NAME

roff − concepts and history of roff typesetting

DESCRIPTION

roff is the general name for a set of type-setting programs, known under names like *troff*, *nroff*, *ditroff*, *groff*, etc. A roff type-setting system consists of an extensible text formatting language and a set of programs for printing and converting to other text formats. Traditionally, it is the main text processing system of Unix; every Unix-like operating system still distributes a roff system as a core package.

The most common roff system today is the free software implementation *GNU roff*, **groff**(1). The pre-groff implementations are referred to as *classical* (dating back as long as 1973). *groff* implements the look-and-feel and functionality of its classical ancestors, but has many extensions. As *groff* is the only roff system that is available for every (or almost every) computer system it is the de-facto roff standard today.

In some ancient Unix systems, there was a binary called **roff** that implemented the even more ancient **runoff** of the *Multics* operating system, cf. section **HISTORY**. The functionality of this program was very restricted even in comparison to ancient troff; it is not supported any longer. Consequently, in this document, the term *roff* always refers to the general meaning of *roff system*, not to the ancient roff binary.

In spite of its age, roff is in wide use today, for example, the manual pages on UNIX systems (*man pages*), many software books, system documentation, standards, and corporate documents are written in roff. The roff output for text devices is still unmatched, and its graphical output has the same quality as other free type-setting programs and is better than some of the commercial systems.

The most popular application of roff is the concept of *manual pages* or shortly *man pages*; this is the standard documentation system on many operating systems.

This document describes the historical facts around the development of the *roff system*; some usage aspects common to all roff versions, details on the roff pipeline, which is usually hidden behind front-ends like **groff**(1); an general overview of the formatting language; some tips for editing roff files; and many pointers to further readings.

HISTORY

The *roff* text processing system has a very long history, dating back to the 1960s. The roff system itself is intimately connected to the Unix operating system, but its roots go back to the earlier operating systems CTSS and Multics.

The Predecessor runoff

The evolution of *roff* is intimately related to the history of the operating systems. Its predecessor **runoff** was written by *Jerry Saltzer* on the *CTSS* operating system (*Compatible Time Sharing System*) as early as 1961. When CTTS was further developed into the operating system Multics ⟨`http://www.multicians.org`⟩, the famous predecessor of Unix from 1963, *runoff* became the main format for documentation and text processing. Both operating systems could only be run on very expensive computers at that time, so they were mostly used in research and for official and military tasks.

The possibilities of the *runoff* language were quite limited as compared to modern roff. Only text output was possible in the 1960s. This could be implemented by a set of requests of length 2, many of which are still identically used in roff. The language was modelled according to the habits of typesetting in the pre-computer age, where lines starting with a dot were used in manuscripts to denote formatting requests to the person who would perform the typesetting manually later on.

The runoff program was written in the *PL/1* language first, later on in *BCPL*, the grandmother of the *C* programming language. In the Multics operating system, the help system was handled by runoff, similar to roff's task to manage the Unix manual pages. There are still documents written in the runoff language; for examples see Saltzer's home page, cf. section **SEE ALSO**.

The Classical nroff/troff System

In the 1970s, the Multics off-spring *Unix* became more and more popular because it could be run on affordable machines and was easily available for universities at that time. At MIT (the Massachusetts Institute of Technology), there was a need to drive the Wang *Graphic Systems CAT* typesetter, a graphical output device from a PDP-11 computer running Unix. As runoff was too limited for this task it was further

developed into a more powerful text formatting system by *Josef F. Osanna*, a main developer of the Multics operating system and programmer of several runoff ports.

The name *runoff* was shortened to *roff*. The greatly enlarged language of Osanna's concept included already all elements of a full *roff system*. All modern roff systems try to implement compatibility to this system. So Joe Osanna can be called the father of all roff systems.

This first *roff system* had three formatter programs.

troff (*typesetter roff*) generated a graphical output for the *CAT* typesetter as its only device.

nroff produced text output suitable for terminals and line printers.

roff was the reimplementation of the former runoff program with its limited features; this program was abandoned in later versions. Today, the name *roff* is used to refer to a troff/nroff sytem as a whole.

Osanna first version was written in the PDP-11 assembly language and released in 1973. *Brian Kernighan* joined the *roff* development by rewriting it in the C programming language. The C version was released in 1975.

The syntax of the formatting language of the **nroff**/**troff** programs was documented in the famous *Troff User's Manual [CSTR #54]*, first published in 1976, with further revisions up to 1992 by Brian Kernighan. This document is the specification of the *classical troff*. All later *roff* systems tried to establish compatibility with this specification.

After Osanna had died in 1977 by a heart-attack at the age of about 50, Kernighan went on with developing troff. The next milestone was to equip troff with a general interface to support more devices, the intermediate output format and the postprocessor system. This completed the structure of a *roff system* as it is still in use today; see section **USING ROFF**. In 1979, these novelties were described in the paper *[CSTR #97]*. This new troff version is the basis for all existing newer troff systems, including *groff*. On some systems, this *device independent troff* got a binary of its own, called **ditroff**(7). All modern **troff** programs already provide the full ditroff capabilities automatically.

Commercialization

A major degradation occurred when the easily available Unix 7 operating system was commercialized. A whole bunch of divergent operating systems emerged, fighting each other with incompatibilities in their extensions. Luckily, the incompatibilities did not fight the original troff. All of the different commercial roff systems made heavy use of Osanna/Kernighan's open source code and documentation, but sold them as "their" system — with only minor additions.

The source code of both the ancient Unix and classical troff weren't available for two decades. Fortunately, Caldera bought SCO UNIX in 2001. In the following, Caldera made the ancient source code accessible online for non-commercial use, cf. section **SEE ALSO**.

Free roff

None of the commercial roff systems could attain the status of a successor for the general roff development. Everyone was only interested in their own stuff. This led to a steep downfall of the once excellent Unix operating system during the 1980s.

As a counter-measure to the galopping commercialization, AT&T Bell Labs tried to launch a rescue project with their *Plan 9* operating system. It is freely available for non-commercial use, even the source code, but has a proprietary license that empedes the free development. This concept is outdated, so Plan 9 was not accepted as a platform to bundle the main-stream development.

The only remedy came from the emerging free operatings systems (386BSD, GNU/Linux, etc.) and software projects during the 1980s and 1990s. These implemented the ancient Unix features and many extensions, such that the old experience is not lost. In the 21st century, Unix-like systems are again a major factor in computer industry — thanks to free software.

The most important free roff project was the GNU port of troff, created by James Clark and put under the GNU Public License ⟨`http://www.gnu.org/copyleft`⟩. It was called *groff* (*GNU roff*). See **groff**(1) for an overview.

The groff system is still actively developed. It is compatible to the classical troff, but many extensions were added. It is the first roff system that is available on almost all operating systems — and it is free. This makes groff the de-facto roff standard today.

USING ROFF

Most people won't even notice that they are actually using roff. When you read a system manual page (man page) roff is working in the background. Roff documents can be viewed with a native viewer called **xditview**(1x), a standard program of the X window distribution, see **X**(7x). But using roff explicitly isn't difficult either.

Some roff implementations provide wrapper programs that make it easy to use the roff system on the shell command line. For example, the GNU roff implementation **groff**(1) provides command line options to avoid the long command pipes of classical troff; a program **grog**(1) tries to guess from the document which arguments should be used for a run of groff; people who do not like specifying command line options should try the **groffer**(1) program for graphically displaying groff files and man pages.

The roff Pipe

Each roff system consists of preprocessors, roff formatter programs, and a set of device postprocessors. This concept makes heavy use of the *piping* mechanism, that is, a series of programs is called one after the other, where the output of each program in the queue is taken as the input for the next program.

sh# `cat` *file* `|` `...` `|` *preproc* `|` `...` `|` `troff` *options* `|` *postproc*

The preprocessors generate roff code that is fed into a roff formatter (e.g. troff), which in turn generates *intermediate output* that is fed into a device postprocessor program for printing or final output.

All of these parts use programming languages of their own; each language is totally unrelated to the other parts. Moreover, roff macro packages that were tailored for special purposes can be included.

Most roff documents use the macros of some package, intermixed with code for one or more preprocessors, spiced with some elements from the plain roff language. The full power of the roff formatting language is seldom needed by users; only programmers of macro packages need to know about the gory details.

Preprocessors

A roff preprocessor is any program that generates output that syntactically obeys the rules of the roff formatting language. Each preprocessor defines a language of its own that is translated into roff code when run through the preprocessor program. Parts written in these languages may be included within a roff document; they are identified by special roff requests or macros. Each document that is enhanced by preprocessor code must be run through all corresponding preprocessors before it is fed into the actual roff formatter program, for the formatter just ignores all alien code. The preprocessor programs extract and transform only the document parts that are determined for them.

There are a lot of free and commercial roff preprocessors. Some of them aren't available on each system, but there is a small set of preprocessors that are considered as an integral part of each roff system. The classical preprocessors are

tbl	for tables
eqn	for mathematical formulæ
pic	for drawing diagrams
refer	for bibliographic references
soelim	for including macro files from standard locations

Other known preprocessors that are not available on all systems include

chem	for drawing chemical formulæ.
grap	for constructing graphical elements.
grn	for including **gremlin**(1) pictures.

Formatter Programs

A *roff formatter* is a program that parses documents written in the roff formatting language or uses some of the roff macro packages. It generates *intermediate output*, which is intended to be fed into a single device

postprocessor that must be specified by a command-line option to the formatter program. The documents must have been run through all necessary preprocessors before.

The output produced by a roff formatter is represented in yet another language, the *intermediate output format* or *troff output*. This language was first specified in *[CSTR #97]*; its GNU extension is documented in **groff_out**(5). The intermediate output language is a kind of assembly language compared to the high-level roff language. The generated intermediate output is optimized for a special device, but the language is the same for every device.

The roff formatter is the heart of the roff system. The traditional roff had two formatters, **nroff** for text devices and **troff** for graphical devices.

Often, the name *troff* is used as a general term to refer to both formatters.

Devices and Postprocessors

Devices are hardware interfaces like printers, text or graphical terminals, etc., or software interfaces such as a conversion into a different text or graphical format.

A roff postprocessor is a program that transforms troff output into a form suitable for a special device. The roff postprocessors are like device drivers for the output target.

For each device there is a postprocessor program that fits the device optimally. The postprocessor parses the generated intermediate output and generates device-specific code that is sent directly to the device.

The names of the devices and the postprocessor programs are not fixed because they greatly depend on the software and hardware abilities of the actual computer. For example, the classical devices mentioned in *[CSTR #54]* have greatly changed since the classical times. The old hardware doesn't exist any longer and the old graphical conversions were quite imprecise when compared to their modern counterparts.

For example, the Postscript device *post* in classical troff had a resolution of 720, while groff's *ps* device has 72000, a refinement of factor 100.

Today the operating systems provide device drivers for most printer-like hardware, so it isn't necessary to write a special hardware postprocessor for each printer.

ROFF PROGRAMMING

Documents using roff are normal text files decorated by roff formatting elements. The roff formatting language is quite powerful; it is almost a full programming language and provides elements to enlarge the language. With these, it became possible to develop macro packages that are tailored for special applications. Such macro packages are much handier than plain roff. So most people will choose a macro package without worrying about the internals of the roff language.

Macro Packages

Macro packages are collections of macros that are suitable to format a special kind of documents in a convenient way. This greatly eases the usage of roff. The macro definitions of a package are kept in a file called *name***.tmac** (classically **tmac.***name*). All tmac files are stored in one or more directories at standardized positions. Details on the naming of macro packages and their placement is found in **groff_tmac**(5).

A macro package that is to be used in a document can be announced to the formatter by the command line option **−m**, see **troff**(1), or it can be specified within a document using the file inclusion requests of the roff language, see **groff**(7).

Famous classical macro packages are *man* for traditional man pages, *mdoc* for BSD-style manual pages; the macro sets for books, articles, and letters are *me* (probably from the first name of its creator *Eric* Allman), *ms* (from *Manuscript Macros*), and *mm* (from *Memorandum Macros*).

The roff Formatting Language

The classical roff formatting language is documented in the *Troff User's Manual [CSTR #54]*. The roff language is a full programming language providing requests, definition of macros, escape sequences, string variables, number or size registers, and flow controls.

Requests are the predefined basic formatting commands similar to the commands at the shell prompt. The user can define request-like elements using predefined roff elements. These are then called *macros*. A doc-

ument writer will not note any difference in usage for requests or macros; both are written on a line on their own starting with a dot.

Escape sequences are roff elements starting with a backslash '\'. They can be inserted anywhere, also in the midst of text in a line. They are used to implement various features, including the insertion of non-ASCII characters with **\(**, font changes with **\f**, in-line comments with \", the escaping of special control characters like \\, and many other features.

Strings are variables that can store a string. A string is stored by the **.ds** request. The stored string can be retrieved later by the ***** escape sequence.

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

FILE NAME EXTENSIONS

Manual pages (man pages) take the section number as a file name extension, e.g., the filename for this document is *roff.7*, i.e., it is kept in section 7 of the man pages.

The classical macro packages take the package name as an extension, e.g. *file*.**me** for a document using the *me* macro package, *file*.**mm** for *mm*, *file*.**ms** for *ms*, *file*.**pic** for *pic* files, etc.

But there is no general naming scheme for roff documents, though *file*.**tr** for *troff file* is seen now and then. Maybe there should be a standardization for the filename extensions of roff files.

File name extensions can be very handy in conjunction with the **less**(1) pager. It provides the possibility to feed all input into a command-line pipe that is specified in the shell environment variable **LESSOPEN**. This process is not well documented, so here an example:

sh# `LESSOPEN='|lesspipe %s'`

where **lesspipe** is either a system supplied command or a shell script of your own.

EDITING ROFF

The best program for editing a roff document is Emacs (or Xemacs), see **emacs**(1). It provides an *nroff* mode that is suitable for all kinds of roff dialects. This mode can be activated by the following methods.

When editing a file within Emacs the mode can be changed by typing '*M-x nroff-mode*', where **M-x** means to hold down the **Meta** key (or **Alt**) and hitting the **x** key at the same time.

But it is also possible to have the mode automatically selected when the file is loaded into the editor.

- The most general method is to include the following 3 comment lines at the end of the file.

  .\" Local Variables:
  .\" mode: nroff
  .\" End:

- There is a set of file name extensions, e.g. the man pages that trigger the automatic activation of the nroff mode.

- Theoretically, it is possible to write the sequence

  .\" -*- nroff -*-

  as the first line of a file to have it started in nroff mode when loaded. Unfortunately, some applications such as the **man** program are confused by this; so this is deprecated.

All roff formatters provide automated line breaks and horizontal and vertical spacing. In order to not disturb this, the following tips can be helpful.

- Never include empty or blank lines in a roff document. Instead, use the empty request (a line consisting of a dot only) or a line comment **.\"** if a structuring element is needed.

- Never start a line with whitespace because this can lead to unexpected behavior. Indented paragraphs can be constructed in a controlled way by roff requests.

- Start each sentence on a line of its own, for the spacing after a dot is handled differently depending on whether it terminates an abbreviation or a sentence. To distinguish both cases, do a line break after each sentence.

- To additionally use the auto-fill mode in Emacs, it is best to insert an empty roff request (a line consisting of a dot only) after each sentence.

The following example shows how optimal roff editing could look.

```
This is an example for a roff document.
.
This is the next sentence in the same paragraph.
.
This is a longer sentence stretching over several
lines; abbreviations like 'cf.' are easily
identified because the dot is not followed by a
line break.
.
In the output, this will still go to the same
paragraph.
```

Besides Emacs, some other editors provide nroff style files too, e.g. **vim**(1), an extension of the **vi**(1) program.

BUGS

UNIX® is a registered trademark of the Open Group. But things have improved considerably after Caldera had bought SCO UNIX in 2001.

SEE ALSO

There is a lot of documentation on roff. The original papers on classical troff are still available, and all aspects of groff are documented in great detail.

Internet sites

troff.org
: The historical troff site ⟨`http://www.troff.org`⟩ provides an overview and pointers to all historical aspects of roff. This web site is under construction; once, it will be the major source for roff history.

Multics
: The Multics site ⟨`http://www.multicians.org`⟩ contains a lot of information on the MIT projects, CTSS, Multics, early Unix, including *runoff*; especially useful are a glossary and the many links to ancient documents.

Unix Archive
: The Ancient Unixes Archive ⟨`http://www.tuhs.org/Archive/`⟩ provides the source code and some binaries of the ancient Unixes (including the source code of troff and its documentation) that were made public by Caldera since 2001, e.g. of the famous Unix version 7 for PDP-11 at the Unix V7 site ⟨`http://www.tuhs.org/Archive/PDP-11/Trees/V7`⟩.

Developers at AT&T Bell Labs
: Bell Labs Computing and Mathematical Sciences Research ⟨`http://cm.bell-labs.com/cm/index.html`⟩ provides a search facility for tracking information on the early developers.

Plan 9
: The Plan 9 operating system ⟨`http://plan9.bell-labs.com`⟩ by AT&T Bell Labs.

runoff
: Jerry Saltzer's home page ⟨`http://web.mit.edu/Saltzer/www/publications/pubs.html`⟩ stores some documents using the ancient runoff formatting language.

CSTR Papers
: The Bell Labs CSTR site ⟨`http://cm.bell-labs.com/cm/cs/cstr.html`⟩ stores the original troff manuals (CSTR #54, #97, #114, #116, #122) and famous historical documents on programming.

GNU roff
: The groff web site ⟨`http://www.gnu.org/software/groff`⟩ provides the free roff implementation groff, the actual standard roff.

Historical roff Documentation

Many classical documents are still available on-line. The two main manuals of the troff language are

[CSTR #54]
J. F. Osanna, *Nroff/Troff User's Manual* ⟨`http://cm.bell-labs.com/cm/cs/54.ps`⟩; Bell Labs, 1976; revised by Brian Kernighan, 1992.

[CSTR #97]
Brian Kernighan, *A Typesetter-independent TROFF* ⟨`http://cm.bell-labs.com/cm/cs/97.ps`⟩, Bell Labs, 1981, revised March 1982.

The "little language" roff papers are

[CSTR #114]
Jon L. Bentley and Brian W. Kernighan, *GRAP — A Language for Typesetting Graphs* ⟨`http://cm.bell-labs.com/cm/cs/114.ps`⟩; Bell Labs, August 1984.

[CSTR #116]
Brian W. Kernighan, *PIC -- A Graphics Language for Typesetting* ⟨`http://cm.bell-labs.com/cm/cs/116.ps`⟩; Bell Labs, December 1984.

[CSTR #122]
J. L. Bentley, L. W. Jelinski, and B. W. Kernighan, *CHEM — A Program for Typesetting Chemical Structure Diagrams, Computers and Chemistry* ⟨`http://cm.bell-labs.com/cm/cs/122.ps`⟩; Bell Labs, April 1986.

Manual Pages

Due to its complex structure, a full roff system has many man pages, each describing a single aspect of roff. Unfortunately, there is no general naming scheme for the documentation among the different roff implementations.

In *groff*, the man page **groff**(1) contains a survey of all documentation available in groff.

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

AUTHORS

This document is part of *groff*, the GNU roff distribution. It was written by Bernd Warken ⟨`bwarken@mayn.de`⟩; it is maintained by Werner Lemberg ⟨`wl@gnu.org`⟩.