Name

groff_out − GNU *roff* intermediate output format

Description

The fundamental operation of the *troff*(1) formatter is the translation of the *groff*(7) input language into a series of instructions concerned primarily with placing glyphs or geometric objects at specific positions on a rectangular page. In the following discussion, the term *command* refers to this intermediate output language, never to the *groff*(7) language intended for use by document authors. Intermediate output commands comprise several categories: glyph output; font, color, and text size selection; motion of the printing position; page advancement; drawing of geometric primitives; and device control commands, a catch-all for other operations. The last includes directives to start and stop output, identify the intended output device, and embed URL hyperlinks in supported output formats.

Because the front-end command *groff*(1) is a wrapper that normally runs the *troff* formatter to generate intermediate output and an output driver ("postprocessor") to consume it, users normally do not encounter this language. The *groff* program's **−Z** option inhibits postprocessing such that this intermediate output is sent to the standard output stream as when *troff* is run manually.

groff's intermediate output facilitates the development of output drivers and other postprocessors by offering a common programming interface. It is an extension of the page description language developed by Brian Kernighan for AT&T device-independent *troff* circa 1980. Where a distinction is necessary, we will say "*troff* output" to describe the output of GNU *troff*, and "intermediate output" to denote the language accepted by the parser implemented in *groff*'s internal C++ library used by most of its output drivers.

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 stacking of 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.

Lines containing only spaces and/or a comment are treated as empty and ignored.

Argument units

Some commands accept integer arguments that represent measurements, but the scaling units of the formatter's language are never used. Most commands assume a scaling unit of "**u**" (basic units), and others use "**z**" (scaled points); These are defined by the parameters specified in the device's *DESC* file; see *groff_font*(5) and, for more on scaling units, *groff*(7) and *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual. Color-related commands use dimensionless integers.

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 subsection "Device Control Commands" below. 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*⟨line-break⟩

A comment. Ignore any characters from the **#** character up to the next newline. 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 necessary only when two integer arguments would clash or if the preceding argument ends with a string argument.

C *id*⟨white-space⟩

Typeset the glyph of the special character *id*. Trailing syntactical space is necessary to allow special character names of arbitrary length. The drawing position is not advanced.

c *c* Typeset the glyph of the ordinary character character *c*. The drawing position is not advanced.

f *n* Select the font mounted at position *n*. *n* cannot be negative.

H *n* Horizontally move the drawing position to *n* basic units from the left edge of the page. *n* cannot be negative.

h *n* Move the drawing position right *n* basic units. AT&T *troff* allowed negative *n;* GNU *troff* does not produce such values, but *groff*'s output driver library handles them.

m *scheme* [*component* . . .]
Select the stroke color using the *component*s in the color space *scheme*. Each *component* is an integer between 0 and 65536. The quantity of components and their meanings vary with each *scheme*. This command is a *groff* extension.

mc *cyan magenta yellow*
Use the CMY color scheme with components cyan, magenta, and yellow.

md Use the default color (no components; black in most cases).

mg *gray*
Use a grayscale color scheme with a component ranging between 0 (black) and 65536 (white).

mk *cyan magenta yellow black*
Use the CMYK color scheme with components cyan, magenta, yellow, and black.

mr *red green blue*
Use the RGB color scheme with components red, green, and blue.

N *n* Typeset the glyph with index *n* in the current font. *n* is normally a non-negative integer. The drawing position is not advanced. The **html** and **xhtml** devices use this command with negative *n* to produce unbreakable space; the absolute value of *n* is taken and interpreted in basic units.

n *b a* Indicate a break. No action is performed; the command is present to make the output more easily parsed. The integers *b* and *a* describe the vertical space amounts before and after the break, respectively. GNU *troff* issues this command but *groff*'s output driver library ignores it. See **v** and **V**.

p *n* Begin a new page, setting its number to *n*. Each page is independent, even from those using the same number. The vertical drawing position is set to 0. All positioning, writing, and drawing commands are interpreted in the context of a page, so a **p** command must precede them.

s *n* Set type size to *n* scaled points (unit **z** in GNU *troff*). AT&T *troff* used unscaled points (**p**) instead; see section "Compatibility" below.

t *xyz* . . .⟨white-space⟩
t *xyz* . . . *dummy-arg*⟨white-space⟩
Typeset word *xyz*; that is, set a sequence of ordinary glyphs named *x*, *y*, *z*, . . . , terminated by a space or newline; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). Each glyph is set at the current drawing position, and the position is then advanced horizontally by the glyph's width. A glyph's width is read from its metrics in the font description file, scaled to the current type size, and rounded to a multiple of the horizontal motion quantum. Use the **C** command to emplace glyphs of special characters. The **t** command is a *groff* extension and is output only for devices whose *DESC* file contains the **tcommand** directive; see *groff_font*(5).

u *n xyz* . . .
u *xyz* . . . *dummy-arg*⟨white-space⟩
Typeset word *xyz* with track kerning. As **t**, but after placing each glyph, the drawing position is further advanced horizontally by *n* basic units. The **u** command is a *groff* extension and is output only for devices whose *DESC* file contains the **tcommand** directive; see *groff_font*(5).

V *n* Vertically move the drawing position to *n* basic units from the top edge of the page. *n* cannot be negative.

v *n* Move the drawing position down *n* basic units. AT&T *troff* allowed negative *n;* GNU *troff* does not produce such values, but *groff*'s output driver library handles them.

w Indicate an inter-word space. No action is performed; the command is present to make the output more easily parsed. Only adjustable, breakable inter-word spaces are thus described; those resulting from **\~** or horizontal motion escape sequences are not. GNU *troff* issues this command but *groff*'s output driver library ignores it. See **h** and **H**.

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 subsection "Separation" above.

D~ $h_1\ v_1\ h_2\ v_2 \ldots h_n\ v_n$ ⟨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 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 analogous 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). 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 in between 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.

  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 current 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 ... 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; otherwise, the line thickness is made proportional to the type size, which is the default. For historical reasons, the horizontal position is changed by adding the argument to the current 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 subsection "Separation" above.

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); see *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" below.

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 currently ignored.

xT *xxx* ⟨line-break⟩
(*Typesetter* control command)
Set the name of the output driver to *xxx*, a sequence of non-whitespace characters terminated by whitespace. The possible names correspond to those of *groff*'s **−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*, it is used only for output to the **X75**, **X75−12**, **X100**, and **X100−12** devices. For others, the commands **t** and **u** provide greater functionality and superior troubleshooting capacity.

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" below.

Example

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, or *gropdf* (1) to output directly to PDF.

- 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

*/usr/local/share/groff/1.23.0/font/dev*name*/DESC*
: describes the output device *name*.

Authors

James Clark wrote an early version of this document, which described only the differences between AT&T device-independent *troff*'s output format and that of GNU *roff*. The present version was completely rewritten in 2001 by Bernd Warken ⟨groff−bernd.warken−72@web.de⟩.

See also

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

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

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

groff(1)
: documents the **−Z** option and contains pointers to further *groff* documentation.

groff(7)
: describes the *groff* language, including its escape sequences and system of units.

groff_font(5)
: details the device scaling parameters of device *DESC* files.

troff(1) generates the device-independent intermediate output documented here.

roff(7) presents historical aspects and the general structure of *roff* systems.

groff_diff(7)
: enumerates differences between the intermediate output produced by AT&T *troff* and that of *groff*.

gxditview(1)
: is a viewer for intermediate output.

Roff.js ⟨https://github.com/Alhadis/Roff.js/⟩ is a viewer for intermediate output written in JavaScript.

grodvi(1), *grohtml*(1), *grolbp*(1), *grolj4*(1), *gropdf*(1), *grops*(1), and *grotty*(1) are *groff* postprocessors.