NAME

groff_out − groff intermediate output format

DESCRIPTION

This manual page describes the intermediate output format of the GNU **roff**(7) text processing system. This output is produced by a run of the GNU **troff**(1) program before it is 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. The pre-groff roff versions are denoted as *classical troff* .

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 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 splitted 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. The names of characters and fonts can be of arbitrary length. A character 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**. But 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 special groff character named *xxx*. The trailing syntactical space or line break is necessary to allow character names of arbitrary length. The character is printed at the current print position; the character's size is read from the font file. The print position is not changed.

c *c*
: Print character *c* at the current print position; the character'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. *[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 character with index *n* (a non-negative integer) of the current font. The print position is not changed. 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 *xxx*⟨white_space⟩
t *xxx dummy_arg*⟨white_space⟩
: Print a word, i.e. a sequence of characters *xxx* 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 character should be printed at the current position, the current horizontal position should then be increased by the width of the first character, and so on for each character.

The widths of the characters are read from the font file, scaled for the current point size, and rounded to a multiple of the horizontal resolution. Special characters cannot be printed using this command (use the **C** command for named characters). 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 xxx*⟨white_space⟩
Print word with track kerning. This is the 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* (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). *[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 arguments called h_1, h_2, ..., h_n stand for horizontal distances where positive means right, negative left. The arguments called v_1, v_2, ..., v_n 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 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.

  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 ... h_n v_n ⟨line_break⟩
: Draw a polygon line from current position to offset (*h1*, *v1*), from there to offset (*h2*, *v2*), etc. up to offset (*hn*, *vn*), 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 ... 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** resp. 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* (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, the writing of a single character was mostly done by a very strange command that combined a horizontal move and the printing of a character. 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 character *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 characters 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> **echo 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> **echo 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 *[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/share/groff/1.18.1.4/font/dev*name*/**DESC**
Device description file for device *name*.

⟨*groff_source_dir*⟩/**src/libs/libdriver/input.cc**
Defines the parser and postprocessor for the intermediate output. It is located relative to the top directory of the *groff* source tree, e.g. *@GROFFSRCDIR@*. 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.

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 ⟨`http://cm.bell-labs.com/cm/cs/cstr.html`⟩.

[CSTR #97]
A Typesetter-independent TROFF by *Brian Kernighan* is the original and most concise documentation on the output language; see CSTR #97 ⟨`http://cm.bell-labs.com/cm/cs/cstr/97.ps.gz`⟩.

[CSTR #54]
The 1992 revision of the *Nroff/Troff User's Manual* by *J. F. Osanna* and *Brian Kernighan* isn't as concise as *[CSTR #97]* regarding the output language; see CSTR #54 ⟨`http://cm.bell-labs.com/cm/cs/cstr/54.ps.gz`⟩.

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 has been rewritten 2002 by Bernd Warken ⟨`bwarken@mayn.de`⟩ and is maintained by Werner Lemberg ⟨`wl@gnu.org`⟩.