NAME

roff - concepts and history of roff typesetting

DESCRIPTION

roff is the general name for a set of text formatting programs, known under names like **troff**, **nroff**, **ditroff**, **groff**, etc. A *roff* system consists of an extensible text formatting language and a set of programs for printing and converting to other text formats. Unix-like operating systems distribute a *roff* system as a core package.

The most common *roff* system today is the free software implementation GNU *roff*, groff(1). *groff* implements the look-and-feel and functionality of its ancestors, with many extensions.

The ancestry of *roff* is described in section **HISTORY**. In this document, the term*r off* always refers to the general class of roff programs, not to the **roff** command provided in early UNIX systems.

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.

roff is used to format UNIX *manual pages*, (or *man pages*), the standard documentation system on many UNIX-derived operating systems.

This document describes the history of 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); a general overview of the formatting language; some tips for editing *roff* files; and many pointers to further readings.

HISTORY

Document formatting by computer dates 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

roff's ancestor **RUNOFF** was written in the MAD language by *Jerry Saltzer* for the *Compatible Time Sharing System (CTSS)*, a project of the Massachusetts Institute of Technology (MIT), in 1963 and 1964 – note that CTSS commands were all uppercase.

In 1965, MIT's Project MAC teamed with Bell Telephone Laboratories (BTL) and General Electric to begin the .I Multics system. A command called**runoff** w as written for Multics in the late 60s in the BCPL language, by *Bob Morris*, *Doug McIlroy*, and other members of the Multics team.

Like its CTSS ancestor, Multics **runoff** formatted an input file consisting of text and command lines; commands began with a period and were two letters. Output from these commands was to terminal devices such as IBM Selectric terminals. Multics **runoff** had additional features added, such as the ability to do two-pass formatting; it became the main format for Multics documentation and text processing.

BCPL and **runoff** were ported to the GCOS system at Bell Labs when BTL left the development of Multics.

There is a free archive about *historical RUNOFF* documents. You can get it anonymously by the shell command

```
$git clone https://github.com/bwarken/RUNOFF_historical.git
```

As well, there is a new project for writing a program that can read *RUNOFF files* , but it does not yet work so far. You can get an early version anonymously by the shell command

```
$git clone https://github.com/bwarken/runoff.git
```

The Classical nroff/troff System

At BTL, there was a need to drive the*Gr aphic 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 *Joseph F. Ossanna*, who already programmed several runoff ports.

The name *runoff* was shortened to *roff*. The greatly enlarged language of Ossanna's version already

included all elements of a full *roff system*. All modern*r off* systems try to implement compatibility to this system. So Joe Ossanna 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*CA T* 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* system as a whole.

Ossanna's 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*r off* systems tried to establish compatibility with this specification.

After Ossanna's death in 1977, Kernighan went on with developing *troff*. In the late 1970s, Kernighan equipped *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*de vice independent troff* got a binary of its own, called ditroff(7). All modern**tr off** programs already provide the full **ditroff** capabilities automatically.

Availability

The source code of both the ancient Unix and classical *troff* weren't available for two decades. Meanwhile, it is accessible again (on-line) for non-commercial use, cf. section **SEE ALSO**.

groff — free GNU roff

The most important free *roff* project was the `GNU` implementation of *troff*, written from scratch by *James Clark* and put under the GNU Public License. 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*tr off*, 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.

Free Heirloom roff

An alternative is .I Gunnar Ritter's Heirloom roff project project, started in 2005, which provides enhanced versions of the various roff tools found in the OpenSolaris and Plan 9 operating systems, now available under free licenses. You can get this package with the shell command:

```
$ git clone https://github.com/n-t-roff/heirloom-doctools
```

Moreover, one finds there the .I Original Documenter's Workbench Release 3.3.

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. *r off* 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.

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 formulae.
pic for drawing diagrams.
refer for bibliographic references.
soelim for including macro files from standard locations.
chem for drawing chemical formulæ.

Other known preprocessors that are not available on all systems include

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 units per inch, 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*r off*. 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*r off* 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 document 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:

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

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

More details for *file name e xtensions* can be found at groff_filenames(7).

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 .I 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.

SEE ALSO

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

Internet sites

troff.org
: The historical troff site provides an overview and pointers to all historical aspects of *roff*.

Multics
: The Multics site 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 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.

Developers at AT&T Bell Labs
: Bell Labs Computing and Mathematical Sciences Research provides a search facility for tracking information on the early developers.

Plan 9
: The Plan 9 operating system by AT&T Bell Labs.

runoff
: Jerry Saltzer's home page stores some documents using the ancient RUNOFF formatting language.

CSTR Papers
: The Bell Labs CSTR site stores the original *troff* manuals (CSTR #54, #97, #114, #116, #122) and famous historical documents on programming.

GNU *roff*
: The groff web site provides the free *roff* implementation *groff*, the actual standard *roff*.

Historical roff Documentation

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

[CSTR #54]
: J. F. Ossanna, .I Nroff/Troff User's Manual; Bell Labs, 1976; revised by Brian Kernighan, 1992.

[CSTR #97]
: Brian Kernighan, .I A Typesetter-independent TROFF, Bell Labs, 1981, revised March 1982.

The “little language” *roff* papers are

[CSTR #114]
: Jon L. Bentley and Brian W. Kernighan, .I GRAP – A Language for Typesetting Graphs; Bell Labs, August 1984.

[CSTR #116]
: Brian W. Kernighan, .I PIC – A Graphics Language for Typesetting; Bell Labs, December 1984.

[CSTR #122]
: J. L. Bentley, L. W. Jelinski, and B. W. Kernighan, .I CHEM – A Program for Typesetting Chemical Structure Diagrams, .I Computers and Chemistry; Bell Labs, April 1986.

You can get an archive with most *classical roff documentation* as reasonable *PDF files* at *github* using the *shell command*

```
$ git clone https://github.com/bwarken/roff_classical.git
```

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.

COPYING

AUTHORS

This man-page was written by Bernd Warken and is maintained by Werner Lemberg.