\[*n]`	nu	u03BD	+
ξ	`\[*c]`	xi	u03BE	+
ο	`\[*o]`	omicron	u03BF	+
π	`\[*p]`	pi	u03C0	+
ρ	`\[*r]`	rho	u03C1	+
"	`\[ts]`	sigma1	u03C2	terminal sigma +
σ	`\[*s]`	sigma	u03C3	+
τ	`\[*t]`	tau	u03C4	+
υ	`\[*u]`	upsilon	u03C5	+
ϕ	`\[*f]`	phi	u03D5	(stroked glyph)+
χ	`\[*x]`	chi	u03C7	+
ψ	`\[*q]`	psi	u03C8	+
ω	`\[*w]`	omega	u03C9	+

Output	*Input name*	*PostScript name*	*Unicode decomposed*	*Notes*
ϑ	`\[+h]`	theta1	u03D1	variant theta
φ	`\[+f]`	phi1	u03C6	variant phi (curly shape)
ϖ	`\[+p]`	omega1	u03D6	variant pi, looking like omega
(N/A)	`\[+e]`	uni03F5	u03F5	variant epsilon
Card symbols				
♣	`\[CL]`	club	u2663	black club suit
♠	`\[SP]`	spade	u2660	black spade suit
♥	`\[HE]`	heart	u2665	black heart suit
(N/A)	`\[u2662]`	uni2662	u2662	white heart suit
♦	`\[DI]`	diamond	u2666	black diamond suit
(N/A)	`\[u2661]`	uni2661	u2661	white diamond suit

AUTHOR

This document is part of *groff*, the GNU roff distribution. It was written by James Clark with additions by Werner Lemberg and Bernd Warken

SEE ALSO

groff(1)
: the GNU roff formatter

groff(7)
: a short reference of the groff formatting language

An extension to the troff character set for Europe, E.G. Keizer, K.J. Simonsen, J. Akkerhuis; EUUG Newsletter, Volume 9, No. 2, Summer 1989

The Unicode Standard

GROFF_DIFF

NAME

groff_diff – differences between GNU troff and classical troff

DESCRIPTION

This manual page describes the language differences between *groff*, the GNU *roff* text processing system, and the classical *roff* formatter of the freely available Unix 7 of the 1970s, documented in the *Troff User's Manual* by *Ossanna* and *Kernighan*. This inludes the roff language as well as the intermediate output format (troff output).

The section *SEE ALSO* gives pointers to both the classical *roff* and the modern *groff* documentation.

GROFF LANGUAGE

In this section, all additional features of *groff* compared to the classical Unix 7 *troff* are described in detail.

Long names

The names of number registers, fonts, strings/macros/diversions, special characters (glyphs), and colors can be of any length. In escape sequences, additionally to the classical '(*xx*' construction for a two-character glyph name, you can use '[*xxx*]' for a name of arbitrary length.

\[*xxx***]** Print the special character (glyph) called *xxx*.

\[*comp1 comp2 . . .***]**
Print composite glyph consisting of multiple components. Example: '\[A ho]' is capital letter A with ogonek which finally maps to glyph name 'u0041_0328'. See the *groff info file* for details how a glyph name for a composite glyph is constructed, and **groff_char**(7) for a list of glyph name components used in composite glyph names.

\f[*xxx***]** Set font *xxx*. Additionally, **\f[]** is a new syntax form equal to **\fP**, i.e., to return to the previous font.

[xxx arg1 arg2 . . .***]**
Interpolate string *xxx*, taking *arg1*, *arg2*, . . . as arguments.

\n[*xxx***]** Interpolate number register *xxx*.

Fractional point sizes

A *scaled point* is equal to **1/sizescale** points, where **sizescale** is specified in the **DESC** file (1 by default). There is a new scale indicator **z** that has the effect of multiplying by sizescale. Requests and escape sequences in troff interpret arguments that represent a point size as being in units of scaled points, but they evaluate each such argument using a default scale indicator of **z**. Arguments treated in this way are the argument to the **ps** request, the third argument to the **cs** request, the second and fourth arguments to the **tkf** request, the argument to the **\H** escape sequence, and those variants of the **\s** escape sequence that take a numeric expression as their argument.

For example, suppose sizescale is 1000; then a scaled point is equivalent to a millipoint; the call **.ps 10.25** is equivalent to **.ps 10.25z** and so sets the point size to 10250 scaled points, which is equal to 10.25 points.

The number register **\n[.s]** returns the point size in points as decimal fraction. There is also a new number register **\n[.ps]** that returns the point size in scaled points.

It would make no sense to use the **z** scale indicator in a numeric expression whose default scale indicator was neither **u** nor **z**, and so **troff** disallows this. Similarly it would make no sense to use a scaling indicator other than **z** or **u** in a numeric expression whose default scale indicator was **z**, and so **troff** disallows this as well.

There is also new scale indicator **s** which multiplies by the number of units in a scaled point. So, for example, **\n[.ps]s** is equal to **1m**. Be sure not to confuse the **s** and **z** scale indicators.

Numeric expressions

Spaces are permitted in a number expression within parentheses.

M indicates a scale of 100ths of an em. **f** indicates a scale of 65536 units, providing fractions for color definitions with the **defcolor** request. For example, 0.5f = 32768u.
*e1***>?***e2* The maximum of *e1* and *e2*.

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

(*c*;*e*) Evaluate *e* using *c* as the default scaling indicator. If *c* is missing, ignore scaling indicators in the evaluation of *e*.

New escape sequences

\A'*anything***'**
This expands to **1** or **0**, depending on whether *anything* is or is not acceptable as the name of a string, macro, diversion, number register, environment, font, or color. It returns **0** if *anything* is empty. This is useful if you want to look up user input in some sort of associative table.

\B'*anything***'**
This expands to **1** or **0**, depending on whether *anything* is or is not a valid numeric expression. It returns **0** if *anything* is empty.

\C'*xxx***'**
Typeset glyph named *xxx*. Normally it is more convenient to use **\[***xxx***]**. But **\C** has the advantage that it is compatible with recent versions of UNIX and is available in compatibility mode.

\E This is equivalent to an escape character, but it is not interpreted in copy mode. For example, strings to start and end superscripting could be defined like this

```
.ds { \v'−.3m'\s'\En[.s]*6u/10u' .ds } \s0\v'.3m'
```

The use of **\E** ensures that these definitions work even if ***{** gets interpreted in copy mode (for example, by being used in a macro argument).

\F*f*
\F(*fm*
\F[*fam***]**
Change font family. This is the same as the **fam** request. **\F[]** switches back to the previous color (note that **\FP** won't work; it selects font family 'P' instead).

\m*x*
\m(*xx*
\m[*xxx***]**
Set drawing color. **\m[]** switches back to the previous color.

\M*x*
\M(*xx*
\M[*xxx***]**
Set background color for filled objects drawn with the **\D'**. . .**'** commands. **\M[]** switches back to the previous color.

\N'*n***'** Typeset the glyph with index *n* in the current font. *n* can be any integer. Most devices only have glyphs with indices between 0 and 255. If the current font does not contain a glyph with that code, special fonts are *not* searched. The **\N** escape sequence can be conveniently used in conjunction with the **char** request, for example

```
.char \[phone] \f(ZD\N'37'
```

The index of each glyph is given in the fourth column in the font description file after the **charset** command. It is possible to include unnamed glyphs in the font description file by using a name of **−−−**; the **\N** escape sequence is the only way to use these.

\O*n*
\O[*n***]** Suppress troff output. The escapes **\O2**, **\O3**, **\O4**, and **\O5** are intended for internal use by **grohtml**.

\O0 Disable any ditroff glyphs from being emitted to the device driver, provided that the escape occurs at the outer level (see **\O3** and **\O4**).

\O1 Enable output of glyphs, provided that the escape occurs at the outer level.

\O0 and **\O1** also reset the registers **\n[opminx]**, **\n[opminy]**, **\n[opmaxx]**, and **\n[opmaxy]** to −1. These four registers mark the top left and bottom right hand corners of a box which encompasses all written glyphs.

\O2 Provided that the escape occurs at the outer level, enable output of glyphs and also write out to stderr the page number and four registers encompassing the glyphs previously written since the last call to **\O**.

\O3 Begin a nesting level. At start-up, **troff** is at outer level. This is really an internal mechanism for **grohtml** while producing images. They are generated by running the troff source through **troff** to the postscript device and **ghostscript** to produce images in PNG format. The **\O3** escape starts a new page if the device is not html (to reduce the possibility of images crossing a page boundary).

\O4 End a nesting level.

\O5[*Pfilename***]**
This escape is **grohtml** specific. Provided that this escape occurs at the outer nesting level, write *filename* to stderr. The position of the image, *P*, must be specified and must be one of **l**, **r**, **c**, or **i** (left, right, centered, inline). *filename* is associated with the production of the next inline image.

\R'*name* ±*n***'**
This has the same effect as

.nr *name* ±*n*

\s(*nn*
\s±(*nn* Set the point size to *nn* points; *nn* must be exactly two digits.

\s[±*n***]**
\s±[*n***]**
\s'±*n***'**
\s±'*n***'** Set the point size to *n* scaled points; *n* is a numeric expression with a default scale indicator of **z**.

\V*x*
\V(*xx*
\V[*xxx***]**
Interpolate the contents of the environment variable *xxx*, as returned by **getenv**(3). **\V** is interpreted in copy mode.

\Y*x*
\Y(*xx*
\Y[*xxx***]**
This is approximately equivalent to **\X'*[***xxx***]'**. However the contents of the string or macro *xxx* are not interpreted; also it is permitted for *xxx* to have been defined as a macro and thus contain newlines (it is not permitted for the argument to **\X** to contain newlines). The inclusion of newlines requires an extension to the UNIX troff output format, and confuses drivers that do not know about this extension.

\Z'*anything***'**
Print anything and then restore the horizontal and vertical position; *anything* may not contain tabs or leaders.

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

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

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

\$^ In a macro, the representation of all parameters as if they were an argument to the **ds** request.

\$(*nn*
\$[*nnn***]** In a macro or string, this gives the *nn*-th or *nnn*-th argument. Macros and strings can have an unlimited number of arguments.

\?*anything***\?**
When used in a diversion, this transparently embeds *anything* in the diversion. *anything* is

read in copy mode. When the diversion is reread, *anything* is interpreted. *anything* may not contain newlines; use **\!** if you want to embed newlines in a diversion. The escape sequence **\?** is also recognized in copy mode and turned into a single internal code; it is this code that terminates *anything*. Thus

```
.nr x 1 .nf .di d \?\\?\\\\?\\\\\\\\nx\\\\?\\?\?  .di .nr x 2 .di e .d .di .nr x 3 .di f .e .di .nr x
4 .f
```

prints **4**.

\/ This increases the width of the preceding glyph so that the spacing between that glyph and the following glyph is correct if the following glyph is a roman glyph. For example, if an italic f is immediately followed by a roman right parenthesis, then in many fonts the top right portion of the f overlaps the top left of the right parenthesis producing *f*), which is ugly. Inserting **\/** produces *f*) and avoids this problem. It is a good idea to use this escape sequence whenever an italic glyph is immediately followed by a roman glyph without any intervening space.

\, This modifies the spacing of the following glyph so that the spacing between that glyph and the preceding glyph is correct if the preceding glyph is a roman glyph. For example, inserting **\,** between the parenthesis and the f changes (*f* to (*f*. It is a good idea to use this escape sequence whenever a roman glyph is immediately followed by an italic glyph without any intervening space.

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

**\ ** This produces an unbreakable space that stretches like a normal inter-word space when a line is adjusted.

\: This causes the insertion of a zero-width break point. It is equal to **\%** within a word but without insertion of a soft hyphen glyph.

\# Everything up to and including the next newline is ignored. This is interpreted in copy mode. It is like **\"** except that **\"** does not ignore the terminating newline.

New requests

.aln *xx yy*
Create an alias *xx* for number register object named *yy*. The new name and the old name are exactly equivalent. If *yy* is undefined, a warning of type **reg** is generated, and the request is ignored.

.als *xx yy*
Create an alias *xx* for request, string, macro, or diversion object named *yy*. The new name and the old name are exactly equivalent (it is similar to a hard rather than a soft link). If *yy* is undefined, a warning of type **mac** is generated, and the request is ignored. The **de**, **am**, **di**, **da**, **ds**, and **as** requests only create a new object if the name of the macro, diversion or string is currently undefined or if it is defined to be a request; normally they modify the value of an existing object.

.am1 *xx yy*
Similar to **.am**, but compatibility mode is switched off during execution. To be more precise, a ‘compatibility save’ token is inserted at the beginning of the macro addition, and a ‘compatibility restore’ token at the end. As a consequence, the requests **am**, **am1**, **de**, and **de1** can be intermixed freely since the compatibility save/restore tokens only affect the macro parts defined by **.am1** and **.ds1**.

.ami *xx yy*
Append to macro indirectly. See the **dei** request below for more information.

.ami1 *xx yy*
Same as the **ami** request but compatibility mode is switched off during execution.

.as1 *xx yy*
Similar to **.as**, but compatibility mode is switched off during expansion. To be more precise, a ‘compatibility save’ token is inserted at the beginning of the string, and a ‘compatibility restore’ token at the end. As a consequence, the requests **as**, **as1**, **ds**, and **ds1** can be intermixed freely since the compatibility save/restore tokens only affect the (sub)strings

defined by **as1** and **ds1**.

.asciify *xx*

This request 'unformats' the diversion *xx* in such a way that ASCII and space characters (and some escape sequences) that were formatted and diverted into *xx* are treated like ordinary input characters when *xx* is reread. Useful for diversions in conjunction with the **writem** request. It can be also used for gross hacks; for example, this

```
.tr @.  .di x @nr n 1 .br .di .tr @@ .asciify x .x
```

sets register **n** to 1. Note that glyph information (font, font size, etc.) is not preserved; use **.unformat** instead.

.backtrace

Print a backtrace of the input stack on stderr.

.blm *xx*

Set the blank line macro to *xx*. If there is a blank line macro, it is invoked when a blank line is encountered instead of the usual troff behaviour.

.box *xx*
.boxa *xx*

These requests are similar to the **di** and **da** requests with the exception that a partially filled line does not become part of the diversion (i.e., the diversion always starts with a new line) but is restored after ending the diversion, discarding the partially filled line which possibly comes from the diversion.

.break Break out of a while loop. See also the **while** and **continue** requests. Be sure not to confuse this with the **br** request.

.brp This is the same as **\p**.

.cflags *n c1 c2*...

Characters *c1*, *c2*,... have properties determined by *n*, which is ORed from the following:

1 The character ends sentences (initially characters **.?!** have this property).

2 Lines can be broken before the character (initially no characters have this property); a line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This can be overridden with value 64.

4 Lines can be broken after the character (initially characters **–\[hy]\[em]** have this property); a line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This can be overridden with value 64.

8 The glyph associated with this character overlaps horizontally (initially characters **\[ul]\[rn]\[ru]\[radicalex]\[sqrtex]** have this property).

16 The glyph associated with this character overlaps vertically (initially glyph **\[br]** has this property).

32 An end-of-sentence character followed by any number of characters with this property is treated as the end of a sentence if followed by a newline or two spaces; in other words the character is transparent for the purposes of end-of-sentence recognition; this is the same as having a zero space factor in TEX (initially characters **"')]*\[dg]\[rq]** have this property).

64 Ignore hyphenation code values of the surrounding characters. Use this in combination with values 2 and 4 (initially no characters have this property).

.char *c string*

[This request can both define characters and glyphs.]

Define entity *c* to be *string*. To be more precise, define (or even override) a groff entity which can be accessed with name *c* on the input side, and which uses *string* on the output side. Every time glyph *c* needs to be printed, *string* is processed in a temporary environment and the result is wrapped up into a single object. Compatibility mode is turned off and the escape character is set to **** while *string* is being processed. Any emboldening, constant spacing or track kerning is applied to this object rather than to individual glyphs in *string*.

A groff object defined by this request can be used just like a normal glyph provided by the output device. In particular other characters can be translated to it with the **tr** request; it can be made the leader glyph by the **lc** request; repeated patterns can be drawn with the glyph using the **\l** and **\L** escape sequences; words containing *c* can be hyphenated correctly, if the **hcode** request is used to give the object a hyphenation code.

There is a special anti-recursion feature: Use of glyph within the glyph's definition is handled like normal glyphs not defined with **char**.

A glyph definition can be removed with the **rchar** request.

.chop *xx*
Chop the last element off macro, string, or diversion *xx*. This is useful for removing the newline from the end of diversions that are to be interpolated as strings.

.close *stream*
Close the stream named *stream*; *stream* will no longer be an acceptable argument to the **write** request. See the **open** request.

.composite *glyph1 glyph2*
Map glyph name *glyph1* to glyph name *glyph2* if it is used in **\[**...**]** with more than one component.

.continue
Finish the current iteration of a while loop. See also the **while** and **break** requests.

.color *n*
If *n* is non-zero or missing, enable colors (this is the default), otherwise disable them.

.cp *n* If *n* is non-zero or missing, enable compatibility mode, otherwise disable it. In compatibility mode, long names are not recognized, and the incompatibilities caused by long names do not arise.

.defcolor *xxx scheme color_components*
Define color *xxx*. *scheme* can be one of the following values: **rgb** (three components), **cmy** (three components), **cmyk** (four components), and **gray** or **grey** (one component). Color components can be given either as a hexadecimal string or as positive decimal integers in the range 0-65535. A hexadecimal string contains all color components concatenated; it must start with either **#** or **##**. The former specifies hex values in the range 0-255 (which are internally multiplied by 257), the latter in the range 0-65535. Examples: #FFC0CB (pink), ##ffff0000ffff (magenta). A new scaling indicator **f** has been introduced which multiplies its value by 65536; this makes it convenient to specify color components as fractions in the range 0 to 1. Example:

```
.defcolor darkgreen rgb 0.1f 0.5f 0.2f
```

Note that **f** is the default scaling indicator for the **defcolor** request, thus the above statement is equivalent to

```
.defcolor darkgreen rgb 0.1 0.5 0.2
```

The color named **default** (which is device-specific) can't be redefined. It is possible that the default color for **\M** and **\m** is not the same.

.de1 *xx yy*
Similar to **.de**, but compatibility mode is switched off during execution. On entry, the current compatibility mode is saved and restored at exit.

.dei *xx yy*
Define macro indirectly. The following example

```
.ds xx aa .ds yy bb .dei xx yy
```

is equivalent to

```
.de aa bb
```

.dei1 *xx yy*
Similar to the **dei** request but compatibility mode is switched off during execution.

.device *anything*
This is (almost) the same as the **\X** escape. *anything* is read in copy mode; a leading " is stripped.

.devicem *xx*
This is the same as the **\Y** escape (to embed the contents of a macro into the intermediate output preceded with ‘x X’).

.do *xxx* Interpret *.xxx* with compatibility mode disabled. For example,

```
.do fam T
```

would have the same effect as

```
.fam T
```

except that it would work even if compatibility mode had been enabled. Note that the previous compatibility mode is restored before any files sourced by *xxx* are interpreted.

.ds1 *xx yy*
Similar to **.ds**, but compatibility mode is switched off during expansion. To be more precise, a ‘compatibility save’ token is inserted at the beginning of the string, and a ‘compatibility restore’ token at the end.

.ecs Save current escape character.

.ecr Restore escape character saved with **ecs**. Without a previous call to **ecs**, ‘\’ will be the new escape character.

.evc *xx* Copy the contents of environment *xx* to the current environment. No pushing or popping of environments is done.

.fam *xx*
Set the current font family to *xx*. The current font family is part of the current environment. If *xx* is missing, switch back to previous font family. The value at start-up is ‘T’. See the description of the **sty** request for more information on font families.

.fchar *c string*
Define fallback character (or glyph) *c* to be *string*. The syntax of this request is the same as the **char** request; the only difference is that a glyph defined with **char** hides the glyph with the same name in the current font, whereas a glyph defined with **fchar** is checked only if the particular glyph isn’t found in the current font. This test happens before checking special fonts.

.fcolor *c*
Set the fill color to *c*. If *c* is missing, switch to the previous fill color.

.fschar *f c string*
Define fallback character (or glyph) *c* for font *f* to be *string*. The syntax of this request is the same as the **char** request (with an additional argument to specify the font); a glyph defined with **fschar** is searched after the list of fonts declared with the **fspecial** request but before the list of fonts declared with **.special**.

.fspecial *f s1 s2*. . .
When the current font is *f*, fonts *s1*, *s2*,. . . are special, that is, they are searched for glyphs not in the current font. Any fonts specified in the **special** request are searched after fonts specified in the **fspecial** request. Without argument, reset the list of global special fonts to be empty.

.ftr *f g* Translate font *f* to *g*. Whenever a font named *f* is referred to in an **\f** escape sequence, in the **F** and **S** conditional operators, or in the **ft**, **ul**, **bd**, **cs**, **tkf**, **special**, **fspecial**, **fp**, or **sty** requests, font *g* is used. If *g* is missing, or equal to *f* then font *f* is not translated.

.fzoom *f zoom*
Set zoom factor *zoom* for font *f*. *zoom* must a non-negative integer multiple of 1/1000th. If it is missing or is equal to zero, it means the same as 1000, namely no magnification. *f* must be a real font name, not a style.

.gcolor *c*
Set the glyph color to *c*. If *c* is missing, switch to the previous glyph color.

.hcode *c1 code1 c2 code2*...
Set the hyphenation code of character *c1* to *code1* and that of *c2* to *code2*. A hyphenation code must be a single input character (not a special character) other than a digit or a space. Initially each lower-case letter a-z has a hyphenation code, which is itself, and each upper-case letter A-Z has a hyphenation code which is the lower-case version of itself. See also the **hpf** request.

.hla *lang*
Set the current hyphenation language to *lang*. Hyphenation exceptions specified with the **hw** request and hyphenation patterns specified with the **hpf** request are both associated with the current hyphenation language. The **hla** request is usually invoked by the **troffrc** file to set up a default language.

.hlm *n* Set the maximum number of consecutive hyphenated lines to *n*. If *n* is negative, there is no maximum. The default value is −1. This value is associated with the current environment. Only lines output from an environment count towards the maximum associated with that environment. Hyphens resulting from **\%** are counted; explicit hyphens are not.

.hpf *file*
Read hyphenation patterns from *file*; this is searched for in the same way that *name***.tmac** is searched for when the **−m***name* option is specified. It should have the same format as (simple) TEX patterns files. More specifically, the following scanning rules are implemented.

- A percent sign starts a comment (up to the end of the line) even if preceded by a backslash.
- No support for 'digraphs' like **\$**.
- ^^*xx* (*x* is 0-9 or a-f) and ^^*x* (character code of *x* in the range 0-127) are recognized; other use of ^ causes an error.
- No macro expansion.
- **hpf** checks for the expression **\patterns{**...**}** (possibly with whitespace before and after the braces). Everything between the braces is taken as hyphenation patterns. Consequently, **{** and **}** are not allowed in patterns.
- Similarly, **\hyphenation{**...**}** gives a list of hyphenation exceptions.
- **\endinput** is recognized also.
- For backwards compatibility, if **\patterns** is missing, the whole file is treated as a list of hyphenation patterns (only recognizing the **%** character as the start of a comment).

Use the **hpfcode** request to map the encoding used in hyphenation patterns files to **groff**'s input encoding.

The set of hyphenation patterns is associated with the current language set by the **hla** request. The **hpf** request is usually invoked by the **troffrc** file; a second call replaces the old patterns with the new ones.

.hpfa *file*
The same as **hpf** except that the hyphenation patterns from *file* are appended to the patterns already loaded in the current language.

.hpfcode *a b c d*...
After reading a hyphenation patterns file with the **hpf** or **hpfa** request, convert all characters with character code *a* in the recently read patterns to character code *b*, character code *c* to *d*, etc. Initially, all character codes map to themselves. The arguments of **hpfcode** must be integers in the range 0 to 255. Note that it is even possible to use character codes which are invalid in **groff** otherwise.

.hym *n* Set the *hyphenation margin* to *n*: when the current adjustment mode is not **b**, the line is not hyphenated if the line is no more than *n* short. The default hyphenation margin is 0. The default scaling indicator for this request is **m**. The hyphenation margin is associated with the current environment. The current hyphenation margin is available in the **\n[.hym]** register.

.hys *n* Set the *hyphenation space* to *n*: When the current adjustment mode is **b** don't hyphenate the line if the line can be justified by adding no more than *n* extra space to each word space. The default hyphenation space is 0. The default scaling indicator for this request is **m**. The hyphenation space is associated with the current environment. The current hyphenation space is available in the **\n[.hys]** register.

.itc *n macro*
Variant of **.it** for which a line interrupted with **\c** counts as one input line.

.kern *n* If *n* is non-zero or missing, enable pairwise kerning, otherwise disable it.

.length *xx string*
Compute the length of *string* and return it in the number register *xx* (which is not necessarily defined before).

.linetabs *n*
If *n* is non-zero or missing, enable line-tabs mode, otherwise disable it (which is the default). In line-tabs mode, tab distances are computed relative to the (current) output line. Otherwise they are taken relative to the input line. For example, the following

```
.ds x a\t\c .ds y b\t\c .ds z c .ta 1i 3i \*x \*y \*z
```

yields

```
a     b     c
```

In line-tabs mode, the same code gives

```
a     b           c
```

Line-tabs mode is associated with the current environment; the read-only number register **\n[.linetabs]** is set to 1 if in line-tabs mode, and 0 otherwise.

.mso *file*
The same as the **so** request except that *file* is searched for in the same directories as macro files for the the **−m** command line option. If the file name to be included has the form *name***.tmac** and it isn't found, **mso** tries to include **tmac.***name* instead and vice versa.

.nop *anything*
Execute *anything*. This is similar to '.if 1'.

.nroff Make the **n** built-in condition true and the **t** built-in condition false. This can be reversed using the **troff** request.

.open *stream filename*
Open *filename* for writing and associate the stream named *stream* with it. See also the **close** and **write** requests.

.opena *stream filename*
Like **open**, but if *filename* exists, append to it instead of truncating it.

.output *string*
Emit *string* directly to the intermediate output (subject to copy-mode interpretation); this is similar to **\!** used at the top level. An initial double quote in *string* is stripped off to allow initial blanks.

.pev Print the current environment and each defined environment state on stderr.

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

.psbb *filename*
Get the bounding box of a PostScript image *filename*. This file must conform to Adobe's Document Structuring Conventions; the command looks for a **%%BoundingBox** comment to extract the bounding box values. After a successful call, the coordinates (in PostScript units) of the lower left and upper right corner can be found in the registers **\n[llx]**, **\n[lly]**, **\n[urx]**, and **\n[ury]**, respectively. If some error has occurred, the four registers are set to zero.

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

command.

.ptr Print the names and positions of all traps (not including input line traps and diversion traps) on stderr. Empty slots in the page trap list are printed as well, because they can affect the priority of subsequently planted traps.

.pvs ±*n* Set the post-vertical line space to *n*; default scale indicator is **p**. This value is added to each line after it has been output. With no argument, the post-vertical line space is set to its previous value.

The total vertical line spacing consists of four components: **.vs** and **\x** with a negative value which are applied before the line is output, and **.pvs** and **\x** with a positive value which are applied after the line is output.

.rchar *c1 c2*...
Remove the definitions of glyphs *c1*, *c2*,... This undoes the effect of a **char** request.

.return Within a macro, return immediately. If called with an argument, return twice, namely from the current macro and from the macro one level higher. No effect otherwise.

.rfschar *c1 c2*...
Remove the font-specific definitions of glyphs *c1*, *c2*,... This undoes the effect of a **fschar** request.

.rj
.rj *n* Right justify the next *n* input lines. Without an argument right justify the next input line. The number of lines to be right justified is available in the **\n[.rj]** register. This implicitly does **.ce 0**. The **ce** request implicitly does **.rj 0**.

.rnn *xx yy*
Rename number register *xx* to *yy*.

.schar *c string*
Define global fallback character (or glyph) *c* to be *string*. The syntax of this request is the same as the **char** request; a glyph defined with **schar** is searched after the list of fonts declared with the **special** request but before the mounted special fonts.

.shc *c* Set the soft hyphen character to *c*. If *c* is omitted, the soft hyphen character is set to the default **\[hy]**. The soft hyphen character is the glyph which is inserted when a word is hyphenated at a line break. If the soft hyphen character does not exist in the font of the glyph immediately preceding a potential break point, then the line is not broken at that point. Neither definitions (specified with the **char** request) nor translations (specified with the **tr** request) are considered when finding the soft hyphen character.

.shift *n* In a macro, shift the arguments by *n* positions: argument *i* becomes argument *i* – *n*; arguments 1 to *n* are no longer available. If *n* is missing, arguments are shifted by 1. Shifting by negative amounts is currently undefined.

.sizes *s1 s2*... *sn* **[0]**
This command is similar to the **sizes** command of a **DESC** file. It sets the available font sizes for the current font to *s1*, *s2*,..., *sn* scaled points. The list of sizes can be terminated by an optional **0**. Each *si* can also be a range of sizes *m*-*n*. Contrary to the font file command, the list can't extend over more than a single line.

.special *s1 s2*...
Fonts *s1*, *s2*,... are special and are searched for glyphs not in the current font. Without arguments, reset the list of special fonts to be empty.

.spreadwarn *limit*
Make **troff** emit a warning if the additional space inserted for each space between words in an output line is larger or equal to *limit*. A negative value is changed to zero; no argument toggles the warning on and off without changing *limit*. The default scaling indicator is **m**. At startup, **spreadwarn** is deactivated, and *limit* is set to 3m. For example, **.spreadwarn 0.2m** causes a warning if **troff** must add 0.2m or more for each interword space in a line. This request is active only if text is justified to both margins (using **.ad b**).

.sty *n f* Associate style *f* with font position *n*. A font position can be associated either with a font or with a style. The current font is the index of a font position and so is also either a font or a style. When it is a style, the font that is actually used is the font the name of which is the concatenation of the name of the current family and the name of the current style. For example, if the current font is 1 and font position 1 is associated with style **R** and the current font family is **T**, then font **TR** is used. If the current font is not a style, then the current family is ignored. When the requests **cs**, **bd**, **tkf**, **uf**, or **fspecial** are applied to a style, then they are applied instead to the member of the current family corresponding to that style. The default family can be set with the **–f** command line option. The **styles** command in the DESC file controls which font positions (if any) are initially associated with styles rather than fonts.

.substring *xx n1* **[***n2***]**
Replace the string named *xx* with the substring defined by the indices *n1* and *n2*. The first character in the string has index 0. If *n2* is omitted, it is taken to be equal to the string's length. If the index value *n1* or *n2* is negative, it is counted from the end of the string, going backwards: The last character has index −1, the character before the last character has index −2, etc.

.tkf *f s1 n1 s2 n2*
Enable track kerning for font *f*. When the current font is *f* the width of every glyph is increased by an amount between *n1* and *n2*; when the current point size is less than or equal to *s1* the width is increased by *n1*; when it is greater than or equal to *s2* the width is increased by *n2*; when the point size is greater than or equal to *s1* and less than or equal to *s2* the increase in width is a linear function of the point size.

.tm1 *string*
Similar to the **tm** request, *string* is read in copy mode and written on the standard error, but an initial double quote in *string* is stripped off to allow initial blanks.

.tmc *string*
Similar to **tm1** but without writing a final newline.

.trf *filename*
Transparently output the contents of file *filename*. Each line is output as if preceded by **\!**; however, the lines are not subject to copy-mode interpretation. If the file does not end with a newline, then a newline is added. For example, you can define a macro *x* containing the contents of file *f*, using

```
.di x .trf f .di
```

Unlike with the **cf** request, the file cannot contain characters such as NUL that are not valid troff input characters.

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

```
.trin ax .di xxx a .br .di .xxx .trin aa .asciify xxx .xxx
```

The result is **x a**. Using **tr**, the result would be **x x**.

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

```
.tr ab .di x \!.tm a .di .x
```

prints **b**; if **trnt** is used instead of **tr** it prints **a**.

.troff Make the **n** built-in condition false, and the **t** built-in condition true. This undoes the effect of the **nroff** request.

.unformat *xx*
This request 'unformats' the diversion *xx*. Contrary to the **asciify** request, which tries to convert formatted elements of the diversion back to input tokens as much as possible, **.unformat** only handles tabs and spaces between words (usually caused by spaces or newlines in the input) specially. The former are treated as if they were input tokens, and the latter are stretchable again. Note that the vertical size of lines is not preserved. Glyph information (font, font size, space width, etc.) is retained. Useful in conjunction with the **box** and **boxa** requests.

.vpt *n* Enable vertical position traps if *n* is non-zero, disable them otherwise. Vertical position traps are traps set by the **wh** or **dt** requests. Traps set by the **it** request are not vertical position traps. The parameter that controls whether vertical position traps are enabled is global. Initially vertical position traps are enabled.

.warn *n*
Control warnings. *n* is the sum of the numbers associated with each warning that is to be enabled; all other warnings are disabled. The number associated with each warning is listed in **troff**(1). For example, **.warn 0** disables all warnings, and **.warn 1** disables all warnings except that about missing glyphs. If *n* is not given, all warnings are enabled.

.warnscale *si*
Set the scaling indicator used in warnings to *si*. Valid values for *si* are **u**, **i**, **c**, **p**, and **P**. At startup, it is set to **i**.

.while *c anything*
While condition *c* is true, accept *anything* as input; *c* can be any condition acceptable to an **if** request; *anything* can comprise multiple lines if the first line starts with **\{** and the last line ends with **\}**. See also the **break** and **continue** requests.

.write *stream anything*
Write *anything* to the stream named *stream*. *stream* must previously have been the subject of an **open** request. *anything* is read in copy mode; a leading **"** is stripped.

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

.writem *stream xx*
Write the contents of the macro or string *xx* to the stream named *stream*. *stream* must previously have been the subject of an **open** request. *xx* is read in copy mode.

Extended escape sequences

\D'...**'** All drawing commands of groff's intermediate output are accepted. See subsection **Drawing Commands** below for more information.

Extended requests

.cf *filename*
When used in a diversion, this embeds in the diversion an object which, when reread, will cause the contents of *filename* to be transparently copied through to the output. In UNIX troff, the contents of *filename* is immediately copied through to the output regardless of whether there is a current diversion; this behaviour is so anomalous that it must be considered a bug.

.de *xx yy*
.am *xx yy*
.ds *xx yy*
.as *xx yy*
In compatibility mode, these requests behaves similar to **.de1**, **.am1**, **.ds1**, and **.as1**, respectively: A 'compatibility save' token is inserted at the beginning, and a 'compatibility restore' token at the end, with compatibility mode switched on during execution.

.ev *xx* If *xx* is not a number, this switches to a named environment called *xx*. The environment should be popped with a matching **ev** request without any arguments, just as for numbered environments. There is no limit on the number of named environments; they are created the first time that they are referenced.

.ss *m n* When two arguments are given to the **ss** request, the second argument gives the *sentence space size*. If the second argument is not given, the sentence space size is the same as the word space size. Like the word space size, the sentence space is in units of one twelfth of the spacewidth parameter for the current font. Initially both the word space size and the sentence space size are 12. Contrary to UNIX troff, GNU troff handles this request in nroff mode also; a given value is then rounded down to the nearest multiple of 12. The sentence space size is used in two circumstances. If the end of a sentence occurs at the end of a line in fill mode, then both an inter-word space and a sentence space are added; if two spaces follow the end of a sentence in the middle of a line, then the second space is a sentence space. Note that the behaviour of UNIX troff are exactly that exhibited by GNU troff if a second argument is never given to the **ss** request. In GNU troff, as in UNIX troff, you should always follow a sentence with either a newline or two spaces.

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

```
.ta T .5i
```

sets tabs every half an inch.

New number registers

The following read-only registers are available:

\n[.br] Within a macro call, it is set to 1 if the macro is called with the 'normal' control character ('.' by default), and set to 0 otherwise. This allows to reliably modify requests.

```
.als bp*orig bp .de bp .tm before bp .ie \\n[.br] .bp*orig .el 'bp*orig .tm after bp ..
```

Using this register outside of a macro makes no sense (it always returns zero in such cases).

\n[.C] 1 if compatibility mode is in effect, 0 otherwise.

\n[.cdp]
The depth of the last glyph added to the current environment. It is positive if the glyph extends below the baseline.

\n[.ce] The number of lines remaining to be centered, as set by the **ce** request.

\n[.cht] The height of the last glyph added to the current environment. It is positive if the glyph extends above the baseline.

\n[.color]
1 if colors are enabled, 0 otherwise.

\n[.csk]
The skew of the last glyph added to the current environment. The *skew* of a glyph is how far to the right of the center of a glyph the center of an accent over that glyph should be placed.

\n[.ev] The name or number of the current environment. This is a string-valued register.

\n[.fam]
The current font family. This is a string-valued register.

\n[.fn] The current (internal) real font name. This is a string-valued register. If the current font is a style, the value of **\n[.fn]** is the proper concatenation of family and style name.

\n[.fp] The number of the next free font position.

\n[.g] Always 1. Macros should use this to determine whether they are running under GNU troff.

\n[.height]
The current height of the font as set with **\H**.

\n[.hla] The current hyphenation language as set by the **hla** request.

\n[.hlc] The number of immediately preceding consecutive hyphenated lines.

\n[.hlm]
The maximum allowed number of consecutive hyphenated lines, as set by the **hlm** request.

\n[.hy] The current hyphenation flags (as set by the **hy** request).

\n[.hym]
The current hyphenation margin (as set by the **hym** request).

\n[.hys]
The current hyphenation space (as set by the **hys** request).

\n[.in] The indentation that applies to the current output line.

\n[.int] Set to a positive value if last output line is interrupted (i.e., if it contains **\c**).

\n[.kern]
1 if pairwise kerning is enabled, 0 otherwise.

\n[.lg] The current ligature mode (as set by the **lg** request).

\n[.linetabs]
The current line-tabs mode (as set by the **linetabs** request).

\n[.ll] The line length that applies to the current output line.

\n[.lt] The title length as set by the **lt** request.

\n[.m] The name of the current drawing color. This is a string-valued register.

\n[.M] The name of the current background color. This is a string-valued register.

\n[.ne] The amount of space that was needed in the last **ne** request that caused a trap to be sprung. Useful in conjunction with the **\n[.trunc]** register.

\n[.ns] 1 if no-space mode is active, 0 otherwise.

\n[.pe] 1 during a page ejection caused by the **bp** request, 0 otherwise.

\n[.pn] The number of the next page, either the value set by a **pn** request, or the number of the current page plus 1.

\n[.ps] The current point size in scaled points.

\n[.psr]
The last-requested point size in scaled points.

\n[.pvs]
The current post-vertical line space as set with the **pvs** request.

\n[.rj] The number of lines to be right-justified as set by the **rj** request.

\n[.slant]
The slant of the current font as set with **\S**.

\n[.sr] The last requested point size in points as a decimal fraction. This is a string-valued register.

\n[.ss]
\n[.sss] These give the values of the parameters set by the first and second arguments of the **ss** request.

\n[.sty] The current font style. This is a string-valued register.

\n[.tabs]
A string representation of the current tab settings suitable for use as an argument to the **ta** request.

\n[.trunc]
The amount of vertical space truncated by the most recently sprung vertical position trap, or, if the trap was sprung by a **ne** request, minus the amount of vertical motion produced by the **ne** request. In other words, at the point a trap is sprung, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is. Useful in conjunction with the **\n[.ne]** register.

\n[.U] Set to 1 if in safer mode and to 0 if in unsafe mode (as given with the **−U** command line option).

\n[.vpt]
1 if vertical position traps are enabled, 0 otherwise.

\n[.warn]
The sum of the numbers associated with each of the currently enabled warnings. The number associated with each warning is listed in **troff**(1).

\n[.x] The major version number. For example, if the version number is 1.03, then **\n[.x]** contains 1.

\n[.y] The minor version number. For example, if the version number is 1.03, then **\n[.y]** contains 03.

\n[.Y] The revision number of groff.

\n[.zoom]
The zoom value of the current font, in multiples of 1/1000th. Zero if no magnification.

\n[llx]
\n[lly]
\n[urx]
\n[ury] These four registers are set by the **psbb** request and contain the bounding box values (in PostScript units) of a given PostScript image.

The following read/write registers are set by the **\w** escape sequence:

\n[rst]
\n[rsb] Like the **st** and **sb** registers, but take account of the heights and depths of glyphs.

\n[ssc] The amount of horizontal space (possibly negative) that should be added to the last glyph before a subscript.

\n[skw]
How far to right of the center of the last glyph in the **\w** argument, the center of an accent from a roman font should be placed over that glyph.

Other available read/write number registers are:

\n[c.] The current input line number. **\n[.c]** is a read-only alias to this register.

\n[hours]
The number of hours past midnight. Initialized at start-up.

\n[hp] The current horizontal position at input line.

\n[minutes]
The number of minutes after the hour. Initialized at start-up.

\n[seconds]
The number of seconds after the minute. Initialized at start-up.

\n[systat]
The return value of the system() function executed by the last **sy** request.

\n[slimit]
If greater than 0, the maximum number of objects on the input stack. If less than or equal to 0, there is no limit on the number of objects on the input stack. With no limit, recursion can continue until virtual memory is exhausted.

\n[year]
The current year. Note that the traditional **troff** number register **\n[yr]** is the current year minus 1900.

Miscellaneous

troff predefines a single (read/write) string-based register, ***[.T]**, which contains the argument given to the **−T** command line option, namely the current output device (for example, *latin1* or *ascii*). Note that this is not the same as the (read-only) number register **\n[.T]** which is defined to be 1 if **troff** is called with the **−T** command line option, and zero otherwise. This behaviour is different to UNIX troff.

Fonts not listed in the DESC file are automatically mounted on the next available font position when they are referenced. If a font is to be mounted explicitly with the **fp** request on an unused font position, it should be mounted on the first unused font position, which can be found in the **\n[.fp]** register; although **troff** does not enforce this strictly, it does not allow a font to be mounted at a position whose number is much greater than that of any currently used position.

Interpolating a string does not hide existing macro arguments. Thus in a macro, a more efficient way of doing

.*xx* **\\$@**

is

\*[*xx***]**

If the font description file contains pairwise kerning information, glyphs from that font are kerned. Kerning between two glyphs can be inhibited by placing a **\&** between them.

In a string comparison in a condition, characters that appear at different input levels to the first delimiter character are not recognized as the second or third delimiters. This applies also to the **tl** request. In a **\w** escape sequence, a character that appears at a different input level to the starting delimiter character is not recognized as the closing delimiter character. The same is true for **\A**, **\b**, **\B**, **\C**, **\l**, **\L**, **\o**, **\X**, and **\Z**. When decoding a macro or string argument that is delimited by double quotes, a character that appears at a different input level to the starting delimiter character is not recognized as the closing delimiter character. The implementation of **\$@** ensures that the double quotes surrounding an argument appear at the same input level, which is different to the input level of the argument itself. In a long escape name **]** is not recognized as a closing delimiter except when it occurs at the same input level as the opening **]**. In compatibility mode, no attention is paid to the input-level.

There are some new types of condition:

.if r*xxx* True if there is a number register named *xxx*.

.if d*xxx* True if there is a string, macro, diversion, or request named *xxx*.

.if m*xxx*
True if there is a color named *xxx*.

.if c*ch* True if there is a character (or glyph) *ch* available; *ch* is either an ASCII character or a glyph (special character) **\N'***xxx***'**, **\(***xx* or **\[***xxx***]**; the condition is also true if *ch* has been defined by the **char** request.

.if F*f* True if font *f* exists. **f** is handled as if it was opened with the **ft** request (this is, font translation and styles are applied), without actually mounting it.

.if S*s* True if style *s* has been registered. Font translation is applied.

The **tr** request can now map characters onto **\ .**

The space width emitted by the **\|** and **\^** escape sequences can be controlled on a per-font basis. If there is a glyph named **\|** or **\^**, respectively (note the leading backslash), defined in the current font file, use this glyph's width instead of the default value.

It is now possible to have whitespace between the first and second dot (or the name of the ending macro) to end a macro definition. Example:

```
.if t \{\ .  de bar .    nop Hello, I'm ‘bar’. . .  .\}
```

INTERMEDIATE OUTPUT FORMAT

This section describes the format output by GNU troff. The output format used by GNU troff is very similar to that used by Unix device-independent troff. Only the differences are documented here.

Units

The argument to the **s** command is in scaled points (units of points/*n*, where *n* is the argument to the **sizescale** command in the DESC file). The argument to the **x Height** command is also in scaled points.

Text Commands

N*n* Print glyph with index *n* (a non-negative integer) of the current font.

If the **tcommand** line is present in the DESC file, troff uses the following two commands.

t*xxx* *xxx* is any sequence of characters terminated by a space or a newline (to be more precise, it is a sequence of glyphs which are accessed with the corresponding characters); the first character should be printed at the current position, the current horizontal position should be increased by the width of the first character, and so on for each character. The width of the glyph is that given in the font file, appropriately scaled for the current point size, and rounded so that it is a multiple of the horizontal resolution. Special characters cannot be printed using this command.

u*n xxx* This is same as the **t** command except that after printing each character, the current horizontal position is increased by the sum of the width of that character and *n*.

Note that single characters can have the eighth bit set, as can the names of fonts and special characters.

The names of glyphs and fonts can be of arbitrary length; drivers should not assume that they are only two characters long.

When a glyph is to be printed, that glyph is always in the current font. Unlike device-independent troff, it is not necessary for drivers to search special fonts to find a glyph.

For color support, some new commands have been added:

mc *cyan magenta yellow*
md
mg *gray*
mk *cyan magenta yellow black*
mr *red green blue*
Set the color components of the current drawing color, using various color schemes. **md** resets the drawing color to the default value. The arguments are integers in the range 0 to 65536.

The **x** device control command has been extended.

x u *n* If *n* is 1, start underlining of spaces. If *n* is 0, stop underlining of spaces. This is needed for the **cu** request in nroff mode and is ignored otherwise.

Drawing Commands

The **D** drawing command has been extended. These extensions are not used by GNU pic if the **–n** option is given.

Df *n*\n Set the shade of gray to be used for filling solid objects to *n*; *n* must be an integer between 0 and 1000, where 0 corresponds solid white and 1000 to solid black, and values in between correspond to intermediate shades of gray. This applies only to solid circles, solid ellipses and solid polygons. By default, a level of 1000 is used. Whatever color a solid object has, it should completely obscure everything beneath it. A value greater than 1000 or less than 0 can also be used: this means fill with the shade of gray that is currently being used for lines and text. Normally this is black, but some drivers may provide a way of changing this.

The corresponding **\D'f**...**'** command shouldn't be used since its argument is always rounded to an integer multiple of the horizontal resolution which can lead to surprising results.

DC *d*\n Draw a solid circle with a diameter of *d* with the leftmost point at the current position.

DE *dx dy*\n
Draw a solid ellipse with a horizontal diameter of *dx* and a vertical diameter of *dy* with the leftmost point at the current position.

Dp dx_1 dy_1 dx_2 dy_2 $\cdots$ dx_n dy_n\n
Draw a polygon with, for $i = 1, \ldots, n+1$, the i-th vertex at the current position $+ \sum_{j=1}^{i-1}(dx_j, dy_j)$. At the moment, GNU pic only uses this command to generate triangles and rectangles.

DP dx_1 dy_1 dx_2 dy_2 $\cdots$ dx_n dy_n\n
Like **Dp** but draw a solid rather than outlined polygon.

Dt *n*\n Set the current line thickness to *n* machine units. Traditionally Unix troff drivers use a line thickness proportional to the current point size; drivers should continue to do this if no **Dt** command has been given, or if a **Dt** command has been given with a negative value of *n*. A zero value of *n* selects the smallest available line thickness.

A difficulty arises in how the current position should be changed after the execution of these commands. This is not of great importance since the code generated by GNU pic does not depend on this. Given a drawing command of the form

\D'*c* x_1 y_1 x_2 y_2 $\cdots$ x_n y_n**'**

where *c* is not one of **c**, **e**, **l**, **a**, or , Unix troff treats each of the x_i as a horizontal quantity, and each of the y_i as a vertical quantity and assumes that the width of the drawn object is $\sum_{i=1}^{n} x_i$, and that the height is $\sum_{i=1}^{n} y_i$. (The assumption about the height can be seen by examining the **st** and **sb** registers after using such a **D** command in a **\w** escape sequence). This rule also holds for all the original drawing commands with the exception of **De**. For the sake of compatibility GNU troff also follows this rule, even though it produces an ugly result in the case of the **Dt** and **Df**, and, to a lesser extent, **DE** commands. Thus after executing a **D** command of the form

D*c* x_1 y_1 x_2 y_2 $\cdots$ x_n y_n\n the current position should be increased by $(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i)$.

Another set of extensions is

DFc *cyan magenta yellow*\n
DFd\n
DFg *gray*\n
DFk *cyan magenta yellow black*\n
DFr *red green blue*\n
Set the color components of the filling color similar to the **m** commands above.

The current position isn't changed by those colour commands (contrary to **Df**).

Device Control Commands

There is a continuation convention which permits the argument to the **x X** command to contain newlines: when outputting the argument to the **x X** command, GNU troff follows each newline in the argument with a **+** character (as usual, it terminates the entire argument with a newline); thus if the line after the line containing the **x X** command starts with **+**, then the newline ending the line containing the **x X** command should be treated as part of the argument to the **x X** command, the **+** should be ignored, and the part of the line following the **+** should be treated like the part of the line following the **x X** command.

The first three output commands are guaranteed to be:

x T *device*
x res *n h v*
x init

INCOMPATIBILITIES

In spite of the many extensions, groff has retained compatibility to classical troff to a large degree. For the cases where the extensions lead to collisions, a special compatibility mode with the restricted, old functionality was created for groff.

Groff Language

groff provides a **compatibility mode** that allows to process roff code written for classical **troff** or for other implementations of roff in a consistent way.

Compatibility mode can be turned on with the **−C** command line option, and turned on or off with the **.cp** request. The number register **\n(.C** is 1 if compatibility mode is on, 0 otherwise.

This became necessary because the GNU concept for long names causes some incompatibilities. *Classical troff* interprets

.dsabcd

as defining a string **ab** with contents **cd**. In *groff* mode, this is considered as a call of a macro named **dsabcd**.

Also *classical troff* interprets ***[** or **\n[** as references to a string or number register called **[** while *groff* takes this as the start of a long name.

In *compatibility mode*, groff interprets these things in the traditional way; so long names are not recognized.

On the other hand, groff in *GNU native mode* does not allow to use the single-character escapes \\ (backslash), **\|** (vertical bar), **\^** (caret), **\&** (ampersand), **\{** (opening brace), **\}** (closing brace), '**** ' (space), **\'** (single quote), **\`** (backquote), **\−** (minus), **_** (underline), **\!** (bang), **\%** (percent), and **\c** (character c) in names of strings, macros, diversions, number registers, fonts or environments, whereas *classical troff* does.

The **\A** escape sequence can be helpful in avoiding these escape sequences in names.

Fractional point sizes cause one noteworthy incompatibility. In *classical troff*, the **ps** request ignores scale indicators and so

.ps 10u

sets the point size to 10 points, whereas in groff native mode the point size is set to 10 scaled points.

In *groff*, there is a fundamental difference between unformatted input characters, and formatted output characters (glyphs). Everything that affects how a glyph is output is stored with the glyph; once a glyph has been constructed it is unaffected by any subsequent requests that are executed, including the **bd**, **cs**, **tkf**, **tr**, or **fp** requests.

Normally glyphs are constructed from input characters at the moment immediately before the glyph is added to the current output line. Macros, diversions and strings are all, in fact, the same type of object; they contain lists of input characters and glyphs in any combination.

Special characters can be both; before being added to the output, they act as input entities, afterwards they denote glyphs.

A glyph does not behave like an input character for the purposes of macro processing; it does not inherit any of the special properties that the input character from which it was constructed might have had. The following example makes things clearer.

```
        .di x \\\\ .br .di .x
```

With *GNU troff* this is printed as \\. So each pair of input backslashes '\\' is turned into a single output backslash glyph '\' and the resulting output backslashes are not interpreted as escape characters when they are reread.

Classical troff would interpret them as escape characters when they were reread and would end up printing a single backslash '\'.

In GNU, the correct way to get a printable version of the backslash character '\' is the **\(rs** escape sequence, but classical troff does not provide a clean feature for getting a non-syntactical backslash. A close method is the printable version of the current escape character using the **\e** escape sequence; this works if the current escape character is not redefined. It works in both GNU mode and compatibility mode, while dirty tricks like specifying a sequence of multiple backslashes do not work reliably; for the different handling in diversions, macro definitions, or text mode quickly leads to a confusion about the necessary number of backslashes.

To store an escape sequence in a diversion that is interpreted when the diversion is reread, either the traditional **\!** transparent output facility or the new **\?** escape sequence can be used.

Intermediate Output

The groff intermediate output format is in a state of evolution. So far it has some incompatibilities, but it is intended to establish a full compatibility to the classical troff output format. Actually the following incompatibilities exist:

- The positioning after the drawing of the polygons conflicts with the classical definition.
- The intermediate output cannot be rescaled to other devices as classical 'device-independent' troff did.

AUTHORS

Copyright (C) 1989, 2001, 2002, 2003, 2004, 2006, 2007, 2008, 2009 Free Software Foundation, Inc.

This document is distributed under the terms of the FDL (GNU Free Documentation License) version 1.3 or later. You should have received a copy of the FDL on your system, it is also available on-line at the GNU copyleft site This document was written by James Clark, with modifications by Werner Lemberg and Bernd Warken

This document is part of *groff*, the GNU roff distribution. Formerly, the contents of this document was kept in the manual page **troff**(1). Only the parts dealing with the language aspects of the different *roff* systems were carried over into this document. The *troff* command line options and warnings are still documented in **troff**(1).

SEE ALSO

The *groff info file*, cf. **info**(1) presents all groff documentation within a single document.

groff(1)
A list of all documentation around *groff*.

groff(7)
A description of the *groff* language, including a short, but complete reference of all predefined requests, registers, and escapes of plain *groff*. From the command line, this is called using

```
man 7 groff
```

roff(7) A survey of *roff* systems, including pointers to further historical documentation.

[*CSTR #54*]
The *Nroff/Troff User's Manual* by *J. F. Ossanna* of 1976 in the revision of *Brian Kernighan* of 1992, being the classical troff documentation

GROFF_HDTBL

NAME

groff_hdtbl – groff ‘hdtbl’ macros for generation of tables

DESCRIPTION

The **hdtbl** macros consist of four base and three optional macros, controlled by about twenty arguments. The syntax is simple and similar to the **HTML** table model and nearly as flexible: You can write sequences of tokens (macro calls with their arguments and content data), separated by blanks and beginning with a macro call, into the same line to get compact and cleanly arrranged input. An advantage of **hdtbl** is that the tables are constructed without calling a preprocessor; this means that **groff**’s full macro capabilities are available. On the other hand, table processing with **hdtbl** is much slower than using the **tbl**(@MAN1EXT) preprocessor. A further advantage is that the HTML-like syntax of **hdtbl** will be easily converted to HTML; this is not implemented yet.

USAGE

The simplest well-formed table consists of just single calls to the four base table macros in the right order. Here we construct a table with only one cell.

```
.TBL
.TR
.TD
contents of the table cell
.ETB
```

Equivalent to the above is the following notation.

```
.TBL .TR .TD contents of the table cell .ETB
```

By default, the formatted table is inserted into the surrounding text at the place of its definition. If the vertical space isn’t sufficient, it is placed at the top of the next page. Tables can also be stored for later insertion.

Using ‘*row-number*column-number*’ as the data for the table cells, a table with two rows and two columns can be written as

```
.TBL cols=2
.  TR .TD 1*1 .TD 1*2
.  TR .TD 2*1 .TD 2*2
.ETB
```

Here we see a difference to HTML tables: The number of columns must be explicitly specified using the ‘cols=*m*’ argument (or indirectly via the ‘width’ argument, see below).

The contents of a table cell is arbitrary; for example, it can be another table, without restriction to the nesting depth. A given table layout can be either constructed with suitably nested tables or with proper arguments to .TD and .TH, controlling column and row spanning. Note, however, that this table

```
.TBL
.  TR
.    TD
.      nop 1*1 1*2
.  TR
.    TD
.      TBL cols=2 border=
.        TR
.          TD
.            nop 2*1
.          TD
.            nop 2*2
.      ETB
.ETB
```

and this table

```
.TBL cols=2
.  TR
.     TD colspan=2
.       nop 1*1 1*2
.   TR
.     TD
.       nop 2*1
.     TD
.       nop 2*2
.ETB
```

are similar but not identical.

Here the latter table in a more compact form.

```
.TBL cols=2 .TR ".TD colspan=2" 1*1 1*2
.             TR .TD 2*1 .TD 2*2 .ETB
```

If a macro has one or more arguments, and it is not starting a line, it must be enclosed in double quotes.

MACROS AND ARGUMENTS

The order of macro calls and other tokens follows the HTML model. In the following list, valid predecessors and successors of all **hdtbl** macros are given, together with the possible arguments.

Macro arguments are separated by blanks. The order of arguments is arbitrary; they are of the form

`key=`*`value`*

or

`key='`*`value1`* [*`value2`* [...]]`'`

with the only exception of the optional argument of the macro `.ETB`, which is the string '`hold`'. Another possible form is

`"key=`*`value1`* [*`value2`* [...]]`"`

However, this is limited to the case where the macro is the first one in the line and not already enclosed in double quotes.

Argument values specified below as *c* are colors predefined by **groff** or colors defined by the user with the `.defcolor` request. Argument values *d* are decimal numbers with or without decimal point. Argument values *m* are natural numbers. Argument values *n* are numerical values with the usual **groff** scaling indicators. Some of the arguments are specific to one or two macros, but most of them can be specified with `.TBL`, `.TR`, `.TD`, and `.TH.` These common arguments are explained in the next subsection.

Most of the argument default values can be changed by the user by setting corresponding default registers or strings, as listed below.

`.TBL` [*args*]

Begin a new table.

predecessor: `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.CPTN`, `.TR`
arguments:

`border=`[*n*]
: Thickness of the surrounding box border. '`border=`' (no value) means neither a surrounding box border nor any horizontal or vertical separator lines between the table rows and cells. '`border=0`' suppresses the surrounding box border, but still allows separator lines between cells and rows.
Default: '`border=.1n`' (register '`t*b`').

`bc=`*c*
: Border color.
Default: '`bc=red4`' (string '`t*bc`').

`cols=`*m*
: Number of table columns. This argument is necessary if more than one column is in the table and no '`width`' arguments are present.
Default: '`cols=1`' (register '`t*cols`').

`cpd=`*n*
: Cell padding, i.e., the extra space between the cell space border and the cell contents.

Default: '`cpd=.5n`' (register '`t*cpd`').

`csp=`*n* Cell spacing, i.e., the extra space between the table border or vertical or horizontal lines between cells and the cellspace.
Default: '`csp=.5n`' (register '`t*csp`').

`tal=l|c|r`
Horizontal alignment of the table, if it is smaller than the line width. '`tal=l`': left alignment. '`tal=c`': centered alignment. '`tal=r`': right alignment.
Default: '`tal=l`' (register '`t*tal`').

`width='`*w1* [*w2* [...]]`'`
Widths of table cells. *w1*, *w2*, ... are either numbers of type *n* or natural numbers with the pseudo-scaling indicator '`%`', with the meaning "percent of the actual line length (or column length for inner tables, respectively)". If there are less width values than table columns, the last width value is used for the remaining cells. The argument

```
width='1.5i 10%'
```

for example indicates that the first column is 1.5 inches wide; the remaining columns take 1/10 of the column length each.
Default: The table width equals the outer line length or column length; the columns have equal widths.

`height=`*n*
Height of the table. If the table with its contents is lower than *n*, the last row is stretched to this value.

.CPTN [*args*]
Text of caption.

The (optionally numbered) table caption. `.CPTN` is optional.

predecessor: `.TBL`
successor: `.TR`
arguments:

`val=t|b`
Vertical alignment of the table caption. '`val=t`': The caption is placed above the table. '`val=b`': The caption is placed below the table.
Default: '`val=t`' (string '`t*cptn`').

.TR [*args*]
Begin a new table row.

predecessor: `.TBL`, `.CPTN`, `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.TD`, `.TH`
arguments:

`height=`*n*
The height of the row. If a cell in the row is higher than *n* this value is ignored; otherwise the row height is stretched to *n*.

.TD [*args* [*cell contents*]]
Begin a table data cell.

.TH [*args* [*cell contents*]]
Begin a table header cell.

Arguments and cell contents can be mixed. The macro `.TH` is not really necessary and differs from `.TD` only in three default settings, similar to the `<TH>` and `<TD>` HTML tags: The contents of `.TH` is horizontally and vertically centered and typeset in boldface.

predecessor: `.TR`, `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.TD`, `.TH`, `.TR`, `.ETB`, `cell contents`
arguments:

`colspan=`*m*
The width of this cell is the sum of the widths of the *m* cells above and

below this row.

`rowspan=`*m*
: The height of this cell is the sum of the heights of the *m* cells left and right of this column.

  Remark: Overlapping of column and row spanning, as in the following table fragment (the overlapping happens in the second cell in the second row), is invalid and causes incorrect results.

```
.TR .TD 1*1 ".TD 1*2 rowspan=2" .TD 1*3
.TR ".TD 2*1 colspan=2"         .TD 2*3
```

.ETB [**hold**]

End of the table.

This macro finishes a table. It causes one of the following actions.

- If the argument '`hold`' is given, the table is held until it is freed by calling the macro `.t*free`, which in turn prints the table immediately, either at the current position or at the top of the next page if its height is larger than the remaining space on the page.
- Otherwise, if the table is higher than the remaining space on the page, it is printed at the top of the next page.
- If none of the two above constraints hold, the table is printed immediately at the place of its definition.

predecessor: `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.TBL`, `.TR`, `.TD`, `.TH`, `.ETB`, `cell contents`
arguments:

`hold` Prevent the table from being printed until it is freed by calling the macro `.t*free`. This argument is ignored for inner (nested) tables.

.t*free [*n*]

Free the next held table or *n* held tables. Call this utility macro to print tables which are held by using the '`hold`' argument of the `.ETB` macro.

Arguments common to .TBL, .TR, .TD, and .TH

The arguments described in this section can be specified with the `.TBL` and `.TR` macros, but they are eventually passed on to the table cells. If omitted, the defaults take place, which the user can change by setting the corresponding default registers or strings, as documented below. Setting an argument with the `.TBL` macro has the same effect as setting it for all rows in the table. Setting an argument with a `.TR` macro has the same effect as setting it for all the `.TH` or `.TD` macro in this row.

`bgc=`[*c*]
: The background color of the table cells. This includes the area specified with the '`csp`' argument. The argument '`bgc=`' (no value) suppresses a background color; this makes the background transparent.
  Default: '`bgc=bisque`' (string '`t*bgc`').

`fgc=`*c* The foreground color of the cell contents.
  Default: '`fgc=red4`' (string '`t*fgc`').

`ff=`*name*
: The font family for the table. *name* is one of the groff font families, for example `A` for the AvantGarde fonts or `HN` for Helvetica-Narrow.
  Default: The font family found before the table (string '`t*ff`').

`fst=`*style*
: The font style for the table. One of `R`, `I`, `B`, or `BI` for roman, **bold**, *italic*, or ***bold italic***, respectively. As with **roff**'s `.ft` request the '`fst`' argument can be used to specify the font family and font style together, for example '`fst=HNBI`' instead of '`ff=HN`' and '`fst=BI`'.
  Default: The font style in use right before the table (string '`t*fst`').

`fsz='`*d1* [*d2*]`'`
: A decimal or fractional factor *d1*, by which the point size for the table is changed, and *d2*, by which the vertical line spacing is changed. If *d2* is omitted, value *d1* is taken for both.
  Default: '`fsz='1.0 1.0'`' (string '`t*fsz`').

`hal=l|c|b|r`
: Horizontal alignment of the cell contents in the table. ‘`hal=l`’: left alignment. ‘`hal=c`’: centered alignment. ‘`hal=b`’: both (left and right) alignment. ‘`hal=r`’: right alignment.
Default: ‘`hal=b`’ (string ‘`t*hal`’).

`val=t|m|b`
: Vertical alignment of the cell contents in the table for cells lower than the current row. ‘`val=t`’: alignment below the top of the cell. ‘`val=m`’: alignment in the middle of the cell. ‘`val=b`’: alignment above the cell bottom.
Default: ‘`val=t`’ (string ‘`t*val`’).

`hl=[s|d]`
: Horizontal line between the rows. If specified with `.TD` or `.TH` this is a separator line to the cell below. ‘`hl=`’ (no value): no separator line. ‘`hl=s`’: a single separator line between the rows. ‘`hl=d`’: a double separator line.

The thickness of the separator lines is the half of the border thickness, but at least 0.1 inches. The distance between the double lines is equal to the line thickness.

Remark: Together with ‘`border=0`’ for proper formatting the value of ‘`csp`’ must be at least .05 inches for single separator lines and .15 inches for double separator lines.
Default: ‘`hl=s`’ (string ‘`t*hl`’).

`vl=[s|d]`
: Vertical separator line between the cells. If specified with `.TD` or `.TH` this is a separator line to the cell on the right. ‘`vl=s`’: a single separator line between the cells. ‘`vl=d`’: a double separator line. ‘`vl=`’ (no value): no vertical cell separator lines. For more information see the documentation of the ‘`hl`’ argument above.
Default: ‘`vl=s`’ (string ‘`t*vl`’).

HDTBL CUSTOMIZATION

A table which does not fit on a partially filled page is printed automatically on the top of the next page if you append the little utility macro `t*hm` to the page header macro of your document’s main macro package. For example, say

```
.am pg@top
.  t*hm
..
```

if you use the **ms** macro package.

hdtbl has built-in page header and page footer macros, **HM** and **BM**. If they interfere with your own header and footer macros, simply say `.rm HM` and `.rm BM` to remove them.

AUTHOR

Joachim Walsdorff

BUGS AND SUGGESTIONS

Please send your commments to the groff mailing list or directly to the author.

GROFF_MAN

NAME

groff_man – groff ‘man’ macros to support generation of man pages

SYNOPSIS

[*options* . . .] [*files* . . .] [*options* . . .] [*files* . . .]

DESCRIPTION

The **man** macros used to generate *man pages* with *groff* were written by James Clark. This document provides a brief summary of the use of each macro in that package.

OPTIONS

The **man** macros understand the following command line options (which define various registers).

–rcR=1
This option (the default if in nroff mode) creates a single, very long page instead of multiple pages. Say **–rcR=0** to disable it.

–rC1 If more than one manual page is given on the command line, number the pages continuously, rather than starting each at 1.

–rD1 Double-sided printing. Footers for even and odd pages are formatted differently.

–rFT=*dist*
Set distance of the footer relative to the bottom of the page if negative or relative to the top if positive. The default is -0.5i.

–rHY=*flags*
Set hyphenation flags. Possible values are 1 to hyphenate without restrictions, 2 to not hyphenate the last word on a page, 4 to not hyphenate the last two characters of a word, and 8 to not hyphenate the first two characters of a word. These values are additive; the default is 14.

–rIN=*width*
Set body text indentation to *width*. The default is 7n for *nroff*, 7.2n for *troff*. For *nroff*, this value should always be an integer multiple of unit ‘n’ to get consistent indentation.

–rLL=*line-length*
Set line length. If this option is not given, the line length is set to respect any value set by a prior ‘.ll’ request, (which *must* be in effect when the ‘.TH’ macro is invoked), if this differs from the built–in default for the formatter; otherwise it defaults to 78n in *nroff* mode and 6.5i in *troff* mode.

Note that the use of a ‘.ll’ request to initialize the line length is supported for backward compatibility with some versions of the **man** program; direct initialization of the ‘LL’ register should *always* be preferred to the use of such a request. In particular, note that a ‘.ll 65n’ request does *not* preserve the normal *nroff* default line length, (the **man** default initialization to 78n prevails), whereas, the ‘-rLL=65n’ option, or an equivalent ‘.nr LL 65n’ request preceding the use of the ‘TH’ macro, *does* set a line length of 65n.

–rLT=*title-length*
Set title length. If this option is not given, the title length defaults to the line length.

–rP*nnn*
Enumeration of pages start with *nnn* rather than with 1.

–rS*xx* Base document font size is *xx* points (*xx* can be 10, 11, or 12) rather than 10 points.

–rSN=*width*
Set sub-subheading indentation to *width*. The default is 3n.

–rX*nnn*
After page *nnn*, number pages as *nnn*a, *nnn*b, *nnn*c, etc. For example, the option ‘–rX2’ produces the following page numbers: 1, 2, 2a, 2b, 2c, etc.

USAGE

This section describes the available macros for manual pages. For further customization, put additional macros and requests into the file **man.local** which is loaded immediately after the **man** package.

.TH *title section* [*extra1*] [*extra2*] [*extra3*]

Set the title of the *man page* to *title* and the section to *section*, which must take on a value between 1 and 8. The value *section* may also have a string appended, e.g. '.pm', to indicate a specific subsection of the *man pages*. Both *title* and *section* are positioned at the left and right in the header line (with *section* in parentheses immediately appended to *title*. *extra1* is positioned in the middle of the footer line. *extra2* is positioned at the left in the footer line (or at the left on even pages and at the right on odd pages if double-sided printing is active). *extra3* is centered in the header line.

For HTML output, headers and footers are completely suppressed.

Additionally, this macro starts a new page; the new line number is 1 again (except if the '-rC1' option is given on the command line) -- this feature is intended only for formatting multiple *man pages*; a single *man page* should contain exactly one **TH** macro at the beginning of the file.

.SH [*text for a heading*]

Set up an unnumbered section heading sticking out to the left. Prints out all the text following **SH** up to the end of the line (or the text in the next input line if there is no argument to **SH**) in bold face (or the font specified by the string **HF**), one size larger than the base document size. Additionally, the left margin and the indentation for the following text is reset to the default values.

.SS [*text for a heading*]

Set up a secondary, unnumbered section heading. Prints out all the text following **SS** up to the end of the line (or the text in the next input line if there is no argument to **SS**) in bold face (or the font specified by the string **HF**), at the same size as the base document size. Additionally, the left margin and the indentation for the following text is reset to the default values.

.TP [*nnn*]

Set up an indented paragraph with label. The indentation is set to *nnn* if that argument is supplied (the default unit is 'n' if omitted), otherwise it is set to the previous indentation value specified with **TP**, **IP**, or **HP** (or to the default value if none of them have been used yet).

The first input line of text following this macro is interpreted as a string to be printed flush-left, as it is appropriate for a label. It is not interpreted as part of a paragraph, so there is no attempt to fill the first line with text from the following input lines. Nevertheless, if the label is not as wide as the indentation the paragraph starts at the same line (but indented), continuing on the following lines. If the label is wider than the indentation the descriptive part of the paragraph begins on the line following the label, entirely indented. Note that neither font shape nor font size of the label is set to a default value; on the other hand, the rest of the text has default font settings.

The **TP** macro is the macro used for the explanations you are just reading.

.TQ The **TQ** macro sets up header continuation for a .TP macro. With it, you can stack up any number of labels (such as in a glossary, or list of commands) before beginning the indented paragraph. For an example, look just past the next paragraph.

This macro is not defined on legacy Unix systems running classic troff. To be certain your page will be portable to those systems, copy its definition from the **an-ext.tmac** file of a **groff** installation.

.LP
.PP
.P These macros are mutual aliases. Any of them causes a line break at the current position, followed by a vertical space downwards by the amount specified by the **PD** macro. The font size and shape are reset to the default value (normally 10pt Roman). Finally, the current left margin and the indentation are restored.

.IP [*designator*] [*nnn*]

Set up an indented paragraph, using *designator* as a tag to mark its beginning. The indentation is set to *nnn* if that argument is supplied (the default unit is ‘n’ if omitted), otherwise it is set to the previous indentation value specified with **TP**, **IP**, or **HP** (or to the default value if none of them have been used yet). Font size and face of the paragraph (but not the designator) are reset to its default values.

To start an indented paragraph with a particular indentation but without a designator, use ‘""’ (two doublequotes) as the second argument.

For example, the following paragraphs were all set up with bullets as the designator, using ‘.IP \(bu 4’. The whole block has been enclosed with ‘.RS’ and ‘.RE’ to set the left margin temporarily to the current indentation value.

- **IP** is one of the three macros used in the **man** package to format lists.
- **HP** is another. This macro produces a paragraph with a left hanging indentation.
- **TP** is another. This macro produces an unindented label followed by an indented paragraph.

.HP [*nnn*]

Set up a paragraph with hanging left indentation. The indentation is set to *nnn* if that argument is supplied (the default unit is ‘n’ if omitted), otherwise it is set to the previous indentation value specified with **TP**, **IP**, or **HP** (or to the default value if none of them have been used yet). Font size and face are reset to its default values. The following paragraph illustrates the effect of this macro with hanging indentation set to 4 (enclosed by **.RS** and **.RE** to set the left margin temporarily to the current indentation):

This is a paragraph following an invocation of the **HP** macro. As you can see, it produces a paragraph where all lines but the first are indented.

Use of this presentation-level macro is deprecated. While it is universally portable to legacy Unix systems, a hanging indentation cannot be expressed naturally under HTML, and many HTML-based manual viewers simply interpret it as a starter for a normal paragraph. Thus, any information or distinction you tried to express with the indentation may be lost.

.RS [*nnn*]

This macro moves the left margin to the right by the value *nnn* if specified (default unit is ‘n’); otherwise it is set to the previous indentation value specified with **TP**, **IP**, or **HP** (or to the default value if none of them have been used yet). The indentation value is then set to the default.

Calls to the **RS** macro can be nested.

.RE [*nnn*]

This macro moves the left margin back to level *nnn*, restoring the previous left margin. If no argument is given, it moves one level back. The first level (i.e., no call to **RS** yet) has number 1, and each call to **RS** increases the level by 1.

.EX
.EE

Example/End Example. After **EX**, filling is disabled and the font is set to constant-width. This is useful for formatting code, command, and configuration-file examples. The **EE** macro restores the previous font.

These macros are defined on many (but not all) legacy Unix systems running classic troff. To be certain your page will be portable to those systems, copy their definitions from the **an-ext.tmac** file of a **groff** installation.

To summarize, the following macros cause a line break with the insertion of vertical space (which amount can be changed with the **PD** macro): **SH**, **SS**, **TP**, **TQ**, **LP** (**PP**, **P**), **IP**, and **HP**. The macros **RS**, **RE**, **EX**, and **EE** also cause a break but no insertion of vertical space.

MACROS TO SET FONTS

The standard font is Roman; the default text size is 10 point.

.SM [*text*]

Causes the text on the same line or the text on the next input line to appear in a font that is one point size smaller than the default font.

.SB [*text*]
Causes the text on the same line or the text on the next input line to appear in boldface font, one point size smaller than the default font.

.BI *text* Causes text on the same line to appear alternately in bold face and italic. The text must be on the same line as the macro call. Thus

```
.BI this "word and" that
```

would cause 'this' and 'that' to appear in bold face, while 'word and' appears in italics.

.IB *text* Causes text to appear alternately in italic and bold face. The text must be on the same line as the macro call.

.RI *text* Causes text on the same line to appear alternately in roman and italic. The text must be on the same line as the macro call.

.IR *text* Causes text on the same line to appear alternately in italic and roman. The text must be on the same line as the macro call.

.BR *text*
Causes text on the same line to appear alternately in bold face and roman. The text must be on the same line as the macro call.

.RB *text*
Causes text on the same line to appear alternately in roman and bold face. The text must be on the same line as the macro call.

.B [*text*]
Causes *text* to appear in bold face. If no text is present on the line where the macro is called the text of the next input line appears in bold face.

.I [*text*] Causes *text* to appear in italic. If no text is present on the line where the macro is called the text of the next input line appears in italic.

MACROS TO DESCRIBE HYPERLINKS AND EMAIL ADDRESSES

The following macros are not defined on legacy Unix systems running classic troff. To be certain your page will be portable to those systems, copy their definitions from the **an-ext.tmac** file of a **groff** installation.

Using these macros helps ensure that you get hyperlinks when your manual page is rendered in a browser or other program that is Web-enabled.

.UR *URL*
.UE [*punctuation*]
Wrap a World Wide Web hyperlink. The argument to **UR** is the URL; thereafter, lines until **UE** are collected and used as the link text. Any argument to the **UE** macro is pasted to the end of the text. On a device that is not a browser,

```
this is a link to .UR http://\:randomsite.org/\:fubar some random site .UE , given as an
example
```

usually displays like this: "this is a link to some random site <http://randomsite.org/fubar>, given as an example".

The use of **\:** to insert hyphenless breakpoints is a groff extension and can be omitted.

.MT *address*
.ME [*punctuation*]
Wrap an email address. The argument of **MT** is the address; text following, until **ME**, is a name to be associated with the address. Any argument to the **ME** macro is pasted to the end of the link text. On a device that is not a browser,

```
contact .UR fred.foonly@\:fubar.net Fred Foonly .UE for more information
```

usually displays like this: "contact Fred Foonly <fred.foonly@fubar.net> for more information".

The use of **\:** to insert hyphenless breakpoints is a groff extension and can be omitted.

MACROS TO DESCRIBE COMMAND SYNOPSES

The following macros are not defined on legacy Unix systems running classic troff. To be certain your page will be portable to those systems, copy their definitions from the **an-ext.tmac** file of a **groff** installation.

These macros are a convenience for authors. They also assist automated translation tools and help browsers in recognizing command synopses and treating them differently from running text.

.SY *command*
Begin synopsis. Takes a single argument, the name of a command. Text following, until closed by **YS**, is set with a hanging indentation with the width of *command* plus a space. This produces the traditional look of a Unix command synopsis.

.OP *key value*
Describe an optional command argument. The arguments of this macro are set surrounded by option braces in the default Roman font; the first argument is printed with a bold face, while the second argument is typeset as italic.

.YS This macro restores normal indentation at the end of a command synopsis.

Here is a real example:

```
.SY groff .OP \-abcegiklpstzCEGNRSUVXZ .OP \-d cs .OP \-f fam .OP \-F dir .OP \-I dir .OP
\-K arg .OP \-L arg .OP \-m name .OP \-M dir .OP \-n num .OP \-o list .OP \-P arg .OP \-r cn
.OP \-T dev .OP \-w name .OP \-W name .RI [ file .IR .\|.\|. ] .YS
```

produces the following output:

[**–abcegiklpstzCEGNRSUVXZ**] [**–d***cs*] [**–f***fam*] [**–F***dir*] [**–I***dir*] [**–K***arg*] [**–L***arg*]
[**–m***name*] [**–M***dir*] [**–n***num*] [**–o***list*] [**–P***arg*] [**–r***cn*] [**–T***dev*] [**–w***name*]
[**–W***name*] [*file* . . .]

If necessary, you might use **br** requests to control line breaking. You can insert plain text as well; this looks like the traditional (unornamented) syntax for a required command argument or filename.

MISCELLANEOUS

The default indentation is 7.2n in troff mode and 7n in nroff mode except for **grohtml** which ignores indentation.

.DT Set tabs every 0.5 inches. Since this macro is always called during a **TH** request, it makes sense to call it only if the tab positions have been changed.

Use of this presentation-level macro is deprecated. It translates poorly to HTML, under which exact whitespace control and tabbing are not readily available. Thus, information or distinctions that you use **DT** to express are likely to be lost. If you feel tempted to use it, you should probably be composing a table using **tbl**(@MAN1DIR@) markup instead.

.PD [*nnn*]
Adjust the empty space before a new paragraph or section. The optional argument gives the amount of space (default unit is ‘v’); without parameter, the value is reset to its default value (1 line in nroff mode, 0.4v otherwise). This affects the macros **SH**, **SS**, **TP**, **LP** (resp. **PP** and **P**), **IP**, and **HP**.

Use of this presentation-level macro is deprecated. It translates poorly to HTML, under which exact control of inter-paragraph spacing is not readily available. Thus, information or distinctions that you use **PD** to express are likely to be lost.

.AT [*system* [*release*]]
Alter the footer for use with `AT&T` *man pages*. This command exists only for compatibility; don’t use it. See the *groff* info manual for more.

.UC [*version*]
Alter the footer for use with `BSD` *man pages*. This command exists only for compatibility; don’t use it. See the *groff* info manual for more.

.PT Print the header string. Redefine this macro to get control of the header.

.BT Print the footer string. Redefine this macro to get control of the footer.

The following strings are defined:

***S** Switch back to the default font size.

***R** The 'registered' sign.

***(Tm** The 'trademark' sign.

***(lq**
***(rq** Left and right quote. This is equal to '\(lq' and '\(rq', respectively.

***(HF** The typeface used to print headings and subheadings. The default is 'B'.

If a preprocessor like **tbl** or **eqn** is needed, it has become usage to make the first line of the *man page* look like this:

> **'\"** *word*

Note the single space character after the double quote. *word* consists of letters for the needed preprocessors: 'e' for **eqn**, 'r' for **refer**, and 't' for **tbl**. Modern implementations of the **man** program read this first line and automatically call the right preprocessor(s).

PORTABILITY AND TROFF REQUESTS

Since the **man** macros consist of groups of *groff* requests, one can, in principle, supplement the functionality of the **man** macros with individual *groff* requests where necessary. See the *groff* info pages for a complete reference of all requests.

Note, however, that using raw troff requests is likely to make your page render poorly on the (increasingly common) class of viewers that render it to HTML. Troff requests make implicit assumptions about things like character and page sizes that may break in an HTML environment; also, many of these viewers don't interpret the full troff vocabulary, a problem which can lead to portions of your text being silently dropped.

For portability to modern viewers, it is best to write your page entirely in the requests described on this page. Further, it is best to completely avoid those we have described as 'presentation-level' (**HP**, **PD**, and **DT**).

The macros we have described as extensions (**.EX/.EE**, **.SY/.OP/.YS**, **.UR/.UE**, and **.MT/.ME**) should be used with caution, as they may not yet be built in to some viewer that is important to your audience. If in doubt, copy the implementation onto your page.

FILES

man.tmac
an.tmac
These are wrapper files to call **andoc.tmac**.

andoc.tmac
Use this file in case you don't know whether the **man** macros or the **mdoc** package should be used. Multiple man pages (in either format) can be handled.

an-old.tmac
Most **man** macros are contained in this file.

an-ext.tmac
The extension macro definitions for **.SY**, **.OP**, **.YS**, **.TQ**, **.EX/.EE**, **.UR/.UE**, and **.MT/.ME** are contained in this file. It is written in classic troff, and released for free re-use, and not copylefted; manual page authors concerned about portability to legacy Unix systems are encouraged to copy these definitions into their pages, and maintainers of troff or its workalikes are encouraged to re-use them.

man.local
Local changes and customizations should be put into this file.

SEE ALSO

tbl(1), **eqn**(1), **refer**(1), **man**(1), **man**(7), **groff_mdoc**(7)

AUTHORS

This manual page was originally written for the Debian GNU/Linux system by Susan G. Kleinmann It was corrected and updated by Werner Lemberg The extension macros were documented (and partly designed) by Eric S. Raymond he also wrote the portability advice. GROFF_MDOC

NAME

groff_mdoc — reference for groff's mdoc implementation

SYNOPSIS

groff -mdoc *file . . .*

DESCRIPTION

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

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

GETTING STARTED

The material presented in the remainder of this document is outlined as follows:

1. TROFF IDIOSYNCRASIES
   Macro Usage
   Passing Space Characters in an Argument
   Trailing Blank Space Characters
   Escaping Special Characters
   Other Possible Pitfalls
2. A MANUAL PAGE TEMPLATE
3. CONVENTIONS
4. TITLE MACROS
5. INTRODUCTION OF MANUAL AND GENERAL TEXT DOMAINS
   What's in a Name . . .
   General Syntax
6. MANUAL DOMAIN
   Addresses
   Author Name
   Arguments
   Configuration Declarations (Section Four Only)
   Command Modifiers
   Defined Variables
   Errno's
   Environment Variables
   Flags
   Function Declarations
   Function Types
   Functions (Library Routines)

TROFF IDIOSYNCRASIES

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

Macro Usage

As in GNU `troff`(1), a macro is called by placing a '`.`' (dot character) at the beginning of a line followed by the two-character (or three-character) name for the macro. There can be space or tab characters between the dot and the macro name. Arguments may follow the macro separated by spaces (but *no* tabs). It is the dot character at the beginning of the line which causes GNU `troff`(1) to interpret the next two (or more) characters as a macro name. A single starting dot followed by nothing is ignored. To place a '`.`' (dot character) at the beginning of an input line in some context other than a macro invocation, precede the '`.`' (dot) with the '`\&`' escape sequence which translates literally to a zero-width space, and is never displayed in the output.

In general, GNU `troff`(1) macros accept an unlimited number of arguments (contrary to other versions of troff which can't handle more than nine arguments). In limited cases, arguments may be continued or extended on the next line (See **Extended Arguments** below). Almost all macros handle quoted arguments (see **Passing Space Characters in an Argument** below).

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

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

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

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

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

In the following, we call an **-mdoc** macro which starts a line (with a leading dot) a *command* if this distinction is necessary.

Passing Space Characters in an Argument

Sometimes it is desirable to give as an argument a string containing one or more blank space characters, say, to specify arguments to commands which expect particular arrangement of items in the argument list. Additionally, it makes **-mdoc** working faster. For example, the function command `.Fn` expects the first argument to be the name of a function and any remaining arguments to be function parameters. As ANSI C stipulates the declaration of function parameters in the parenthesized parameter list, each parameter is guaranteed to be at minimum a two word string. For example, *`int foo`*.

There are two possible ways to pass an argument which contains an embedded space. One way of passing a string containing blank spaces is to use the hard or unpaddable space character '`\ `', that is, a blank space preceded by the escape character '`\`'. This method may be used with any macro but has the side effect of interfering with the adjustment of text over the length of a line. `Troff` sees the hard space as if it were any other printable character and cannot split the string into blank or newline separated pieces as one would expect. This method is useful for strings which are not expected to overlap a line boundary. An alternative is to use '`\ `', a paddable (i.e. stretchable), unbreakable space (this is a GNU `troff`(1) extension). The second method is to enclose the string with double quotes.

For example:

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

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

If the '`\`' before the space in the first example or double quotes in the second example were omitted, `.Fn` would see three arguments, and the result would be:

fetch(*char*, **str*)

Trailing Blank Space Characters

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

Escaping Special Characters

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

Other Possible Pitfalls

A warning is emitted when an empty input line is found outside of displays (see below). Use `.sp` instead. (Well, it is even better to use **-mdoc** macros to avoid the usage of low-level commands.)

Leading spaces will cause a break and are output directly. Avoid this behaviour if possible. Similarly, do not use more than one space character between words in an ordinary text line; contrary to other text formatters, they are *not* replaced with a single space.

You can't pass '`"`' directly as an argument. Use `\*[q]` (or `\*q`) instead.

By default, `troff`(1) inserts two space characters after a punctuation mark closing a sentence; characters like '`)`' or '`'`' are treated transparently, not influencing the sentence-ending behaviour. To change this, insert '`\&`' before or after the dot:

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

gives

The ''. character

The '`.`' character.

test. test

test. test

As can be seen in the first and third line, **-mdoc** handles punctuation characters specially in macro arguments. This will be explained in section **General Syntax** below. In the same way, you have to protect trailing full stops of abbreviations with a trailing zero-width space: `e.g.\&`.

A comment in the source file of a man page can be either started with `.\"` on a single line, '`\"`' after some input, or '`\#`' anywhere (the latter is a GNU `troff`(1) extension); the rest of such a line is ignored.

A MANUAL PAGE TEMPLATE

The body of a man page is easily constructed from a basic template:

```
.\" The following commands are required for all man pages.
.Dd Month day, year
.Os [OPERATING_SYSTEM] [version/release]
.Dt DOCUMENT_TITLE [section number] [architecture/volume]
.Sh NAME
.Nm name
.Nd one line description of name
.\" This next command is for sections 2 and 3 only.
.\" .Sh LIBRARY
.Sh SYNOPSIS
.Sh DESCRIPTION
.\" The following commands should be uncommented and
.\" used where appropriate.
.\" .Sh IMPLEMENTATION NOTES
.\" This next command is for sections 2, 3 and 9 function
.\" return values only.
.\" .Sh RETURN VALUES
.\" This next command is for sections 1, 6, 7 and 8 only.
.\" .Sh ENVIRONMENT
.\" .Sh FILES
.\" .Sh EXAMPLES
.\" This next command is for sections 1, 6, 7, 8 and 9 only
.\"     (command return values (to shell) and
.\"     fprintf/stderr type diagnostics).
.\" .Sh DIAGNOSTICS
.\" .Sh COMPATIBILITY
.\" This next command is for sections 2, 3 and 9 error
.\"     and signal handling only.
.\" .Sh ERRORS
.\" .Sh SEE ALSO
.\" .Sh STANDARDS
.\" .Sh HISTORY
.\" .Sh AUTHORS
.\" .Sh BUGS
```

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

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

CONVENTIONS

In the description of all macros below, optional arguments are put into brackets. An ellipsis ('...') represents zero or more additional arguments. Alternative values for a parameter are separated with '|'. If there are alternative values for a mandatory parameter, braces are used (together with '|') to enclose the value set. Meta-variables are specified within angles.

Example:

`.Xx` ⟨foo⟩ {bar1 | bar2} [−test1 [−test2 | −test3]] ...

Except stated explicitly, all macros are parsed and callable.

Note that a macro takes effect up to the next nested macro. For example, `.Ic foo Aq bar` doesn't produce '**`foo <bar>`**' but '**`foo`** ⟨bar⟩'. Consequently, a warning message is emitted for most commands if the first argument is a macro itself since it cancels the effect of the calling command completely. Another consequence is that quoting macros never insert literal quotes; '**`foo <bar>`**' has been produced by `.Ic "foo <bar>"`.

Most macros have a default width value which can be used to specify a label width (**`-width`**) or offset (**`-offset`**) for the `.Bl` and `.Bd` macros. It is recommended not to use this rather obscure feature to avoid dependencies on local modifications of the **`-mdoc`** package.

TITLE MACROS

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

`.Dt` [⟨document title⟩] [⟨section number⟩] [⟨volume⟩]

The document title is the subject of the man page and must be in CAPITALS due to troff limitations. If omitted, 'UNTITLED' is used. The section number may be a number in the range 1, ..., 9 or `unass`, `draft`, or `paper`. If it is specified, and no volume name is given, a default volume name is used.

Under BSD, the following sections are defined:

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

A volume name may be arbitrary or one of the following:

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

For compatibility, `MMI` can be used for `IND`, and `LOC` for `LOCAL`. Values from the previous table will specify a new volume name. If the third parameter is a keyword designating a computer architecture, its value is prepended to the default volume name as specified by the second parameter. By default, the following architecture keywords are defined:

alpha, acorn26, acorn32, algor, amd64, amiga, arc, arm26, arm32, atari, bebox, cats, cesfic, cobalt, dreamcast, evbarm, evbmips, evbppc, evbsh3, hp300, hp700, hpcmips, i386, luna68k, m68k, mac68k, macppc, mips, mmeye, mvme68k, mvmeppc, netwinder, news68k, newsmips, next68k, ofppc, pc532, pmax, pmppc, powerpc, prep, sandpoint, sgimips, sh3, shark, sparc, sparc64, sun3, tahoe, vax, x68k, x86_64

If the section number is neither a numeric expression in the range 1 to 9 nor one of the above described keywords, the third parameter is used verbatim as the volume name.

In the following examples, the left (which is identical to the right) and the middle part of the manual page header strings are shown. Note how '`\&`' prevents the digit 7 from being a valid numeric expression.

```
.Dt FOO 7      FOO(7) BSD Miscellaneous Information Manual
.Dt FOO 7 bar  FOO(7) BSD Miscellaneous Information Manual
.Dt FOO \&7 bar
               FOO(7) bar
.Dt FOO 2 i386 FOO(2) BSD/i386 System Calls Manual
.Dt FOO "" bar FOO bar
```

Local, OS-specific additions might be found in the file `mdoc.local`; look for strings named `volume-ds-XXX` (for the former type) and `volume-as-XXX` (for the latter type); `XXX` then denotes the keyword to be used with the `.Dt` macro.

This macro is neither callable nor parsed.

`.Os` [⟨operating system⟩] [⟨release⟩]

If the first parameter is empty, the default 'BSD' is used. This may be overridden in the local configuration file, `mdoc.local`. In general, the name of the operating system should be the common acronym, e.g. BSD or ATT. The release should be the standard release nomenclature for the system specified. In the following table, the possible second arguments for some predefined operating systems are listed. Similar to `.Dt`, local additions might be defined in `mdoc.local`; look for strings named `operating-system-XXX-YYY`, where `XXX` is the acronym for the operating system and `YYY` the release ID.

ATT	7th, 7, III, 3, V, V.2, V.3, V.4
BSD	3, 4, 4.1, 4.2, 4.3, 4.3t, 4.3T, 4.3r, 4.3R, 4.4
NetBSD	0.8, 0.8a, 0.9, 0.9a, 1.0, 1.0a, 1.1, 1.2, 1.2a, 1.2b, 1.2c, 1.2d, 1.2e, 1.3, 1.3a, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1, 1.6.2, 1.6.3, 2.0, 2.0.1, 2.0.2, 2.0.3, 2.1, 3.0, 3.0.1, 3.0.2, 3.1, 4.0, 4.0.1
FreeBSD	1.0, 1.1, 1.1.5, 1.1.5.1, 2.0, 2.0.5, 2.1, 2.1.5, 2.1.6, 2.1.7, 2.2, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 4.9, 4.10, 4.11, 5.0, 5.1, 5.2, 5.2.1, 5.3, 5.4, 5.5, 6.0, 6.1, 6.2, 6.3, 6.4, 7.0, 7.1
DragonFly	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 1.8.1, 1.10, 1.12, 1.12.2, 2.0
Darwin	8.0.0, 8.1.0, 8.2.0, 8.3.0, 8.4.0, 8.5.0, 8.6.0, 8.7.0, 8.8.0, 8.9.0, 8.10.0, 8.11.0, 9.0.0, 9.1.0, 9.2.0, 9.3.0, 9.4.0, 9.5.0, 9.6.0

For ATT, an unknown second parameter will be replaced with the string UNIX; for the other predefined acronyms it will be ignored and a warning message emitted. Unrecognized arguments are displayed as given in the page footer. For instance, a typical footer might be:

```
.Os BSD 4.3
```

giving `4.3 Berkeley Distribution`, or for a locally produced set

```
.Os CS Department
```

which will produce `CS Department`.

If the `.Os` macro is not present, the bottom left corner of the manual page will be ugly.

This macro is neither callable nor parsed.

`.Dd` [⟨month⟩ ⟨day⟩, ⟨year⟩]
If ‘`Dd`’ has no arguments, `Epoch` is used for the date string. If it has exactly three arguments, they are concatenated, separated with unbreakable space:

```
.Dd January 25, 2001
```

The month’s name shall not be abbreviated.

With any other number of arguments, the current date is used, ignoring the parameters.

This macro is neither callable nor parsed.

INTRODUCTION OF MANUAL AND GENERAL TEXT DOMAINS

What’s in a Name...

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

In the first case, `troff`(1) macros are themselves a type of command; the general syntax for a troff command is:

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

`.Xx` is a macro command, and anything following it are arguments to be processed. In the second case, the description of a UNIX command using the content macros is a bit more involved; a typical **SYNOPSIS** command line might be displayed as:

filter [**-flag**] ⟨*infile*⟩ ⟨*outfile*⟩

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

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

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

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

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

```
.Nm make
.Op Fl eiknqrstv
.Op Fl D Ar variable
.Op Fl d Ar flags
.Op Fl f Ar makefile
.Op Fl I Ar directory
```

```
.Op Fl j Ar max_jobs
.Op Ar variable Ns = Ns Ar value
.Bk
.Op Ar target ...
.Ek
```

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

General Syntax

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

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

The result is:

sptr, ptr),

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

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

The result is:

sptr, *ptr*),

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

The following punctuation characters are recognized by **-mdoc**:

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

`Troff` is limited as a macro language, and has difficulty when presented with a string containing a member of the mathematical, logical or quotation set:

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

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

MANUAL DOMAIN

Addresses

The address macro identifies an address construct.

`Usage: .Ad` ⟨address⟩ ...

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

The default width is 12n.

Author Name

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

```
Usage: .An ⟨author name⟩ ...

          .An "Joe Author"        Joe Author

          .An "Joe Author" ,      Joe Author,

          .An "Joe Author" Aq nobody@FreeBSD.org
                                  Joe Author ⟨nobody@FreeBSD.org⟩

          .An "Joe Author" ) ) ,  Joe Author)),
```

The default width is 12n.

In the **AUTHORS** section, the `.An` command causes a line break allowing each new name to appear on its own line. If this is not desirable,

```
.An -nosplit
```

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

```
.An -split
```

Arguments

The `.Ar` argument macro may be used whenever an argument is referenced. If called without arguments, the '*file ...*' string is output.

```
Usage: .Ar [⟨argument⟩] ...

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

The default width is 12n.

Configuration Declaration (Section Four Only)

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

```
Usage: .Cd ⟨argument⟩ ...

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

In the **SYNOPSIS** section a `.Cd` command causes a line break before and after its arguments are printed.

The default width is 12n.

Command Modifiers

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

The default width is 10n.

Defined Variables

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

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

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

The default width is 12n.

Errno's

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

```
Usage: .Er ⟨errno type⟩ ...

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

The default width is 17n.

Environment Variables

The `.Ev` macro specifies an environment variable.

```
Usage: .Ev ⟨argument⟩ ...

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

The default width is 15n.

Flags

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

```
Usage: .Fl ⟨argument⟩ ...

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

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

The default width is 12n.

Function Declarations

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

```
Usage: .Fd ⟨argument⟩ ...

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

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

The `.In` macro, while in the **SYNOPSIS** section, represents the `#include` statement, and is the short form of the above example. It specifies the C header file as being included in a C program. It also causes a line break.

While not in the **SYNOPSIS** section, it represents the header file enclosed in angle brackets.

```
Usage: .In ⟨header file⟩

       .In stdio.h #include <stdio.h>
       .In stdio.h <stdio.h>
```

Function Types

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

```
Usage: .Ft ⟨type⟩ ...

       .Ft struct stat struct stat
```

Functions (Library Routines)

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

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

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

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

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

Example:

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

Produces:

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

In the **SYNOPSIS** section, the function will always begin at the beginning of line. If there is more than one function presented in the **SYNOPSIS** section and a function type has not been given, a line break will occur, leaving a nice vertical space between the current function name and the one prior.

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

Function Arguments

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

`Usage: .Fa` ⟨function argument⟩ ...

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

The default width is 12n.

Return Values

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

`Usage: .Rv` [−std] [⟨function⟩ ...]

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

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

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

Exit Status

The `.Ex` macro generates text for use in the **DIAGNOSTICS** section.

`Usage: .Ex` [−std] [⟨utility⟩ ...]

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

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

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

Interactive Commands

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

`Usage: .Ic` ⟨argument⟩ ...

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

The default width is 12n.

Library Names

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

`Usage: .Lb` ⟨argument⟩ ...

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

`libarm`	ARM Architecture Library (libarm, −larm)
`libarm32`	ARM32 Architecture Library (libarm32, −larm32)
`libc`	Standard C Library (libc, −lc)
`libcdk`	Curses Development Kit Library (libcdk, −lcdk)
`libcompat`	Compatibility Library (libcompat, −lcompat)
`libcrypt`	Crypt Library (libcrypt, −lcrypt)
`libcurses`	Curses Library (libcurses, −lcurses)

`libedit`	Command Line Editor Library (libedit, −ledit)
`libevent`	Event Notification Library (libevent, −levent)
`libform`	Curses Form Library (libform, −lform)
`libi386`	i386 Architecture Library (libi386, −li386)
`libintl`	Internationalized Message Handling Library (libintl, −lintl)
`libipsec`	IPsec Policy Control Library (libipsec, −lipsec)
`libkvm`	Kernel Data Access Library (libkvm, −lkvm)
`libm`	Math Library (libm, −lm)
`libm68k`	m68k Architecture Library (libm68k, −lm68k)
`libmagic`	Magic Number Recognition Library (libmagic, −lmagic)
`libmenu`	Curses Menu Library (libmenu, −lmenu)
`libossaudio`	OSS Audio Emulation Library (libossaudio, −lossaudio)
`libpam`	Pluggable Authentication Module Library (libpam, −lpam)
`libpcap`	Packet Capture Library (libpcap, −lpcap)
`libpci`	PCI Bus Access Library (libpci, −lpci)
`libpmc`	Performance Counters Library (libpmc, −lpmc)
`libposix`	POSIX Compatibility Library (libposix, −lposix)
`libpthread`	POSIX Threads Library (libpthread, −lpthread)
`libresolv`	DNS Resolver Library (libresolv, −lresolv)
`librt`	POSIX Real-time Library (librt, −lrt)
`libtermcap`	Termcap Access Library (libtermcap, −ltermcap)
`libusbhid`	USB Human Interface Devices Library (libusbhid, −lusbhid)
`libutil`	System Utilities Library (libutil, −lutil)
`libx86_64`	x86_64 Architecture Library (libx86_64, −lx86_64)
`libz`	Compression Library (libz, −lz)

Local, OS-specific additions might be found in the file `mdoc.local`; look for strings named `str-Lb-XXX`. `XXX` then denotes the keyword to be used with the `.Lb` macro.

In the **LIBRARY** section an `.Lb` command causes a line break before and after its arguments are printed.

Literals

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

Usage: `.Li` ⟨argument⟩ ...

```
.Li \en          \n
.Li M1 M2 M3 ;  M1 M2 M3;
.Li cntrl-D ) , cntrl-D),
.Li 1024 ...    1024 . . .
```

The default width is 16n.

Names

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

Usage: `.Nm` [⟨argument⟩] ...

```
.Nm groff_mdoc groff_mdoc
```

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

The default width is 10n.

Options

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

`Usage: .Op` [⟨option⟩] ...

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

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

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

Produces:

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

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

Pathnames

The `.Pa` macro formats path or file names. If called without arguments, the ‘ ’ string is output, which represents the current user’s home directory.

`Usage: .Pa` [⟨pathname⟩] ...

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

The default width is 32n.

Standards

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

`Usage: .St` ⟨abbreviation⟩ ...

Available pairs for “Abbreviation/Formal Name” are:

ANSI/ISO C

`-ansiC`	ANSI X3.159-1989 (“ANSI C89”)
`-ansiC-89`	ANSI X3.159-1989 (“ANSI C89”)

`-isoC`	ISO/IEC 9899:1990 (“ISO C90”)
`-isoC-90`	ISO/IEC 9899:1990 (“ISO C90”)
`-isoC-99`	ISO/IEC 9899:1999 (“ISO C99”)

POSIX Part 1: System API

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

POSIX Part 2: Shell and Utilities

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

X/Open

`-susv2`	Version 2 of the Single UNIX Specification (“SUSv2”)
`-susv3`	
`-svid4`	System V Interface Definition, Fourth Edition (“SVID4”)
`-xbd5`	X/Open System Interface Definitions Issue 5 (“XBD5”)
`-xcu5`	X/Open Commands and Utilities Issue 5 (“XCU5”)
`-xcurses4.2`	X/Open Curses Issue 4, Version 2 (“XCURSES4.2”)
`-xns5`	X/Open Networking Services Issue 5 (“XNS5”)
`-xns5.2`	X/Open Networking Services Issue 5.2 (“XNS5.2”)
`-xpg3`	X/Open Portability Guide Issue 3 (“XPG3”)
`-xpg4`	X/Open Portability Guide Issue 4 (“XPG4”)
`-xpg4.2`	X/Open Portability Guide Issue 4, Version 2 (“XPG4.2”)
`-xsh5`	X/Open System Interfaces and Headers Issue 5 (“XSH5”)

Miscellaneous

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

Variable Types

The `.Vt` macro may be used whenever a type is referenced. In the **SYNOPSIS** section, it causes a line break (useful for old style variable declarations).

`Usage: .Vt` ⟨type⟩ ...

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

Variables

Generic variable reference.

`Usage: .Va` ⟨variable⟩ ...

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

The default width is 12n.

Manual Page Cross References

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

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

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

The default width is 10n.

GENERAL TEXT DOMAIN

AT&T Macro

`Usage: .At` [⟨version⟩] ...

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

The following values for ⟨version⟩ are possible:

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

BSD Macro

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

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

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

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

NetBSD Macro

`Usage: .Nx` [⟨version⟩] ...

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

For possible values of ⟨version⟩ see the description of the `.Os` command above in section **TITLE MACROS**.

FreeBSD Macro

`Usage: .Fx` [⟨version⟩] ...

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

For possible values of ⟨version⟩ see the description of the `.Os` command above in section **TITLE MACROS**.

DragonFly Macro

`Usage: .Dx` [⟨version⟩] ...

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

For possible values of ⟨version⟩ see the description of the `.Os` command above in section **TITLE MACROS**.

OpenBSD Macro

`Usage: .Ox` [⟨version⟩] ...

`.Ox 1.0` OpenBSD 1.0

BSD/OS Macro

`Usage: .Bsx` [⟨version⟩] ...

`.Bsx 1.0` BSD/OS 1.0

UNIX Macro

`Usage: .Ux ...`

`.Ux` UNIX

Emphasis Macro

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

`Usage: .Em` ⟨argument⟩ ...

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

The default width is 10n.

Font Mode

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

`.Bf` has the following syntax:

`.Bf` ⟨font mode⟩

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

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

Both macros are neither callable nor parsed.

Enclosure and Quoting Macros

The concept of enclosure is similar to quoting. The object being to enclose one or more strings between a pair of characters like quotes or parentheses. The terms quoting and enclosure are used interchangeably throughout this document. Most of the one-line enclosure macros end in small letter ‘`q`’ to give a hint of quoting, but there are a few irregularities. For each enclosure macro there is also a pair of open and close macros which end in small letters ‘`o`’ and ‘`c`’ respectively.

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

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

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

`.Es`, `.En` Due to the nine-argument limit in the original troff program two other macros have been implemented which are now rather obsolete: `.Es` takes the first and second parameter as the left and right enclosure string, which are then used to enclose the arguments of `.En`. The default width value is 12n for both macros.

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

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

The default width is 16n.

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

`.Pf ( Fa name2` (*name2*

The default width is 12n.

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

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

Examples of quoting:

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

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

No-Op or Normal Text Macro

The `.No` macro can be used in a macro command line for parameters which should *not* be formatted. Be careful to add '`\&`' to the word '`No`' if you really want that English word (and not the macro) as a parameter.

`Usage: .No` ⟨argument⟩ ...

`.No test Ta with Ta tabs` test with tabs

The default width is 12n.

No-Space Macro

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

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

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

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

Section Cross References

The `.Sx` macro designates a reference to a section header within the same document.

`Usage: .Sx` ⟨section reference⟩ ...

`.Sx FILES` **FILES**

The default width is 16n.

Symbolics

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

`Usage: .Sy` ⟨symbol⟩ ...

`.Sy Important Notice` **Important Notice**

The default width is 6n.

Mathematical Symbols

Use this macro for mathematical symbols and similar things.

`Usage: .Ms` ⟨math symbol⟩ ...

`.Ms sigma` **sigma**

The default width is 6n.

References and Citations

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

`.Rs`	Reference start (does not take arguments). Causes a line break in the **SEE ALSO** section and begins collection of reference information until the reference end macro is read.
`.Re`	Reference end (does not take arguments). The reference is printed.
`.%A`	Reference author name; one name per invocation.
`.%B`	Book title.

`.%C`	City/place (not implemented yet).
`.%D`	Date.
`.%I`	Issuer/publisher name.
`.%J`	Journal name.
`.%N`	Issue number.
`.%O`	Optional information.
`.%P`	Page number.
`.%Q`	Corporate or foreign author.
`.%R`	Report name.
`.%T`	Title of article.
`.%V`	Volume.

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

Example:

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

produces

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

Trade Names (or Acronyms and Type Names)

The trade name macro prints its arguments in a smaller font. Its intended use is to imitate a small caps fonts for uppercase acronyms.

`Usage: .Tn` ⟨symbol⟩ `...`

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

The default width is 10n.

Extended Arguments

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

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

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

produces

I*operation*\n*count*\n

Another one:

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

produces

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

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

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

produces

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

PAGE STRUCTURE DOMAIN

Section Headers

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

The default width is 8n.

`.Sh NAME`	The `.Sh NAME` macro is mandatory. If not specified, headers, footers and page layout defaults will not be set and things will be rather unpleasant. The **NAME** section consists of at least three items. The first is the `.Nm` name macro naming the subject of the man page. The second is the name description macro, `.Nd`, which separates the subject name from the third item, which is the description. The description should be the most terse and lucid possible, as the space available is small. `.Nd` first prints '-', then all its arguments.
`.Sh LIBRARY`	This section is for section two and three function calls. It should consist of a single `.Lb` macro call; see **Library Names**.
`.Sh SYNOPSIS`	The **SYNOPSIS** section describes the typical usage of the subject of a man page. The macros required are either `.Nm`, `.Cd`, or `.Fn` (and possibly `.Fo`, `.Fc`, `.Fd`, and `.Ft`). The function name macro `.Fn` is required for manual page sections 2 and 3; the command and general name macro `.Nm` is required for sections 1, 5, 6, 7, and 8. Section 4 manuals require a `.Nm`, `.Fd` or a `.Cd` configuration device usage macro. Several other macros may be necessary to produce the synopsis line as shown below:

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

The following macros were used:

```
.Nm cat
.Op Fl benstuv
.Op Fl
.Ar
```

`.Sh DESCRIPTION`	In most cases the first text in the **DESCRIPTION** section is a brief paragraph on the command, function or file, followed by a lexical list of options and respective explanations. To create such a list, the `.Bl` (begin list), `.It` (list item) and `.El` (end list) macros are used (see **Lists and Columns** below).
`.Sh IMPLEMENTATION NOTES`	Implementation specific information should be placed here.
`.Sh RETURN VALUES`	Sections 2, 3 and 9 function return values should go here. The `.Rv` macro may be used to generate text for use in the **RETURN VALUES** section for most section 2 and 3 library functions; see **Return Values**.

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

`.Sh ENVIRONMENT`	The **ENVIRONMENT** section should reveal any related environment variables and clues to their behavior and/or usage.
`.Sh FILES`	Files which are used or created by the man page subject should be listed via the `.Pa` macro in the **FILES** section.
`.Sh EXAMPLES`	There are several ways to create examples. See the **EXAMPLES** section below for details.
`.Sh DIAGNOSTICS`	Diagnostic messages from a command should be placed in this section. The `.Ex` macro may be used to generate text for use in the **DIAGNOSTICS** section for most section 1, 6 and 8 commands; see **Exit Status**.
`.Sh COMPATIBILITY`	Known compatibility issues (e.g. deprecated options or parameters) should be listed here.
`.Sh ERRORS`	Specific error handling, especially from library functions (man page sections 2, 3, and 9) should go here. The `.Er` macro is used to specify an error (errno).
`.Sh SEE ALSO`	References to other material on the man page topic and cross references to other relevant man pages should be placed in the **SEE ALSO** section. Cross references are specified using the `.Xr` macro. Currently `refer`(1) style references are not accommodated. It is recommended that the cross references are sorted on the section number, then alphabetically on the names within a section, and placed in that order and comma separated. Example: `ls`(1), `ps`(1), `group`(5), `passwd`(5)
`.Sh STANDARDS`	If the command, library function or file adheres to a specific implementation such as IEEE Std 1003.2 (“POSIX.2”) or ANSI X3.159-1989 (“ANSI C89”) this should be noted here. If the command does not adhere to any standard, its history should be noted in the **HISTORY** section.
`.Sh HISTORY`	Any command which does not adhere to any specific standards should be outlined historically in this section.
`.Sh AUTHORS`	Credits should be placed here. Use the `.An` macro for names and the `.Aq` macro for e-mail addresses within optional contact information. Explicitly indicate whether the person authored the initial manual page or the software or whatever the person is being credited for.
`.Sh BUGS`	Blatant problems with the topic go here.

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

```
.Sh "PAGE STRUCTURE DOMAIN"
```

Subsection Headers

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

The default width is 8n.

Paragraphs and Line Spacing

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

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

Keeps

The only keep that is implemented at this time is for words. The macros are `.Bk` (begin keep) and `.Ek` (end keep). The only option that `.Bk` accepts currently is **`-words`** (this is also the default if no option is given) which is useful for preventing line breaks in the middle of options. In the example for the make command line arguments (see **What's in a Name**), the keep prevented `nroff` from placing up the flag and the argument on separate lines.

Both macros are neither callable nor parsed.

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

Examples and Displays

There are seven types of displays.

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

> **`-ldghfstru`**

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

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

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

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

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

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

`-ragged`	Fill, but do not adjust the right margin (only left-justify).
`-centered`	Center lines between the current left and right margin. Note that each single line is centered.
`-unfilled`	Do not fill; display a block of text as typed, using line breaks as specified by the user. This can produce overlong lines without warning messages.
`-filled`	Display a filled block. The block of text is formatted (i.e., the text is justified on both the left and right side).
`-literal`	Display block with literal font (usually fixed-width). Useful for source code or simple tabbed or spaced text.
`-file` ⟨*`file name`*⟩	The file whose name follows the **`-file`** flag is read and displayed before any data enclosed with `.Bd` and `.Ed`, using the selected display type. Any `troff`/**`-mdoc`** commands in the file will be processed.

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

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

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

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

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

Lists and Columns

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

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

It has the following syntax forms:

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

And now a detailed description of the list types.

-bullet A bullet list.

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

Produces:

- Bullet one goes here.
- Bullet two here.

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

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

Produces:

– Dash one goes here.
– Dash two here.

-enum An enumerated list.

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

The result:

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

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

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

Result:

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

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

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

```
.It
Item two here.
Item two here.
Item two here.
.El
```

Produces:

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

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

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

SL	sleep time of the process (seconds blocked)
PAGEIN	number of disk I/O's resulting from references by the process to pages not loaded in core.
UID	numerical user-id of process owner
PPID	numerical id of parent of process priority (non-positive when in non-interruptible wait)

The raw text:

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

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

Example:

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

produces

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

-hang A list with hanging tags.

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

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

And the unformatted text which created it:

```
.Bl -hang -offset indent
.It Em Hanged
labels appear similar to tagged lists when the
```

```
label is smaller than the label width.
.It Em Longer hanged list labels
blend into the paragraph unlike
tagged paragraph labels.
.El
```

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

SL
sleep time of the process (seconds blocked)

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

UID
numerical user-id of process owner

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

The raw text:

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

-inset Here is an example of inset labels:

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

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

Hang Hanged labels are a matter of taste.

Ohang Overhanging labels are nice when space is constrained.

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

Here is the source text which produced the above example:

```
.Bl -inset -offset indent
.It Em Tag
The tagged list (also called a tagged paragraph)
is the most common type of list used in the
Berkeley manuals.
.It Em Diag
Diag lists create section four diagnostic lists
```

```
and are similar to inset lists except callable
macros are ignored.
.It Em Hang
Hanged labels are a matter of taste.
.It Em Ohang
Overhanging labels are nice when space is constrained.
.It Em Inset
Inset labels are useful for controlling blocks of
paragraphs and are valuable for converting
.Nm -mdoc
manuals to other formats.
.El
```

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

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

The table:

String	**Nroff**	**Troff**
`<=`	<=	≤
`>=`	>=	≥

was produced by:

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

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

Other keywords:

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

Example:

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

gives:

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

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

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

If a width is not specified for the tag list type, every time `.It` is invoked, an attempt is made to determine an appropriate width. If the first argument to `.It` is a callable macro, the default width for that macro will be used; otherwise, the default width of `.No` is used.

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

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

MISCELLANEOUS MACROS

Here a list of the remaining macros which do not fit well into one of the above sections. We couldn't find real examples for the following macros: `.Me` and `.Ot`. They are documented here for completeness – if you know how to use them properly please send a mail to `bug-groff@gnu.org` (including an example).

`.Bt` prints

is currently in beta test.

It is neither callable nor parsed and takes no arguments.

`.Fr`

`Usage: .Fr` ⟨function return value⟩ ...

Don't use this macro. It allows a break right before the return value (usually a single digit) which is bad typographical behaviour. Use '`\ `' to tie the return value to the previous word.

`.Hf` Use this macro to include a (header) file literally. It first prints `File:` followed by the file name, then the contents of ⟨file⟩.

`Usage: .Hf` ⟨file⟩

It is neither callable nor parsed.

`.Lk` To be written.

`.Me` Exact usage unknown. The documentation in the **-mdoc** source file describes it as a macro for "menu entries".

Its default width is 6n.

`.Mt` To be written.

`.Ot` Exact usage unknown. The documentation in the **-mdoc** source file describes it as “old function type (fortran)”.

`.Sm` Activate (toggle) space mode.

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

If space mode is off, no spaces between macro arguments are inserted. If called without a parameter (or if the next parameter is neither ‘`on`’ nor `off`, `.Sm` toggles space mode.

`.Ud` prints

currently under development.

It is neither callable nor parsed and takes no arguments.

PREDEFINED STRINGS

The following strings are predefined:

String	**Nroff**	**Troff**	**Meaning**
`<=`	<=	≤	less equal
`>=`	>=	≥	greater equal
`Rq`	”	”	right double quote
`Lq`	“	“	left double quote
`ua`	^	↑	upwards arrow
`aa`	´	´	acute accent
`ga`	`	`	grave accent
`q`	"	"	straight double quote
`Pi`	pi	π	greek pi
`Ne`	!=	≠	not equal
`Le`	<=	≤	less equal
`Ge`	>=	≥	greater equal
`Lt`	<	<	less than
`Gt`	>	>	greater than
`Pm`	+−	±	plus minus
`If`	infinity	∞	infinity
`Am`	&	&	ampersand
`Na`	*NaN*	*NaN*	not a number
`Ba`	|	|	vertical bar

The names of the columns **Nroff** and **Troff** are a bit misleading; **Nroff** shows the ASCII representation, while **Troff** gives the best glyph form available. For example, a Unicode enabled TTY-device will have proper glyph representations for all strings, whereas the enhancement for a Latin1 TTY-device is only the plus-minus sign.

String names which consist of two characters can be written as `\*(xx`; string names which consist of one character can be written as `\*x`. A generic syntax for a string name of any length is `\*[xxx]` (this is a GNU `troff`(1) extension).

DIAGNOSTICS

The debugging macro `.Db` available in previous versions of **-mdoc** has been removed since GNU `troff`(1) provides better facilities to check parameters; additionally, many error and warning messages have been added to this macro package, making it both more robust and verbose.

The only remaining debugging macro is `.Rd` which yields a register dump of all global registers and strings. A normal user will never need it.

FORMATTING WITH GROFF, TROFF, AND NROFF

By default, the package inhibits page breaks, headers, and footers if displayed with a TTY device like 'latin1' or 'unicode', to make the manual more efficient for viewing on-line. This behaviour can be changed (e.g. to create a hardcopy of the TTY output) by setting the register 'cR' to zero while calling groff(1), resulting in multiple pages instead of a single, very long page:

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

For double-sided printing, set register 'D' to 1:

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

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

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

Register 'S' is ignored for TTY devices.

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

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

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

FILES

`doc.tmac`	The main manual macro package.
`mdoc.tmac`	A wrapper file to call `doc.tmac`.
`mdoc/doc-common`	Common strings, definitions, stuff related typographic output.
`mdoc/doc-nroff`	Definitions used for a TTY output device.
`mdoc/doc-ditroff`	Definitions used for all other devices.
`mdoc.local`	Local additions and customizations.
`andoc.tmac`	Use this file if you don't know whether the **-mdoc** or the **-man** package should be used. Multiple man pages (in either format) can be handled.

SEE ALSO

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

BUGS

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

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

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

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

GROFF_ME

NAME

groff_me – troff macros for formatting papers

SYNOPSIS

groff –me [options] file ...
groff –m me [options] file ...

DESCRIPTION

This manual page describes the GNU version of the −me macros, which is part of the groff document formatting system. This version can be used with both GNU troff and Unix troff. This package of *troff* macro definitions provides a canned formatting facility for technical papers in various formats.

The macro requests are defined below. Many *troff* requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines

Output of the *pic, eqn, refer,* and *tbl* preprocessors is acceptable as input.

FILES

c:/progra 1/groff/share/groff/1.20/tmac/me.tmac (a wrapper file for e.tmac)
c:/progra 1/groff/share/groff/1.20/tmac/e.tmac

SEE ALSO

groff(1), **troff**(1)
−me Reference Manual, Eric P. Allman
Writing Papers with Groff Using −me

REQUESTS

This list is incomplete; see *The −me Reference Manual* for interesting details.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, e.g., abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to

			center the equation.
.GE	-	yes	End *gremlin* picture.
.GS	-	yes	Begin *gremlin* picture.
.PE	-	yes	End *pic* picture.
.PS	-	yes	Begin *pic* picture.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bu	-	yes	Begin bulleted paragraph
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *´x´y´z´*	*´´´´*	no	Set even footer to x y z
.eh *´x´y´z´*	*´´´´*	no	Set even header to x y z
.fo *´x´y´z´*	*´´´´*	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *´x´y´z´*	*´´´´*	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.np	1	yes	Start numbered paragraph.
.of *´x´y´z´*	*´´´´*	no	Set odd footer to x y z
.oh *´x´y´z´*	*´´´´*	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sm *x*	-	no	Set *x* in a smaller pointsize.
.sz +*n*	10p	no	Augment the point size by *n* points.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x.*

GROFF_MM

NAME

groff_mm – groff mm macros

SYNOPSIS

groff –mm [*options*. . .] [*files*. . .]

DESCRIPTION

The groff mm macros are intended to be compatible with the DWB mm macros with the following limitations:

- No Bell Labs localisms are implemented.
- The macros OK and PM are not implemented.
- groff mm does not support cut marks.

mm is intended to support easy localization. Use **mmse** as an example how to adapt the output format to a national standard. Localized strings are collected in the file 'c:/progra 1/groff/share/groff/1.20/tmac/*xx*.tmac', where *xx* denotes the two-letter code for the *language*, as defined in the ISO 639 standard. For Swedish, this is 'sv.tmac' – not 'se', which is the ISO 3166 two-letter code for the *country* (as used for the output format localization).

A file called **locale** or *country***_locale** is read after the initialization of the global variables. It is therefore possible to localize the macros with a different company name and so on.

In this manual, square brackets are used to show optional arguments.

Number registers and strings

Many macros can be controlled by number registers and strings. A number register is assigned with the **nr** command:

.nr *XXX* [±]*n* [*i*]

XXX is the name of the register, *n* is the value to be assigned, and *i* is the increment value for auto-increment. *n* can have a plus or minus sign as a prefix if an increment or decrement of the current value is wanted. (Auto-increment or auto-decrement occurs if the number register is used with a plus or minus sign, **\n+[***XXX***]** or **\n-[***XXX***]**.)

Strings are defined with **ds**.

.ds *YYY string*

The string is assigned everything to the end of the line, even blanks. Initial blanks in *string* should be prefixed with a double-quote. (Strings are used in the text as ***[***YYY***]**.)

Special formatting of number registers

A number register is printed with normal digits if no format has been given. Set the format with **af**:

.af *R c*

R is the name of the register, *c* is the format.

Form	**Sequence**
1	0, 1, 2, 3, . . .
001	000, 001, 002, 003, . . .
i	0, i, ii, iii, iv, . . .
I	0, I, II, III, IV, . . .
a	0, a, b, c, . . ., z, aa, ab, . . .
A	0, A, B, C, . . ., Z, AA, AB, . . .

Fonts

In **mm**, the fonts (or rather, font styles) **R** (normal), **I** (italic), and **B** (bold) are hardwired to font positions **1**, **2**, and **3**, respectively. Internally, font positions are used for backwards compatibility. From a practical point of view it doesn't make a big difference – a different font family can still be selected with a call to the **.fam** request or using **groff**'s **–f** command line option. On the other hand, if you want to replace just, say, font **B**, you have to replace the font at position 2 (with a call to '.fp 2 . . .').

Macros

)E *level text*
: Add heading text *text* to the table of contents with *level*, which is either 0 or in the range 1 to 7. See also **.H**. This macro is used for customized tables of contents.

1C [**1**]
: Begin one-column processing. A **1** as an argument disables the page break. Use wide footnotes, small footnotes may be overprinted.

2C
: Begin two-column processing. Splits the page in two columns. It is a special case of **MC**. See also **1C**.

AE
: Abstract end, see **AS**.

AF [*name-of-firm*]
: Author's firm, should be called before **AU**, see also **COVER**.

AL [*type* [*text-indent* [**1**]]]
: Start auto-increment list. Items are numbered beginning with one. The *type* argument controls the format of numbers.

Arg	**Description**
1	Arabic (the default)
A	Upper-case letters (A-Z)
a	Lower-case letters (a-z)
I	Upper-case roman
i	Lower-case roman

text-indent sets the indentation and overrides **Li**. A third argument prohibits printing of a blank line before each item.

APP *name text*
: Begin an appendix with name *name*. Automatic naming occurs if *name* is **""**. The appendices start with **A** if automatic naming is used. A new page is ejected, and a header is also produced if the number variable **Aph** is non-zero. This is the default. The appendix always appears in the 'List of contents' with correct page numbers. The name 'APPENDIX' can be changed by setting the string **App** to the desired text. The string **Apptxt** contains the current appendix text.

APPSK *name pages text*
: Same as **.APP**, but the page number is incremented with *pages*. This is used when diagrams or other non-formatted documents are included as appendices.

AS [*arg* [*indent*]]
: Abstract start. Indentation is specified in 'ens', but scaling is allowed. Argument *arg* controls where the abstract is printed.

An abstract is not printed at all in external letters (**MT 5**). The *indent* parameter controls the indentation of both margins, otherwise normal text indentation is used.

AST [*title*]
: Abstract title. Default is 'ABSTRACT'. Sets the text above the abstract text.

AT *title1* [*title2* [...]]
: Author's title. **AT** must appear just after each **AU**. The title shows up after the name in the signature block.

AU [*name* [*initials* [*loc* [*dept* [*ext* [*room* [*arg* [*arg* [*arg*]]]]]]]]]
: Author information. Specifies the author of the memo or paper, and is printed on the cover sheet and on other similar places. **AU** must not appear before **TL**. The author information can contain initials, location, department, telephone extension, room number or name and up to three extra arguments.

AV [*name* [**1**]]
: Approval signature. Generates an approval line with place for signature and date. The string 'APPROVED:' can be changed with variable **Letapp**; it is replaced with an empty lin if there is a second argument. The string 'Date' can be changed with variable **Letdate**.

AVL [*name*]
: Letter signature. Generates a line with place for signature.

B [*bold-text* [*prev-font-text* [*bold* [. . .]]]]
Begin boldface. No limit on the number of arguments. All arguments are concatenated to one word; the first, third and so on is printed in boldface.

B1 Begin box (as the ms macro). Draws a box around the text. The text is indented one character, and the right margin is one character shorter.

B2 End box. Finishes the box started with **B1**.

BE End bottom block, see **BS**.

BI [*bold-text* [*italic-text* [*bold-text* [. . .]]]]
Bold-italic. No limit on the number of arguments, see **B**.

BL [*text-indent* [**1**]]
Start bullet list. Initializes a list with a bullet and a space in the beginning of each list item (see **LI**). *text-indent* overrides the default indentation of the list items set by number register **Pi**. A third argument prohibits printing of a blank line before each item.

BR [*bold-text* [*roman-text* [*bold-text* [. . .]]]]
Bold-roman. No limit on the number of arguments.

BS Bottom block start. Begins the definition of a text block which is printed at the bottom of each page. The block ends with **BE**.

BVL *text-indent* [*mark-indent* [**1**]]
Start of broken variable-item list. Broken variable-item list has no fixed mark, it assumes that every **LI** has a mark instead. The text always begins at the next line after the mark. *text-indent* sets the indentation to the text, and *mark-indent* the distance from the current indentation to the mark. A third argument prohibits printing of a blank line before each item.

COVER [*arg*]
Begin a coversheet definition. It is important that **.COVER** appears before any normal text. This macro uses *arg* to build the filename 'c:/progra 1/groff/share/groff/1.20/tmac/mm/*arg*.cov'. Therefore it is possible to create unlimited types of cover sheets. 'ms.cov' is supposed to look like the ms cover sheet. **.COVER** requires a **.COVEND** at the end of the cover definition. Always use this order of the cover macros:

```
.COVER
.TL
.AF
.AU
.AT
.AS
.AE
.COVEND
```

However, only **.TL** and **.AU** are required.

COVEND
Finish the cover description and print the cover page. It is defined in the cover file.

DE Display end. Ends a block of text or display that begins with **DS** or **DF**.

DF [*format* [*fill* [*rindent*]]]
Begin floating display (no nesting allowed). A floating display is saved in a queue and is printed in the order entered. *Format*, *fill*, and *rindent* are the same as in **DS**. Floating displays are controlled by the two number registers **De** and **Df**.

De register

Df register

DL [*text-indent* [**1** [**1**]]]
Dash list start. Begins a list where each item is printed after a dash. *text-indent* changes the default indentation of the list items set by number register **Pi**. A second argument prevents an empty line between each list item. See **LI**. A third argument prohibits printing of a blank line

before each item.

DS [*format* [*fill* [*rindent*]]]
Static display start. Begins collection of text until **DE**. The text is printed together on the same page, unless it is longer than the height of the page. **DS** can be nested arbitrarily.

format

The values ‘L’, ‘I’, ‘C’, and ‘CB’ can also be specified as ‘0’, ‘1’, ‘2’, and ‘3’, respectively, for compatibility reasons.

fill

""	Line-filling turned off.
none	Line-filling turned off.
N	Line-filling turned off.
F	Line-filling turned on.

‘N’ and ‘F’ can also be specified as ‘0’ and ‘1’, respectively.

By default, an empty line is printed before and after the display. Setting number register **Ds** to 0 prevents this. *rindent* shortens the line length by that amount.

EC [*title* [*override* [*flag* [*refname*]]]]
Equation title. Sets a title for an equation. The *override* argument changes the numbering.

flag

EC uses the number register **Ec** as a counter. It is possible to use **.af** to change the format of the number. If number register **Of** is 1, the format of title uses a dash instead of a dot after the number.

The string **Le** controls the title of the List of Equations; default is ‘LIST OF EQUATIONS’. The List of Equations is only printed if number register **Le** is 1. The default is 0. The string **Liec** contains the word ‘Equation’, which is printed before the number. If *refname* is used, then the equation number is saved with **.SETR**, and can be retrieved with ‘**.GETST** *refname*’.

Special handling of the title occurs if **EC** is used inside **DS/DE**; it is not affected by the format of **DS**.

EF [*arg*]
Even-page footer, printed just above the normal page footer on even pages. See **PF**.

This macro defines string **EOPef**.

EH [*arg*]
Even-page header, printed just below the normal page header on even pages. See **PH**.

This macro defines string **TPeh**.

EN Equation end, see **EQ**.

EOP End-of-page user-defined macro. This macro is called instead of the normal printing of the footer. The macro is executed in a separate environment, without any trap active. See **TP**.

strings available to EOP

EOPf	argument of **PF**
EOPef	argument of **EF**
EOPof	argument of **OF**

EPIC [**−L**] *width height* [*name*]
Draw a box with the given *width* and *height*. It also prints the text *name* or a default string if *name* is not specified. This is used to include external pictures; just give the size of the picture. **−L** left-adjusts the picture; the default is to center. See **PIC**.

EQ [*label*]
Equation start. **EQ/EN** are the delimiters for equations written for **eqn**(1). **EQ/EN** must be inside of a **DS/DE** pair, except if **EQ** is used to set options for **eqn** only. The *label* argument appears at the right margin of the equation, centered vertically within the **DS/DE** block, unless number register **Eq** is 1. Then the label appears at the left margin.

If there are multiple **EQ/EN** blocks within a single **DS/DE** pair, only the last equation label (if any) is printed.

EX [*title* [*override* [*flag* [*refname*]]]]
Exhibit title. The arguments are the same as for **EC**. **EX** uses the number register **Ex** as a counter. The string **Lx** controls the title of the List of Exhibits; default is 'LIST OF EXHIBITS'. The List of Exhibits is only printed if number register **Lx** is 1, which is the default. The string **Liex** contains the word 'Exhibit', which is printed before the number. If *refname* is used, the exhibit number is saved with **.SETR**, and can be retrieved with '**.GETST** *refname*'.

Special handling of the title occurs if **EX** is used inside **DS/DE**; it is not affected by the format of **DS**.

FC [*closing*]
Print 'Yours very truly,' as a formal closing of a letter or memorandum. The argument replaces the default string. The default is stored in string variable **Letfc**.

FD [*arg* [**1**]]
Footnote default format. Controls the hyphenation (hyphen), right margin justification (adjust), and indentation of footnote text (indent). It can also change the label justification (ljust).

arg	**hyphen**	**adjust**	**indent**	**ljust**
0	no	yes	yes	left
1	yes	yes	yes	left
2	no	no	yes	left
3	yes	no	yes	left
4	no	yes	no	left
5	yes	yes	no	left
6	no	no	no	left
7	yes	no	no	left
8	no	yes	yes	right
9	yes	yes	yes	right
10	no	no	yes	right
11	yes	no	yes	right

An argument greater than or equal to 11 is considered as value 0. Default for **mm** is 10.

FE Footnote end.

FG [*title* [*override* [*flag* [*refname*]]]]
Figure title. The arguments are the same as for **EC**. **FG** uses the number register **Fg** as a counter. The string **Lf** controls the title of the List of Figures; default is 'LIST OF FIGURES'. The List of Figures is only printed if number register **Lf** is 1, which is the default. The string **Lifg** contains the word 'Figure', which is printed before the number. If *refname* is used, then the figure number is saved with **.SETR**, and can be retrieved with '**.GETST** *refname*'.

Special handling of the title occurs if **FG** is used inside **DS/DE**, it is not affected by the format of **DS**.

FS [*label*]
Footnote start. The footnote is ended by **FE**. By default, footnotes are automatically numbered; the number is available in string **F**. Just add ***F** in the text. By adding *label*, it is possible to have other number or names on the footnotes. Footnotes in displays are now possible. An empty line separates footnotes; the height of the line is controlled by number register **Fs**, default value is 1.

GETHN *refname* [*varname*]
Include the header number where the corresponding '**SETR** *refname*' was placed. This is displayed as 'X.X.X.' in pass 1. See **INITR**. If *varname* is used, **GETHN** sets the string variable *varname* to the header number.

GETPN *refname* [*varname*]
Include the page number where the corresponding '**SETR** *refname*' was placed. This is

displayed as ‘9999’ in pass 1. See **INITR**. If *varname* is used, **GETPN** sets the stringvariable *varname* to the page number.

GETR *refname*

Combine **GETHN** and **GETPN** with the text ‘chapter’ and ‘, page’. The string **Qrf** contains the text for the reference:

.ds Qrf See chapter \\*[Qrfh], page \\*[Qrfp].

Qrf may be changed to support other languages. Strings **Qrfh** and **Qrfp** are set by **GETR** and contain the page and header number, respectively.

GETST *refname* [*varname*]

Include the string saved with the second argument to **.SETR**. This is a dummy string in pass 1. If *varname* is used, **GETST** sets it to the saved string. See **INITR**.

H *level* [*heading-text* [*heading-suffix*]]

Numbered section heading. Section headers can have a level between 1 and 14; level 1 is the top level. The text is given in *heading-text*, and must be surrounded by double quotes if it contains spaces. *heading-suffix* is added to the header in the text but not in the table of contents. This is normally used for footnote marks and similar things. Don’t use ***F** in *heading-suffix*, it doesn’t work. A manual label must be used, see **FS**.

A call to the paragraph macro **P** directly after **H** is ignored. **H** takes care of spacing and indentation.

Page ejection before heading

Number register **Ej** controls page ejection before the heading. By default, a level-one heading gets two blank lines before it; higher levels only get one. A new page is ejected before each first-level heading if number register **Ej** is 1. All levels below or equal the value of **Ej** get a new page. Default value for **Ej** is 0.

Heading break level

A line break occurs after the heading if the heading level is less or equal to number register **Hb**. Default value is 2.

Heading space level

A blank line is inserted after the heading if the heading level is less or equal to number register **Hs**. Default value is 2.

Text follows the heading on the same line if the level is greater than both **Hb** and **Hs**.

Post-heading indent

Indentation of the text after the heading is controlled by number register **Hi**. Default value is 0.

Hi

Centered section headings

All headings whose level is equal or below number register **Hc** and also less than or equal to **Hb** or **Hs** are centerered.

Font control of the heading

The font of each heading level is controlled by string **HF**. It contains a font number or font name for each level. Default value is

2 2 2 2 2 2 2 2 2 2 2 2 2 2

(all headings in italic). This could also be written as

I I I I I I I I I I I I I I

Note that some other implementations use **3 3 2 2 2 2 2** as the default value. All omitted values are presumed to have value 1.

Point size control

String **HP** controls the point size of each heading, in the same way as **HF** controls the font. A value of 0 selects the default point size. Default value is

0 0 0 0 0 0 0 0 0 0 0 0 0 0

Beware that only the point size changes, not the vertical size. The latter can be controlled by the user-specified macros **HX** and/or **HZ**.

Heading counters

Fourteen number registers named **H1** up to **H14** contain the counter for each heading level. The values are printed using arabic numerals; this can be changed with the macro **HM** (see below). All marks are concatenated before printing. To avoid this, set number register **Ht** to 1. This only prints the current heading counter at each heading.

Automatic table of contents

All headings whose level is equal or below number register **Cl** are saved to be printed in the table of contents. Default value is 2.

Special control of the heading, user-defined macros

The following macros can be defined by the user to get a finer control of vertical spacing, fonts, or other features. Argument *level* is the level-argument to **H**, but 0 for unnumbered headings (see **HU**). Argument *rlevel* is the real level; it is set to number register **Hu** for unnumbered headings. Argument *heading-text* is the text argument to **H** and **HU**.

HX *level rlevel heading-text*

This macro is called just before the printing of the heading. The following registers are available for **HX**. Note that **HX** may alter **}0**, **}2**, and **;3**.

}0 (string)

Contains the heading mark plus two spaces if *rlevel* is non-zero, otherwise empty.

;0 (register)

Contains the position of the text after the heading. 0 means that the text should follow the heading on the same line, 1 means that a line break should occur before the text, and 2 means that a blank line should separate the heading and the text.

}2 (string)

Contains two spaces if register **;0** is 0. It is used to separate the heading from the text. The string is empty if **;0** is non-zero.

;3 (register)

Contains the needed space in units after the heading. Default is 2v. Can be used to change things like numbering (**}0**), vertical spacing (**}2**), and the needed space after the heading.

HY *dlevel rlevel heading-text*

This macro is called after size and font calculations and might be used to change indentation.

HZ *dlevel rlevel heading-text*

This macro is called after the printing of the heading, just before **H** or **HU** exits. Can be used to change the page header according to the section heading.

HC [*hyphenation-character*]

Set hyphenation character. Default value is '\'. Resets to the default if called without argument. Hyphenation can be turned off by setting number register **Hy** to 0 at the beginning of the file.

HM [*arg1* [*arg2* [... [*arg14*]]]]
Heading mark style. Controls the type of marking for printing of the heading counters. Default is 1 for all levels.

Argument	
1	Arabic numerals.
0001	Arabic numerals with leading zeroes, one or more.
A	upper-case alphabetic
a	lower-case alphabetic
I	upper-case roman numerals
i	lower-case roman numerals
""	Arabic numerals.

HU *heading-text*
Unnumbered section header. **HU** behaves like **H** at the level in number register **Hu**. See **H**.

HX *dlevel rlevel heading-text*
User-defined heading exit. Called just before printing the header. See **H**.

HY *dlevel rlevel heading-text*
User-defined heading exit. Called just before printing the header. See **H**.

HZ *dlevel rlevel heading-text*
User-defined heading exit. Called just after printing the header. See **H**.

I [*italic-text* [*prev-font-text* [*italic-text* [...]]]]
Italic. Changes the font to italic if called without arguments. With one argument it sets the word in italic. With two arguments it concatenates them and sets the first word in italic and the second in the previous font. There is no limit on the number of argument; all are concatenated.

IA [*addressee-name* [*title*]]
Begin specification of the addressee and addressee's address in letter style. Several names can be specified with empty **IA/IE**-pairs, but only one address. See **LT**.

IB [*italic-text* [*bold-text* [*italic-text* [...]]]]
Italic-bold. Even arguments are printed in italic, odd in boldface. See **I**.

IE End the address specification after **IA**.

INITI *type filename* [*macro*]
Initialize the new index system and set the filename to collect index lines in with **IND**. Argument *type* selects the type of index: page number, header marks or both. The default is page numbers.

It is also possible to create a macro that is responsible for formatting each row; just add the name of the macro as a third argument. The macro is then called with the index as argument(s).

type

INITR *filename*
Initialize the refence macros. References are written to stderr and are supposed to be written to '*filename*.qrf'. Requires two passes with groff; this is handled by a separate program called **mmroff**(1). This program exists because **groff**(1) by default deactivates the unsafe operations that are required by **INITR**. The first pass looks for references, and the second one includes them. **INITR** can be used several times, but it is only the first occurrence of **INITR** that is active.

See also **SETR**, **GETPN**, and **GETHN**.

IND *arg1* [*arg2* [...]]
Write a line in the index file selected by **INITI** with all arguments and the page

number or header mark separated by tabs.

Examples

```
arg1\tpage number
arg1\targ2\tpage number
arg1\theader mark
arg1\tpage number\theader mark
```

INDP Print the index by running the command specified by string variable **Indcmd**, which has 'sort –t\t' as the default value. **INDP** reads the output from the command to form the index, by default in two columns (this can be changed by defining **TYIND**). The index is printed with string variable **Index** as header, default is 'INDEX'. One-column processing is reactivated after the list. **INDP** calls the user-defined macros **TXIND**, **TYIND**, and **TZIND** if defined. **TXIND** is called before printing the string 'INDEX', **TYIND** is called instead of printing 'INDEX', and **TZIND** is called after the printing and should take care of restoring to normal operation again.

ISODATE [**0**]
Change the predefined date string in **DT** to ISO-format, this is, 'YYYY-MM-DD'. This can also be done by adding **–rIso=1** on the command line. Reverts to old date format if argument is **0**.

IR [*italic-text* [*roman-text* [*italic-text* [...]]]]
Italic-roman. Even arguments are printed in italic, odd in roman. See **I**.

LB *text-indent mark-indent pad type* [*mark* [*LI-space* [*LB-space*]]]
List-begin macro. This is the common macro used for all lists. *text-indent* is the number of spaces to indent the text from the current indentation.

pad and *mark-indent* control where to put the mark. The mark is placed within the mark area, and *mark-indent* sets the number of spaces before this area. By default it is 0. The mark area ends where the text begins. The start of the text is still controlled by *text-indent*.

The mark is left-justified whitin the mark area if *pad* is 0. If *pad* is greater than 0, *mark-indent* is ignored, and the mark is placed *pad* spaces before the text. This right-justifies the mark.

If *type* is 0 the list either has a hanging indentation or, if argument *mark* is given, the string *mark* as a mark.

If *type* is greater than 0 automatic numbering occurs, using arabic numbers if *mark* is empty. *mark* can then be any of '1', 'A', 'a', 'I', or 'i'.

type selects one of six possible ways to display the mark.

type

1	x.
2	x)
3	(x)
4	[x]
5	<x>
6	{x}

Every item in the list gets *LI-space* number of blank lines before them. Default is 1.

LB itself prints *LB-space* blank lines. Default is 0.

LC [*list-level*]
List-status clear. Terminates all current active lists down to *list-level*, or 0 if no argmuent is given. This is used by **H** to clear any active list.

LE [**1**] List end. Terminates the current list. **LE** outputs a blank line if an argument is given.

LI [*mark* [**1**|**2**]]
List item preceding every item in a list. Without argument, **LI** prints the mark

determined by the current list type. By giving **LI** one argument, it uses that as the mark instead. Two arguments to **LI** makes *mark* a prefix to the current mark. There is no separating space between the prefix and the mark if the second argument is '2' instead of '1'. This behaviour can also be achieved by setting number register **Limsp** to zero. A zero length *mark* makes a hanging indentation instead.

A blank line is printed before the list item by default. This behaviour can be controlled by number register **Ls**. Pre-spacing occurs for each list level less than or equal to **Ls**. Default value is 99. There is no nesting limit.

The indentation can be changed through number register **Li**. Default is 6.

All lists begin with a list initialization macro, **LB**. There are, however, seven predefined list types to make lists easier to use. They all call **LB** with different default values.

AL	Automatically Incremented List
ML	Marked List
VL	Variable-Item List
BL	Bullet List
DL	Dash List
RL	Reference List
BVL	Broken Variable List.

These lists are described at other places in this manual. See also **LB**.

LT [*arg*]
Format a letter in one of four different styles depending on the argument. See also section **INTERNALS**.

LO *type* [*arg*]
Specify options in letter (see **.LT**). This is a list of the standard options:

MC *column-size* [*column-separation*]
Begin multiple columns. Return to normal with **1C**. **MC** creates as many columns as the current line length permits. *column-size* is the width of each column, and *column-separation* is the space between two columns. Default separation is *column-size*/15. See also **1C**.

ML *mark* [*text-indent* [**1**]]
Marked list start. The *mark* argument is printed before each list item. *text-indent* sets the indent and overrides **Li**. A third argument prohibits printing of a blank line before each item.

MT [*arg* [*addressee*]]
Memorandum type. The argument *arg* is part of a filename in 'c:/progra 1/groff/share/groff/1.20/tmac/mm/*.MT'. Memorandum types 0 to 5 are supported, including type 'string' (which gets internally mapped to type 6). *addressee* just sets a variable, used in the AT.T macros.

arg

0	Normal memorandum, no type printed.
1	Memorandum with 'MEMORANDUM FOR FILE' printed.
2	Memorandum with 'PROGRAMMER'S NOTES' printed.
3	Memorandum with 'ENGINEER'S NOTES' printed.
4	Released paper style.
5	External letter style.

See also **COVER/COVEND**, a more flexible type of front page.

MOVE *y-pos* [*x-pos* [*line-length*]]
Move to a position, setting page offset to *x-pos*. If *line-length* is not given, the difference between current and new page offset is used. Use **PGFORM** without arguments to return to normal.

MULB *cw1 space1* [*cw2 space2* [*cw3* ...]]
Begin a special multi-column mode. All columns widths must be specified. The space between the columns must be specified also. The last column does not need any space definition. **MULB** starts a diversion, and **MULE** ends the diversion and prints the columns. The unit for the width and space arguments is ‘n’, but **MULB** accepts all normal unit specifications like ‘c’ and ‘i’. **MULB** operates in a separate environment.

MULN Begin the next column. This is the only way to switch the column.

MULE End the multi-column mode and print the columns.

nP [*type*]
Print numbered paragraph with header level two. See **.P**.

NCOL Force printing to the next column. Don’t use this together with the **MUL*** macros, see **2C**.

NS [*arg* [**1**]]
Print different types of notations. The argument selects between the predefined type of notations. If the second argument is available, then the argument becomes the entire notation. If the argument doesn’t select a predefined type, it is printed as ‘Copy (*arg*) to’. It is possible to add more standard notations, see the string variables **Letns** and **Letnsdef**.

Arg	**Notation**
none	Copy To
""	Copy To
1	Copy To (with att.) to
2	Copy To (without att.) to
3	Att.
4	Atts.
5	Enc.
6	Encs.
7	Under separate cover
8	Letter to
9	Memorandum to
10	Copy (with atts.) to
11	Copy (without atts.) to
12	Abstract Only to
13	Complete Memorandum to
14	CC

ND *new-date*
New date. Overrides the current date. Date is not printed if *new-date* is an empty string.

OF [*arg*]
Odd-page footer, a line printed just above the normal footer. See **EF** and **PF**.

This macro defines string **EOPof**.

OH [*arg*]
Odd-page header, a line printed just below the normal header. See **EH** and **PH**.

This macro defines string **TPoh**.

OP Make sure that the following text is printed at the top of an odd-numbered page. Does not output an empty page if currently at the top of an odd page.

P [*type*]
Begin new paragraph. **P** without argument produces left-justified text, even the first line of the paragraph. This is the same as setting *type* to 0. If the argument is 1, the first line of text following **P** is indented by the number of spaces in number register **Pi**, by default 5.

Instead of giving an argument to **P** it is possible to set the paragraph type in number register **Pt**. Using 0 and 1 is the same as adding that value to **P**. A value of 2 indents all paragraphs, except after headings, lists, and displays (this value can't be used as an argument to **P** itself).

The space between two paragraphs is controlled by number register **Ps**, and is 1 by default (one blank line).

PGFORM [*linelength* [*pagelength* [*pageoffset* [**1**]]]]
: Set line length, page length, and/or page offset. This macro can be used for special formatting, like letter heads and other. It is normally the first command in a file, though it is not necessary. **PGFORM** can be used without arguments to reset everything after a **MOVE** call. A line break is done unless the fourth argument is given. This can be used to avoid the page number on the first page while setting new width and length. (It seems as if this macro sometimes doesn't work too well. Use the command line arguments to change line length, page length, and page offset instead.)

PGNH No header is printed on the next page. Used to get rid of the header in letters or other special texts. This macro must be used before any text to inhibit the page header on the first page.

PIC [**−L**] [**−C**] [**−R**] [**−I** *n*] *filename* [*width* [*height*]]
: Include a PostScript file in the document. The macro depends on **mmroff**(1) and **INITR**. The arguments **−L**, **−C**, **−R**, and **−I** *n* adjust the picture or indent it. The optional *width* and *height* can also be given to resize the picture.

PE Picture end. Ends a picture for **pic**(@MAN1EXT).

PF [*arg*]
: Page footer. **PF** sets the line to be printed at the bottom of each page. Empty by default. See **PH** for the argument specification.

This macro defines string **EOPf**.

PH [*arg*]
: Page header, a line printed at the top of each page. The argument should be specified as

"'*left-part*'*center-part*'*right-part*'"

where *left-part*, *center-part,* and *right-part* are printed left-justified, centered, and right justified, respectively. Within the argument to **PH**, the character ‘’ is changed to the current page number. The default argument is

"’’- -’’"

which gives the page number between two dashes.

This macro defines string **TPh**.

PS Picture start (from pic). Begins a picture for **pic**(1).

PX Page header user-defined exit. This macro is called just after the printing of the page header in *no-space* mode.

R Roman. Return to roman font, see also **I**.

RB [*roman-text* [*bold-text* [*roman-text* [. . .]]]]
: Roman-bold. Even arguments are printed in roman, odd in boldface. See **I**.

RD [*prompt* [*diversion* [*string*]]]
: Read from standard input to diversion and/or string. The text is saved in a diversion named *diversion*. Recall the text by writing the name of the diversion after a dot on an empty line. A string is also defined if *string* is given. *Diversion* and/or *prompt* can be empty ("").

RF Reference end. Ends a reference definition and returns to normal processing. See **RS**.

RI [*roman-text* [*italic-text* [*roman-text* [...]]]]
Print even arguments in roman, odd in italic. See **I**.

RL [*text-indent*[**1**]]
Reference list start. Begins a list where each item is preceded with an automatically incremented number between square brackets. *text-indent* changes the default indentation.

RP [*arg1* [*arg2*]]
Produce reference page. This macro can be used if a reference page is wanted somewhere in the document. It is not needed if **TC** is used to produce a table of contents. The reference page is then printed automatically.

The reference counter is not reset if *arg1* is 1.

arg2 tells **RP** whether to eject a page or not.

arg2

The reference items are separated by a blank line. Setting number register **Ls** to 0 suppresses the line.

The string **Rp** contains the reference page title and is set to 'REFERENCES' by default.

RS [*string-name*]
Begin an automatically numbered reference definition. Put the string ***(Rf** where the reference mark should be and write the reference between **RS/RF** at next new line after the reference mark. The reference number is stored in number register **:R**. If *string-name* is given, a string with that name is defined and contains the current reference mark. The string can be referenced as ***[***string-name***]** later in the text.

S [*size* [*spacing*]]
Set point size and vertical spacing. If any argument is equal to 'P', the previous value is used. A 'C' means current value, and 'D' the default value. If '+' or '−' is used before the value, the current value is incremented or decremented, respectively.

SA [*arg*]
Set right-margin justification. Justification is turned on by default. No argument or value '0' turns off justification, and '1' turns on justification.

SETR *refname* [*string*]
Remember the current header and page number as *refname*. Saves *string* if *string* is defined. *string* is retrieved with **.GETST**. See **INITR**.

SG [*arg* [**1**]]
Signature line. Prints the authors name(s) after the formal closing. The argument is appended to the reference data, printed at either the first or last author. The reference data is the location, department, and initials specified with **.AU**. It is printed at the first author if the second argument is given, otherwise at the last. No reference data is printed if the author(s) is specified through **.WA/.WE**. See section **INTERNALS**.

SK [*pages*]
Skip pages. If *pages* is 0 or omitted, a skip to the next page occurs unless it is already at the top of a page. Otherwise it skips *pages* pages.

SM *string1* [*string2* [*string3*]]
Make a string smaller. If *string2* is given, *string1* is made smaller and *string2* stays at normal size, concatenated with *string1*. With three arguments, everything is concatenated, but only *string2* is made smaller.

SP [*lines*]
Space vertically. *lines* can have any scaling factor, like '3i' or '8v'. Several **SP** calls in a line only produces the maximum number of lines, not the sum. **SP** is ignored also until the first text line in a page. Add **\&** before a call to **SP** to avoid this.

TAB Reset tabs to every 5n. Normally used to reset any previous tab positions.

TB [*title* [*override* [*flag* [*refname*]]]]
Table title. The arguments are the same as for **EC**. **TB** uses the number register **Tb** as a counter. The string **Lt** controls the title of the List of Tables; default value is 'LIST OF TABLES'. The List of Tables is only printed if number register **Lt** is 1, which is the default. The string **Litb** contains the word 'TABLE', which is printed before the number.

Special handling of the title occurs if **TB** is used inside **DS/DE**, it is not affected by the format of **DS**.

TC [*slevel* [*spacing* [*tlevel* [*tab* [*h1* [*h2* [*h3* [*h4* [*h5*]]]]]]]]]
Table of contents. This macro is normally used as the last line of the document. It generates a table of contents with headings up to the level controlled by number register **Cl**. Note that **Cl** controls the saving of headings, it has nothing to do with **TC**. Headings with a level less than or equal to *slevel* get *spacing* number of lines before them. Headings with a level less than or equal to *tlevel* have their page numbers right-justified with dots or spaces separating the text and the page number. Spaces are used if *tab* is greater than zero, dots otherwise. Other headings have the page number directly at the end of the heading text (*ragged-right*).

The rest of the arguments is printed, centered, before the table of contents.

The user-defined macros **TX** and **TY** are used if **TC** is called with at most four arguments. **TX** is called before the printing of the string 'CONTENTS', and **TY** is called instead of printing 'CONTENTS'.

Equivalent macros can be defined for list of figures, tables, equations and exhibits by defining **TX***xx* or **TY***xx*, where *xx* is 'Fg', 'TB', 'EC', or 'EX', respectively.

String **Ci** can be set to control the indentations for each heading-level. It must be scaled, like

```
.ds Ci .25i .5i .75i 1i 1i
```

By default, the indentation is controlled by the maximum length of headings in each level.

The string variables **Lifg**, **Litb**, **Liex**, **Liec**, and **Licon** contain 'Figure', 'TABLE', 'Exhibit', 'Equation', and 'CONTENTS', respectively. These can be redefined to other languages.

TE Table end. See **TS**.

TH [**N**] Table header. See **TS**. **TH** ends the header of the table. This header is printed again if a page break occurs. Argument 'N' isn't implemented yet.

TL [*charging-case-number* [*filing-case-number*]]
Begin title of memorandum. All text up to the next **AU** is included in the title. *charging-case-number* and *filing-case-number* are saved for use in the front page processing.

TM [*num1* [*num2* [. . .]]]
Technical memorandum numbers used in **.MT**. An unlimited number of arguments may be given.

TP Top-of-page user-defined macro. This macro is called instead of the normal page header. It is possible to get complete control over the header. Note that the header and the footer are printed in a separate environment. Line length is preserved, though. See **EOP**.

strings available to TP

TPh	argument of **PH**
TPeh	argument of **EH**
TPoh	argument of **OH**

TS [**H**] Table start. This is the start of a table specification to **tbl**(1). **TS** ends with **TE**. Argument 'H' tells **mm** that the table has a header. See **TH**.

TX User-defined table of contents exit. This macro is called just before **TC** prints the word 'CONTENTS'. See **TC**.

TY User-defined table of contents exit. This macro is called instead of printing 'CONTENTS'. See **TC**.

VERBON [*flag* [*point-size* [*font*]]]
Begin verbatim output using Courier font. Usually for printing programs. All characters have equal width. The point size can be changed with the second argument. By specifying a third argument it is possible to use another font instead of Courier. *flag* controls several special features. Its value is the sum of all wanted features.

VERBOFF
End verbatim output.

VL *text-indent* [*mark-indent* [**1**]]
Variable-item list. It has no fixed mark, it assumes that every **LI** has a mark instead. *text-indent* sets the indent to the text, and *mark-indent* the distance from the current indentation to the mark. A third argument prohibits printing of a blank line before each item.

VM [**−T**] [*top* [*bottom*]]
Vertical margin. Adds extra vertical top and margin space. Option **−T** sets the total space instead. If no argument is given, reset the margin to zero, or the default ('7v 5v') if **−T** has been used. It is higly recommended that macros **TP** and/or **EOP** are defined if using **−T** and setting top and/or bottom margin to less than the default.

WA [*writer-name* [*title*]]
Begin specification of the writer and writer's address. Several names can be specified with empty **WA/WE** pairs, but only one address.

WE End the address specification after **.WA**.

WC [*format1*] [*format2*] [. . .]
Footnote and display width control.

Strings used in mm

App A string containing the word 'APPENDIX'.

Apptxt The current appendix text.

EM Em dash string

H1txt Updated by **.H** and **.HU** to the current heading text. Also updated in table of contents . friends.

HF Font list for headings, '2 2 2 2 2 2 2' by default. Non-numeric font names may also be used.

HP Point size list for headings. By default, this is '0 0 0 0 0 0 0' which is the same as '10 10 10 10 10 10 10'.

Index Contains the string 'INDEX'.

Indcmd
Contains the index command. Default value is 'sort −t\t'.

Lifg String containing 'Figure'.

Litb String containing 'TABLE'.

Liex String containing 'Exhibit'.

Liec String containing 'Equation'.

Licon String containing 'CONTENTS'.

Lf Contains the string 'LIST OF FIGURES'.

Lt Contains the string 'LIST OF TABLES'.

Lx Contains the string 'LIST OF EXHIBITS'.

Le Contains the string 'LIST OF EQUATIONS'.

Letfc Contains the string 'Yours very truly,', used in **.FC**.

Letapp Contains the string 'APPROVED:', used in **.AV**.

Letdate
Contains the string 'Date', used in **.AV**.

LetCN Contains the string 'CONFIDENTIAL', used in **.LO CN**.

LetSA Contains the string 'To Whom It May Concern:', used in **.LO SA**.

LetAT Contains the string 'ATTENTION:', used in **.LO AT**.

LetSJ Contains the string 'SUBJECT:', used in **.LO SJ**.

LetRN Contains the string 'In reference to:', used in **.LO RN**.

Letns is an array containing the different strings used in **.NS**. It is really a number of string variables prefixed with **Letns!**. If the argument doesn't exist, it is included between () with **Letns!copy** as a prefix and **Letns!to** as a suffix. Observe the space after 'Copy' and before 'to'.

Name	**Value**
Letns!0	Copy to
Letns!1	Copy (with att.) to
Letns!2	Copy (without att.) to
Letns!3	Att.
Letns!4	Atts.
Letns!5	Enc.
Letns!6	Encs.
Letns!7	Under separate cover
Letns!8	Letter to
Letns!9	Memorandum to
Letns!10	Copy (with atts.) to
Letns!11	Copy (without atts.) to
Letns!12	Abstract Only to
Letns!13	Complete Memorandum to
Letns!14	CC
Letns!copy	Copy \"
Letns!to	" to

Letnsdef
Define the standard notation used when no argument is given to **.NS**. Default is 0.

MO1 - MO12
Strings containing the month names 'January' through 'December'.

Qrf String containing 'See chapter \\*[Qrfh], page \\n[Qrfp].'.

Rp Contains the string 'REFERENCES'.

Tcst Contains the current status of the table of contents and list of figures, etc. Empty outside of **.TC**. Useful in user-defined macros like **.TP**.

Value	**Meaning**
co	Table of contents
fg	List of figures
tb	List of tables
ec	List of equations
ex	List of exhibits
ap	Appendix

Tm Contains the string '\(tm', the trade mark symbol.

Verbnm
Argument to **.nm** in the **.VERBON** command. Default is 1.

Number variables used in mm

Aph Print an appendix page for every new appendix if this number variable is non-zero. No output occurs if **Aph** is zero, but there is always an appendix entry in the 'List of contents'.

Cl Contents level (in the range 0 to 14). The contents is saved if a heading level is lower than or equal to the value of **Cl**. Default is 2.

Cp Eject page between list of table, list of figure, etc., if the value of **Cp** is zero. Default is 0.

D Debug flag. Values greater than zero produce debug information of increasing verbosity. A value of 1 gives information about the progress of formatting. Default is 0.

De If set to 1, eject after floating display is output. Default is 0.

Dsp If defined, it controls the space output before and after static displays. Otherwise the value of **Lsp** is used.

Df Control floating keep output. This is a number in the range 0 to 5, with a default value of 5. See **.DF**.

Ds If set to 1, use the amount of space stored in register **Lsp** before and after display. Default is 1.

Ej If set to 1, eject page before each first-level heading. Default is 0.

Eq Equation labels are left-adjusted if set to 0 and right-adjusted if set to 1. Default is 0.

Fs Footnote spacing. Default is 1.

H1 - H7
Heading counters

H1dot Append a dot after the level-one heading number if value is greater than zero. Default is 1.

H1h A copy of number register **H1**, but it is incremented just before the page break. Useful in user-defined header macros.

Hb Heading break level. A number in the range 0 to 14, with a default value of 2. See **.H**.

Hc Heading centering level. A number in the range 0 to 14, with a default value value of 0. See **.H**.

Hi Heading temporary indent. A number in the range 0 to 2, with a default value of 1.

Hps Heading pre-space level. If the heading level is less than or equal to **Hps**, two lines precede the section heading instead of one. Default is first level only. The real amount of lines is controlled by the variables **Hps1** and **Hps2**.

Hps1 Number of lines preceding **.H** if the heading level is greater than **Hps**. Value is in units, default is 0.5.

Hps2 Number of lines preceding **.H** if the heading level is less than or equal to **Hps**. Value is in units, default is 1.

Hs Heading space level. A number in the range 0 to 14, with a default value of 2. See **.H**.

Hss Number of lines following **.H** if the heading level is less than or equal to **Hs**. Value is in units, default is 1.

Ht Heading numbering type.

0	multiple levels (1.1.1, 1.1.2, etc.)
1	single level

Default is 0.

Hu Unnumbered heading level. Default is 2.

Hy Hyphenation status of text body.

0	no hyphenation
1	hyphenation on, set to value 14

Default is 0.

Iso Set this variable to 1 on the command line to get an ISO-formatted date string (**−rIso=1**). Useless inside of a document.

L Page length, only for command line settings.

Letwam
Maximum lines in return-address, used in **.WA/.WE**. Default is 14.

Lf, **Lt**, **Lx**, **Le**
Enable (1) or disable (0) the printing of List of figures, List of tables, List of exhibits and List of equations, respectively. Default values are Lf=1, Lt=1, Lx=1, and Le=0.

Li List indentation, used by **.AL**. Default is 6.

Limsp A flag controlling the insertion of space between prefix and mark in automatic lists (**.AL**).

0	no space
1	emit space

Ls List space threshold. If current list level is greater than **Ls** no spacing occurs around lists. Default is 99.

Lsp The vertical space used by an empty line. The default is 0.5v in troff mode and 1v in nroff mode.

N Page numbering style.

Default is 0. See also the number registers **Sectf** and **Sectp**.

Np A flag to control whether paragraphs are numbered.

0	not numbered
1	numbered in first-level headings.

Default is 0.

O Page offset, only for command line settings.

Of Format of figure, table, exhibit, and equation titles.

0	". "
1	" - "

Default is 0.

P Current page-number, normally the same as ‘’ unless ‘section-page’ numbering style is enabled.

Pi Paragraph indentation. Default is 5.

Pgps A flag to control whether header and footer point size should follow the current settings or just change when the header and footer are defined.

Ps Paragraph spacing. Default is 1.

Pt Paragraph type.

Default is 0.

Sectf A flag controlling ‘section-figures’ numbering style. A non-zero value enables this. See also register **N**.

Sectp A flag controlling ‘section-page’ numbering style. A non-zero value enables this. See also register **N**.

Si Display indentation. Default is 5.

Verbin Indentation for **.VERBON**. Default is 5n.

W Line length, only for command line settings.

.mgm Always 1.

INTERNALS

The letter macros are using different submacros depending on the letter type. The name of the submacro has the letter type as suffix. It is therefore possible to define other letter types, either in the national macro-file, or as local additions. **.LT** sets the number variables **Pt** and **Pi** to 0 and 5, respectively. The following strings and macros must be defined for a new letter type.

let@init_*type*
  This macro is called directly by **.LT**. It is supposed to initialize variables and other stuff.

let@head_*type*
  This macro prints the letter head, and is called instead of the normal page header. It is supposed to remove the alias **let@header**, otherwise it is called for all pages.

let@sg_*type name title n flag* [*arg1* [*arg2* [. . .]]]
  .SG is calling this macro only for letters; memorandums have its own processing. *name* and *title* are specified through **.WA**/**.WB**. *n* is the counter, 1-max, and *flag* is true for the last name. Any other argument to **.SG** is appended.

let@fc_*type closing*
  This macro is called by **.FC**, and has the formal closing as the argument.

.LO is implemented as a general option-macro. It demands that a string named **Let***type* is defined, where *type* is the letter type. **.LO** then assigns the argument to the string variable **let*lo-***type*.

AUTHOR

Jörgen Hägg, Lund, Sweden <jh@axis.se>.

FILES

c:/progra 1/groff/share/groff/1.20/tmac/m.tmac

c:/progra 1/groff/share/groff/1.20/tmac/mm/*.cov

c:/progra 1/groff/share/groff/1.20/tmac/mm/*.MT

c:/progra 1/groff/share/groff/1.20/tmac/mm/locale

SEE ALSO

groff(1), **troff**(1), **tbl**(1), **pic**(1), **eqn**(1)
groff_mmse(7)

GROFF_MMSE

NAMN

groff_mmse – svenska mm makro för groff

SYNTAX

groff –mmse [*flaggor*. . .] [*filer*. . .]

BESKRIVNING

mmse är en svensk variant av **mm**. Alla texter är översatta. En A4 sida får text som är 13 cm bred, 3.5 cm indragning samt är 28.5 cm hög. Det finns stöd för brevuppställning enligt svensk standard för vänster och högerjusterad text.

COVER kan använda *se_ms* som argument. Detta ger ett svenskt försättsblad. Se **groff_mm**(7) för övriga detaljer.

BREV

Tillgängliga brevtyper:

.LT SVV
Vänsterställd löptext med adressat i position T0 (vänsterställt).

.LT SVH
Högerställd löptext med adressat i position T4 (passar fönsterkuvert).

Följande extra LO-variabler används.

.LO DNAMN *namn*
Anger dokumentets namn.

.LO MDAT *datum*
Mottagarens datum, anges under **Ert datum:** (**LetMDAT**).

.LO BIL *sträng*
Anger bilaga, nummer eller sträng med **Bilaga** (**LetBIL**) som prefix.

.LO KOMP *text*
Anger kompletteringsuppgift.

.LO DBET *beteckning*
Anger dokumentbeteckning eller dokumentnummer.

.LO BET *beteckning*
Anger beteckning (ärendebeteckning i form av diarienummer eller liknande).

.LO SIDOR *antal*
Anger totala antalet sidor och skrivs ut efter sidnumret inom parenteser.

Om makrot **.TP** är definierat anropas det efter utskrift av brevhuvudet. Där lägger man lämpligen in postadress och annat som brevfot.

SKRIVET AV

Jörgen Hägg, Lund, Sweden <Jorgen.Hagg@axis.se>

FILER

c:/progra 1/groff/share/groff/1.20/tmac/mse.tmac

c:/progra 1/groff/share/groff/1.20/tmac/mm/se_∗.cov

SE OCKSÅ

groff(1), **troff**(1), **tbl**(1), **pic**(1), **eqn**(1)
groff_mm(7)

GROFF_MOM

NAME

groff_mom – groff ‘mom’ macros

SYNOPSIS

groff –mom [*files*. . .]
groff –m mom [*files*. . .]

DESCRIPTION

mom (“my own macros”, “my other macros”, “maximum overdrive macros”, . . .) is a macro set for groff, designed primarily to format documents for PostScript output.

mom provides two categories of macros: macros for typesetting and macros for document processing. The typesetting macros provide access to groff’s typesetting power in ways that are simpler to master and to use than groff’s primitives. The document processing macros provide customizable markup “tags” that allow the user to design and output professional-looking documents with a minimum of typesetting intervention.

mom comes with her own (very) complete documentation in HTML format.

FILES

om.tmac
– the main macro file

mom.tmac
– a wrapper file that calls om.tmac directly.

c:/progra 1/groff/share/doc/groff/1.20/html/momdoc/toc.html
– entry point to the HTML documentation

c:/progra 1/groff/share/doc/groff/1.20/examples/*.mom
– example files using mom

AUTHOR

mom was written by Peter Schaffter Please send bug reports to the groff bug mailing list or directly to the author at the address, above.

GROFF_MS

NAME

groff_ms – groff ms macros

SYNOPSIS

groff –ms [*options*. . .] [*files*. . .]
groff –m ms [*options*. . .] [*files*. . .]

DESCRIPTION

This manual page describes the GNU version of the *ms* macros, part of the *groff* typesetting system. The *ms* macros are mostly compatible with the documented behavior of the 4.3 BSD Unix *ms* macros (see *Differences from troff ms* below for details). The *ms* macros are suitable for reports, letters, books, and technical documentation.

USAGE

The *ms* macro package expects files to have a certain amount of structure. The simplest documents can begin with a paragraph macro and consist of text separated by paragraph macros or even blank lines. Longer documents have a structure as follows:

Document type
If you use the **RP** (report) macro at the beginning of the document, *groff* prints the cover page information on its own page; otherwise it prints the information on the first page with your document text immediately following. Other document formats found in AT.T *troff* are specific to AT.T or Berkeley, and are not supported in *groff ms*.

Format and layout
By setting number registers, you can change your document's type (font and size), margins, spacing, headers and footers, and footnotes. See *Document control registers* below for more details.

Cover page
A cover page consists of a title, and optionally the author's name and institution, an abstract, and the date. See *Cover page macros* below for more details.

Body Following the cover page is your document. It consists of paragraphs, headings, and lists.

Table of contents
Longer documents usually include a table of contents, which you can add by placing the **TC** macro at the end of your document.

Document control registers

The following table lists the document control number registers. For the sake of consistency, set registers related to margins at the beginning of your document, or just after the **RP** macro.

Margin settings

Reg.	Definition	Effective	Default
`PO`	Page offset (left margin)	next page	1i
`LL`	Line length	next paragraph	6i
`LT`	Header/footer length	next paragraph	6i
`HM`	Top (header) margin	next page	1i
`FM`	Bottom (footer) margin	next page	1i

Text settings

Reg.	Definition	Effective	Default
PS	Point size	next paragraph	10p
VS	Line spacing (leading)	next paragraph	12p
PSINCR	Point size increment for section headings of increasing importance	next heading	1p
GROWPS	Heading level beyond which PSINCR is ignored	next heading	0

Paragraph settings

Reg.	Definition	Effective	Default
PI	Initial indent	next paragraph	5n
PD	Space between paragraphs	next paragraph	0.3v
QI	Quoted paragraph indent	next paragraph	5n
PORPHANS	Number of initial lines to be kept together	next paragraph	1
HORPHANS	Number of initial lines to be kept with heading	next heading	1

Footnote settings

Reg.	Definition	Effective	Default
FL	Footnote length	next footnote	\n[LL]∗5/6
FI	Footnote indent	next footnote	2n
FF	Footnote format	next footnote	0
FPS	Point size	next footnote	\n[PS]-2
FVS	Vert. spacing	next footnote	\n[FPS]+2
FPD	Para. spacing	next footnote	\n[PD]/2

Other settings

Reg.	Definition	Effective	Default
MINGW	Minimum width between columns	next page	2n

Cover page macros

Use the following macros to create a cover page for your document in the order shown.

.RP [no]

Specifies the report format for your document. The report format creates a separate cover page. With no **RP** macro, *groff* prints a subset of the cover page on page 1 of your document.

If you use the optional **no** argument, *groff* prints a title page but does not repeat any of the title page information (title, author, abstract, etc.) on page 1 of the document.

.P1 (P-one) Prints the header on page 1. The default is to suppress the header.

.DA [*xxx***]**

(optional) Print the current date, or the arguments to the macro if any, on the title page (if specified) and in the footers. This is the default for *nroff*.

.ND [*xxx***]**

(optional) Print the current date, or the arguments to the macro if any, on the title page (if specified) but not in the footers. This is the default for *troff*.

.TL Specifies the document title. *Groff* collects text following the **TL** macro into the title, until reaching the author name or abstract.

.AU Specifies the author's name. You can specify multiple authors by using an **AU** macro for each author.

.AI Specifies the author's institution. You can specify multiple institutions.

.AB [no]
Begins the abstract. The default is to print the word **ABSTRACT**, centered and in italics, above the text of the abstract. The option **no** suppresses this heading.

.AE End the abstract.

Paragraphs

Use the **PP** macro to create indented paragraphs, and the **LP** macro to create paragraphs with no initial indent.

The **QP** macro indents all text at both left and right margins. The effect is identical to the HTML **<BLOCKQUOTE>** element. The next paragraph or heading returns margins to normal.

The **XP** macro produces an exdented paragraph. The first line of the paragraph begins at the left margin, and subsequent lines are indented (the opposite of **PP**).

For each of the above paragraph types, and also for any list entry introduced by the **IP** macro (described later), the document control register **PORPHANS**, sets the *minimum* number of lines which must be printed, after the start of the paragraph, and before any page break occurs. If there is insufficient space remaining on the current page to accommodate this number of lines, then a page break is forced *before* the first line of the paragraph is printed.

Similarly, when a section heading (see subsection *Headings* below) preceeds any of these paragraph types, the **HORPHANS** document control register specifies the *minimum* number of lines of the paragraph which must be kept on the same page as the heading. If insufficient space remains on the current page to accommodate the heading and this number of lines of paragraph text, then a page break is forced *before* the heading is printed.

Headings

Use headings to create a hierarchical structure for your document. By default, the *ms* macros print headings in **bold** using the same font family and point size as the body text. For output devices which support scalable fonts, this behaviour may be modified, by defining the document control registers, **GROWPS** and **PSINCR**.

The following heading macros are available:

.NH *xx* Numbered heading. The argument *xx* is either a numeric argument to indicate the level of the heading, or *S xx xx* "..." to set the section number explicitly. If you specify heading levels out of sequence, such as invoking **.NH 3** after **.NH 1**, *groff* prints a warning on standard error.

If the **GROWPS** register is set to a value greater than the level of the heading, then the point size of the heading will be increased by **PSINCR** units over the text size specified by the **PS** register, for each level by which the heading level is less than the value of **GROWPS**. For example, the sequence:

```
.nr PS 10
.nr GROWPS 3
.nr PSINCR 1.5p
.
.NH 1
Top Level Heading
.
.NH 2
Second Level Heading
.
.NH 3
Third Level Heading
```

will cause "*1. Top Level Heading*" to be printed in 13pt **bold** text, followed by "*1.1. Second Level Heading*" in 11.5pt **bold** text, while "*1.1.1. Third Level Heading*", and all more deeply nested heading levels, will remain in the 10pt **bold** text which is specified by the **PS** register.

Note that the value stored in **PSINCR** is interpreted in *groff* basic units; the *p* scaling factor should be employed, when assigning a value specified in points.

The style used to represent the section number, within a numbered heading, is controlled by the **SN-STYLE** string; this may be set to either the **SN-DOT** or the **SN-NO-DOT** style, (described below), by aliasing **SN-STYLE** accordingly. By default, **SN-STYLE** is initialised by defining the alias

```
.als SN-STYLE SN-DOT
```

it may be changed to the **SN-NO-DOT** style, if preferred, by defining the alternative alias

```
.als SN-STYLE SN-NO-DOT
```

Any such change becomes effective with the first use of **.NH**, *after* the new alias is defined.

After invoking **.NH**, the assigned heading number is available in the strings **SN-DOT** (as it appears in the default formatting style for numbered headings, with a terminating period following the number), and **SN-NO-DOT** (with this terminating period omitted). The string **SN** is also defined, as an alias for **SN-DOT**; if preferred, the user may redefine it as an alias for **SN-NO-DOT**, by including the initialisation:

```
.als SN SN-NO-DOT
```

at any time; the change becomes effective with the next use of **.NH**, *after* the new alias is defined.

.SH [*xx*]

Unnumbered subheading. The use of the optional *xx* argument is a GNU extension, which adjusts the point size of the unnumbered subheading to match that of a numbered heading, introduced using **.NH** *xx* with the same value of *xx*. For example, given the same settings for **PS**, **GROWPS** and **PSINCR**, as used in the preceeding **.NH** example, the sequence:

```
.SH 2
An Unnumbered Subheading
```

will print "*An Unnumbered Subheading*" in 11.5pt **bold** text.

Highlighting

The *ms* macros provide a variety of methods to highlight or emphasize text:

.B [*txt* [*post* [*pre*]]]

Sets its first argument in **bold type**. If you specify a second argument, *groff* prints it in the previous font after the bold text, with no intervening space (this allows you to set punctuation after the highlighted text without highlighting the punctuation). Similarly, it prints the third argument (if any) in the previous font **before** the first argument. For example,

```
.B foo ) (
```

prints (**foo**).

If you give this macro no arguments, *groff* prints all text following in bold until the next highlighting, paragraph, or heading macro.

.R [*txt* [*post* [*pre*]]]

Sets its first argument in roman (or regular) type. It operates similarly to the **B** macro otherwise.

.I [*txt* [*post* [*pre*]]]

Sets its first argument in *italic type*. It operates similarly to the **B** macro otherwise.

.CW [*txt* [*post* [*pre*]]]

Sets its first argument in a constant width face. It operates similarly to the **B** macro otherwise.

.BI [*txt* [*post* [*pre*]]]

Sets its first argument in bold italic type. It operates similarly to the **B** macro otherwise.

.BX [*txt*]

Prints its argument and draws a box around it. If you want to box a string that contains spaces, use a digit-width space (\0).

.UL [*txt* [*post*]]

Prints its first argument with an underline. If you specify a second argument, *groff* prints it in the previous font after the underlined text, with no intervening space.

.LG Prints all text following in larger type (2 points larger than the current point size) until the next font size, highlighting, paragraph, or heading macro. You can specify this macro multiple times to enlarge the point size as needed.

.SM Prints all text following in smaller type (2 points smaller than the current point size) until the next type size, highlighting, paragraph, or heading macro. You can specify this macro multiple times to reduce the point size as needed.

.NL Prints all text following in the normal point size (that is, the value of the **PS** register).

{text****}**
Print the enclosed *text* as a superscript.

Indents

You may need to indent sections of text. A typical use for indents is to create nested lists and sublists.

Use the **RS** and **RE** macros to start and end a section of indented text, respectively. The **PI** register controls the amount of indent.

You can nest indented sections as deeply as needed by using multiple, nested pairs of **RS** and **RE**.

Lists

The **IP** macro handles duties for all lists. Its syntax is as follows:

.IP [*marker* **[***width***]]**

The *marker* is usually a bullet character **\(bu** for unordered lists, a number (or auto-incrementing number register) for numbered lists, or a word or phrase for indented (glossary-style) lists.

The *width* specifies the indent for the body of each list item. Once specified, the indent remains the same for all list items in the document until specified again.

Tab stops

Use the **ta** request to set tab stops as needed. Use the **TA** macro to reset tabs to the default (every 5n). You can redefine the **TA** macro to create a different set of default tab stops.

Displays and keeps

Use displays to show text-based examples or figures (such as code listings). Displays turn off filling, so lines of code can be displayed as-is without inserting **br** requests in between each line. Displays can be *kept* on a single page, or allowed to break across pages. The following table shows the display types available.

Use the **DE** macro to end any display type. The macros **Ds** and **De** were formerly provided as aliases for **DS** and **DE**, respectively, but they have been removed, and should no longer be used. X11 documents which actually use **Ds** and **De** always load a specific macro file from the X11 distribution (macros.t) which provides proper definitions for the two macros.

To *keep* text together on a page, such as a paragraph that refers to a table (or list, or other item) immediately following, use the **KS** and **KE** macros. The **KS** macro begins a block of text to be kept on a single page, and the **KE** macro ends the block.

You can specify a *floating keep* using the **KF** and **KE** macros. If the keep cannot fit on the current page, *groff* holds the contents of the keep and allows text following the keep (in the source file) to fill in the remainder of the current page. When the page breaks, whether by an explicit **bp** request or by reaching the end of the page, *groff* prints the floating keep at the top of the new page. This is useful for printing large graphics or tables that do not need to appear exactly where specified.

The macros **B1** and **B2** can be used to enclose a text within a box; **.B1** begins the box, and **.B2** ends it. Text in the box is automatically placed in a diversion (keep).

Tables, figures, equations, and references

The *-ms* macros support the standard *groff* preprocessors: *tbl*, *pic*, *eqn*, and *refer*. Mark text meant for preprocessors by enclosing it in pairs of tags as follows:

.TS [H] and **.TE**
Denotes a table, to be processed by the *tbl* preprocessor. The optional **H** argument instructs *groff* to create a running header with the information up to the **TH** macro. *Groff* prints the header at the beginning of the table; if the table runs onto another page, *groff* prints the header

on the next page as well.

.PS and **.PE**
: Denotes a graphic, to be processed by the *pic* preprocessor. You can create a *pic* file by hand, using the AT.T *pic* manual available on the Web as a reference, or by using a graphics program such as *xfig*.

.EQ [*align*] and **.EN**
: Denotes an equation, to be processed by the *eqn* preprocessor. The optional *align* argument can be **C**, **L**, or **I** to center (the default), left-justify, or indent the equation.

.[and **.]**
: Denotes a reference, to be processed by the *refer* preprocessor. The GNU *refer*(1) manual page provides a comprehensive reference to the preprocessor and the format of the bibliographic database.

Footnotes

The *ms* macros provide a flexible footnote system. You can specify a numbered footnote by using the ** escape, followed by the text of the footnote enclosed by **FS** and **FE** macros.

You can specify symbolic footnotes by placing the mark character (such as **\(dg** for the dagger character) in the body text, followed by the text of the footnote enclosed by **FS \(dg** and **FE** macros.

You can control how *groff* prints footnote numbers by changing the value of the **FF** register as follows:

0	Prints the footnote number as a superscript; indents the footnote (default).
1	Prints the number followed by a period (like 1.) and indents the footnote.
2	Like 1, without an indent.
3	Like 1, but prints the footnote number as a hanging paragraph.

You can use footnotes safely within keeps and displays, but avoid using numbered footnotes within floating keeps. You can set a second ** between a ** and its corresponding **.FS**; as long as each **.FS** occurs *after* the corresponding ****** and the occurrences of **.FS** are in the same order as the corresponding occurrences of ******.

Headers and footers

There are three ways to define headers and footers:

- Use the strings **LH**, **CH**, and **RH** to set the left, center, and right headers; use **LF**, **CF**, and **RF** to set the left, center, and right footers. This works best for documents that do not distinguish between odd and even pages.
- Use the **OH** and **EH** macros to define headers for the odd and even pages; and **OF** and **EF** macros to define footers for the odd and even pages. This is more flexible than defining the individual strings. The syntax for these macros is as follows:

  .OH '*left***'***center***'***right***'**

  You can replace the quote (') marks with any character not appearing in the header or footer text.

You can also redefine the **PT** and **BT** macros to change the behavior of the header and footer, respectively. The header process also calls the (undefined) **HD** macro after **PT ;** you can define this macro if you need additional processing after printing the header (for example, to draw a line below the header).

Margins

You control margins using a set of number registers. The following table lists the register names and defaults:

Reg.	**Definition**	**Effective**	**Default**

`PO`	Page offset (left margin)	next page	1i
`LL`	Line length	next paragraph	6i
`LT`	Header/footer length	next paragraph	6i
`HM`	Top (header) margin	next page	1i
`FM`	Bottom (footer) margin	next page	1i

Note that there is no right margin setting. The combination of page offset and line length provide the information necessary to derive the right margin.

Multiple columns

The *ms* macros can set text in as many columns as will reasonably fit on the page. The following macros are available. All of them force a page break if a multi-column mode is already set. However, if the current mode is single-column, starting a multi-column mode does *not* force a page break.

.1C Single-column mode.

.2C Two-column mode.

.MC [*width* **[***gutter***]]**
Multi-column mode. If you specify no arguments, it is equivalent to the **2C** macro. Otherwise, *width* is the width of each column and *gutter* is the space between columns. The **MINGW** number register is the default gutter width.

Creating a table of contents

Wrap text that you want to appear in the table of contents in **XS** and **XE** macros. Use the **TC** macro to print the table of contents at the end of the document, resetting the page number to **i** (Roman numeral 1).

You can manually create a table of contents by specifying a page number as the first argument to **XS**. Add subsequent entries using the **XA** macro. For example:

```
.XS 1
Introduction
.XA 2
A Brief History of the Universe
.XA 729
Details of Galactic Formation
. . .
.XE
```

Use the **PX** macro to print a manually-generated table of contents without resetting the page number.

If you give the argument **no** to either **PX** or **TC**, *groff* suppresses printing the title specified by the ***[TOC]** string.

Fractional point sizes

Traditionally, the *ms* macros only support integer values for the document's font size and vertical spacing. To overcome this restriction, values larger than or equal to 1000 are taken as fractional values, multiplied by 1000. For example, '.nr PS 10250' sets the font size to 10.25 points.

The following four registers accept fractional point sizes: **PS**, **VS**, **FPS**, and **FVS**.

Due to backwards compatibility, the value of **VS** must be smaller than 40000 (this is 40.0 points).

DIFFERENCES FROM troff ms

The *groff ms* macros are a complete re-implementation, using no original AT.T code. Since they take advantage of the extended features in *groff*, they cannot be used with AT.T *troff*. Other differences include:

- The internals of *groff ms* differ from the internals of Unix *ms*. Documents that depend upon implementation details of Unix *ms* may not format properly with *groff ms*.
- The error-handling policy of *groff ms* is to detect and report errors, rather than silently to ignore them.
- Some Bell Labs localisms are not implemented by default. However, if you call the otherwise undocumented **SC** section-header macro, you will enable implementations of three other archaic Bell Labs macros: **UC**, **P1**, and **P2**. These are not enabled by default because (a) they were not documented, in the original *ms manual*, and (b) the **P1** and **UC** macros both collide with different

macros in the Berkeley version of *ms*.

These emulations are sufficient to give back the 1976 Kernighan . Cherry paper *Typsetting Mathematics – User's Guide* its section headings, and restore some text that had gone missing as arguments of undefined macros. No warranty express or implied is given as to how well the typographic details these produce match the original Bell Labs macros.

- Berkeley localisms, in particular the **TM** and **CT** macros, are not implemented.
- *Groff ms* does not work in compatibility mode (e.g., with the **−C** option).
- There is no support for typewriter-like devices.
- *Groff ms* does not provide cut marks.
- Multiple line spacing is not supported (use a larger vertical spacing instead).
- Some Unix *ms* documentation says that the **CW** and **GW** number registers can be used to control the column width and gutter width, respectively. These number registers are not used in *groff ms*.
- Macros that cause a reset (paragraphs, headings, etc.) may change the indent. Macros that change the indent do not increment or decrement the indent, but rather set it absolutely. This can cause problems for documents that define additional macros of their own. The solution is to use not the **in** request but instead the **RS** and **RE** macros.
- The number register **GS** is set to 1 by the *groff ms* macros, but is not used by the Unix *ms* macros. Documents that need to determine whether they are being formatted with Unix *ms* or *groff ms* should use this number register.
- To make *groff ms* use the default page offset (which also specifies the left margin), the **PO** number register must stay undefined until the first **ms** macro is evaluated. This implies that **PO** should not be used early in the document, unless it is changed also: Remember that accessing an undefined register automatically defines it.

Strings

You can redefine the following strings to adapt the *groff ms* macros to languages other than English:

String	Default Value
`REFERENCES`	References
`ABSTRACT`	ABSTRACT
`TOC`	Table of Contents
`MONTH1`	January
`MONTH2`	February
`MONTH3`	March
`MONTH4`	April
`MONTH5`	May
`MONTH6`	June
`MONTH7`	July
`MONTH8`	August
`MONTH9`	September
`MONTH10`	October
`MONTH11`	November
`MONTH12`	December

The **\*-** string produces an em dash — like this.

Use **\*Q** and **\*U** to get a left and right typographer's quote, respectively, in *troff* (and plain quotes in *nroff*).

Text Settings

The **FAM** string sets the default font family. If this string is undefined at initialization, it is set to Times.

The point size, vertical spacing, and inter-paragraph spacing for footnotes are controlled by the number registers **FPS**, **FVS**, and **FPD**; at initialization these are set to **\n(PS-2**, **\n[FPS]+2**, and **\n(PD/2**, respectively. If any of these registers are defined before initialization, the initialization macro does not change them.

The hyphenation flags (as set by the **hy** request) are set from the **HY** register; the default is 14.

Improved accent marks (as originally defined in Berkeley's *ms* version) are available by specifying the **AM** macro at the beginning of your document. You can place an accent over most characters by specifying the string defining the accent directly after the character. For example, **n*** produces an n with a tilde over it.

NAMING CONVENTIONS

The following conventions are used for names of macros, strings and number registers. External names available to documents that use the *groff ms* macros contain only uppercase letters and digits.

Internally the macros are divided into modules; naming conventions are as follows:

- Names used only within one module are of the form *module***name*.
- Names used outside the module in which they are defined are of the form *module*@*name*.
- Names associated with a particular environment are of the form *environment*:*name*; these are used only within the **par** module.
- *name* does not have a module prefix.
- Constructed names used to implement arrays are of the form *array*!*index*.

Thus the groff ms macros reserve the following names:

- Names containing the characters *, @, and **:**.
- Names containing only uppercase letters and digits.

FILES

c:/progra 1/groff/share/groff/1.20/tmac/ms.tmac (a wrapper file for **s.tmac**)
c:/progra 1/groff/share/groff/1.20/tmac/s.tmac

SEE ALSO

groff(1), **troff**(1), **tbl**(1), **pic**(1), **eqn**(1), **refer**(1), *Groff: The GNU Implementation of troff* by Trent Fisher and Werner Lemberg.

AUTHOR

Original manual page by James Clark *et al*; rewritten by Larry Kollar (*lkollar@despammed.com*).

GROFF_TRACE

NAME

groff_trace – groff macro package trace.tmac

SYNOPSIS

[*options* . . .] [*files* . . .]

DESCRIPTION

The *trace* macro package of **groff**(1) can be a valuable tool for debugging documents written in the roff formatting language. A call stack trace is protocolled on standard error, this is, a diagnostic message is emitted on entering and exiting of a macro call. This greatly eases to track down an error in some macro.

This tracing process is activated by specifying the groff or troff command line option **–m trace**. This works also with the **groffer**(1) viewer program. A finer control can be obtained by including the macro file within the document by the groff macro call **.mso trace.tmac**. Only macros that are defined after this line are traced.

If command line option **–r trace-full=1** is given (or if this register is set in the document), number and string register assignments together with some other requests are traced also.

If some other macro package should be traced as well it must be specified after **–m trace** on the command line.

The macro file **trace.tmac** is unusual because it does not contain any macros to be called by a user. Instead, the existing macro definition and appending facilities are modified such that they display diagnostic messages.

EXAMPLES

In the following examples, a roff fragment is fed into groff via standard input. As we are only interested in the diagnostic messages (standard error) on the terminal, the normal formatted output (standard output) is redirected to the nirvana device */dev/null*. The resulting diagnostic messages are displayed directly below the corresponding example.

Command line option

Example:

sh# echo '. > .de test_macro > .. > .test_macro > .test_macro some dummy arguments > ' | groff -m trace >/dev/null *** .de test_macro *** de trace enter: .test_macro *** trace exit: .test_macro *** de trace enter: .test_macro "some" "dummy" "arguments" *** trace exit: .test_macro "some" "dummy" "arguments"

The entry and the exit of each macro call is displayed on the terminal (standard output) — together with the arguments (if any).

Nested macro calls

Example:

```
sh# echo '. > .de child > .. > .de parent > .child > .. > .parent > ' | groff -m trace >/dev/null
*** .de child *** .de parent *** de trace enter: .parent
 *** de trace enter: .child
 *** trace exit: .child *** trace exit: .parent
```

This shows that macro calls can be nested. This powerful feature can help to tack down quite complex call stacks.

Activating with .mso

Example:

sh# echo '. > .de before > .. > .mso trace.tmac > .de after > .. > .before > .after > .before > ' | groff >/dev/null *** de trace enter: .after *** trace exit: .after

Here, the tracing is activated within the document, not by a command line option. As tracing was not active when macro *before* was defined, no call of this macro is protocolled; on the other hand, the macro *after* is fully protocolled.

PROBLEMS

Because **trace.tmac** wraps the **.de** request (and its cousins), macro arguments are expanded one level more. This causes problems if an argument contains four backslashes or more to prevent too early expansion of the backslash. For example, this macro call

```
.foo \\\\n[bar]
```

normally passes '\\n[bar]' to macro '.foo', but with the redefined **.de** request it passes '\n[bar]' instead.

The solution to this problem is to use groff's **\E** escape which is an escape character not interpreted in copy mode, for example

```
.foo \En[bar]
```

FILES

The *trace* macros are kept in the file **trace.tmac** located in the *tmac directory*; see **groff_tmac**(5) for details.

ENVIRONMENT

$GROFF_TMAC_PATH
A colon-separated list of additional tmac directories in which to search for macro files; see **groff_tmac**(5) for details.

AUTHOR

This document is part of *groff*, the GNU roff distribution. It was written by Bernd Warken.

SEE ALSO

groff(1)
An overview of the groff system.

troff(1) For details on option **−m**.

groffer(1)
A viewer program for all kinds of roff documents.

groff_tmac(5)
A general description of groff macro packages.

groff(7)
A short reference for the groff formatting language.

A complete reference for all parts of the groff system is found in the groff **info**(1) file.

GROFF_WWW

NAME

groff_www – groff macros for authoring web pages

SYNOPSIS

groff –mwww [options] file ...

DESCRIPTION

This manual page describes the GNU –mwww macro package, which is part of the groff document formatting system. The manual page is very a basic guide, and the html device driver (**grohtml**) has been completely rewritten but still remains as in an alpha state. It has been included into the distribution so that a lot of people have a chance to test it. Note that this macro file is automatically called (via the **troffrc** file) if you use **–Thtml** or **–Txhtml**.

To see the hyperlinks in action, please format this man page with the **grohtml** device.

Here is a summary of the functions found in this macro set.

.JOBNAME	split output into multiple files
.HX	automatic heading level cut off
.BCL	specify colours on a web page
.BGIMG	specify background image
.URL	create a url using two parameters
.FTP	create an ftp reference
.MTO	create a html email address
.FTP	create an ftp reference
.TAG	generate an html name
.IMG	include an image file
.PIMG	include png image
.MPIMG	place png on the margin and wrap text around it
.HnS	begin heading
.HnE	end heading
.LK	emit automatically collected links.
.HR	produce a horizontal rule
.NHR	suppress automatic generation of rules.
.HTL	only generate HTML title
.HEAD	add data to <head> block
.ULS	unorder list begin
.ULE	unorder list end
.OLS	ordered list begin
.OLE	ordered list end
.DLS	definition list begin
.DLE	definition list end
.LI	insert a list item
.DC	generate a drop capital
.HTML	pass an html raw request to the device driver
.CDS	code example begin
.CDE	code example end
.ALN	place links on left of main text.
.LNS	start a new two-column table with links in the left.
.LNE	end the two-column table.
.LINKSTYLE	initialize default url attributes.

Output of the **pic**, **eqn**, **refer**, and **tbl** preprocessors is acceptable as input.

REQUESTS

.JOBNAME filename

Split output into multiple HTML files. A file is split whenever a .SH or .NH 1 is encountered. Its argument is the file stem name for future output files. This option is equivalent to **grohtml**'s **–j** option.

.HX n Specify the cut off depth when generating links from section headings. For example, a parameter of 2 would cause **grohtml** to generate a list of links for **.NH 1** and **.NH 2** but not for

.NH 3. Whereas

> **.HX 0**

tells **grohtml** that no heading links should be created at all. Another method for turning automatic headings off is by issuing the the command line switch **−P−l** to **groff**.

.BCL foreground background active not-visited visited
This macro takes five parameters: foreground, background, active hypertext link, hypertext link not yet visited, and visited hypertext link colour.

.BGIMG imagefile
the only parameter to this macro is the background image file.

.URL url [description] [after]
generates a URL using either one, two or three arguments. The first parameter is the actual URL, the second is the name of the link, and the third is optional stuff to be printed immediately afterwards. If **description** and **after** are absent then the **url** becomes the anchor text. Hyphenation is disabled while printing the actual URL; explicit breakpoints should be inserted with the **\:** escape. Here is how to encode foo ⟨`http://foo.org/`⟩:

> **.URL http://\:foo.\:org/ foo :**

If this is processed by a device other than **−Thtml** or **−Txhtml** it appears as:

> foo ⟨`http://foo.org`⟩:

The URL macro can be of any type; for example we can reference Eric Raymond's pic guide ⟨`pic.html`⟩ by:

> **.URL pic.html "Eric Raymond's pic guide"**

.MTO address [description] [after]
Generate an email html reference. The first argument is mandatory as the email address. The optional second argument is the text you see in your browser If an empty argument is given, **address** is used instead. An optional third argument is stuff printed immediately afterwards. Hyphenation is disabled while printing the actual email address. For example, Joe User ⟨`joe@user.org`⟩ was achieved by the following macro:

> **.MTO joe@user.org "Joe User"**

Note that all the URLs actually are treated as consuming no textual space in groff. This could be considered as a bug since it causes some problems. To circumvent this, **www.tmac** inserts a zero-width character which expands to a harmless space (only if run with **−Thtml** or **−Txhtml**).

.FTP url [description] [after]
indicates that data can be obtained via ftp. The first argument is the url and the second is the browser text. A third argument, similar to the macros above, is intended for stuff printed immediately afterwards. The second and the third parameter are optional. Hyphenation is disabled while printing the actual URL. As an example, here the location of the GNU ftp server ⟨`ftp://ftp.gnu.org/`⟩. The macro example above was specified by:

> **.FTP ftp://\:ftp.gnu.org/ "GNU ftp server" .**

.TAG name
Generates an html name tag from its argument. This can then be referenced using the URL ⟨`#URL`⟩ macro. As you can see, you must precede the tag name with **#** since it is a local reference. This link was achieved via placing a TAG in the URL description above; the source looks like this:

```
.TP
.B URL
generates
.TAG URL
a URL using either two or three arguments.
. . .
```

.IMG [-R|-L|-C] filename [width] [height]
Include a picture into the document. The first argument is the horizontal location: right, left, or center (**–R**, **–L**, or **–C**). Alignment is centered by default (-C). The second argument is the filename. The optional third and fourth arguments are the width and height. If the width is absent it defaults to 1 inch. If the height is absent it defaults to the width. This maps onto an html img tag. If you are including a png image then it is advisable to use the **PIMG** macro.

.PIMG [-R|-L|-C] filename [width [height]]
Include an image in PNG format. This macro takes exactly the same parameters as the **IMG** macro; it has the advantage of working with postscript and html devices also since it can automatically convert the image into the EPS format, using the following programs of the **netpbm** package: **pngtopnm**, **pnmcrop**, and **pnmtops**. If the document isn't processed with **–Thtml** or **–Txhtml** it is necessary to use the **–U** option of groff.

.MPIMG [-R|-L] [-G gap] filename [width [height]]
Place a PNG image on the margin and wrap text around it. The first parameters are optional. The alignment: left or right (**–L** or **–R**) specifies the margin where the picture is placed at. The default alignment is left (**-L**). Optionally, **–G** *gap* can be used to arrange a gap between the picture and the text that wraps around it. The default gap width is zero.
The first non-optional argument is the filename. The optional following arguments are the width and height. If the width is absent it defaults to 1 inch. If the height is absent it defaults to the width. Example:

```
.MPIMG -L -G 2c foo.png 3c 1.5c
```

The height and width may also be given as percentages. The PostScript device calculates the width from the **.l** register and the height from the **.p** register. For example:

```
.MPIMG -L -G 2c foo.png 15
```

.HnS n Begin heading. The numeric heading level *n* is specified by the first parameter. Use this macro if your headings contain URL, FTP or MTO macros. Example:

```
.HnS 1
.HR
GNU Troff
.URL http://groff.ffii.org (Groff)
— a
.URL http://www.gnu.org/ GNU
project.
Hosted by
.URL http://ffii.org/ FFII .
.HR
.HnE
```

In this case you might wish to disable automatic links to headings. This can be done via **–P–l** from the command line.

.HnE End heading.

.LK Force grohtml to place the automatically generated links at this position. If this manual page has been processed with **–Thtml** or **–Txhtml** those links can be seen right here.

.HR Generate a full-width horizontal rule for **–Thtml** and **–Txhtml**. No effect for all other devices.

.NHR Suppress generation of the top and bottom rules which grohtml emits by default.

.HTL Generate an HTML title only. This differs from the **TL** macro of the **ms** macro package which generates both an HTML title and an <H1> heading. Use it to provide an HTML title as search engine fodder but a graphic title in the document. The macro terminates when a space or break is seen (.sp, .br).

.HEAD
Add arbitrary HTML data to the <head> block. Ignored if not processed with **–Thtml** or **–Txhtml**. Example:

```
.HEAD "<link \
 rel=""icon"" \
 type=""image/png"" \
 href=""http://foo.org//bar.png""/>"
```

.HTML
All text after this macro is treated as raw html. If the document is processed without **−Thtml** or **−Txhtml** then the macro is ignored. Internally, this macro is used as a building block for other higher-level macros.

For example, the **BGIMG** macro is defined as

```
.de BGIMG
.  HTML <body background=\$1>
..
```

.DC l text [color]
Produce a drop capital. The first parameter is the letter to be dropped and enlarged, the second parameter **text** is the ajoining text whose height the first letter should not exceed. The optional third parameter is the color of the dropped letter. It defaults to black.

.CDS Start displaying a code section in constant width font.

.CDE End code display

.ALN [color] [percentage]
Place section heading links automatically to the left of the main text. The color argument is optional and if present indicates which HTML background color is to be used under the links. The optional percentage indicates the amount of width to devote to displaying the links. The default values are #eeeeee and 30 for color and percentage width, respectively. This macro should only be called once at the beginning of the document. After calling this macro each section heading emits an HTML table consisting of the links in the left and the section text on the right.

.LNS Start a new two-column table with links in the left column. This can be called if the document has text before the first .SH and if .ALN is used. Typically this is called just before the first paragraph and after the main title as it indicates that text after this point should be positioned to the right of the left-hand navigational links.

.LNE End a two-column table. This should be called at the end of the document if .ALN was used.

.LINKSTYLE color [fontstyle [openglyph closeglyph]]
Initialize default url attributes to be used if this macro set is not used with the HTML device. The macro set initializes itself with the following call

```
.LINKSTYLE blue C \[la] \[ra]
```

but these values will be superseded by a user call to LINKSTYLE.

SECTION HEADING LINKS

By default **grohtml** generates links to all section headings and places these at the top of the html document. (See LINKS ⟨#LK⟩ for details of how to switch this off or alter the position).

LIMITATIONS OF GROHTML

tbl information is currently rendered as a PNG image.

FILES

c:/progra 1/groff/share/groff/1.20/tmac/www.tmac

SEE ALSO

groff(1), **troff**(1) **grohtml**(1), **netpbm**(1)

AUTHOR

grohtml was written by Gaius Mulley ⟨gaius@glam.ac.uk⟩

BUGS

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

ROFF

NAME

roff – concepts and history of roff typesetting

DESCRIPTION

roff is the general name for a set of text formatting programs, known under names like **troff**, **nroff**, **ditroff**, **groff**, etc. A *roff* system consists of an extensible text formatting language and a set of programs for printing and converting to other text formats. Unix-like operating systems distribute a *roff* system as a core package.

The most common *roff* system today is the free software implementation GNU *roff*, **groff**(1). *groff* implements the look-and-feel and functionality of its ancestors, with many extensions.

The ancestry of *roff* is described in section **HISTORY**. In this document, the term *roff* always refers to the general class of roff programs, not to the **roff** command provided in early UNIX systems.

In spite of its age, *roff* is in wide use today, for example, the manual pages on UNIX systems (*man pages*), many software books, system documentation, standards, and corporate documents are written in roff. The *roff* output for text devices is still unmatched, and its graphical output has the same quality as other free type-setting programs and is better than some of the commercial systems.

roff is used to format UNIX *manual pages*, (or *man pages*), the standard documentation system on many UNIX-derived operating systems.

This document describes the history of the development of the *roff system*; some usage aspects common to all *roff* versions, details on the *roff* pipeline, which is usually hidden behind front-ends like **groff**(1); a general overview of the formatting language; some tips for editing *roff* files; and many pointers to further readings.

HISTORY

Document formatting by computer dates back to the 1960s. The *roff* system itself is intimately connected to the Unix operating system, but its roots go back to the earlier operating systems CTSS and Multics.

The Predecessor RUNOFF

roff's ancestor **RUNOFF** was written in the MAD language by *Jerry Saltzer* for the *Compatible Time Sharing System (CTSS)*, a project of the Massachusetts Institute of Technology (MIT), in 1963 and 1964 – note that CTSS commands were all uppercase.

In 1965, MIT's Project MAC teamed with Bell Telephone Laboratories (BTL) and General Electric to begin the *Multics* system A command called **runoff** was written for Multics in the late 60s in the BCPL language, by *Bob Morris*, *Doug McIlroy*, and other members of the Multics team.

Like its CTSS ancestor, Multics **runoff** formatted an input file consisting of text and command lines; commands began with a period and were two letters. Output from these commands was to terminal devices such as IBM Selectric terminals. Multics **runoff** had additional features added, such as the ability to do two-pass formatting; it became the main format for Multics documentation and text processing.

BCPL and **runoff** were ported to the GCOS system at Bell Labs when BTL left the development of Multics.

The Classical nroff/troff System

At BTL, there was a need to drive the *Graphic Systems CAT* typesetter, a graphical output device from a PDP-11 computer running Unix. As **runoff** was too limited for this task it was further developed into a more powerful text formatting system by *Joseph F. Ossanna*, who already programmed several runoff ports.

The name *runoff* was shortened to *roff*. The greatly enlarged language of Ossanna's version already included all elements of a full *roff system*. All modern *roff* systems try to implement compatibility to this system. So Joe Ossanna can be called the father of all *roff* systems.

This first *roff system* had three formatter programs.

troff	(*typesetter roff*) generated a graphical output for the *CAT* typesetter as its only device.
nroff	produced text output suitable for terminals and line printers.

roff was the reimplementation of the former **runoff** program with its limited features; this program was abandoned in later versions. Today, the name *roff* is used to refer to a *troff/nroff* sytem as a whole.

Ossanna's first version was written in the PDP-11 assembly language and released in 1973. *Brian Kernighan* joined the *roff* development by rewriting it in the C programming language. The C version was released in 1975.

The syntax of the formatting language of the **nroff/troff** programs was documented in the famous *Troff User's Manual [CSTR #54]*, first published in 1976, with further revisions up to 1992 by Brian Kernighan. This document is the specification of the *classical troff*. All later *roff* systems tried to establish compatibility with this specification.

After Ossanna's death in 1977, Kernighan went on with developing *troff*. In the late 1970s, Kernighan equipped *troff* with a general interface to support more devices, the intermediate output format, and the postprocessor system. This completed the structure of a *roff system* as it is still in use today; see section **USING ROFF**. In 1979, these novelties were described in the paper *[CSTR #97]*. This new *troff* version is the basis for all existing newer troff systems, including *groff*. On some systems, this *device independent troff* got a binary of its own, called **ditroff**(7). All modern **troff** programs already provide the full **ditroff** capabilities automatically.

Availability

The source code of both the ancient Unix and classical *troff* weren't available for two decades. Meanwhile, it is accessible again (on-line) for non-commercial use, cf. section **SEE ALSO**.

Free roff

The most important free *roff* project was the GNU implementation of *troff*, written from scratch by *James Clark* and put under the GNU Public License It was called *groff* (GNU *roff*). See **groff**(1) for an overview.

The *groff* system is still actively developed. It is compatible to the classical *troff*, but many extensions were added. It is the first *roff* system that is available on almost all operating systems – and it is free. This makes *groff* the de-facto *roff* standard today.

An alternative is *Gunnar Ritter*'s Heirloom Documentation Tools project, started in 2005, which provides enhanced versions of the various roff tools found in the OpenSolaris and Plan 9 operating systems, now available under free licenses.

USING ROFF

Most people won't even notice that they are actually using *roff*. When you read a system manual page (man page) *roff* is working in the background. *roff* documents can be viewed with a native viewer called **xditview**(1x), a standard program of the X window distribution, see **X**(7x). But using *roff* explicitly isn't difficult either.

Some *roff* implementations provide wrapper programs that make it easy to use the *roff* system on the shell command line. For example, the GNU *roff* implementation **groff**(1) provides command line options to avoid the long command pipes of classical *troff*; a program **grog**(1) tries to guess from the document which arguments should be used for a run of **groff**; people who do not like specifying command line options should try the **groffer**(1) program for graphically displaying *groff* files and man pages.

The roff Pipe

Each *roff* system consists of preprocessors, *roff* formatter programs, and a set of device postprocessors. This concept makes heavy use of the *piping* mechanism, that is, a series of programs is called one after the other, where the output of each program in the queue is taken as the input for the next program.

```
cat file | ... | preproc | ... | troff options | postproc
```

The preprocessors generate *roff* code that is fed into a *roff* formatter (e.g. **troff**), which in turn generates *intermediate output* that is fed into a device postprocessor program for printing or final output.

All of these parts use programming languages of their own; each language is totally unrelated to the other parts. Moreover, *roff* macro packages that were tailored for special purposes can be included.

Most *roff* documents use the macros of some package, intermixed with code for one or more preprocessors, spiced with some elements from the plain *roff* language. The full power of the *roff* formatting language is seldom needed by users; only programmers of macro packages need to know about the gory details.

Preprocessors

A *roff* preprocessor is any program that generates output that syntactically obeys the rules of the *roff* formatting language. Each preprocessor defines a language of its own that is translated into *roff* code when run through the preprocessor program. Parts written in these languages may be included within a *roff* document; they are identified by special *roff* requests or macros. Each document that is enhanced by preprocessor code must be run through all corresponding preprocessors before it is fed into the actual *roff* formatter program, for the formatter just ignores all alien code. The preprocessor programs extract and transform only the document parts that are determined for them.

There are a lot of free and commercial *roff* preprocessors. Some of them aren't available on each system, but there is a small set of preprocessors that are considered as an integral part of each *roff* system. The classical preprocessors are

tbl	for tables.
eqn	for mathematical formulæ.
pic	for drawing diagrams.
refer	for bibliographic references.
soelim	for including macro files from standard locations.
chem	for drawing chemical formulæ.

Other known preprocessors that are not available on all systems include

grap	for constructing graphical elements.
grn	for including **gremlin**(1) pictures.

Formatter Programs

A *roff formatter* is a program that parses documents written in the *roff* formatting language or uses some of the *roff* macro packages. It generates *intermediate output*, which is intended to be fed into a single device postprocessor that must be specified by a command-line option to the formatter program. The documents must have been run through all necessary preprocessors before.

The output produced by a *roff* formatter is represented in yet another language, the *intermediate output format* or *troff output*. This language was first specified in *[CSTR #97]*; its GNU extension is documented in **groff_out**(5). The intermediate output language is a kind of assembly language compared to the high-level *roff* language. The generated intermediate output is optimized for a special device, but the language is the same for every device.

The *roff* formatter is the heart of the *roff* system. The traditional *roff* had two formatters, **nroff** for text devices and **troff** for graphical devices.

Often, the name *troff* is used as a general term to refer to both formatters.

Devices and Postprocessors

Devices are hardware interfaces like printers, text or graphical terminals, etc., or software interfaces such as a conversion into a different text or graphical format.

A *roff* postprocessor is a program that transforms *troff* output into a form suitable for a special device. The *roff* postprocessors are like device drivers for the output target.

For each device there is a postprocessor program that fits the device optimally. The postprocessor parses the generated intermediate output and generates device-specific code that is sent directly to the device.

The names of the devices and the postprocessor programs are not fixed because they greatly depend on the software and hardware abilities of the actual computer. For example, the classical devices mentioned in *[CSTR #54]* have greatly changed since the classical times. The old hardware doesn't exist any longer and the old graphical conversions were quite imprecise when compared to their modern counterparts.

For example, the Postscript device *post* in classical *troff* had a resolution of 720 units per inch, while *groff*'s *ps* device has 72000, a refinement of factor 100.

Today the operating systems provide device drivers for most printer-like hardware, so it isn't necessary to write a special hardware postprocessor for each printer.

ROFF PROGRAMMING

Documents using *roff* are normal text files decorated by *roff* formatting elements. The *roff* formatting language is quite powerful; it is almost a full programming language and provides elements to enlarge the language. With these, it became possible to develop macro packages that are tailored for special applications. Such macro packages are much handier than plain *roff*. So most people will choose a macro package without worrying about the internals of the *roff* language.

Macro Packages

Macro packages are collections of macros that are suitable to format a special kind of documents in a convenient way. This greatly eases the usage of *roff*. The macro definitions of a package are kept in a file called *name***.tmac** (classically **tmac.***name*). All tmac files are stored in one or more directories at standardized positions. Details on the naming of macro packages and their placement is found in **groff_tmac**(5).

A macro package that is to be used in a document can be announced to the formatter by the command line option **−m**, see **troff**(1), or it can be specified within a document using the file inclusion requests of the *roff* language, see **groff**(7).

Famous classical macro packages are *man* for traditional man pages, *mdoc* for BSD-style manual pages; the macro sets for books, articles, and letters are *me* (probably from the first name of its creator *Eric* Allman), *ms* (from *Manuscript Macros*), and *mm* (from *Memorandum Macros*).

The roff Formatting Language

The classical *roff* formatting language is documented in the *Troff User's Manual [CSTR #54]*. The *roff* language is a full programming language providing requests, definition of macros, escape sequences, string variables, number or size registers, and flow controls.

Requests are the predefined basic formatting commands similar to the commands at the shell prompt. The user can define request-like elements using predefined *roff* elements. These are then called *macros*. A document writer will not note any difference in usage for requests or macros; both are written on a line on their own starting with a dot.

Escape sequences are *roff* elements starting with a backslash '\'. They can be inserted anywhere, also in the midst of text in a line. They are used to implement various features, including the insertion of non-ASCII characters with **\(**, font changes with **\f**, in-line comments with **\"**, the escaping of special control characters like \\, and many other features.

Strings are variables that can store a string. A string is stored by the **.ds** request. The stored string can be retrieved later by the ***** escape sequence.

Registers store numbers and sizes. A register can be set with the request **.nr** and its value can be retrieved by the escape sequence **\n**.

FILE NAME EXTENSIONS

Manual pages (man pages) take the section number as a file name extension, e.g., the filename for this document is *roff.7*, i.e., it is kept in section 7 of the man pages.

The classical macro packages take the package name as an extension, e.g. *file.***me** for a document using the *me* macro package, *file.***mm** for *mm*, *file.***ms** for *ms*, *file.***pic** for *pic* files, etc.

But there is no general naming scheme for *roff* documents, though *file.***tr** for *troff file* is seen now and then. Maybe there should be a standardization for the filename extensions of *roff* files.

File name extensions can be very handy in conjunction with the **less**(1) pager. It provides the possibility to feed all input into a command-line pipe that is specified in the shell environment variable **LESSOPEN**. This process is not well documented, so here an example:

```
LESSOPEN='|lesspipe s'
```

where **lesspipe** is either a system supplied command or a shell script of your own.

EDITING ROFF

The best program for editing a *roff* document is Emacs (or Xemacs), see **emacs**(1). It provides an *nroff* mode that is suitable for all kinds of *roff* dialects. This mode can be activated by the following methods.

When editing a file within Emacs the mode can be changed by typing '*M-x nroff-mode*', where **M-x** means to hold down the **Meta** key (or **Alt**) and hitting the **x** key at the same time.

But it is also possible to have the mode automatically selected when the file is loaded into the editor.

- The most general method is to include the following 3 comment lines at the end of the file.

  .\" Local Variables: .\" mode: nroff .\" End:

- There is a set of file name extensions, e.g. the man pages that trigger the automatic activation of the *nroff* mode.
- Theoretically, it is possible to write the sequence

  .\" -*- nroff -*-

  as the first line of a file to have it started in *nroff* mode when loaded. Unfortunately, some applications such as the **man** program are confused by this; so this is deprecated.

All *roff* formatters provide automated line breaks and horizontal and vertical spacing. In order to not disturb this, the following tips can be helpful.

- Never include empty or blank lines in a *roff* document. Instead, use the empty request (a line consisting of a dot only) or a line comment **.\"** if a structuring element is needed.
- Never start a line with whitespace because this can lead to unexpected behavior. Indented paragraphs can be constructed in a controlled way by *roff* requests.
- Start each sentence on a line of its own, for the spacing after a dot is handled differently depending on whether it terminates an abbreviation or a sentence. To distinguish both cases, do a line break after each sentence.
- To additionally use the auto-fill mode in Emacs, it is best to insert an empty *roff* request (a line consisting of a dot only) after each sentence.

The following example shows how optimal *roff* editing could look.

> This is an example for a .I roff document. . This is the next sentence in the same paragraph. . This is a longer sentence stretching over several lines; abbreviations like 'cf.' are easily identified because the dot is not followed by a line break. . In the output, this will still go to the same paragraph.

Besides Emacs, some other editors provide *nroff* style files too, e.g. **vim**(1), an extension of the **vi**(1) program.

SEE ALSO

There is a lot of documentation on *roff*. The original papers on classical *troff* are still available, and all aspects of *groff* are documented in great detail.

Internet sites

troff.org
: The historical troff site provides an overview and pointers to all historical aspects of *roff*.

Multics
: The Multics site contains a lot of information on the MIT projects, CTSS, Multics, early Unix, including *runoff*; especially useful are a glossary and the many links to ancient documents.

Unix Archive
: The Ancient Unixes Archive provides the source code and some binaries of the ancient Unixes (including the source code of *troff* and its documentation) that were made public by Caldera since 2001, e.g. of the famous Unix version 7 for PDP-11 at the Unix V7 site

Developers at AT.T Bell Labs
: Bell Labs Computing and Mathematical Sciences Research provides a search facility for tracking information on the early developers.

Plan 9
: The Plan 9 operating system by AT.T Bell Labs.

runoff
: Jerry Saltzer's home page stores some documents using the ancient RUNOFF formatting language.

CSTR Papers
: The Bell Labs CSTR site stores the original *troff* manuals (CSTR #54, #97, #114, #116, #122) and famous historical documents on programming.

GNU *roff*
: The groff web site provides the free *roff* implementation *groff*, the actual standard *roff*.

Historical roff Documentation

Many classical **troff** documents are still available on-line. The two main manuals of the *troff* language are

[CSTR #54]
: J. F. Ossanna, *Nroff/Troff User's Manual* Bell Labs, 1976; revised by Brian Kernighan, 1992.

[CSTR #97]
: Brian Kernighan, *A Typesetter-independent TROFF* Bell Labs, 1981, revised March 1982.

The "little language" *roff* papers are

[CSTR #114]
: Jon L. Bentley and Brian W. Kernighan, *GRAP – A Language for Typesetting Graphs* Bell Labs, August 1984.

[CSTR #116]
: Brian W. Kernighan, *PIC – A Graphics Language for Typesetting* Bell Labs, December 1984.

[CSTR #122]
: J. L. Bentley, L. W. Jelinski, and B. W. Kernighan, *CHEM – A Program for Typesetting Chemical Structure Diagrams, Computers and Chemistry* Bell Labs, April 1986.

Manual Pages

Due to its complex structure, a full *roff* system has many man pages, each describing a single aspect of *roff*. Unfortunately, there is no general naming scheme for the documentation among the different *roff* implementations.

In *groff*, the man page **groff**(1) contains a survey of all documentation available in *groff*.

On other systems, you are on your own, but **troff**(1) might be a good starting point.

AUTHORS

This document is part of *groff*, the `GNU` *roff* distribution. It was written by Bernd Warken it is maintained by Werner Lemberg