Neatroff

Ali Gholami Rudi

Updated in November 2025

Neatroff is a new implementation of the Troff typesetting system in the C programming language, which tries to address, as neatly as possible, some of the shortcomings of the original Troff based on the ideas and features available in Plan 9 Troff, Heirloom Troff, and Groff. The latest versions of Neatroff, its PostScript and PDF post-processor, Neatpost, and its eqn preprocessor, Neateqn, are available at their home page (link). This document enumerates Neatroff's features, its new requests, and its differences compared to other Troff implementations. On the other hand, the document "Getting Started with Neatroff" (link) explains how to set up and use Neatroff.

Nomenclature

Since there are several implementations of this family of roff typesetting suites available, the Neatroff documentation needs to distinguish this implementation from others to avoid confusion. "roff" for example is the main typesetting application, which may also exist under this name in other typesetting suites of the same kind. To avoid misinterpretation, the name "Neatroff" will be used in these documents when addressing either the whole typesetting suite as such or the roff binary of this typeseeting suit in the sense of a brand name. If these documents address the commandline usage of "roff" or more general handling the binary itself, it uses the name "roff" to name the binary being the result of the compilation process. Same goes for "neatpost," "neateqn," "neatmkfn," and "neatrefer" accordingly.

Noteworthy Features

The following list describes the main extensions of Neatroff compared to the original Troff (many of these extensions are available in Groff and Heirloom Troff as well). The number register .neat, which is always one in Neatroff, can be used to distinguish Neatroff from other Troff implementations.

UTF-8 encoding

In Neatroff, input files and characters, glyph names, ligatures, hyphenation patterns and dictionary, as well as quoted escape sequence delimiters and arguments of commands like .tr, .hc, .tc, .lc, .mc, and .fc are in UTF-8 encoding.

Long macro, register, and environment names

When not in compatibility mode (activated with the -C command line option or the .cp request, as in Groff), Neatroff supports long macro, register, and environment names. It also supports Groff-style escape sequences with long arguments (for \[], *[], \$[], \f[], \g[], \k[], \m[], \n[], and \s[]) and interpolating string registers with arguments (*[xyz arg1 arg2 ...]). Note that like Groff, Neatroff supports named environments and is not limited to original Troff's three fixed environments.

Advanced font features, ligature, and pairwise kerning

Neatroff and Neatmkfn (which generates Neatroff's font descriptions) support many of the advanced font features available in OpenType fonts. In a font, a set of substitution and positioning rules may be specified, which are grouped into several features and scripts. In Neatroff, features can be enabled with .ff and the active script and language can be selected with .ffsc. Neatmkfn supports PostScript Type 1 fonts, TrueType fonts (TTF), and OpenType fonts (OTF). For the latter, however, it cannot extract glyph bounding boxes, which are used by the Neateqn preprocessor. Therefore, if an OpenType font is supposed to be used in Neateqn blocks, it should be converted to TrueType first.

Whole paragraph text formatting

Neatroff supports filling whole paragraphs at once, to achieve more uniform word spacing. Like Heirloom Troff, the .ad request accepts arguments p or pb, pl, and pr, which are equivalent to b, l, and r, except that the filling is done for whole paragraphs, i.e., words are collected until a line break is issued. This inevitably changes the behaviour of some requests and traps: several lines may be collected and ready to be output while executing them. For the end macro, Troff invokes the macro specified with .em request without flushing the last incomplete line. Neatroff follows the same behaviour even when formatting whole paragraphs and does not write any of the collected lines to the output. Since after the end macro no new page is started, collected lines may be unexpectedly written to the end of the last page of the document. To change that, the end macro can invoke the .br request. For requests that cause a break, using ' as the control character prevents writing any line of the collected paragraph to the output, as expected. The exception to this rule is 'br, which formats the words collected so far and outputs all resulting lines except the final incomplete line (this is useful, for instance, for footnotes, which should be inserted on the same page).

Paragraph formatting algorithm

For deciding at what points to break a paragraph into lines, Neatroff assigns a cost to each possible outcome: a cost of 100 is assigned to each stretchable space that has to be stretched 100 percent. The cost grows quadratically and the cost of stretching a space 200 percent is 400. There are requests that adjust the algorithm Neatroff uses for performing paragraph formatting. The .hycost request changes the cost of hyphenating words. The default value is zero. The \j escape sequence, as in Heirloom Troff, specifies the extra cost of a line break after a word; for instance, in "Hello\j'10000' world", the words are not split by the line breaking algorithm, unless absolutely necessary (i.e., if other options are more costly). The escape sequence \~ introduces non-breakable stretchable space. Also, to prevent paragraphs with very short last lines, the .pmll (paragraph minimum line length) sets the minimum length of a formatted line, specified as a percentage of \n(.l; ".pmll 15", for instance, ensures that the length of last line of each paragraph is at least 15% of its other lines; otherwise, a cost proportional to the value specified as its second argument is added.

Controlling word spaces

The .ssh request sets the amount (in percentage) by which the stretchable spaces in a line may be shrunk while formatting lines. The default value is zero. Also, the second argument of the .ss request specifies sentence space, as in Groff or Heirloom Troff.

Macros and their arguments

In a macro, \$* is replaced with the macro's arguments separated by spaces. \$@ is like \$*, but quotes the arguments as well. \$^ is like \$@, except that it escapes the double quotes in the arguments. The arguments can be shifted with a .shift request. Neatroff also supports blank line macro (.blm) and leading space macro (.lsm).

Text direction

Neatroff supports changing the text direction to render right-to-left languages. The .<< and .>> requests specify text direction and the \< and \> escape sequences change it temporarily for including words in the reverse direction. The value of number registers .td and .cd indicate the current text and temporary directions respectively; zero means left-to-right and one means right-to-left. Neatroff starts processing text direction, after the first invocation of .<< or .>>.

Keshideh justification and cursive scripts

A new adjustment type (.ad k) allows inserting Keshideh characters before justifying text with hyphenation and spaces. Neatroff also supports cursive scripts, which require connecting glyphs at their cursive attachment positions, as defined in the fonts.

Font manipulation

In Neatroff, the mapping between Troff character names and glyphs in a font can be modified with the .fmap request: ".fmap F C G" maps Troff character C to the glyph with device-specific name G for font F. When this glyph does not exist in F, Neatroff assumes that the character C is not present in the font. When G is missing, the effect of .fmap for character C is cancelled. Neatroff also implements Groff's .fspecial and .fzoom requests: after ".fspecial FN S1 S2

...],"when the current font is FN, the fonts S1, S2, ... are assumed to be special. Also, ".fzoom FN zoom" scales font FN by the second argument after dividing it by 1000.

Colour support

Neatroff supports colours with the .cl request and the \m[] escape sequence. Unlike Groff, colours need not be defined beforehand and can be specified directly. The argument of \m can be predefined colour names (e.g. blue), predefined colour numbers (0 for black, 1 for red, 2 for green, 3 for yellow, 4 for blue, 5 for magenta, 6 for cyan, and 7 for white), #rgb and #rrggbb for specifying colours in hexadecimal RGB format, #g and #gg for specifying grey with the given hexadecimal level, and empty (\m[]) for the previous colour. The current colour is available in the .m number register.

Hyphenation

The .hpf request loads hyphenation patterns, exceptions, and character mappings from the addresses specified via its arguments. The specified files should contain nothing but utf-8 patterns, exceptions and mappings respectively (i.e. no TeX code), just like the files whose names end with .pat.txt, .hyp.txt and .chr.txt in CTAN for TeX (link). The .hpfa request is like .hpf, except that it does not clear the previous hyphenation patterns and exceptions. The second and third arguments of these requests are optional. With no arguments, these requests load English hyphenation patterns and exceptions. Also the ".hcode abcd..." request, assigns the hyphenation code of b to a and the hyphenation code of d to c; initially all upper-case ASCII letters are mapped to their lower-case forms.

Filled drawing objects

Neatroff supports Groff-style polygons and filled drawing objects (p, C, E and P commands for \D escape sequence). In Neatroff, however, there is no specific background colour; objects are filled with the current colour (.m number register). Furthermore, in Neatroff the edges of polygons can be lines, arcs, or splines; a letter among the arguments of \D'p ..' specifies the type of the subsequent edges: 'l', 'a', and '~' for lines, arcs, and splines respectively.

Conditional escape sequence

Neatroff supports a new escape sequence for conditional interpolation: the escape sequence \?'cond@expr1@expr2@', evaluates cond (exactly as if it were a .if condition) and interpolates expr1, if the condition is true, and expr2, otherwise. The delimiter (@ in this example) can be any character, but may not be part of the condition; for numerical expressions, for instance, it cannot be a digit, an operator sign, or a scale indicator, unless separated from the condition with \&. The final delimiter, and even expr2, may be omitted, thus \?'cond@expr' is valid; Neatroff interpolates expr if cond is true.

Neatpost-specific device functions

Neatpost can produce both PostScript and PDF. The escape sequences \X'eps img.eps [width [height]]' and \X'pdf img.pdf [width [height]]' in Neatroff instruct Neatpost to include the given EPS or PDF file; the former works only when the output is PostScript and the latter when the output is PDF. They include the given EPS/PDF file with its lower left corner at the current point. If the width or height are given (in basic units), the image is scaled appropriately. Neatroff also supports \X'rotate deg' for rotating the current page around the current point. For creating a document outline (bookmark tree), Neatpost supports \X'mark desc page offset level', and for creating named references, Neatpost supports \X'name label page offset'. Also, \X'link destination width height' can be used to create links; its first argument for internal references (defined with \X'name ...') should start with a # sign.

Helper Macro Packages

In addition to the standard Troff macro packages, such as -ms, -mm, and -me, which are imported from Plan 9 Troff, Neatroff comes with a few convenient helper macro packages as follows (these macros are included in neatroff_make): for drawing simple tables without the tbl preprocessor -mtbl, for invoking Neatpost device functions like including EPS and PDF images -mpost, for drawing simple charts and graphs -mgr, for floating objects -mkeep. Some Groff-specific macros are implemented in -mgnu, such as open, opena, close, write, pso, and mso. Also, -men and -mfa include -ms-like macros for creating short English and Farsi documents.

Summary of New Requests

This is the list of new requests available in Neatroff compared to those documented in “Troff User’s Manual” by Ossanna and Kernighan.

`.ad p*`	b	E

With values pl, pr, pb, and p, this request instructs Neatroff to perform whole-paragraph line formatting. Also, the value k enables Keshideh justification (kp is the equivalent for whole-paragraph formatting).

`.blm M`	–	–

Specify the blank line macro. If specified, each blank line is treated as an invocation of this macro.

`.chop R`	–	–

Remove the last character of a string register.

`.cl C`	0	E

Change text colour. The current colour is available in the number register \n(.m. With no arguments, the previous colour is selected. The format of the argument and the \m escape sequence are described in the previous section.

`.co SRC DST`	–	–

Copy the contents of register SRC into register DST.

`.co+ SRC DST`	–	–

Append the contents of register SRC to register DST.

`.co> R F`	–	–

Copy the contents of register R into file F.

`.co< R F`	–	–

Read the contents of register R from file F.

`.char C DEF` – –

Define Troff character C. If DEF is missing, previous definitions for character C are removed.

`.ochar FN C DEF` – –

Define Troff character C only for font FN. If DEF is missing, previous definitions for character C are removed.

`.rchar C` – –

Remove the definition of character C.

`.evc E` – E

Copy text-formatting settings from environment E to the currently active environment. Pending input lines are not copied.

`.eos S T` S=.?! T='")]* –

Specify sentence characters. The first argument specifies the characters that end a sentence and the second argument specifies the characters ignored after them.

`.fzoom F N` 1000 –

Magnify the given font by N/1000.

`.fp N F L` – –

In Neatroff, if instead of the position of the font to be mounted, N is a dash, the position of the font is decided automatically: if a font with the same name is already mounted, the same position is reused. Otherwise the font is mounted on the next available position.

`.ff F +F1 -F2` – –

Enable or disable font features; the first argument specifies the font and the rest of the arguments specify feature names, prefixed with a plus to enable or a minus to disable. When a feature is enabled, all substitution and positioning rules of a font with that feature name are applied when laying out the glyphs.

`.ffsc F SC LN` – –

Specify font's script and language. A Neatroff font description specifies a set of rules for each script and language, grouped into several features. With this request, only the rules for the specified script and language are enabled. By default, or when SC is missing, all scripts are selected. When LN is missing, the rules of the default language of the selected script are enabled.

`.fspecial F S1 S2` – –

Set special fonts when the current font is F.

`.fmap FN CH GID` – –

Map Troff character CH to glyph with device dependent name GID for font FN. When gid is missing, the effect of mapping CH is cancelled. Neatroff implicitly calls .fmap for all aliases in font descriptions (character definitions whose second column is ").

`.hycost N N2 N3` 0 E

Change the cost of hyphenating words when adjusting lines. An argument of 100 assigns to each hyphenation the cost of stretching a space one hundred percent while formatting. The second and third arguments specify additional costs for two and three consecutive hyphenated lines (only when formatting whole paragraphs).

`.hlm n` 0 E

Set the maximum number of consecutive hyphenated lines (only when formatting whole paragraphs). The current value is available via \n[.hlm]. An argument of zero or a negative number implies no limit.

`.hydash C` \:\(hy\(en\(em-\-\(-- –

Specify the list of characters after which words may be broken (even when hyphenation is disabled) without inserting hyphens.

`.hystop C` \% –

Specify hyphenation inhibiting characters. Words containing any of the given characters are not hyphenated, unless after dashes (characters specified via

.hydash) or hyphenation indicators (\%).

.hpf P H C – –

Set hyphenation files for patterns, exceptions, and mappings. With no arguments, loads English hyphenation patterns and exceptions.

.hpfa P H C – –

Like, .hpf, but do not clear the previous hyphenation patterns.

.hcode abcd... – –

Assign the hyphenation code of b to a and the hyphenation code of d to c.

.in2 0 E

Right-side indentation. The current right-side indentation is available in register \n(.I.

.ti2 0 E

Right-side temporary indentation.

.kn N 1 E

Enable or disable pairwise kerning (the current value is available through \n[.kn]).

.lsm M – –

Specify the leading space macro. If specified, for each line with leading spaces, this macro is invoked. The register \n[lsn] holds the number of leading spaces removed from the line.

.pmll N C 0 E

Set paragraph minimum line length in percentage. To shorter lines, Neatroff assigns a cost proportional to the value specified as the second argument (or 100, if missing) when formatting paragraphs. Number registers \n[.pmll] and \n[.pmllcost] store the values passed to .pmll.

`.>>   .<<`	left-to-right	E

Render text in left-to-right or right-to-left direction. See the first section for an explanation of the escape sequences \> and \<.

`.shift N`	–	–

Shift macro arguments by N positions.

`.ssh N`	0	E

Set the amount stretchable spaces in formatted lines may be shrunk in percentage (available through \n[.ssh]).

`.ss M N`	M=12 N=12	E

The second argument sets sentence space size (available in \n[.sss]).

`.tkf FN S1 N1 S2 N2`	–	–

Enable track kerning for font FN. If the point size is at most S1, the width of each character is increased by N1 points, if it is at least S2, the width of each character is increased by N2 points, and if it is between S1 and S2, the width of each character is increased by a value between N1 and N2, relative to the difference between the current point size and S1.

Notes

The standard macro packages

The standard Troff macro packages and a top-level build script to obtain and install Neatroff are available in the neatroff_make git repository (link). "Getting Started with Neatroff" (link) explains how to use this repository.

Formatting equations with Neateqn

Neateqn is an eqn preprocessor for Neatroff. It implements many of the extensions introduced in Groff's eqn preprocessor. It can use TeX's Computer Modern-style bracket-building symbols, if available. "Typesetting Mathematics with Neateqn" (link) introduces Neateqn.

Generating the output device

The Neatmkfn program (link) generates Neatroff font descriptions for AFM, TrueType, and OpenType fonts. It includes a script to create a complete output device for Neatroff.

Missing requests

A few requests of the original Troff are not implemented: .pi, .cf, .rd, .pm, .ul, .cu, .uf, \H, and \S.

Porting and distribution

Given that Neatroff, Neatpost, Neatmkfn, and Neateqn can be compiled with Neatcc, porting them to other Unix variants besides Linux should not be difficult. Note that Neatroff is released under the ISC licence.

List of OpenType font features

As mentioned in previous sections, font features can be enabled and disabled with the .ff request. For a list of OpenType features in general and their descriptions, see the list of typographic features in Wikipedia (link) or OpenType specification (link).

Font Description Files

The format of font description files in Neatroff, although still mostly backward compatible, has been slightly changed. The value of special, spacewidth, and ligatures parameters retain their old meanings; sizes and name parameters are ignored, however. The value of the fontname parameter in Neatroff specifies the device name of the font (e.g. Times-Roman); Neatpost uses it to map Troff fonts to PostScript fonts. In the charset section, the forth field is always the device-specific name of the glyph (accessible with the \N escape sequence) and the optional fifth field specifies the glyph's code (the fourth field of the original Troff).

In addition to the old charset section of the original Troff, Neatroff supports a new syntax for defining characters and kerning pairs. Lines starting with the word "char" define characters (similar to lines in the charset section) and lines starting with "kern" specify kerning pairs. For the latter, "kern" is followed by three tokens: the name of the first glyph, the name of the second glyph, and the amount of kerning between them. Specifying the name of glyphs (the fourth field after "char") instead of character names allows specifying kerning pairs for glyphs not mapped to any characters (may be done later with a .fmap request) or specifying kerning pairs only once for all aliases of a character. Here are a few lines of a font description file for Neatroff, created with Neatmkfn.

```
name R
fontname Times-Roman
spacewidth 25
ligatures fi fl 0
# the list of characters
char    !       33      2       exclam  33
char    .       25      0       period  46
char    A       72      2       A       65
char    B       67      2       B       66
char    C       67      2       C       67
# the kerning pairs
kern    A       C       -5
kern    A       period  -1
```

The width column of the character definition lines can optionally include four more numbers, separated with commas, that describe the bounding boxes of the glyphs. The bounding boxes are used in the \w escape sequence; after this escape sequence, the value of the bbllx, bblly, bburx and bbury number registers are modified to represent the bounding box of the argument of \w.

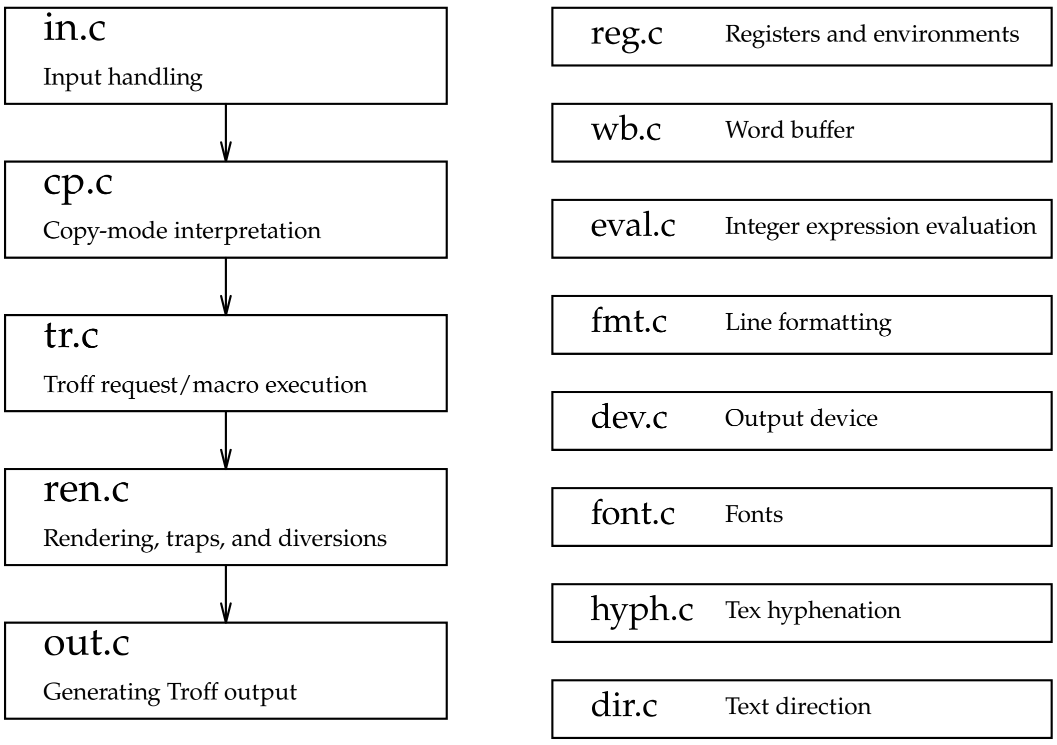
To use the advanced features present in TrueType and OpenType fonts, Neatroff supports lines that define substitution and positioning rules (lines starting with “gsub” and “gpos” respectively). Note that unlike Heirloom Troff, which implements non-contextual single-character substitutions, Neatroff implements many of the more complex OpenType substitution and positioning features. The following example shows how such features are defined in Neatroff font descriptions:

```
gsub liga:latn 4 -gl1 -gl2 -gl3 +gl123
gpos kern:latn 2 gl1:+0+0-5+0 gl2
```

In this example, the first line defines a 3-character ligature (with feature name “liga” and script name “latn”) and the second defines pairwise kerning for the pair of glyphs gl1 and gl2 (decreasing the horizontal advance of gl1 by 5 basic units; with feature name “kern” and script name “latn”). The patterns can be longer and more detailed, defining context or glyph groups, to support OpenType features that require them; for examples, see the files generated by Neatmkfn.

Source Code Organization

The following figure shows where Neatroff's major layers and features are implemented in its source tree.



in.c
Input handling
cp.c
Copy-mode interpretation
tr.c
Troff request/macro execution
ren.c
Rendering, traps, and diversions
out.c
Generating Troff output
reg.c
Registers and environments
wb.c
Word buffer
eval.c
Integer expression evaluation
fmt.c
Line formatting
dev.c
Output device
font.c
Fonts
hyph.c
Tex hyphenation
dir.c
Text direction