Name

roff – concepts and history of *roff* typesetting

Description

The term *roff* denotes a family of document formatting systems known by names like *troff*, *nroff*, and *ditroff*. A *roff* system consists of an interpreter for an extensible text formatting language and a set of programs for preparing output for various devices and file formats. Unix-like operating systems often distribute a *roff* system. The manual pages on Unix systems ("man pages") and bestselling books on software engineering, including Brian Kernighan and Dennis Ritchie's *The C Programming Language* and W. Richard Stevens's *Advanced Programming in the Unix Environment* have been written using *roff* systems. GNU *roff*—*groff*—is arguably the most widespread *roff* implementation.

Below we present typographical concepts that form the background of all *roff* implementations, narrate the development history of some *roff* systems, detail the command pipeline managed by *groff*(1), survey the formatting language, suggest tips for editing *roff* input, and recommend further reading materials.

Concepts

roff input files contain text interspersed with instructions to control the formatter. Even in the absence of such instructions, a *roff* formatter still processes its input in several ways, by filling, hyphenating, breaking, and adjusting it, and supplementing it with inter-sentence space. These processes are basic to typesetting, and can be controlled at the input document's discretion.

When a device-independent *roff* formatter starts up, it obtains information about the device for which it is preparing output from the latter's description file (see *groff_font*(5)). An essential property is the length of the output line, such as "6.5 inches".

The formatter interprets plain text files employing the Unix line-ending convention. It reads input a character at a time, collecting words as it goes, and fits as many words together on an output line as it can—this is known as *filling.* To a *roff* system, a *word* is any sequence of one or more characters that aren't spaces or newlines. The exceptions separate words.

A *roff* formatter attempts to detect boundaries between sentences, and supplies additional inter-sentence space between them. It flags certain characters (normally "**!**", "**?**", and "**.**") as potentially ending a sentence. When the formatter encounters one of these *end-of-sentence characters* at the end of an input line, or one of them is followed by two (unescaped) spaces on the same input line, it appends an inter-word space followed by an inter-sentence space in the output. The dummy character escape sequence **\&** can be used after an end-of-sentence character to defeat end-of-sentence detection on a per-instance basis. Normally, the occurrence of a visible non-end-of-sentence character (as opposed to a space or tab) immediately after an end-of-sentence character cancels detection of the end of a sentence. However, several characters are treated *transparently* after the occurrence of an end-of-sentence character. That is, a *roff* does not cancel end-of-sentence detection when it processes them. This is because such characters are often used as footnote markers or to close quotations and parentheticals. The default set is **"**, **'**, **)**, **]**, *****, **\[dg]**, **\[dd]**, **\[rq]**, and **\[cq]**. The last four are examples of *special characters,* escape sequences whose purpose is to obtain glyphs that are not easily typed at the keyboard, or which have special meaning to the formatter (like ****).

When an output line is nearly full, it is uncommon for the next word collected from the input to exactly fill it—typically, there is room left over only for part of the next word. The process of splitting a word so that it appears partially on one line (with a hyphen to indicate to the reader that the word has been broken) with its remainder on the next is *hyphenation.* Hyphenation points can be manually specified; *groff* also uses a hyphenation algorithm and language-specific pattern files to decide which words can be hyphenated and where. Hyphenation does not always occur even when the hyphenation rules for a word allow it; it can be disabled, and when not disabled there are several parameters that can prevent it in certain circumstances.

Once an output line is full, the next word (or remainder of a hyphenated one) is placed on a different output line; this is called a *break.* In this document and in *roff* discussions generally, a "break" if not further qualified always refers to the termination of an output line. When the formatter is filling text, it introduces breaks automatically to keep output lines from exceeding the configured line length. After an automatic break, a *roff* formatter *adjusts* the line if applicable (see below), and then resumes collecting and filling text on the next output line.

Sometimes, a line cannot be broken automatically. This usually does not happen with natural language text unless the output line length has been manipulated to be extremely short, but it can with specialized text like program source code. *groff* provides a means of telling the formatter where the line may be broken without hyphens. This is done with the non-printing break point escape sequence **\:**.

There are several ways to cause a break at a predictable location. A blank input line not only causes a break, but by default it also outputs a one-line vertical space (effectively a blank output line). Macro packages may discourage or disable this "blank line method" of paragraphing in favor of their own macros. A line that begins with one or more spaces causes a break. The spaces are output at the beginning of the next line without being *adjusted* (see below). Again, macro packages may provide other methods of producing indented paragraphs. Trailing spaces on *text lines* (see below) are discarded. The end of input causes a break.

After the formatter performs an automatic break, it may then *adjust* the line, widening inter-word spaces until the text reaches the right margin. Extra spaces between words are preserved. Leading and trailing spaces are handled as noted above. Text can be aligned to the left or right margin only, or centered, using *requests.*

A *roff* formatter translates horizontal tab characters, also called simply "tabs", in the input into movements to the next tab stop. These tab stops are by default located every half inch measured from the current position on the input line. With them, simple tables can be made. However, this method can be deceptive, as the appearance (and width) of the text in an editor and the results from the formatter can vary greatly, particularly when proportional typefaces are used. A tab character does not cause a break and therefore does not interrupt filling. The formatter provides facilities for sophisticated table composition; there are many details to track when using the "tab" and "field" low-level features, so most users turn to the *tbl*(1) preprocessor to lay out tables.

Requests and macros

A *request* is an instruction to the formatter that occurs after a *control character,* which is recognized at the beginning of an input line. The regular control character is a dot "**.**". Its counterpart, the *no-break control character,* a neutral apostrophe "**'**", suppresses the break implied by some requests. These characters were chosen because it is uncommon for lines of text in natural languages to begin with them. If you require a formatted period or apostrophe (closing single quotation mark) where the formatter is expecting a control character, prefix the dot or neutral apostrophe with the dummy character escape sequence, "**\&**".

An input line beginning with a control character is called a *control line.* Every line of input that is not a control line is a *text line.*

Requests often take *arguments,* words (separated from the request name and each other by spaces) that specify details of the action the formatter is expected to perform. If a request is meaningless without arguments, it is typically ignored. Of key importance are the requests that define macros. Macros are invoked like requests, enabling the request repertoire to be extended or overridden.

A *macro* can be thought of as an abbreviation you can define for a collection of control and text lines. When the macro is *called* by giving its name after a control character, it is replaced with what it stands for. The process of textual replacement is known as *interpolation.* Interpolations are handled as soon as they are recognized, and once performed, a *roff* formatter scans the replacement for further requests, macro calls, and escape sequences.

In *roff* systems, the "**de**" request defines a macro.

Page geometry

roff systems format text under certain assumptions about the size of the output medium, or page. For the formatter to correctly break a line it is filling, it must know the line length, which it derives from the page width. For it to decide whether to write an output line to the current page or wait until the next one, it must know the page length. A device's *resolution* converts practical units like inches or centimeters to *basic units,* a convenient length measure for the output device or file format. The formatter and output driver use basic units to reckon page measurements. The device description file defines its resolution and page dimensions (see *groff_font*(5)).

A *page* is a two-dimensional structure upon which a *roff* system imposes a rectangular coordinate system with its upper left corner as the origin. Coordinate values are in basic units and increase down and to the right. Useful ones are therefore always positive and within numeric ranges corresponding to the page boundaries.

While the formatter (and, later, output driver) is processing a page, it keeps track of its *drawing position,* which is the location at which the next glyph will be written, from which the next motion will be measured, or where a geometric object will commence rendering. Notionally, glyphs are drawn from the text baseline upward and to the right. (*groff* does not yet support right-to-left scripts.) The *text baseline* is a (usually invisible) line upon which the glyphs of a typeface are aligned. A glyph therefore "starts" at its bottom-left corner. If drawn at the origin, a typical letter glyph would lie partially or wholly off the page, depending on whether, like "g", it features a descender below the baseline.

Such a situation is nearly always undesirable. It is furthermore conventional not to write or draw at the extreme edges of the page. Therefore the initial drawing position of a *roff* formatter is not at the origin, but below and to the right of it. This rightward shift from the left edge is known as the *page offset.* (*groff*'s terminal output devices have page offsets of zero.) The downward shift leaves room for a text output line.

Text is arranged on a one-dimensional lattice of text baselines from the top to the bottom of the page. *Vertical spacing* is the distance between adjacent text baselines. Typographic tradition sets this quantity to 120% of the type size. The initial vertical drawing position is one unit of vertical spacing below the page top. Typographers term this unit a *vee.*

Vertical spacing has an impact on page-breaking decisions. Generally, when a break occurs, the formatter moves the drawing position to the next text baseline automatically. If the formatter were already writing to the last line that would fit on the page, advancing by one vee would place the next text baseline off the page. Rather than let that happen, *roff* formatters instruct the output driver to eject the page, start a new one, and again set the drawing position to one vee below the page top; this is a *page break.*

When the last line of input text corresponds to the last output line that fits on the page, the break caused by the end of input will also break the page, producing a useless blank one. Macro packages keep users from having to confront this difficulty by setting "traps"; moreover, all but the simplest page layouts tend to have headers and footers, or at least bear vertical margins larger than one vee.

Other language elements

Escape sequences start with the *escape character,* a backslash \, and are followed by at least one additional character. They can appear anywhere in the input.

With requests, the escape and control characters can be changed; further, escape sequence recognition can be turned off and back on.

Strings store character sequences. In *groff*, they can be parameterized as macros can.

Registers store numerical values, including measurements. The latter are generally in basic units; *scaling units* can be appended to numeric expressions to clarify their meaning when stored or interpolated. Some read-only predefined registers interpolate text.

Fonts are identified either by a name or by a mounting position (a non-negative number). Four styles are available on all devices. **R** is "roman": normal, upright text. **B** is **bold**, an upright typeface with a heavier weight. **I** is *italic*, a face that is oblique on typesetter output devices and usually underlined instead on terminal devices. **BI** is ***bold-italic***, man(7) evil here combining both of the foregoing style variations. Typesetting devices group these four styles into *families* of text fonts; they also typically offer one or more *special* fonts that provide unstyled glyphs; see *groff_char*(7).

groff supports named *colors* for glyph rendering and drawing of geometric objects. Stroke and fill colors are distinct; the stroke color is used for glyphs.

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 our scope). In brief, "A" is a character when we consider it in the abstract: to make it a glyph, we must select a typeface with which to render it, and determine its type size and color. The formatting process turns input characters into output glyphs. A few characters commonly seen on keyboards are treated specially by the *roff* language and may not look correct in output if used unthinkingly; they are the (double) quotation mark ("), the neutral apostrophe ('), the minus sign (−), the backslash (\), the caret or circumflex accent (^), the grave accent (`), and the tilde (~). All of these and more can be produced with *special character* escape sequences; see *groff_char*(7).

groff offers *streams*, identifiers for writable files, but for security reasons this feature is disabled by default.

A further few language elements arise as page layouts become more sophisticated and demanding. *Environments* collect formatting parameters like line length and typeface. A *diversion* stores formatted output for later use. A *trap* is a condition on the input or output, tested automatically by the formatter, that is associated with a macro, calling it when that condition is fulfilled.

Footnote support often exercises all three of the foregoing features. A simple implementation might work as follows. A pair of macros is defined: one starts a footnote and the other ends it. The author calls the first macro where a footnote marker is desired. The macro establishes a diversion so that the footnote text is collected at the place in the body text where its corresponding marker appears. An environment is created for the footnote so that it is set at a smaller typeface. The footnote text is formatted in the diversion using that environment, but it does not yet appear in the output. The document author calls the footnote end macro, which returns to the previous environment and ends the diversion. Later, after much more body text in the document, a trap, set a small distance above the page bottom, is sprung. The macro called by the trap draws a line across the page and emits the stored diversion. Thus, the footnote is rendered.

History

Computer-driven document formatting dates back to the 1960s. The *roff* system is intimately connected with Unix, but its origins lie with the earlier operating systems CTSS, GECOS, and Multics.

The predecessor—*RUNOFF*

roff's ancestor *RUNOFF* was written in the MAD language by Jerry Saltzer to prepare his Ph.D. thesis on the Compatible Time Sharing System (CTSS), a project of the Massachusetts Institute of Technology (MIT). This program is referred to in full capitals, both to distinguish it from its many descendants, and because bits were expensive in those days; five- and six-bit character encodings were still in widespread usage, and mixed-case alphabetics in file names seen as a luxury. *RUNOFF* introduced a syntax of inlining formatting directives amid document text, by beginning a line with a period (an unlikely occurrence in human-readable material) followed by a "control word". Control words with obvious meaning like ".line length *n*" were supported as well as an abbreviation system; the latter came to overwhelm the former in popular usage and later derivatives of the program. A sample of control words from a Unknown 〈〉 was documented as follows (with the parameter notation slightly altered). The abbreviations will be familiar to *roff* veterans.

Abbreviation	Control word
.ad	.adjust
.bp	.begin page
.br	.break
.ce	.center
.in	.indent *n*
.ll	.line length *n*
.nf	.nofill
.pl	.paper length *n*
.sp	.space [*n*]

In 1965, MIT's Project MAC teamed with Bell Telephone Laboratories and General Electric (GE) to inaugurate the Multics project. After a few years, Bell Labs discontinued its participation in Multics, famously prompting the development of Unix. Meanwhile, Saltzer's *RUNOFF* proved influential, seeing many ports and derivations elsewhere.

In 1969, Doug McIlroy wrote one such reimplementation, adding extensions, in the BCPL language for a GE 645 running GECOS at the Bell Labs location in Murray Hill, New Jersey. In its manual, the control commands were termed "requests", their two-letter names were canonical, and the control character was configurable with a **.cc** request. Other familiar requests emerged at this time; no-adjust (**.na**), need (**.ne**), page offset (**.po**), tab configuration (**.ta**, though it worked differently), temporary indent (**.ti**), character translation (**.tr**), and automatic underlining (**.ul**; on *RUNOFF* you had to backspace and underscore in the input yourself). **.fi** to enable filling of output lines got the name it retains to this day. McIlroy's program also featured a heuristic system for automatically placing hyphenation points, designed and implemented by Molly Wagner. It furthermore introduced numeric variables, termed registers. By 1971, this program had been ported to Multics and was known as *roff*, a name McIlroy attributes to Bob Morris, to distinguish it from CTSS *RUNOFF*.

Unix and *roff*

McIlroy's *roff* was one of the first Unix programs. In Ritchie's term, it was "transliterated" from BCPL to DEC PDP-7 assembly language for the fledgling Unix operating system. Automatic hyphenation was managed with **.hc** and **.hy** requests, line spacing control was generalized with the **.ls** request, and what later *roff*s would call diversions were available via "footnote" requests. This *roff* indirectly funded operating systems research at Murray Hill; AT&T prepared patent applications to the U.S. government with it. This arrangement enabled the group to acquire a PDP-11; *roff* promptly proved equal to the task of formatting the manual for what would become known as "First Edition Unix", dated November 1971.

Output from all of the foregoing programs was limited to line printers and paper terminals such as the IBM 2471 (based on the Selectric line of typewriters) and the Teletype Corporation Model 37. Proportionally spaced type was unavailable.

New *roff* and Typesetter *roff*

The first years of Unix were spent in rapid evolution. The practicalities of preparing standardized documents like patent applications (and Unix manual pages), combined with McIlroy's enthusiasm for macro languages, perhaps created an irresistible pressure to make *roff* extensible. Joe Ossanna's *nroff*, literally a "new roff", was the outlet for this pressure. By the time of Unix Version 3 (February 1973)—and still in PDP-11 assembly language—it sported a swath of features now considered essential to *roff* systems: definition of macros (**.de**), diversion of text thither (**.di**), and removal thereof (**.rm**); trap planting (**.wh**; "when") and relocation (**.ch**; "change"); conditional processing (**.if**); and environments (**.ev**). Incremental improvements included assignment of the next page number (**.pn**); no-space mode (**.ns**) and restoration of vertical spacing (**.rs**); the saving (**.sv**) and output (**.os**) of vertical space; specification of replacement characters for tabs (**.tc**) and leaders (**.lc**); configuration of the no-break control character (**.c2**); shorthand to disable automatic hyphenation (**.nh**); a condensation of what were formerly six different requests for configuration of page "titles" (headers and footers) into one (**.tl**) with a length controlled separately from the line length (**.lt**); automatic line numbering (**.nm**); interactive input (**.rd**), which necessitated buffer-flushing (**.fl**), and was made convenient with early program cessation (**.ex**); source file inclusion in its modern form (**.so**; though *RUNOFF* had an ".append" control word for a similar purpose) and early advance to the next file argument (**.nx**); ignorable content (**.ig**); and programmable abort (**.ab**).

Third Edition Unix also brought the *pipe*(2) system call, the explosive growth of a componentized system based around it, and a "filter model" that remains perceptible today. Equally importantly, the Bell Labs site in Murray Hill acquired a Graphic Systems C/A/T phototypesetter, and with it came the necessity of expanding the capabilities of a *roff* system to cope with a variety of proportionally spaced typefaces at multiple sizes. Ossanna wrote a parallel implementation of *nroff* for the C/A/T, dubbing it *troff* (for "typesetter roff"). Unfortunately, surviving documentation does not illustrate what requests were implemented at this time for C/A/T support; the *troff*(1) man page in Fourth Edition Unix (November 1973) does not feature a request list, troff(1) (1975) unlike *nroff*(1). Apart from typesetter-driven features, Unix Version 4 *roff*s added string definitions (**.ds**); made the escape character configurable (**.ec**); and enabled the user to write diagnostics to the standard error stream (**.tm**). Around 1974, empowered with multiple type sizes, italics, and a symbol font specially commissioned by Bell Labs from Graphic Systems, Kernighan and Lorinda Cherry implemented *eqn* for typesetting mathematics. In the same year, for Fifth Edition Unix, Ossanna combined and reimplemented the two *roff*s in C, using that language's preprocessor to generate both from a single source tree.

Ossanna documented the syntax of the input language to the *nroff* and *troff* programs in the "Troff User's Manual", first published in 1976, with further revisions as late as 1992 by Kernighan. (The original version was entitled "Nroff/Troff User's Manual", which may partially explain why *roff* practitioners have tended to refer to it by its AT&T document identifier, "CSTR #54".) Its final revision serves as the *de facto* specification of AT&T *troff*, and all subsequent implementors of *roff* systems have done so in its shadow.

A small and simple set of *roff* macros was first used for the manual pages of Unix Version 4 and persisted for two further releases, but the first macro package to be formally described and installed was *ms* by Michael Lesk in Version 6. He also wrote a manual, "Typing Documents on the Unix System", describing *ms* and basic *nroff*/*troff* usage, updating it as the package accrued features. Sixth Edition additionally saw the debut of the *tbl* preprocessor for formatting tables, also by Lesk.

For Unix Version 7 (January 1979), McIlroy designed, implemented, and documented the *man* macro package, introducing most of the macros described in *groff_man*(7) today, and edited volume 1 of the Version 7 manual using it. Documents composed using *ms* featured in volume 2, edited by Kernighan.

Meanwhile, *troff* proved popular even at Unix sites that lacked a C/A/T device. Tom Ferrin of the University of California at San Francisco combined it with Allen Hershey's popular vector fonts to produce *vtroff*, which translated *troff*'s output to the command language used by Versatec and Benson-Varian plotters.

Ossanna had passed away unexpectedly in 1977, and after the release of Version 7, with the C/A/T typesetter becoming supplanted by alternative devices such as the Mergenthaler Linotron 202, Kernighan undertook a revision and rewrite of *troff* to generalize its design. To implement this revised architecture, he developed the font and device description file formats and the page description language that remain in use today. He described these novelties in the article "A Typesetter-independent TROFF", last revised in 1982, and like the *troff* manual itself, it is widely known by a shorthand, "CSTR #97".

Kernighan's innovations prepared *troff* well for the introduction of the Adobe PostScript language in 1982 and a vibrant market in laser printers with built-in interpreters for it. An output driver for PostScript, *dpost*, was swiftly developed. However, AT&T's software licensing practices kept Ossanna's *troff*, with its tight coupling to the C/A/T's capabilities, in parallel distribution with device-independent *troff* throughout the 1980s. Today, however, all actively maintained *troff*s follow Kernighan's device-independent design.

***groff*—a free *roff* from GNU**

The most important free *roff* project historically has been *groff*, the GNU implementation of *troff*, developed by James Clark starting in 1989 and distributed under copyleft licenses, ensuring to all the availability of source code and the freedom to modify and redistribute it, properties unprecedented in *roff* systems to that point. *groff* rapidly attracted contributors, and has served as a replacement for almost all applications of AT&T *troff* (exceptions include *mv*, a macro package for preparation of viewgraphs and slides, and the *ideal* preprocessor, which produces diagrams from mathematical constraints). Beyond that, it has added numerous features; see *groff_diff*(7). Since its inception and for at least the following three decades, it has been used by practically all GNU/Linux and BSD operating systems.

groff continues to be developed, is available for almost all operating systems in common use (along with several obscure ones), and is free. These factors make *groff* the *de facto roff* standard today.

Other free *roff*s

In 2007, Caldera/SCO and Sun Microsystems, having acquired rights to AT&T Documenter's Workbench (DWB) *troff* (a descendant of the Bell Labs code), released it under a free but GPL-incompatible license. This implementation was made portable to modern POSIX systems, and adopted and enhanced first by Gunnar Ritter and then Carsten Kunze to produce Heirloom Doctools .I troff.

In July 2013, Ali Gholami Rudi announced .I neatroff, a permissively licensed new implementation.

Another descendant of DWB *troff* is part of Plan 9 from User Space. Since 2021, this *troff* has been available under permissive terms.

Using *roff*

When you read a man page, often a *roff* is the program rendering it. Some *roff* implementations provide wrapper programs that make it easy to use the *roff* system from the shell's command line. These can be specific to a macro package, like *mmroff*(1), or more general. *groff*(1) provides command-line options sparing the user from constructing the long, order-dependent pipelines familiar to AT&T *troff* users. Further, a heuristic program, *grog*(1), is available to infer from a document's contents which *groff* arguments should be used to process it.

The *roff* pipeline

A typical *roff* document is prepared by running one or more processors in series, followed by a a formatter program and then an output driver (or "device postprocessor"). Commonly, these programs are structured into a pipeline; that is, each is run in sequence such that the output of one is taken as the input to the next, without passing through secondary storage. (On non-Unix systems, pipelines may have to be simulated with temporary files.)

```
$ preproc1 < input-file | preproc2 | ... | troff [option] ... \
    | output-driver
```

Once all preprocessors have run, they deliver pure *roff* language input to the formatter, which in turn generates a document in a page description language that is then interpreted by a postprocessor for viewing, printing, or further processing.

Each program interprets input in a language that is independent of the others; some are purely descriptive, as with *tbl*(1) and *roff* output, and some permit the definition of macros, as with *eqn*(1) and *roff* input. Most *roff* input files employ the macros of a document formatting package, intermixed with instructions for one or more preprocessors, and seasoned with escape sequences and requests from the *roff* language. Some documents are simpler still, since their formatting packages discourage direct use of *roff* requests; man pages are a prominent example. Many features of the *roff* language are seldom needed by users; only authors of macro packages require a substantial command of them.

Preprocessors

A *roff* preprocessor is a program that, directly or ultimately, generates output in the *roff* language. Typically, each preprocessor defines a language of its own that transforms its input into that for *roff* or another preprocessor. As an example of the latter, *chem* produces *pic* input. Preprocessors must consequently be run in an appropriate order; *groff*(1) handles this automatically for all preprocessors supplied by the GNU *roff* system.

Portions of the document written in preprocessor languages are usually bracketed by tokens that look like *roff* macro calls. *roff* preprocessor programs transform only the regions of the document intended for them. When a preprocessor language is used by a document, its corresponding program must process it before the input is seen by the formatter, or incorrect rendering is almost guaranteed.

GNU *roff* provides several preprocessors, including *eqn*, *grn*, *pic*, *tbl*, *refer*, and *soelim*. See *groff*(1) for a complete list. Other preprocessors for *roff* systems are known.

dformat	depicts data structures;
grap	constructs statistical charts; and
ideal	draws diagrams using a constraint-based language.

Formatter programs

A *roff* formatter transforms *roff* language input into a single file in a page description language, described in *groff_out*(5), intended for processing by a selected device. This page description language is specialized in its parameters, but not its syntax, for the selected device; the format is device-*independent*, but not device-*agnostic*. The parameters the formatter uses to arrange the document are stored in *device* and *font description files*; see *groff_font*(5).

AT&T Unix had two formatters—*nroff* for terminals, and *troff* for typesetters. Often, the name *troff* is used loosely to refer to both. When generalizing thus, *groff* documentation prefers the term "*roff*". In GNU *roff*, the formatter program is always *troff*(1).

Devices and output drivers

To a *roff* system, a *device* is a hardware interface like a printer, a text or graphical terminal, or a standardized file format that unrelated software can interpret. An *output driver* is a program that parses the output of *troff* and produces instructions specific to the device or file format it supports. An output driver might support multiple devices, particularly if they are similar.

The names of the devices and their driver programs are not standardized. Technological fashions evolve; the devices used for document preparation when AT&T *troff* was first written in the 1970s are no longer used in production environments. Device capabilities have tended to increase, improving resolution and font repertoire, and adding color output and hyperlinking. Further, to reduce file size and processing time, AT&T *troff*'s page description language placed low limits on the magnitudes of some quantities it could represent. Its PostScript output driver, *dpost*(1), had a resolution of 720 units per inch; *groff*'s *grops*(1) uses 72,000.

roff **programming**

Documents using *roff* are normal text files interleaved with *roff* formatting elements. The *roff* language is powerful enough to support arbitrary computation and it supplies facilities that encourage extension. The primary such facility is macro definition; with this feature, macro packages have been developed that are tailored for particular applications.

Macro packages

Macro packages can have a much smaller vocabulary than *roff* itself; this trait combined with their domain-specific nature can make them easy to acquire and master. The macro definitions of a package are typically kept in a file called *name***.tmac** (historically, **tmac.***name*). Find details on the naming and placement of macro packages in *groff_tmac*(5).

A macro package anticipated for use in a document can be declared to the formatter by the command-line option **−m**; see *troff*(1). It can alternatively be specified within a document using the **mso** request of the *groff* language; see *groff*(7).

Well-known macro packages include *man* for traditional man pages and *mdoc* for BSD-style manual pages. Macro packages for typesetting books, articles, and letters include *ms* (from "manuscript macros"), *me* (named by a system administrator from the first name of its creator, Eric Allman), *mm* (from "memorandum macros"), and *mom*, a punningly named package exercising many *groff* extensions. See *groff_tmac*(5) for more.

The *roff* formatting language

The *roff* language provides requests, escape sequences, macro definition facilities, string variables, registers for storage of numbers or dimensions, and control of execution flow. The theoretically minded will observe that a *roff* is not a mere markup language, but Turing-complete. It has storage (registers), it can perform tests (as in conditional expressions like "**(\n[i] >= 1)**"), its "**if**" and related requests alter the flow of control, and macro definition permits unbounded recursion.

Requests and *escape sequences* are instructions, predefined parts of the language, that perform formatting operations, interpolate stored material, or otherwise change the state of the parser. The user can define their own request-like elements by composing together text, requests, and escape sequences *ad libitum.* A document writer will not (usually) note any difference in usage for requests or macros; both are found on control lines. However, there is a distinction; requests take either a fixed number of arguments (sometimes zero), silently ignoring any excess, or consume the rest of the input line, whereas macros can take a variable number of arguments. Since arguments are separated by spaces, macros require a means of embedding a space in an argument; in other words, of quoting it. This then demands a mechanism of embedding the quoting character itself, in case *it* is needed literally in a macro argument. AT&T *troff* had complex rules involving the placement and repetition of the double quote to achieve both aims. *groff* cuts this knot by supporting a special character escape sequence for the neutral double quote, "**\[dq]**", which never performs quoting in the typesetting language, but is simply a glyph, '"'.

Escape sequences start with a backslash, "\". They can appear almost anywhere, even in the midst of text on a line, and implement various features, including the insertion of special characters with "**\(*xx***" or "**\[*xxx*]**", break suppression at input line endings with "**\c**", font changes with "**\f**", type size changes with "**\s**", in-line comments with "**\"**", and many others.

Strings store text. They are populated with the **ds** request and interpolated using the ***** escape sequence.

Registers store numbers and measurements. A register can be set with the request **nr** and its value can be retrieved by the escape sequence **\n**.

File naming conventions

The structure or content of a file name, beyond its location in the file system, is not significant to *roff* tools. *roff* documents employing "full-service" macro packages (see *groff_tmac*(5)) tend to be named with a suffix identifying the package; we thus see file names ending in *.man*, *.ms*, *.me*, *.mm*, and *.mom*, for instance. When installed, man pages tend to be named with the manual's section number as the suffix. For example, the file name for this document is *roff.7*. Practice for "raw" *roff* documents is less consistent; they are sometimes seen with a *.t* suffix.

Input conventions

Since *troff* fills text automatically, it is common practice in the *roff* language to avoid visual composition of text in input files: the esthetic appeal of the formatted output is what matters. Therefore, *roff* input should be arranged such that it is easy for authors and maintainers to compose and develop the document, understand the syntax of *roff* requests, macro calls, and preprocessor languages used, and predict the behavior of the formatter. Several traditions have accrued in service of these goals.

- Follow sentence endings in the input with newlines to ease their recognition. It is frequently convenient to end text lines after colons and semicolons as well, as these typically precede independent clauses. Consider doing so after commas; they often occur in lists that become easy to scan when itemized by line, or constitute supplements to the sentence that are added, deleted, or updated to clarify it. Parenthetical and quoted phrases are also good candidates for placement on text lines by themselves.
- Set your text editor's line length to 72 characters or fewer; see the subsections below. This limit, combined with the previous item of advice, makes it less common that an input line will wrap in your text editor, and thus will help you perceive excessively long constructions in your text. Recall that natural languages originate in speech, not writing, and that punctuation is correlated with pauses for breathing and changes in prosody.
- Use **\&** after "**!**", "**?**", and "**.**" if they are followed by space, tab, or newline characters and don't end a sentence.
- In filled text lines, use **\&** before "**.**" and "**'**" if they are preceded by space, so that reflowing the input doesn't turn them into control lines.
- Do not use spaces to perform indentation or align columns of a table. Leading spaces are reliable when text is not being filled.
- Comment your document. It is never too soon to apply comments to record information of use to future document maintainers (including your future self). The **\"** escape sequence causes *troff* to ignore the remainder of the input line.
- Use the empty request—a control character followed immediately by a newline—to visually manage separation of material in input files. Many of the *groff* project's own documents use an empty request between sentences, after macro definitions, and where a break is expected, and two empty requests between paragraphs or other requests or macro calls that will introduce vertical space into the document. You can combine the empty request with the comment escape sequence to include whole-line comments in your document, and even "comment out" sections of it.

An example sufficiently long to illustrate most of the above suggestions in practice follows. An arrow → indicates a tab character.

```
.\"   nroff this_file.roff | less
.\"   groff -T ps this_file.roff > this_file.ps
→The theory of relativity is intimately connected with
the theory of space and time.
.
I shall therefore begin with a brief investigation of
the origin of our ideas of space and time,
although in doing so I know that I introduce a
controversial subject.  \" remainder of paragraph elided
.
.

→The experiences of an individual appear to us arranged
```

```
        in a series of events;
        in this series the single events which we remember
        appear to be ordered according to the criterion of
        \[lq]earlier\[rq] and \[lq]later\[rq], \" punct swapped
        which cannot be analysed further.
        .
        There exists,
        therefore,
        for the individual,
        an I-time,
        or subjective time.
        .
        This itself is not measurable.
        .
        I can,
        indeed,
        associate numbers with the events,
        in such a way that the greater number is associated with
        the later event than with an earlier one;
        but the nature of this association may be quite
        arbitrary.
        .
        This association I can define by means of a clock by
        comparing the order of events furnished by the clock
        with the order of a given series of events.
        .
        We understand by a clock something which provides a
        series of events which can be counted,
        and which has other properties of which we shall speak
        later.
        .\" Albert Einstein, _The Meaning of Relativity_, 1922
```

Editing with Emacs

Official GNU doctrine holds that the best program for editing a *roff* document is Emacs; see *emacs*(1). It provides an *nroff* major 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 (often labelled "Alt") while pressing and releasing the "x" key.

It is also possible to have the mode automatically selected when a *roff* file is loaded into the editor.

- The most general method is to include file-local variables at the end of the file; we can also configure the fill column this way.

  ```
  .\" Local Variables:
  .\" fill-column: 72
  .\" mode: nroff
  .\" End:
  ```

- Certain file name extensions, such as those commonly used by man pages, trigger the automatic activation of the *nroff* mode.

- Technically, having the sequence

  ```
  .\" -*- nroff -*-
  ```

  in the first line of a file will cause Emacs to enter the *nroff* major mode when it is loaded into the buffer. Unfortunately, some implementations of the *man*(1) program are confused by this practice, so we discourage it.

Editing with Vim

Other editors provide support for *roff*-style files too, such as *vim*(1), an extension of the *vi*(1) program. Vim's highlighting can be made to recognize *roff* files by setting the **filetype** option in a Vim *modeline*. For this feature to work, your copy of *vim* must be built with support for, and configured to enable, several features; consult the editor's online help topics "auto−setting", "filetype", and "syntax". Then put the following at the end of your *roff* files, after any Emacs configuration:

```
.\" vim: set filetype=groff textwidth=72:
```

Replace "groff" in the above with "nroff" if you want highlighting that does *not* recognize many of the GNU extensions to *roff*, such as request, register, and string names longer than two characters.

Authors

This document was written by Bernd Warken and GBranden Robinson.

See also

Much *roff* documentation is available. The Bell Labs papers describing AT&T *troff* remain available, and *groff* is documented comprehensively.

Internet sites

.I Unix Text Processing, by Dale Dougherty and Tim O'Reilly, 1987, Hayden Books. This well-regarded text brings the reader from a state of no knowledge of Unix or text editing (if necessary) to sophisticated computer-aided typesetting. It has been placed under a free software license by its authors and updated by a team of *groff* contributors and enthusiasts.

"History of Unix Manpages", an online article maintained by the mdocml project, provides an overview of *roff* development from Saltzer's *RUNOFF* to 2008, with links to original documentation and recollections of the authors and their contemporaries.

troff.org, Ralph Corderoy's *troff* site, provides an overview and pointers to much historical *roff* information.

Multicians, a site by Multics enthusiasts, contains a lot of information on the MIT projects CTSS and Multics, including *RUNOFF*; it is especially useful for its glossary and the many links to historical documents.

The Unix Archive, curated by the Unix Heritage Society, provides the source code and some binaries of historical Unices (including the source code of some versions of *troff* and its documentation) contributed by their copyright holders.

Unknown stores some documents using the original *RUNOFF* formatting language.

.I groff, GNU *roff*'s web site, provides convenient access to *groff*'s source code repository, bug tracker, and mailing lists (including archives and the subscription interface).

Historical *roff* documentation

Many AT&T *troff* documents are available online, and can be found at Ralph Corderoy's site (see above) or via Internet search.

Of foremost significance are two mentioned in section "History" above, describing the language and its device-independent implementation, respectively.

"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.

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97.

You can obtain many relevant Bell Labs papers in PDF from Bernd Warken's "roff classical" GitHub repository ⟨⟩.

Manual pages

As a system of multiple components, a *roff* system potentially has many man pages, each describing an aspect of it. Unfortunately, there is no consistent naming scheme for these pages among the different *roff* implementations.

For GNU *roff*, the *groff*(1) man page enumerates all man pages distributed with the system, and individual pages frequently refer to external resources as well as manuals distributed with *groff* on a variety of topics.

With other *roff*s, you are on your own, but *troff*(1) might be a good starting point.