The Groff and Friends HOWTO

Dean Allen Provins

University of Calgary
dprovins@ucalgary.ca

1. Introduction

This HOWTO attempts to introduce the document formatting system known as *groff*. It isn't intended to be a reference document; other individuals are working on that. It is aimed at two groups: firstly, the individuals who are new to UNIX, Linux or another UNIX variant, and are discovering that operating system's enormous potential and versatility; and secondly, at those who are considering the use of a document formatting system for producing high quality and consistently formatted papers, articles and possibly books.

groff is not the only document formatting system available to writers. In recent years there have been developed "What You See Is What You Get" or **WYSIWYG** types of programs such as the many that are available under Microsoft's various operating systems. Other similar programs include those available from Sun Microsystems or Applix. In fact, these are not, or at least haven't been, document formatting programs so much as tools to generate shorter works, with their emphasis being on look, rather than content. The only true alternatives (known to this writer) are *troff* and the versatile TEX system with its accompanying macro packages such as LATEX.

As this writer understands the history of text formatting, the predecessor to TEX was *troff*. It was developed at AT&T Bell Labs and was in use from about 1971 according to [Kernighan, Lesk, and Ossanna 1978]. *troff* and related programs were provided as part of the **Documenter's WorkBench**, an optional extra for those purchasing a licence to use UNIX. In the late 1980's, the Free Software Foundation's version of these tools was developed, and was named *groff*. At about the same time (possibly a little earlier), the TEX formatting system was developed.

Like *troff/groff*, TEX and subsequently LATEX, use text formatting commands interspersed with document text to describe how a page is to be formatted by a type setting device.[1] *troff/groff* however, tend to be much more like typical UNIX applications in that the commands used to describe page formatting are terse. These commands are typically 3 characters in length and may be combined in sometimes difficult to read combinations (not unlike the regular expressions used with **sed** or **grep**). The macro packages available for use with *troff/groff* are somewhat more easily read, but are still brief. The advantage offered by TEX is that the commands are much more descriptive or readable. Their disadvantage (in the opinion of this writer) is that they take more time to type. Both however offer writers the opportunity to focus on content, without the distraction of look.

While the early systems were designed to drive specific devices, this is no longer the case. Both systems generate a neutral device-independent page description which subsequent processes format for specific devices. This is often via Postscript, which on UNIX or UNIX-clone systems is easily formatted for a wide variety of available printing and plotting devices, but need not be. Today, plain text, TEX format, PCL5 (HP's printer description language) and HTML are available.

Although I have acknowledged the availability of TEX and LATEX, this document will not dwell on them, or compare *groff* to them.[2] For further information on TEX, interested readers are referred to [Lamport 1985]

[1] The original *troff* program was designed to drive a specific machine, a Graphic Systems Incorporated C/A/T (Computer Assisted Typesetter) [Kernighan, Lesk, and Ossanna 1978; Sun Microsystems 1988].

[2] At least not in this revision.

and [Knuth 1986].

This document then will describe how one can profit by the use of *groff* to generate high quality articles, reports and even books (many of which have been written using *groff* or *troff*. See for example [Dougherty and O'Reilly 1987; Kernighan and Plauger 1974; Kernighan and Plauger 1976] and [Sequin and Tompsett 1975] to name a few).

2. Disclaimer

This document is intended to be one that assists new users by introducing them to *groff* and friends. It is not intended to be a definitive reference manual. Other documents cover these subjects very well, and are available either with the current distribution or at a web site identified in the sequel. *groff* comes with well-written **man pages** that describe the differences between *groff* and *troff*. These with the aforementioned reference documents are all that have been available for some time. Under development however, are complete reference documents for the *groff* suite.

Given the nature of this document, and the availability of other documentation, please do not hold me responsible for any damages that you or your organization may incur as a result of using the *groff* suite or this HOWTO. As with other free software, you use it at your own risk. No one associated with this software, this document, or any others mentioned herein has intended that you incur any damages. All products described here are made available freely, with the hope that they benefit you as much as they have benefited us.

3. Credits and Copyright

The *groff* document preparation suite was written by James J. Clark who re-implemented the *troff* system in some 60,000 lines of C++ code. Since then he has gone on to develop SGML parsers and related processors in both C and C++. All users of *groff* owe considerable thanks to his not inconsiderable efforts.

The current suite of programs are being maintained by Werner Lemberg and Ted Harding, with contributions by many others. As this document is being written, the current release of the suite, with enhancements for HTML and graph rendering, is version 1.13.

Some of the current and past contributors to this document, either directly or indirectly, plus contributors to the *groff* suite in general are:

Contributors to the *groff* suite

James J. Clark	groff's author
Paul Eggert	various patches
Trent Fisher	tex.info file
Ted Harding	user assistance
Werner LEMBERG	release coordination
Eddie Maddox	web site manager
Bernhard Reiter	mail list manager

This document is copyright by the author[3], who grants all readers a non-exclusive right to copy and to distribute it without compensation under the terms of the GNU General Public Licence.[4]

4. ms Macro Overview

As indicated earlier, there are *macro* packages available for *groff*; but what is a macro package? Think of it as a high level page description, which is written in a low level page description language. An analogy to a macro package might be a procedural language like C, which is relatively easy to read, but which (when compiled) is turned into a low level language like assembler, and ultimately machine instructions. Few people code in assembly language any more,[5] preferring to use higher level instruction sets (which

[3] © 1999 Dean Allen Provins

[4] For the terms of this licence, see the file COPYING that comes with the *groff* distribution.

incidentally greatly improves productivity).

In a similar manner, a macro package for *groff* such as the package of *ms* macros[6] is a collection of statements, which when interpreted are translated into low level *troff* instructions. You are welcome to use only the low level instructions in your documents, but you will likely find that the use of a package like *ms* greatly facilitates your productivity. On the other hand, you may find that the macro packages available don't meet your specific formatting needs, and you may wish to alter an existing package, or write your own package.

In this document, you won't see any new macro package development, but later I'll show you how one might customize some of the *ms* commands.

When *groff* invokes the GNU version of *troff*, it causes the replacement of the high level macro commands that you introduce to describe how you have organized your text (into chapters, sections, tables and so on) into low level formatting commands. These are interpreted by the code that actually manipulates your text into an attractive document. If you're familiar with **HTML**, which is a subset of Structured Generalized Markup Language, or **SGML** then you already know about page formatting and markup languages. The *troff* macros (and consequently the *groff* macros) are simply the earliest implementation of machine interpreted markup languages.

The *ms* macros are designed to assist you in formatting letters, proposals, memos, technical papers and reports. I have used them for all of these purposes, and have also created (admittedly short) books. To meet my particular needs, I have added to the package, and also modified the supplied macros. In the sequel, I'll show some of these things[7] so that you are left with a sufficiently comfortable feeling about what you can do, and how you might go about it.

4.1. A Simple Example

Suppose that we need to write a short report. It requires a title, some author information and an abstract to precede several chapters (or sections), some of which may have subsections. We can format it as follows:

```
.TL
Our Important Report
.AU
Dean Provins
.AI
Noted Author and Hacker
Student of Life and the University of Calgary
.AB
Noted author and hacker, Dean Provins reveals all in the much heralded
HOWTO for \fIgroff\fP. In this classic document, Provins describes the
inner workings of one of the earliest machine document processing systems
available.

Readers are encouraged to try any or all the examples provided.
.AE
.NH
Machine Document Processing
.LP
In this chapter, we see that computers have finally come into their own.
No more must workers labour over Gutenberg presses, as they have for
```

[5] Some of us older folks found it rather enjoyable. It sort of set apart the men from the boys so-to-speak.

[6] There are several macro packages available with the *groff* distribution, including *man*, *ms* and *me*: see 'man 7 groff_ms' and 'man 7 groff_me' for details.

[7] I won't describe all, as some are embarrassingly poor hacks!

```
20 generations.
21 .NH
22 Macros
23 .LP
24 There are many macro packages available for \fIgroff\fP and friends.  There
25 are even more available for its predecessor \fItroff\fP.
26 .ds MS \fIms\fP
27 .NH 2
28 The \*[MS] Macro Package
29 .LP
30 The \*[MS] macros provide a much needed lift for those toiling with the
31 basic \fItroff\fP commands.  They (the \*[MS] macros) are now used
32 extensively by writers the world over.
```

The result looks like the following (I trimmed the bottom of the page off for this display):

Our Important Report

Dean Provins

Noted Author and Hacker
Student of Life and the University of Calgary

ABSTRACT

Noted author and hacker, Dean Provins reveals all in the much heralded HOWTO for *groff*. In this classic document, Provins describes the inner workings of one of the earliest machine document processing systems available.

Readers are encouraged to try any or all the examples provided.

1. Machine Document Processing

In this chapter, we see that computers have finally come into their own. No more must workers labour over Gutenberg presses, as they have for generations.

2. Macros

There are many macro packages available for *groff* and friends. There are even more available for its predecessor *troff*.

2.1. The *ms* Macro Package

The *ms* macros provide a much needed lift for those toiling with the basic *troff* commands. They (the *ms* macros) are now used extensively by writers the world over.

So what happened? The *ms* macros which are introduced by a period, followed by 2 uppercase letters in the example text were interpreted by the *troff* processor as representing *troff* commands. These then controlled the formatting of the final result. With the exception of line 26 (which is a *troff* command to be explained later) all others represented font size changes, centering, emboldening and so on.

We start with the .**TL** command. This is a macro for a **title**. It appears at the beginning of a document, centers the following text, increases the text size and sets it in bold. The author is identified after line 3 (.**AU**) which indicates that an **author name** (or several author's names) follows. This macro sets the name in italics.

The author is further identified by lines of text following the .**AI** macro in line 5. The **abstract** is identified by a .**AB** macro call, and is terminated by a .**AE** which presumably means **abstract end**.

The text of the document follows line 15, which indicates a **chapter** or main **section** via the macro call .**NH**. The section/chapter title follows immediately, and is terminated by .**LP** on line 17. Until the next macro call on line 21, all text is read and after setting in the current font and pitch, with line filling[8] turned on, The text is output to be later formatted as PostScript (or whatever format you wish, and is available).

At line 21, another chapter is identified and text is read in lines 24 and 25.

At line 26, the author thought he might take a short cut, and declared a string expression named **MS** as '\fBms\fP'. From that point on, the author could use the string expansion expression '*[MS]' to represent '\fBms\fP'. Admittedly, this wasn't much of a saving, but it could have been if the string definition was for '\fItroff\fP' which takes more keystrokes than '*[MS]'. The .**ds** definition is an example of a *troff* command inserted into a document to facilitate text entry.

All that remains in the example is a **subsection** definition at line 27. As you can see in the resulting text, sections receive numeric identifiers followed by a period. Subsections received subordinate numeric identifiers. The *ms* macros allow for up to 4 levels of sections/subsections.

For the record, another macro command named .**SH** introduces **unnumbered sections**.

As you can see, using the *ms* macros is quite straightforward. There is no need for you to fill every line of text; in fact, *groff* is quite happy with a single word per line (although you may find it difficult to read). When entering text, use your favourite text editor (**vi** or **emacs**), and type away. If you need to add a few lines, just insert it at the appropriate point. With a text editor, it will be unlikely that you will ever encounter the formatting and overflow problems that have plagued WYSIWYG word processing programs on longer documents. They simply don't have the overhead; just one more reason to use a document processing system!

Just a word about **blank lines**. Blank lines terminate the filling of lines. They also indicate the ends of paragraphs. Note that the chapter/section/subsection commands generate blank lines before and after the chapter/section/subsection title.

5. Tables

Periodically, you may find that you need to set out some text in a tabular form. Perhaps it's a list of names and numbers, or perhaps it is much more complicated. The table capabilities of the *groff* suite are handled by a processor known as **tbl**.

To use **tbl**, one needs to describe the general format (it needn't be fixed for the entire table), and then enter the text in a manner that indicates which text belongs in which column. While I describe text as appearing in tables, I know that equations can also be entered. Whether figures can be or not, is not something I have ever experimented with.

The only limitation that I have ever encountered with tables is the use to footnotes. As I recall, automatically numbered footnotes are incorrectly numbered; but we'll see if that still holds in the following example.

Suppose that we have some names and telephone numbers of key individuals, and that we would like to organize them in a table to be contained in a document. The following might do the job for an individual who uses *groff* to keep his diary:[9]

```
.SH
Another Day in the Life of Dean P.
```

[8] If insufficient text is read on an input line, more text from the next and subsequent lines is added to fill the output line. This happens until a blank line or another macro call is encountered.

[9] Let us assume that we don't need numbered chapters.

```
3   .LP
4   Lately, my life has become more complicated.  Now the boss wants me to
5   remember the names and telephone numbers of key suppliers, and selected
6   contacts.  As if I don't already have enough to do.  Well my memory used
7   to be up to the task, but in these declining years, if I want to hang on
8   to this job, I better work smarter.  I think I'll use that new-fangled
9   groff program.  I hear it's great!
10  .TS
11  center box tab (:);
12  cb cb cb cb
13  r | n | l | l.
14  Telephone Number:Serial Number:Contact:Supplier
15  _
16  123-3456:123.:Jim:ABS Supplies
17  123-5437:.456::Genco Paints
18  557-234-3333:123.456:Suzy:HAL Computer Systems
19  .TE
20  Well that should do it.  As long as I have this diary, I'll never forget
21  again.
22  .SH
23  The Next Day
24  .LP
25  Wow!  I sure impressed the boss today.  I got ABS Supplies contact name
26  with just one small glance at my diary.  Now I'm in his good books!
```

And what is the result of this?

Another Day in the Life of Dean P.

Lately, my life has become more complicated. Now the boss wants me to remember the names and telephone numbers of key suppliers, and selected contacts. As if I don't already have enough to do. Well my memory used to be up to the task, but in these declining years, if I want to hang on to this job, I better work smarter. I think I'll use that new-fangled *groff* program. I hear it's great!

Telephone Number	**Serial Number**	**Contact**	**Supplier**
123-3456	123.	Jim	ABS Supplies
123-5437	.456		Genco Paints
557-234-3333	123.456	Suzy	HAL Computer Systems

Well that should do it. As long as I have this diary, I'll never forget again.

The Next Day

Wow! I sure impressed the boss today. I got ABS Supplies contact name with just one small glance at my diary. Now I'm in his good books!

In this example, we used the section command .**SH** to provide unnumbered section titles (lines 1-3). Line 10 saw the beginning of our table, which went to line 19. As you can see, tables begin with a .**TS** command, and end with a .**TE** command.

The first line after .**TS** (up to the semi-colon) describes some general parameters. I have indicated that the table should be centered, enclosed in a box, and that the text information will be divided into columns by a colon (the **tab** command).

The next line applies to the first line of the table data. In this case it says that there will be four columns, and that the text for each column (if given) is to be centered and emboldened. The final line (up to the period on line 13) describes the subsequent lines of text which make up the table. Here I indicated that the first column was to be right justified (**r**) and followed by a vertical line (**|**). The next column will be treated as numeric data and the decimal points will be aligned. The last two columns will be left justified. Vertical lines will separate all columns. The period at the end of this line indicates that the formatting information is complete.

The next line is presumed to be the column headers (corresponding to the first line of format description). Then comes a horizontal line, followed by the remaining lines of text. Notice the colons separating each column, and notice that line 17 lacks one column of data. The **tbl** program leaves space for missing data, if necessary. The **tab** character (a colon here) can be any character not in the text (a real TAB works fine too).

This is a very simple table example, but is useful to illustrate how easy it is to generate one. A very slightly more complicated table might look like,

A Table Containing Equations	
Equation 1	$a = \sqrt{b^2 + \frac{c}{d^3}}$
Equation 2	$a_k = \sum_{i=1}^{N} C_i \int_0^{\pi} x_k^2 dx$
Don't you wish there were **no more**	Equations

It was generated by means of

```
1   .TS
2   center doublebox delim (@#) tab (	);
3   cb s
4   l l.
5   A Table Containing Equations
6   _
7   Equation 1	@a = sqrt {b sup 2 + c over {d sup 3}}#

8   Equation 2	@a sub k = sum from i=1 to N C sub i  int from 0 to pi x sub k sup 2 dx#
9   .T&
10  r l.
11  T{
12  Don't you wish there were
13  \fBno more\fP
14  T}	Equations
15  .TE
```

This time I asked for a **doublebox** centered, with equation delimiters of @ and #.[10] and a TAB character as the separator between the columns.[11]

In this example, I have also illustrated inline equation definitions. As you can see, they are textual, which makes them reasonably easy to read (and key in).

Between lines 7 and 8 there is a blank line (unnumbered), and at line 9, I have given a **tbl** command (**.T&**) which says that I wish to restate the format of the columns (which I define on the next line, ending it in a period).

Just for fun, and to illustrate how one can enter long strings of text (although mine is very short), I have illustrated the use of the text block entry commands **T{** and **T}**. These must appear as follows:

[10] In my documents, for historical reasons, I set these as the equation delimiters. More typically, you would use two @ symbols.

[11] A real TAB entered at this location on the page causes the parentheses to appear separated by this amount. At another location, the amount might be different.

- Enter the **T{** as the first two characters immediately following the **tab**, or the first two characters if the text block belongs in the first column.
- Place the **T}** on the line immediately following the last line of text in the block. If another column follows, enter the **tab** and continue with that column's text.

The **tbl** program will wrap blocks of text into a single column for you. As you can see, **tbl** wraps the text where it likes (not necessarily where you would like); nevertheless, text wrap, especially in the last column can be very useful.

6. Equations

7. Figures

8. References

9. Useful Scripts

10. Customizing the ms Macros

11. Customizing Equations and References

12. A Sample Document

13. Document Organization and a Sample Makefile

14. Formats Other Than PostScript

15. Questions About *groff*; Some With Answers

15.1. How Do I Convert PostScript to ASCII or HTML?

PostScript to ASCII conversions may be performed via *ps2ascii*. This program is a Ghostscript translator from PostScript or PDF to ASCII (i.e. plain text). Apart from first converting PostScript to ASCII, and then adding suitable formatting commands to generate HTML, I have no idea how one can convert directly to HTML.

15.2. Is There a *groff info* File?

Trent Fisher has contributed a rudimentary groff.texinfo file and this is now a part of the *groff* distribution. Should an interested party wish to extend this work, their assistance would be gratefully received.

15.3. How can I do Cyrillic with *groff*?

Recently the following question was posed:

On 29-Oct-99 Grigoriy Strokin wrote:
> Hello,
>
> How do I prepare Cyrillic documents in PostScript
> using groff, provided I have Cyrillic fonts for Ghostscript?
>
> In fact, some cyrillic letters appear as expected,
> but some other are replaced by groff to funny characters
> such as Thorn or multiply sign.

The following is a transcription of the response:

I'm not familiar with Cyrillic fonts for Ghostscript, and these may possibly have a problem with missing characters, for all I know. The following assumes that you have a PostScript font with a complete Cyrillic character set (possibly from ghostscript, of course).

Provided you have "copied" the Cyrillic fonts to groff (i.e. you have installed them as groff fonts in ..../groff/fonts/devps) you could handle this according to some variant of one of the two following methods.

METHOD 1

Look, in this directory, at the font file for a particular font (e.g. I have a font which I pulled from the Web called "College Cyrillic" whose font file I have called "CCy", quite arbitrarily, to which the following applies).

This may start with a list of "kernpairs", but in any case you will come to a line with the word "charset".

After the "charset" line you will see several columns of information (the number of columns depends on the font), most of which is "font metrics", one line for each character. Some characters will be Latin, others Cyrillic.

The first column will be either a single ASCII character, a groff name for the character, or "---". The last column but one will be an octal number of the form "0pqr". For instance, the standard octal code for the letter "A" in English is "0101". In decimal, this is "65" which is the ASCII code for "A". The rule for conversion from octal "0pqr" to decimal is 64*p + 8*q + r.

You now need to identify which decimal codes correspond to which Cyrillic characters (if they also have groff names, so much the better; but it doesn't matter). The most direct way to do this is to print out a table consisting (in groff input) of lines like

241 \N'241'

where (in this example) the "241" is the decimal form of the octal 0361.

The groff escape sequence "\N'237'" causes the character with octal code "0361" in the correct font to be printed. So you must make sure that the current font is the Cyrillic font when this is printed (e.g. by using "\f[...]", or the request ".ft ..." on a line by itself, where "..." is the name of the font, i.e. the name of the font file you were looking at above, such as "\f[CCy]" or ".ft CCy").

Once you have got this information, you can then link this to your input.

Myself, being primarily in a Latin environment with Cyrillic being required exceptionally, I use two schemes. One is for single Russian words or very short pieces of Russian text, and is based on transliteration. For instance, the Russian "YA" (looking like an upper-case reversed "R") is denoted by the groff escape "\(YA". Similarly, the lower-case version is "\(ya".

This is pre-defined (in private macros) by

```
.char \(YA \N'241'
```

(This character is at 241 in most -- but not all -- of the Cyrillic PostScript fonts which I have. Therefore, if that is also the case with you, you should switch to Cyrillic using a macro like

```
.Cyr
.fnt fontname
.tr \(xx \N'nnn'
.tr ...  ...
  ...
.
```

If the Cyrillic character encoding depends on which font you are using, it might be better to make the name of the macro the same as the name of the font. Then you can use font-specific translations.

The other (for extended text where using escapes like "\(YA" throughout would be too tedious) is based on a similar character translation which allows me to type in Latin characters using a transliteration-like convention (I can't be bothered to fiddle with keyboards) like (using Russian alphabetical order)

```
.tr A \N'225'      [Cyrillic char identical to Latin "A"]
.tr B \N'226'      [Cyrillic char like large 'b']
.tr V \N'247'      [Cyrillic char identical to Latin "B"]
.tr G \N'231'      [Cyrillic char identical to Greek GAMMA]
..
.tr \(YA \N'241'
```

so that, for instance, the word for "Probability" would be entered as "Vero\(yatnoste\(i:". This reduces the use of escape sequences to the minimum. (The sequence "\(i:" stands for "i-kratkoi"

You could also, of course, Grigoriy, define the compound character

```
.char \(iy i\(i:
```

so that your name would be typed "Grigor\(iy".

METHOD 2 (which depends on info from Method 1)

In your environment, which is presumably primarily Cyrillic, you will probably be using a Russian keyboard, and a screen font based on the ISO font encoding iso-8859-5 (Cyrillic) where various numerical codes above 128 correspond to Cyrillic characters.

For the reasons given above, the iso-latin-5 character encoding may differ from the encoding in the PostScript font you are using. Therefore, you should carry out the above check as to which decimal codes correspond to which printed characters in the Cyrillic PostScript font. Some of these may coincide with the iso-8859-5 encoding; others may be different. In that case, what you need to do is to define a translation for the anomalous codes, as above, so that you can type the characters normally using the keyboard, they will print correctly, and you will also see them on the screen correctly.

In the first place, this is a lot of work; but once it has been done you will have an easy life!

The best of luck, Ted.

15.4. Where can I get Windows binaries of *groff*?

15.5. Is There a Document Like the "Troff User's Manual" for *groff*?

One of the maintainers (Ted Harding) has embarked on such a document. As this is not a trivial undertaking, it may not be available for some time. Readers are encouraged to subscribe to the *groff* mail list to follow the development, or to volunteer to assist Ted in order to speed it along.

For immediate use, are a series of overheads that Ted used in a recent presentation to his local Linux user's group. These foils, with their *groff* source files, are available from the following European site: **http://www.manlug.mcc.ac.uk/19991120/talk.tar.gz**. See the file **talk.txt** for a description of the contents.

16. Other Sources of Information

Several of the references cited earlier are worth looking at, if copies can be located in your local library. In particular, the book by Dougherty and O'Reilly is useful, although not particularly so as a reference[12]. The Sun Microsystems document is very good, and is based on the PostScript files mentioned below.

There are several documents available in PostScript format which describe the use of *troff* and associated processors. These are available at **http://www.kohala.com/start/** in the *troff resources* link and at **http://cm.bell-labs.com/cm/cs/cstr.html**.

At **http://plan9.bell-labs.com/7thEdMan/index.html** may be found the original *troff* sources for the documentation that was shipped with the **Unix Seventh Edition Manual**. Included is material on *troff* and related processors.

The *groff* maintainers have set up a mailing list and web site to provide interested users with the latest releases, bug fixes and problem solving facilities. In addition, a moderated newsgroup named **gnu.groff.bug** is available.

To subscribe to the **mail list**, send an Email to **groff-request@ffii.org** with the word **subscribe** in either the header or the mail body. Then, to post to the list, send your email to **groff@ffii.org**. An acknowledgement of your request to join will be received, and you will be assigned a password. Mail received by the list will be archived at **http://ffii.org/archive/mails/groff/**.

The web site is **http://groff.ffii.org/**. There, the latest version of *groff* and friends is always available.

References

Dougherty and O'Reilly 1987.
Dale Dougherty and Tim O'Reilly, *UNIX Text Processing,* Hayden Books, A Division of Howrd W. Sams and Company, Indianapolis, Indiana, U.S.A. (1987).

Kernighan, Lesk, and Ossanna 1978.
B.W. Kernighan, M.E. Lesk, and J.F. Ossanna, Jr., "Document Preparation," *The Bell System Technical Journal,* 57, 6, part 2, pp. 2115-2135, American Telephone and Telegraph Company, Short Hills, New Jersey, U.S.A. (July/August 1978).

[12] This writer finds the text too wordy, and the index not sufficiently so.

Kernighan and Plauger 1974.
B.W. Kernighan and P.J. Plauger, *The Elements of Programming Style,* McGraw-Hill, New York, U.S.A. (1974).

Kernighan and Plauger 1976.
B.W. Kernighan and P.J. Plauger, *Software Tools,* Addison-Wesley, Reading Massechusetts, U.S.A. (1976).

Knuth 1986.
Donald E. Knuth, *The TeXbook (ISBN 0-201-13447-0),* Addison-Wesley (1986).

Lamport 1985.
Leslie Lamport, *LaTeX - A Document Preparation System,* Addison-Wesley (ISBN 0-201-15790-X) (1985).

Sequin and Tompsett 1975.
C.H. Sequin and M.F. Tompsett, *Charge Transfer Devices,* Academic Press, New York, U.S.A. (1975).

Sun Microsystems 1988.
Sun Microsystems, "Using NROFF and TROFF" in *Sun Microsystems System Documentation,* Sun Microsystems (May 1988 Revision A).