nroff, The Text-Formatting Language

nroff is the COHERENT system's text-formatting language. You write a file that mingles the text you want formatted with commands to control the formatting. **nroff** then uses the commands to format the text, and writes the formatted text onto the standard-output device.

This tutorial describes how to work with **nroff.** It assumes you are familiar with the basic features of the COHERENT system. In particular, you should know what a *command* is, what a *file* is, and how to create and edit a file. If you are not familiar with these concepts, read *Using the COHERENT System* before you read this tutorial.

The Lexicon also contains a number of articles that relate to **nroff**. In particular, you should read the article for **printer**, which describes how you can print text under the COHERENT system.

What is nroff?

nroff is the text processor for COHERENT. A *text processor* is a utility that accepts commands and text, and uses the commands to format the text on a page. The commands may call for simple formatting, such as indenting each new paragraph five spaces, to complex formatting of columns and entire pages.

A file that contains text mixed with **nroff** commands is called a *script*. For example, the following **nroff** script

```
.nr Z 0 5
.nf
I tire of love,
.ti \n+Z
I sometimes tire of rhyme;
.ti \n-Z
But money makes me happy
.ti \n+Z
All the time!
.fi
```

produces the following printed text:

```
I tire of love,
        I sometimes tire of rhyme;
But money makes me happy
        All the time!
```

An **nroff** script allows you to change your output very easily. For example, change the minus sign '-' in line 7 of the **nroff** to a plus sign '+', and the formatted text suddenly becomes:

```
I tire of love,
        I sometimes tire of rhyme;
                But money makes me happy
                        All the time!
```

As you can see, **nroff** is a powerful and versatile formatter.

In truth, however, **nroff** is both a text formatter and a text formatting *language*. With **nroff**, you can write your own text-formatting commands to handle automatically the unique requirements of whatever formatting you need.

nroff Input and Output

Input is what you give to **nroff**. *Output* is what **nroff** returns to you. If you simply type

```
nroff
```

then **nroff** accepts input from your keyboard, and prints its output on your screen. For example, if you want **nroff** to process the contents of a file named **script.r**, type the command line

```
nroff script.r
```

nroff then takes the file **script.r**, processes it, and in a few moments it displays the formatted text on your screen. Note that the suffix **.r** is used by convention to indicate that a file contains an unprocessed **nroff** script.

You can save **nroff**'s output by *redirecting* it into another file. For example, you can redirect **nroff**'s processed output of the file **script.r** into the file named **target** by using the following command:

```
nroff script.r > target
```

Printing nroff Output

The COHERENT system's implementation of **nroff** currently can be used with any variety of printer. COHERENT, however, fully supports three varieties of printer: Epson-compatible dot-matrix printers, printers that use the Hewlett-Packard Page Control Language (PCL) (including the Hewlett-Packard LaserJet and DeskJet families of printers), and any printer that has implemented the PostScript page-control language. The following descriptions assume that you have plugged your printer into a parallel port on your computer, and have installed COHERENT correctly so that it can access your printer.

To print **nroff** output on an Epson-compatible printer, use the commands **epson** and **lpr**. For example, to print the **nroff** output that you have directed into file **text.out**, use the following command:

```
epson text.out | lpr
```

Or, you can pipe the output of **nroff** directly into **epson**, as follows:

```
nroff -ms text.r | epson | lpr
```

In the above example, **text.r** is your input, and **-ms** invokes the **ms** package of macros.

To print on a printer that uses PCL, use the commands **hp** and **hpr**. For example, to print the file **text.out** on a PCL printer, use the command:

```
hp text.out | hpr -B
```

The option **-B** to **hpr** suppresses the printing of a banner page. If you wish, you can pipe the output of **nroff** directly into **hp**, as follows:

```
nroff -ms text.r | hp | hpr -B
```

To access a printer that uses PostScript, use the command **hpr**, but do not use the command **hp**. Also, you use must the **-p** switch to **nroff**, which tells it to generate PostScript output. For example, the following command processes file **text.r** into PostScript output, and passes that output to a PostScript printer:

```
nroff -p -ms text.r | hpr -B
```

All of the above commands are described in their respective entries in the Lexicon.

You can also print the output of **nroff** through the **lp** spooler. For information on that spooler, see its entry in the Lexicon. For a summary of how the COHERENT system manages printers, see the Lexicon entry for **printer**.

nroff Limitations

Because **nroff** is a text-formatting language rather than a text-formatter *per se,* it makes no assumptions about how you want to lay out your page. It does not automatically leave margins at the top and bottom of pages; it does not automatically number pages; it does not automatically format paragraphs. You must use or create a set of formatting commands, called *macros,* to generate these features. This tutorial will teach you how to write macros that can solve nearly every conceivable formatting problem. As you have seen, too, your copy of COHERENT comes with a set of predefined macros, the **-ms** macro package.

The ms Macro Package

A macro package called **-ms** is included with your copy of **nroff**. It provides macros to format paragraphs, produce headers and footers (the areas at the top and bottom of pages, respectively), and perform most other page-formatting tasks. **-ms** is easy to use. The command

```
nroff -ms
```

tells **nroff** to accept input from your keyboard, process it using the **-ms** macro package, and print the output on your screen. The command

```
nroff -ms script.r
```

tells **nroff** to process **script.r** with the **-ms** package and print the output on your terminal; while the command

```
nroff -ms script.r >target
```

redirects the output of **nroff** into the file **target**; and

```
nroff -ms script.r | lpr
```

prints the output on the line printer.

Working with the **-ms** macro package is a good way to gain confidence in working with **nroff** commands. Soon you will learn the correct way to encode **nroff** commands in your scripts.

Using this Tutorial

The only way to learn about **nroff** is to use it. You should type all the examples in this tutorial into your computer and observe how they work. You should also alter the example and examine how your changes affect what **nroff** produces. Don't hesitate to experiment! You can learn more from analyzing why something unexpected happens than you can from simply copying an example that works as you were told it would.

The first section describes how to use **nroff** with the **-ms** macro package. The second section describes how to perform sophisticated formatting. For most users, this chapter contains all the information they need to know.

The rest of the tutorial describes how **nroff** actually works with the input text to produce its output. This will teach you how to write your own **nroff** macros for your special word processing needs.

The ms Macro Package

As explained above, **nroff** is the text formatter for COHERENT. You give **nroff** a *script* — that is, text interspersed with commands that control its processing; **nroff**, in turn, formats your text in the manner dictated by your commands.

nroff's most outstanding feature is its flexibility: you can control line length, page offset, page length, paragraph format, beginning- and end-of-page format, and every other aspect of formatting a document.

nroff has built into it a set of basic commands, called *primitives*, that are used to control formatting. A basic formatting function might require several primitives. For example, formatting a new paragraph requires one primitive to force the printing of the fragment of a line left at the end of the previous paragraph; another primitive to skip a blank line; and a third primitive to indent the first line of the new paragraph. If you were to type directly into your script all the primitives required to control every feature of your document, formatting would be a very difficult task, and mistakes would be common.

Fortunately, another feature of **nroff** makes it easier for you to prepare input: **nroff** allows you to bundle together a group of primitives and give the bundle its own name. Such a bundle is called a *macro.* Whenever you want all the commands in that bundle to be executed, you simply insert the name of the macro into the text. For example, you might group the primitives needed to format a paragraph, and call that bundle **PP**. Then, instead of retyping the primitives, all you need to do is insert the command **.PP** before the start of a paragraph.

-ms is a package of macros that are ready for you to use. When you include the option **-ms** on the **nroff** command line, **nroff** automatically uses the the macros that have been defined in the **-ms** package. These macros will take care of setting line length and page length, numbering pages, formatting paragraphs, and all other formatting tasks. You do not need to know how **nroff**'s primitives are used in the macros; you only need to know the names of the macros and what they do, so that you can insert them correctly into your text.

Using the **-ms** package is a good way to become accustomed to preparing input for **nroff,** so that the features of the primitives will not seem so alien when you eventually choose to work with them. When you become familiar with **nroff**, you may wish to your own macro packages, to handle the unique requirements of different types of documents. For now, however, you will find that the **-ms** package will get you up and running with **nroff**.

Text and Commands

nroff input includes both *text* and *commands.* The commands control the processing of the text. **nroff** distinguishes between text and commands by looking at the first character of each input line. If that character is a period or an apostrophe, the line is a *command*; otherwise, it is *text*.

Earlier in this tutorial, you used the **-ms** package to format a text file that had already been prepared for you. To become more accustomed to using **nroff,** try entering the following text into a file that can be formatted later. Use a text editor (either **ed** or MicroEMACS) to create a file named **script2.r** that contains the following text. It is important for this exercise that you break up the lines as they are shown here:

```
London.  Michaelmas Term lately over,
and the Lord Chancellor sitting in
Lincoln’s Inn Hall.  Implacable November weather.
As much mud in the streets, as if the waters
had but newly retired from the face of the
earth, and it would not be wonderful to meet
a Megalosaurus, forty feet long or so, waddling
like an elephantine lizard up Holborn Hill.
```

Note that this file contains no commands; every line is a text line. Process the file with the command:

```
nroff script2.r | more
```

The output is piped to **more** so that it will not all rush past your screen. **nroff** will process the text, and in a moment you will see the following:

```
London.  Michaelmas Term lately over, and the Lord Chancellor sitting in Lincoln’s Inn Hall.
Implacable November weather.  As much mud in the streets, as if the waters had but newly retired
from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long
or so, waddling like an elephantine lizard up Holborn Hill.
```

When you see this example, the spacing will be different; the spacing for the examples in this tutorial is adjusted to conform to the rest of the tutorial text. Notice that **nroff** automatically adjusts the spacing between words to justify the right margin, even though the input text has a ragged right margin. Each output line contains 65 characters, and each output page contains 66 lines.

Now try processing **script.r** again, this time with the **-ms** macro package. Type

```
nroff -ms script.r | more
```

As you can see, **nroff** again adjusted the spacing to keep a strict right margin. Each line was indented with ten leading spaces, followed by 65 characters of text. The pages output by both the **nroff** command and the **nroff -ms** command both contain 66 lines, but the page built with the **-ms** package left blank lines at the top of the page and printed the page number in a blank space at the bottom of the page. When **nroff** constructs its output, it assumes that your printer prints ten characters per inch (Pica, or 10-pitch spacing) and six lines per inch. Given these assumptions, each page of output from **nroff -ms** fits onto an 8.5 by 11 inch page, with an inch of blank space at the top, at the bottom, and on each side.

As this example shows, **nroff** adjusts the spacing between words to keep a strict right margin. When you type in the text, don’t worry about the right margin. You must, however, keep a strict left margin, because when **nroff** encounters a line of text that begins with blank spaces, it breaks the line it was working on and begins a new, indented line.

Also, do not hyphenate words; if you do, **nroff** treats each part as a separate “word” (the first ending with the hyphen character), rather than keeping them joined, as you want.

nroff normally interprets as a command every line that begins with a *period* or an *apostrophe*. However, to include an initial apostrophe or period as a literal part of your document, you must place the characters **\&** before the period or apostrophe.

The remainder of this will show you how to use commands in input text to change the appearance of the output. You can control many aspects of the printed document simply by including the appropriate commands within your text.

Command Names

The name of every **nroff** primitive consists of two lower-case letters. Some commands can also include additional information, or *arguments*. For example, **.sp** is the command to leave vertical space between output lines. The command line

```
.sp
```

leaves one space, whereas

```
.sp 2
```

leaves two spaces. The information that follows the command name on the command line is an argument. Each macro defined in the **-ms** macro package is named with one or two upper-case letters. For example, **.PP** is the name of the macro that begins a new paragraph.

Paragraphs

Every time you want to begin a new paragraph, enter the *paragraph* command **.PP;** that is, place the command line **.PP** in the text. To test this macro, enter the following text under the name **script3.r**:

```
.PP
It is a truth universally acknowledged,
that a single man in possession of a good fortune,
must be in want of a wife.
.PP
However little known the feelings or views of such
a man may be on first entering a neighbourhood, the
truth is so well fixed in the minds of the surrounding
families, that he is considered as the rightful
property of some one or the other of their daughters.
```

When you process this text with the command

```
nroff -ms script3.r | more
```

the result resembles the following:

```
            It is a truth universally acknowledged, that a single man in possession of a good
fortune, must be in want of a wife.

            However little known the feelings or views of such a man may be on first entering a
neighbourhood, the truth is so well fixed in the minds of the surrounding families, that he is
considered as the rightful property of some one or the other of their daughters.
```

As the output shows, the **.PP** command inserts a blank line before beginning a new paragraph, and indents the first line of the new paragraph by half an inch.

The **-ms** package also provides another paragraph format: the **.IP** command. This macro creates an *indented paragraph.* The **.PP** macro indents only the first line of each paragraph; however, **.IP** indents every line *except* the first. For example,

```
.IP
This is an indented paragraph.
All the lines are indented by
the same amount.
.PP
This is a normal paragraph.
nroff indents the first line
but does not indent the following lines.
```

gives the output

```
        This is an indented paragraph.  All the lines are indented by the same amount.

        This is a normal paragraph.  nroff indents the first line but does not indent the
following lines.
```

Several options are available for the basic **.IP** macro. You can add two *arguments* to it. **nroff** interprets the first argument after the **.IP** as a *tag* to the paragraph, and it interprets the second argument as the amount of indentation you want. For example,

```
.IP A. 8
This is the first line of text.
nroff indents the following lines by the same
amount as the first.
The indent is eight spaces.
The paragraph includes a tag in the indent.
```

produces

```
A.      This is the first line of text.  nroff indents the following lines by the same amount as
        the first.  The indent is eight spaces.  The paragraph includes a tag in the indent.
```

You must make sure the indent leaves enough spaces for the tag. If the tag contains blank spaces, enclose it within quotation marks. To see how this works, enter the following script under the title **script4.r**:

```
.IP "King Lear:" 16
Is man no more than this?
Consider him well.
Thou owest the worm no silk,
the beast no hide,
the sheep no wool,
the cat no perfume...
Unaccommodated man is no more
but such a poor, bare, forked
animal as thou art.
```

When processed with the command

```
nroff -ms script4.r >script4.p
```

you see:

```
King Lear:     Is man no more than this?  Consider him well.  Thou owest the worm no silk, the
               beast no hide, the sheep no wool, the cat no perfume...  Unaccommodated man is no
               more but such a poor, bare, forked animal as thou art.
```

As this example shows, this form of the **.IP** macro can be used to format the script for a play.

If you do not want a tag, but merely wish to set the indentation to something other than the default setting of five spaces, then use a pair of quotation marks with nothing between them for the first field:

```
.IP "" 8
```

If you forget the quotation marks, you will not get what you expect: **nroff** will interpret '8' as a tag and use the normal indentation of five spaces.

Once you set the amount of indentation, the new indentation stays in effect until you change it again. For example, if you format a paragraph with

```
.IP "" 8
```

and follow it with another paragraph that begins with **.IP**, **nroff** will also indent the second paragraph by eight spaces. The indentation will remain in effect until you explicitly change it — for example, by beginning a paragraph with

```
.IP "" 6
```

which resets the indent to six spaces.

Normally, **nroff** measures the paragraph indentation from the left margin. Another variation of **IP** allows you to measure the indentation of a new indented paragraph from the left-hand edge of a previous indented paragraph, thus producing *relative indentation*. To do this, enclose the new paragraph between the macros **RS** and **RE** (for **r**elative indent **s**tart and **r**elative indent **e**nd). Copy the following script into the file **script5.r**:

```
.IP
And it came to pass in an eveningtide,
that David arose from off his bed ...
and from the roof he saw a woman washing
herself; and the woman was very beautiful
to look upon.  And David sent and enquired
after the woman.  And one said,
.RS
.IP
Is not this Bathsheba, the daughter of Eliam,
the wife of Uriah the Hittite?
.RE
.IP
And David sent messengers and took her; and
she came in unto him, and he ...
and she returned unto her house.
```

When processed through **nroff** with the command

```
nroff -ms script5.r >script5.p
```

the output resembles the following:

```
And it came to pass in an eveningtide, that David arose from off his bed ... and from the roof he
saw a woman washing herself; and the woman was very beautiful to look upon. And David sent and
enquired after the woman. And one said,

        Is not this Bathsheba, the daughter of Eliam, the wife of Uriah the Hittite?

And David sent messengers and took her; and she came in unto him, and he ... and she returned unto
her house.
```

You can include any number of indented paragraphs between **.RS** and **.RE**. Also, you can specify tags and different indents just as for ordinary indented paragraphs. You can even nest **.RS** and **.RE** pairs inside each other to produce multiple relative indents. Just remember that an **.RS** must always be balanced by an **.RE**. Type the following into the file **script6.r** to see how **nroff** handles nested flashbacks:

```
.IP
In England during World War II, a captain tells the
story of his Free French bomber squadron.
.RS
.IP
In the early days of the war, a French ship picks up
five men adrift in a small boat.  One tells of their
life on Devil's Island.
.RS
.IP
A convict tells others of his past.
.RS
.IP
Publication of anti-Nazi material leads to arrest on
false charges.
.RE
.IP
The convicts escape to help France in the war.
.RE
.IP
When France surrenders, the crew overpowers pro-Vichy
officers and heads for England instead of Marseilles.
.RE
.IP
The captain concludes his story as the bombers return
from a mission.
```

When you process this file with the **-ms** package, the output file **script6.p** should resemble the following:

```
In England during World War II, a captain tells the story of his Free French bomber squadron.

        In the early days of the war, a French ship picks up five men adrift in a small boat. One
        tells of their life on Devil's Island.

                A convict tells others of his past.

                        Publication of anti-Nazi material leads to arrest on false charges.

                The convicts escape to help France in the war.

        When France surrenders, the crew overpowers pro-Vichy officers and heads for England
        instead of Marseilles.

The captain concludes his story as the bombers return from a mission.
```

As you can see, each **.RE** command peels away the current layer of indentation and moves you into the previous one. To return to an even earlier level, you must input the appropriate number of **.RE** commands before you begin a paragraph.

A third type of paragraph is the *quoted* paragraph. It produces a paragraph that is indented on both on the right side and on the left side, in order to set off a quotation from the surrounding text. To produce such a paragraph, precede it with the **.QS** macro and follow it with the **.QE** macro. To break the quotation into different sections, insert a blank line in the text before each line that you want to begin a new section. For example, type the following example as **script7.r**:

```
Form of Tender of Rescue from Strange Young Gentleman
to Strange Young Lady at a Fire.
.QS
Although through the fiat of a cruel fate, I have been
debarred the gracious privilege of your acquaintance,
permit me, Miss [here insert name, if known], the
inestimable honor of offering you the aid of a true
and loyal arm against the fiery doom which now
o'ershadows you with its crimson wing.  [This form
to be memorized, and practiced in private.]
.QE
Should she accept, the young gentleman should offer
his arm - bowing, and observing "Permit me" -
and so escort her to the fire escape and deposit
her in it.
```

After processing with the **-ms** package, the output file **script7.p** should resemble the following:

```
Form of Tender of Rescue from Strange Young Gentleman to Strange Young Lady at a Fire.

        Although through the fiat of a cruel fate, I have been debarred the gracious
        privilege of your acquaintance, permit me, Miss [here insert name, if known],
        the inestimable honor of offering you the aid of a true and loyal arm against the
        fiery doom which now o'ershadows you with its crimson wing.  [This form to be
        memorized, and practiced in private.]

Should she accept, the young gentleman should offer his arm - bowing, and observing "Permit me" -
and so escort her to the fire escape and deposit her in it.
```

Section Headings

The *section heading* macro **.SH** prints a heading or title. For example:

```
.SH
Section Headings
```

The heading may be more than one line long; consequently, you should follow a section heading with a **.PP** or an **.IP** macro. **nroff** leaves a blank line before the heading and prints the heading flush with the left margin in **boldface** type, as described below in the section on Fonts.

The *numbered heading* macro **.NH** produces consecutively numbered section headings. For example:

```
.NH
Guess What's Coming to Dinner?
.NH
Guess Why I Won't be There?
```

produces

```
1.   Guess What's Coming to Dinner?

2.   Guess Why I Won't Be There?
```

You can number subsection headings by entering a number from two to five to the **.NH** macro. The number indicates the level of section headings; for example, **.NH 2** numbers subsection headings, **.NH 3** numbers sub-subsection headings. For example:

```
.NH
Guess What’s Coming to Dinner?
.NH 2
Guess What it Looks Like?
.NH 3
Teeth Like That Might Frighten the Children!
.NH 2
What Does it Eat?
.NH
Guess Why I Won’t be There?
```

produces:

```
1.   Guess What’s Coming to Dinner?

1.1   Guess What it Looks Like?

1.1.1   Teeth Like That Might Frighten the Children!

1.2   What Does it Eat?

2.   Guess Why I Won’t be There?
```

The number on the **.NH** command line is *not* the number that appears before the heading; instead, it controls how many “parts” appear in the number. For example, **.NH 3** produces a three-part number, such as **2.5.3**, whereas **.NH 4** produces a four-part number, such as **7.4.5.2**.

You can reset the entire numbering scheme by using the command **NH 0**; for example,

```
.NH 0
Through The Mandelbrot Set With Rod and Gun
```

produces

```
1.   Through The Mandelbrot Set With Rod and Gun
```

with numbering starting at one.

Title Page

If you want your output to begin with a title page, begin the input with the following.

```
.TL
Title of Document (may be more than one line)
.AU
Name(s) of Author(s) (may be more than one line)
.AI
Institution(s) of Author(s)
.AB
Abstract (line length 5.5 inches)
.AE
```

The **.TL** macro indicates the *title,* the **.AU** macro indicates the *author,* the **.AI** macro indicates the *author's institution,* and the **.AB** macro precedes the *abstract.* The **.AE** macro, for **a**bstract **e**nd, marks the end of the abstract. If you do not want some of these headings to appear, simply omit the relevant macros. Begin the body of the document immediately after the **.AE** macro. The body must begin with a formatting command, such as **.PP** or **.SH.**

Note that the *end abstract* macro **.AE** also prints today’s date automatically. To do so, **nroff** reads the date as encoded in COHERENT. Before you use these macros, be sure that you have set the correct date in COHERENT.

To see how these macros work, type the following script into file **script8.r**:

```
.TL
Tickling in the Therapy of
von Muenchausen’s Syndrome
.AU
P. R. Sanserif
.AI
The Department of Parapsychology
The University of Southern North Dakota
```

```
at Hoople
.AB
Study of 150 subjects (75 men and 76 women)
indicated that hard tickling may prove beneficial
to patients with von Muenchausen’s syndrome.
Applications for a seven-figure grant have been
made to continue research in this area.
.AE
.PP
Due to complications in our experiment, this paper
has now been withdrawn.
```

After processing with the **-ms** macro package, you will see that in the outputfile **script8.p**, **nroff** placed the text on the same page as the title information. You may or may not want this to happen. If you do not, one solution is to insert two additional commands between the **.AE** macro and the body of your text:

```
.PP
.bp
```

Headers and Footers

The *header* macro controls the format of the top of each page. It automatically skips one inch at the top of the page. The *footer* macro controls the format of the bottom of each page. It stops printing text one inch above the bottom of the page, and prints the page number.

It is easy to print either a page header or a page footer. Both the page header and the page footer are three-part titles: **nroff** prints the first part on the left side of the page, the second part in the middle, and the third part on the right side of the page. The parts of the header title are named:

LT: left, top
CT: center, top
RT: right, top

and the parts of the footer title are named:

LF: left, footer
CF: center, footer
RF: right, footer

These parts are called *strings.* A later section of this tutorial describes strings in detail. Normally, these strings are undefined, except for **CF**, which prints the current page number; therefore, the header macro normally prints nothing, and the footer macro prints only the page number in the center of the block of space at the bottom of each page. However, you can set any portion of the header or footer to print what you like. To set the left portion of the header, for example, type the following:

```
.ds LT "Walnuts in History"
```

Note that you do *not* type a period before the **LT**. After you define **LT** in this fashion, **nroff** will print

```
Walnuts in History
```

at the top of each page on the left-hand side. If you want the date to appear on the right-hand side of the header, type:

```
.ds RT "\*(Ds"
```

The string **Ds** is automatically initialized to today’s date, as set on your COHERENT system. A later section of this tutorial will present strings in detail. For now, all you need to know is that whenever you want **nroff** to insert today’s date into your script automatically, just type the entry ***(Ds**. This entry does *not* have to be at the beginning of a line to work.

Use the same procedure to define the strings in the footer title. If you want something other than the page number to appear in the position allocated to **CF**, use the **.ds** primitive to redefine **CF**. If you want nothing to appear there, type

```
.ds CF ""
```

Wherever you want the current page number to appear in the header or footer, use the symbol ‘%’. For example, if you want the page number to appear in the upper right-hand corner of each page, type

```
.ds RT "Page %"
```

Be sure to type in all of the macros to define headers and footers *before* you begin to type in your text. Otherwise, your headers and footers will not appear on the first page of the formatted output.

To see how this works, try editing the file **script1.r**. At the top, insert the macro

```
.ds RT "\*(Ds"
```

and reprocess the file using the **-ms** macro package. Each output page should have today's date written in the upper right-hand corner.

Fonts

nroff normally prints ordinary, or "Roman", characters. In addition, **nroff** can print **boldface** and *italic* characters. Each of the three styles of type — Roman, boldface, and italic — is called a *font*, in keeping with typesetting terminology.

nroff prints each boldface and italic character by generating a special three-character output sequence. It prints the boldface character **c**, for example, by printing a 'c', then the backspace character **<ctrl-H>**, and then another 'c'. This sequence emaphasizes 'c' by forcing your printer to print it twice. **nroff** represents an italic character *c* with the underscore character '_', followed by the backspace character **<ctrl-H>,** followed by 'c'.

Because of these special representations, the appearance of **nroff** boldface and italic fonts depends on the device on which you see the output. On your terminal, the **<ctrl-H>** backspaces the cursor, and the third character of each sequence replaces the first; therefore, boldface and italic characters appear the same as Roman characters. On a printer, the appearance depends on the characteristics of the printer. The COHERENT system provides programs to print boldface and italic characters appropriately on certain devices.

The **-ms** macro package includes three commands for easy printing in specific fonts: the *boldface* command **.B**, the *italic* command **.I**, and the *Roman* command **.R**. To print a single word in boldface, do the following:

```
The last word is printed in
.B boldface.
```

Likewise for italics:

```
The last word is printed in
.I italics.
```

These example printed a word in a different font. You can print several words in a different font by enclosing the words within quotation marks on the command line:

```
This sentence ends with
.B "three bold words".
```

You can also switch fonts by using one of the font commands with nothing after it on the command line. For example,

```
.B
This entire sentence is printed in boldface.
.R
```

or

```
.I
This entire sentence is printed in italics.
.R
```

In these examples, the Roman font command **.R** is needed to return to the normal font after completing the boldface or italic text.

On rare occasions, you might want different parts of one word to be in different fonts. You cannot use the **-ms** macros to produce mixed-font words directly. A later section of this tutorial gives additional information about **nroff** fonts. As explained there, the input

```
This manual describes \fBnroff\fR's powerful features.
```

produces the output:

This manual describes **nroff**'s powerful features.

The word **nroff** is boldface but the following apostrophe and ‘s’ are Roman.

Special Characters

A few characters have special meaning to **nroff.** You should be aware of these characters if you want **nroff** to process your text properly.

As mentioned earlier, the period and the apostrophe introduce **nroff** command lines. Each is a special character if it is the *first non-space character* on an input line. If you wish to use a period or an apostrophe at the start of an input line simply as part of your text, you must precede it with a backslash and ampersand “**\&**”. For example, the input

```
The footnote command
.DS
\&.FT
.DE
generates footnotes for you automatically.
```

produces the output

```
The footnote command

.FT

generates footnotes for you automatically.
```

Neither the period nor the apostrophe is a special character unless it is the first non-space character on a line.

The most important special character for **nroff** is the backslash ‘\’. It changes the meaning of the following character or characters. If you simply want a backslash to appear as part of your text, you must follow it with the letter ‘e’; that is, use “\e” in your input to have ‘\’ appear in your output. Later sections of this tutorial describe other special uses for backslash.

Footnotes

You can place footnotes between the *footnote start* command **.FS** and the *footnote end* command **.FE,** as in the following example:

```
.FS
*MicroKVETCH Electronic Nag is a
copyrighted trademark of Caveat Emptor
Software, Inc.
.FE
```

You should insert each footnote into your text where the reference to it occurs; **nroff** will see to it that the footnote appears at the bottom of the correct page. Footnotes should be inserted as follows:

```
The notion that we have been visited
by visitors from outer space may seem
outlandish(1)
.FS
1. Raucus J, O’Hooligan R: "Viruses
from Venus?" \fIJ Earth Med Assoc\fR,
1985;36:412-414.
.FE
but reason compels us to exclude no ...
```

The journal article cited in the footnote will appear at the bottom of the page, with the journal name in italics.

Displays and Keeps

A *display* is a portion of text, such as a graph or a table, that should appear in the output exactly as it is typed in the input. **nroff** normally alters the spacings between elements in your text, which, of course, would destroy the appearance of a display. Therefore, **nroff** has macros to tell it that a portion of text is a display, and so not to alter spacings between elements or split it between two pages. These macros are the *display start* macro **.DS** and the *display end* macro **.DE**. You should place your display between these macros, as follows:

```
.DS
The text of the display goes here,
exactly
as
you
want
it
to appear in the output.
.DE
```

The **.DS** macro comes in three varieties. The *display start centered* macro **.DS C** centers every line of your display. Because **nroff** centers each line individually, both right and left margins are ragged. The *display start block-centered* macro **.DS B** takes the entire display at once and centers it. You can think of this as simply shifting the display to the right by an appropriate amount. The *display start indented* macro **.DS I** indents the entire display by half an inch.

If your display is longer than one page, do not use **.DS** or any of its variants. Instead, begin the display with one of the following.

The *centered display* macro **.CD** centers each line of the display. The *block-centered display* macro **.BD** considers the entire display as a block and centers it. The *left display* macro **.LD** performs no indenting or centering, but simply begins each line at the left margin. Finally, the *indented display* macro **.ID** indents each line by half an inch. If you begin the display with one of these macros, do *not* end it with **.DE**; rather, just type **.PP** or **.SH** or whatever other macro is needed at that point.

To see how displays work, type the following into the file **script9.r** and process it with the **-ms** macro package:

```
.PP
.DS C
Tyger! Tyger! burning bright
In the forests of the night,
What immortal hand or eye
Could frame thy fearful symmetry?
Burma Shave
.DE
```

When the output file **script9.p** is read, the results will appear as follows:

```
           Tyger! Tyger! burning bright
            In the forests of the night,
              What immortal hand or eye
         Could frame thy fearful symmetry?
                    Burma Shave
```

You must remember one important fact when you use display macros: the normal length of output lines is 6.5 inches, but if the display contains lines longer than this **nroff** simply prints them as they are. If a line is too long to fit onto the page, what occurs afterwards depends upon the output device. If you are displaying the output on the screen, the text will be displayed as far as possible to the right, then the remainder will be wrapped around onto the next line, without indentation. On most printers, however, the chunk of text that extends past the right margin will simply be lopped off and thrown away. In any event, the effect is usually quite unsightly. The only restriction on what you can safely put in a display, then, is that lines should be no longer than 6.5 inches. If you are using an indented display, lines should be no longer than six inches.

A *keep* is a display macro: you put text between the *keep start* macro **.KS** and the *keep end* macro **.KE** when you want it all kept on the same page. If you put a block of text between these macros that proves to be longer than one page, **nroff** moves the excess text onto a new page.

The major difference between the keep and the display is that normal processing occurs in the keep: **nroff** adjusts spacings between words, hyphenates words, justifies lines, and performs all other formatting tasks, just as it normally does.

Other Commands

Several of **nroff**'s primitives can be used with the **-ms** macro package. The primitive

.sp *N*

skips *N* lines on the output page; for example, **.sp 4** skips four lines.

The *begin page* primitive **.bp** tells **nroff** to begin a new page, no matter where it is on the current page.

The remaining sections of this tutorial provide more information about these other **nroff** primitives.

Introducing nroff's Primitives

The rest of this tutorial describe **nroff**'s *basic commands* — the commands that are "built into" **nroff**, and from which macros are assembled. These basic commands, or *primitives,* form **nroff**'s text formatting language. Once you have mastered the primitives, you will be able to write macros to control automatically even the most difficult text formatting tasks.

The rest of this tutorial includes a number of exercises. You should type them into your system and execute them as described in the tutorial; this will greatly increase the rate at which you master **nroff**. None of the following examples should be processed with the **-ms** macro package; the purpose of this portion of the tutorial is to teach you how to create you own text processing routines, rather than how to use ones that have already been written.

Page Format

When deciding how to process text, you must first decide how to position the text on the printed page. You must control line length, left and right margins, page offset (i.e., how far from the left edge of the page each line begins), and page length. Controlling these functions is quite easy with the appropriate **nroff** commands.

The *line length* primitive **.ll** controls the line length; and the *page offset* command **.po** controls the page offset. If you are writing an **nroff** script, you should include these commands before the beginning of your text, so that **nroff** can put them into effect immediately. The following example uses a line length of three inches and a page offset of two inches. Type this into your system under the name **ex1.r**. Note, by the way, that the text to the right of the characters '\"' is a *comment*, and there is no need for you to type it into your system:

```
.ll 3i         \" set line length
.po 2i         \" set page offset
Along outside of the front fence ran the country
road, dusty in the summertime, and a good place for
snakes -- they liked to lie in it and sun themselves;
when they were rattlesnakes or puff adders, we killed
them; when they were black snakes, or racers, or belonged
to the fabled "hoop" breed, we fled, without shame; when
they were "house snakes", or "garters", we carried them
home and put them in Aunt Patsy's work basket for a
surprise; for she was prejudiced against snakes, and
always when she took the basket in her lap and they
began to climb out of it it disordered her mind.
```

Process this script by typing the command

```
nroff ex1.r >ex1.p
```

From this point on, you should *not* use the **-ms** macro package with your **nroff** examples. When you display the output stored in the file **ex1.p**, you will see that the length of each line is three inches, and each line begins two inches from the left-hand margin.

As you noticed, line length and page offset were set in *inches*. **nroff** output can be controlled using a number of different units of measurements, including inches, number of characters, or lines, or *machine units*. A following section discusses **nroff** units of measurement in detail.

As noted above, this example contains two *comments*. **nroff** ignores any text that appears on a line after "\"". You should use comments, for the benefit of anyone who must read your **nroff** script (including yourself). The above example used the comments

```
\" set line length
\" set page offset
```

to help you understand the **.ll** and **.po** commands. Judicious comments can make a complex script much easier to understand.

Breaks

Before you look at the *break* primitive **.br**, it is helpful to examine how **nroff** constructs a finished line of output. Suppose, for example, that you tell **nroff** that you want each output line to be five inches long. **nroff** takes your input one word at a time, and attempts to squeeze that word into the space that has not yet been taken up in the line. When **nroff** finally picks up a word that is too large to fit into the amount of space left in the line, it either puts the word aside entirely, or hyphenates the word and places the hyphenated portion into the line. **nroff** then inserts extra blank spaces between the words to justify the line. The *break* primitive **.br**, however, tells **nroff** to print whatever words have already been put into the line, even if they do not form a complete line, and without performing right justification.

The idea of a break might seem strange at first, but you are familiar with a simple example: the end of a paragraph. You do not want the start of a new paragraph to be on the same line as the end of the previous paragraph: you want to print the end of the previous paragraph whether or not it fills a complete line; and you want to begin the new paragraph on a new line. As you will learn later, some **nroff** commands cause breaks automatically; you should be aware of this when you use them.

Fill and Adjust Modes

Two terms describe how **nroff** processes your input to create its output: *filling*, and *adjusting* or *justifying*. Unless you order it not to, **nroff** operates in the *fill* and *adjust* modes. The *no-fill* primitive **.nf** tells **nroff** to stop using fill mode. The *fill* primitive **.fi** tells it to resume using the fill mode. In a similar way, the *adjust* primitive **.ad** tells **nroff** to use adjust mode, whereas the *no adjust* primitive **.na** tells it to use no-adjust mode.

As mentioned above, **nroff** by default is in both fill mode and adjust mode, so you do not need to begin your script with **.fi** and **.ad** if you want **nroff** to fill and adjust your text. However, if you turn off filling and adjusting by using the **.nf** and **.na** commands, you must use the **.fi** and **.ad** commands to turn filling and adjusting back on.

When you use **.nf** to turn off fill mode, **nroff** no longer tries to fill lines to a fixed line length. It prints each line of input text exactly as received. However, a sufficiently long line of text would run off the right-hand edge of the page if **nroff** were to print it as entered. If the input line cannot fit on one line, **nroff** prints as much as it can fit on one line, then breaks the line and prints the rest on the next line with no page offset.

In adjust mode, **nroff** inserts extra spaces between words to justify lines of text, as described above. When **nroff** is in no-fill mode, it is automatically in no-adjust mode: with no fixed line length, there is no need to insert extra spaces. *Moral:* you can fill without adjusting, but you cannot adjust without filling.

If you request filling but not adjusting, **nroff** fills the output line as described earlier, but does not insert extra spaces between words; that is, it does not try to keep an even right margin. Every output line either is shorter than the line length you specified, or exactly as long.

The **.ad** primitive includes several options. If you use the command **.ad** without an argument, **nroff** keeps strict left and right margins. The primitive **.ad l** justifies the left margin only; **.ad r** justifies the right margin only; and **.ad b** justifies both margins (this, of course, is the default). Finally, **.ad c** centers output lines while keeping their lengths less than or equal to the line length, as set with the **.ll** command.

Remember that **nroff** ignores adjustment requests if you are in no-fill mode. If **nroff** is in fill mode and you request any variety of adjustment, it adjusts accordingly until you issue either a no-fill or a no-adjust command. If you give a no-fill command, only a fill command restores adjustment; any plea for a different kind of adjustment is ignored while **nroff** is in no-fill mode.

To see how this works, type the following script under the name **ex2.r**, and process it as above:

```
.ll 6.75i
.sp          \" space
When we were alone, I introduced the subject
of death, and endeavored to maintain that the fear
of it might be got over.  I told [Johnson] that
David Hume said to me, he was no more uneasy to
think that he should not be after this life, than
that he had not been before he began to exist.
.sp
.na          \"no adjust
JOHNSON:  "Sir, if he really thinks so,
his perceptions are disturbed;
he is mad: if he does not think so, he
```

```
lies .... When he dies, he at
least gives up all he has."
.sp
.ad r          \"right-adjust
BOSWELL:  "Foote, sir, told me that
he was not afraid to die."
.sp
.nf            \"no-fill
JOHNSON:  "It is not true, sir.
Hold a pistol to Foote's
breast or to Hume's breast,
and threaten to kill them,
and you'll see how they behave."
.sp
.fi            \"fill
BOSWELL:  "But may we not fortify our minds for
the approach of death?"
.sp
JOHNSON:  "No, sir, let it alone.  It matters not
how a man
dies, but how he lives.  The act of dying is not of
importance, it lasts so short a time .... A man
knows it must be so, and submits.
It will do him no good to whine."
```

When you process this input with **nroff,** your output should look like this:

```
When we were alone, I introduced the subject of death, and endeavored to maintain that the fear
of it might be got over. I told [Johnson] that David Hume said to me, he was no more uneasy to
think that he should not be after this life, than that he had not been before he began to exist.

JOHNSON: "Sir, if he really thinks so, his perceptions are disturbed; he is mad: if he does not
think so, he lies .... When he dies, he at least gives up all he has."

BOSWELL: "Foote, sir, told me that he was not afraid to die."

JOHNSON:  "It is not true, sir.
Hold a pistol to Foote's
breast or to Hume's breast,
and threaten to kill them,
and you'll see how they behave."

BOSWELL: "But may we not fortify our minds for the approach of death?"

     JOHNSON: "No, sir, let it alone.  It matters not how a man dies, but how he lives.  The act of
  dying is not of importance, it lasts so short a time .... A man knows it must be so, and submits.
It will do him no good to whine."
```

After the **.na** primitive, **nroff** fills but does not adjust the second paragraph. After **.ad r**, it fills and right adjusts the third paragraph. After **.nf**, it neither fills nor adjusts the fourth paragraphs. Finally, after **.fi**, it fills the fifth and sixth paragraphs and uses the **.ad r** adjust option that was in effect previously.

Under certain extreme conditions, **nroff** cannot adjust a line even though it is in adjust mode. If, for example, you specified a line length of one inch, a seven-letter or eight-letter word would then take up most of a line. When such a word was then followed by a word that could not fit into the line after it, **nroff** would begin a new line with the second word rather than violate the right margin by inserting the into the line. When a line has only one word in it, **nroff** obviously cannot adjust the line by inserting extra spaces between words; therefore, the right margin is left uneven, as though **nroff** were in no-adjust mode.

Defining Paragraphs

What happens if you copy text from several pages of a book into a file without adding any formatting commands, and then process the file with **nroff**? There is no page offset, because **nroff**'s default page-offset setting is zero; and the processed lines are set to the default length of 6.5 inches (65 Pica characters).

More interesting things happen with paragraphs. Suppose you skip a line between paragraphs and begin each paragraph by indenting five spaces. The blank line in the input text causes a break, and forces **nroff** to print a blank line. The last line of each paragraph is unadjusted, and a blank line appears before the next paragraph. Initial blank spaces in a line of input also cause a break. In this example, the breaks caused by initial blank spaces at the beginning of each paragraph do nothing, because the preceding blank line forces out the last line of the preceding paragraph. **nroff** always considers initial blank spaces in a line to be significant, and preserves them in the output.

To see how blank lines and initial spaces affect **nroff**'s output, copy the following example and run it through **nroff:**

```
     Here is a little text so you can see
whether nroff will ignore the initial
indentation
            in this very very long sentence.
Here is a little bit more text.

     And here is something to mimic
the beginning of a new paragraph.
```

The output should look like this:

```
Here is a little text so you can see whether nroff will ignore the initial indentation
    in this very very long sentence. Here is a little bit more text.

And here is something to mimic the beginning of a new paragraph.
```

Instead of leaving a blank line in the text, you could use the *space* primitive **.sp 1,** which causes a break and inserts one blank line into the output. In a similar way, **.sp 5** causes a break and inserts five blank lines in the output. Edit the example and replace the blank line with the command line:

```
.sp 1
```

You will see that it has the same effect. You can also use the form **.sp**; **nroff** assumes you want one space if you omit the argument.

Most **nroff** input consists of many paragraphs that contain text, and you probably want each paragraph to have the same format in the output. Rather than formatting each paragraph explicitly, as in this example, you can use the *macro* facility of **nroff** to define a sequence of commands to format a paragraph. Macros are described in detail later in this tutorial.

Centering

The *center* primitive **.ce** centers one or more lines of text. For example, you can center a two-line heading as follows:

```
.ce 2
Heading Printed
In Center of Page
```

If you use the **.ce** command with no argument, **nroff** assumes a default argument of one, and centers only the next line of input. The command **ce 0** cancels any earlier centering command that is in operation.

Tabs

If your **nroff** input includes tables, you may find it convenient to use tabs to separate items in a line of the table. **nroff** recognizes the **<tab>** character and expands it into spaces. If you use tabs to format a table, remember to use no-fill mode; otherwise, **nroff** tries to fill and adjust your output lines.

By default, **nroff** sets a tab stops after every eight characters. You can use the *tab* primitive **.ta** to change the positions of the tab stops. For example,

```
.ta 10 20 30 40 50 60
```

sets tab stops ten characters apart rather than eight. **.ta** can also be used to fix tab stops in inches rather than after a number of characters; for example,

```
.ta 0.8i 2.0i
```

sets tab stops after 0.8 inches and 2.0 inches on the output line. This is quite helpful when you are designing a table.

You can use the *tab character* command **.tc** to change the character **nroff** prints between its current position and the next tab stop. Enter the following text to see how this primitive works:

```
.ta 9 19 29 39
.tc *
.nf
<tab>1<tab>2<tab>3<tab>4
```

The output file, **ex3.p**, should appear as follows:

```
*********1*********2*********3*********4
```

Page Breaks

The *begin page* primitive **.bp** causes a break and forces **nroff** to the next output page. By default, **nroff** assumes a page length of 11 inches (66 lines). You can change the page length with the *page length* command **.pl.** For example,

```
.pl 2i
```

specifies a two-inch page length.

At this point, the question arises about how **nroff** handles top and bottom page margins, number pages, and other aspects of page layout. The answer is it merely keeps track of the current output page number and the current line number on the current output page; designing top and bottom margins, page headers and footers, and other aspects of page layout is up to you.

Can **nroff** execute a set of commands whenever it reaches a certain position on the page? This would solve the problem of producing top and bottom margins, and you would not have to guess where to insert the commands in your script. In fact, you can tell **nroff** to do this, by using *traps.* The next section of this tutorial describes macros and traps and how to use them to format a page.

Macros and Traps

This section presents **nroff** macros: how to write them, how to tell **nroff** to execute them at a give point on every output page, and how to install a macro file under the COHERENT system

As with previous sections, this one uses a number of exercises. Working the exercises will help you master **nroff** quickly. When you format the exercise scripts, do not use the **-ms** option. Also, it is not necessary for you to copy the comments into your system; they are here to help you understand what each **nroff** command does, but they have no effect on how the script executes.

What Is a Macro?

To become familiar with the idea of a *macro*, consider the problem of formatting a paragraph. Whenever you come to a new paragraph, you want **nroff** to skip a line and indent the first line five spaces. Because **nroff** preserves blank lines and initial indentations, you could force **nroff** to break your text into paragraphs simply by inserting a blank line and spaces directly into your text. The same effect, however, can be achieved by inserting following set of **nroff** commands

```
.br             \" break
.sp             \" skip a line
.ti 5           \" indent next line 5 spaces
```

between the end of each paragraph and the start of the next paragraph. You should recognize the first two commands: **.br** causes a break, so that **nroff** prints the last line of the previous paragraph even though it might not be a complete line; **.sp** skips a line before the next paragraph begins. The third command is the *temporary indent* command **.ti**, which tells **nroff** to indent the next line; the number indicates how many spaces to indent. The following exercise, **ex4.r**, demonstrates how this works:

```
.ll 3i          \" line length
.po 3i          \" page offset
```

```
.ti 5             \" indent next line
Adam was human--this explains it all.  He did
not want the apple for the apple’s sake, he
wanted it because it was forbidden.  The mistake
was in not forbidding the serpent; then he would
have eaten the serpent.
.br               \" break
.sp               \" skip a line
.ti 5             \" indent next line
Training is everything.  The peach was once a bitter
almond; cauliflower is nothing but cabbage with a
college education.
.br
.sp
.ti 5
Habit is habit, and not to be flung out of the window
by any man, but coaxed downstairs a step at a time.
```

After you have processed this file, the output file **ex4.p** should resemble the following:

```
         Adam was human--this explains it all. He did not want the apple for the apple’s sake,
he wanted it because it was forbidden. The mistake was in not forbidding the serpent; then he
would have eaten the serpent.

         Training is everything. The peach was once a bitter almond; cauliflower is nothing but
cabbage with a college education.

         Habit is habit, and not to be flung out of the window by any man, but coaxed downstairs a
step at a time.
```

Now, in a small file it would be easy to type all of the **nroff** primitives directly into your input text; however, what if your file is very long, with hundreds of paragraphs? Every time you wanted to begin a paragraph, you would have to include that set of commands within the text. You would save considerable agony if you could bundle these commands together under a common *name*; then you could simply put that *name* into your text whenever you wanted **nroff** to perform these commands, rather than typing the commands themselves over and over again.

As you probably have guessed by now, you can do just that; the set of commands is called a *macro.* The following shows the selections from Pudd’nhead Wilson’s calendar set with a macro called **.PP** that takes care of formatting each paragraph. The following exercise, **ex5.r**, shows how to bundle together the **nroff** primitives for formatting paragraphs into the **.PP** macro:

```
.de PP            \" define the PP macro
.br               \" break the line
.sp               \" insert a blank line
.ti 5             \" indent next line 5 spaces
..                \" two periods ends the macro definition
.PP               \" execute PP macro
Adam was human--this explains it all.  He did
not want the apple for the apple’s sake, he
wanted it because it was forbidden.  The mistake
was in not forbidding the serpent; then he would
have eaten the serpent.
.PP
Training is everything.  The peach was once a bitter
almond; cauliflower is nothing but cabbage with a
college education.
.PP
Habit is habit, and not to be flung out of the window
by any man, but coaxed downstairs a step at a time.
```

As you can see, using a macro can save you a considerable amount of work when you prepare your script.

Introducing Traps

Now, consider the problem of formatting the beginning and ending of each page of output. You could define what are traditionally called *header* and *footer* macros, which contain the commands you want performed at the top and bottom of each page. However, how can you tell **nroff** when to execute these macros? You cannot possibly know where to call these macros in the input text, because you cannot know where any given text line will appear in the output until you have processed it through **nroff**. This problem is solved by using *traps.*

nroff keeps track of its vertical position on each output page. You can set a *trap* that tells **nroff** to execute a macro at a particular vertical position on every page. When a line of output reaches or extends past the position that is specified in your *trap*, **nroff** then executes the commands named in the trap command before processing any more input text.

You can set a trap by using the *when* command **.wh**. For example, if you want **nroff** to call your header macro **.HD** at the very top of each page, the command

```
.wh 0 HD            \" set header trap
```

sets a trap for the macro **.HD** at vertical position 0 (the very top of the page) of every output page. The macro **.HD** will then be executed every time **nroff** begins a new page. To have your footer macro **.FO** execute one inch from the bottom of each page, use the command

```
.wh -1i FO          \" set footer trap
```

The negative number tells **nroff** to measure distance from the *bottom* of the page rather than from the top; the **i** is an abbreviation for inches. (**nroff** recognizes various units of measurement; this will be described in more detail later.)

Headers and Footers

Suppose you want to design the output page by defining the header and footer macros. A simple header macro merely skips an inch of space at the top of each page; a simple footer macro forces printing to stop an inch from the bottom of each page and prints the page number. **nroff** does not print page numbers automatically, but it does automatically keep track of which output page it is on. It stores the page number internally in a *number register* that you can access with the symbol '%'. (A later section gives more information about number registers and how to use them.)

The following gives a simple footer macro that prints the page number:

```
.de FO              \" define footer macro FO
'sp 4v              \" skip four vertical lines (no break)
.tl ''- % -''       \" print hyphen, page number, hyphen
'bp                 \" jump to new page
..                  \" end macro definition
```

There are several points of interest raised by this macro.

First, notice that some commands are preceded with an apostrophe rather than with a period. The use of the apostrophe instead of the period tells **nroff** to suppress the break these commands normally cause. You might run into problems if you define your header macro as follows:

```
.de HD              \" header macro
.sp 1i              \" skip an inch (break)
..
```

You want this to leave a blank space of one inch at the top of each page; however, the **.sp** command causes a break, so that if a word were left over from the last line on the preceding page, **nroff** would print it at the very top of the next page. The effect would be quite unsightly. However, if you use **'sp** instead of **.sp** in the macro, **nroff** suppresses the break and does not print the partial word until *after* it performs the macro commands. The same is true for the footer macro: you do not want anything unplanned to be printed in the blank space at the bottom of the page. You should always be conscious of these considerations when you use commands that cause breaks.

Another new item in the above example is the *title* command **.tl,** which prints a three-part title. A three-part title contains a left part (aligned to the left margin of the page), a center part (centered), and a right part (aligned to the right margin). The command name **.tl** is followed by four apostrophes: **nroff** prints the characters between the first two apostrophes as the left part of the title line, those between the second and third apostrophes as the center part, and those between the third and fourth apostrophes as the right part of the three-part title. If you do not want **nroff** to print anything in one of these positions, simply put nothing between the appropriate pair of quotes.

In the above example, the **.tl** primitive tells **nroff** to print nothing in the left and right portions of the footer title line, but to print the page number in the center. If you want an apostrophe to appear as a part of the title, precede it with the backslash character '\'.

nroff considers the length of the title line to be independent of the length of normal output lines; therefore, you must set it with the *length of title* primitive **.lt** unless you want **nroff** to use the default title length of 6.5 inches. For example, to set the length of the title to five inches, use the command

```
.lt 5i
```

In light of all you now know, you should give Pudd'nhead Wilson's calendar the treatment it deserves:

```
.ll 3i            \" set line length to 3 inches
.po 2i            \" set page offset to 3 inches
.pl 9i            \" set page length to 9 inches
.wh 0 HD          \" set the header trap
.wh -1i FO        \" set the footer trap
.de HD            \" define header macro HD
'sp 1i            \" skip 1 inches of space
..                \" end macro definition
.de FO            \" define footer macro
'sp 2             \" skip 2 lines
.tl ''- % -''     \" define footer title
'bp               \" begin new page
..                \" end macro definition
.de PP            \" define paragraph macro
.sp 1             \" skip 1 line of space
.ti 5             \" indent the first line 5 characters
..                \" end macro definition
.PP
Adam was human--this explains it all.  He did
not want the apple for the apple's sake, he
wanted it because it was forbidden.  The mistake
was in not forbidding the serpent; then he would
have eaten the serpent.
.PP
Training is everything.  The peach was once a bitter
almond; cauliflower is nothing but cabbage with a
college education.
.PP
Habit is habit, and not to be flung out of the window
by any man, but coaxed downstairs a step at a time.
```

As a point of technique, always set header and footer traps early in your input script; otherwise, **nroff** may not print the header on the first page.

Macro Arguments

You can affect how macros function by passing them modifiers, called *arguments*. An argument may be a bit of text that is arranged in a special way by the macro, or it may be a number or other parameter that dictates exactly what the macro does.

As an example of how a macro can handle arguments, consider a macro to format the list of ingredients for a recipe. You want the ingredients to be printed as follows:

```
3 cups of pumpkin
1 cup of milk
1 cup of sugar
1 tsp of ground ginger
1 tbl of cinnamon
```

Each of these lines has the same format: the amount of ingredient, the unit of measurement, the word "of", and the name of the ingredient. You can create a macro (call it **.RE**, for **re**cipe) that encodes the format of these lines and contains three "slots": one slot for the amount, one for the unit of measurement, and one for the name of the ingredient. Each time you use the macro, you indicate what you want to go into each slot, and **nroff** substitutes it for you. The macro **.RE** can be constructed as follows:

```
.de RE              \" define macro RE
\\$1 \\$2 of \\$3   \" set RE's arguments
..                  \" end definition
Here is some text.
.nf                 \" don't fill the recipe
.RE 3 cups pumpkin
.RE 1 cup milk
.RE 1 cup sugar
.RE 1 tsp "ground ginger"
.RE 1 tbl cinnamon
.fi                 \" resume filling
Here is some more text.
.bp                 \" begin a new page, to force printing
```

When you call a macro that takes arguments, the arguments must appear on the same line as the macro command itself. A macro may have up to nine arguments; they are denoted by **\$1**, through **\$9**, respectively: the first field after the macro name is called **\$1**, the second **\$2**, and so on.

If you want to use as an argument a string of characters that includes blank spaces, you must enclose the string within quotation marks, as with the words "ground ginger", in the example above. If you forget to include the quotation marks, **nroff** regards each word in the string as a separate argument, and treats them accordingly.

Note that macros that are called by traps cannot accept arguments.

Double vs. Single Backslashes

If you carefully examine the definition of **RE**, you will see that it identifies each argument with two backslashes:

```
\\$1 \\$2 of \\$3
```

Whenever you identify an argument within a macro, always preface it with two backslashes, rather than one. The reason is that **nroff** in effect processes a macro *twice*: when it first reads it, and later when you call it within your text. Prefacing an argument with *one* backslash tells **nroff** that you want to expand that argument when the macro is first read; prefacing it with two backslashes tells **nroff** that you want to expand it when the macro is called in your text. In nearly every circumstance, you want to expand the arguments in your text, so you should use two backslashes. As you will see, this rule also applies to the use of *strings* and *number registers* within macros.

To see how this works, consider again the **.RE** macro:

```
.de RE
\\$1 \\$2 of \\$3
..
Here is some text.
.nf
.RE 3 cups pumpkin
.RE 1 cup milk
.RE 1 cup sugar
.RE 1 tsp "ground ginger"
.RE 1 tbl cinnamon
.fi
Here is some more text.
.bp
```

Using two backslashes, as above, allows you to redefine what **$1**, **$2**, and $3 mean many times throughout your text, to generate the following output:

```
Here is some text.
3 cups of pumpkin
1 cup of milk
1 cup of sugar
1 tsp of ground ginger
1 tbl of cinnamon
Here is some more text.
```

If you used only one backslash, however, your output would appear as follows:

```
Here is some text.
  of
  of
  of
  of
  of
Here is some more text.
```

nroff could not expand the argument calls (**\$1** etc.), because you had not yet defined them; therefore, it threw them away; and because all of the argument calls had been thrown away, **nroff** then threw all the arguments away. All that was left was word **of**.

Designing and Installing Macros

Now that you have been shown how to write a macro, the next step is to design some macros and install them, so you can call them over and over again.

The first step in designing a macro is to analyze the problem that you want to solve. Suppose that in this instance you want to print a list of names. Each name will consist of a first name, a last name, and the department with which he is associated, and the list will be printed in columns; for example:

```
Firstname      Lastname       Department
```

Moreover, you want to be able to switch the order in which the columns appear without having to retype your list; for example:

```
Lastname       Firstname      Department
```

or

```
Department     Lastname       Firstname
```

In effect, then, you want three macros: one for each of the three orders of columns shown above.

When you have finished designing your macros, they should look something the following. Type the following into the file **tmac.lst**; note that the symbol **<tab>** represents a *tab* character, and should not be entered literally:

```
.\" List macros. $1 represents first name,
.\" $2 last name, $3 department
.de LA
.nf
.ta 1.5i 2.75i
\\$1<tab>\\$2:<tab>\\$3
.Rt
..
.de LB
.nf
.ta 1.5i 2.75i
\\$2,<tab>\\$1:<tab>\\$3
.Rt
..
.de LC
.nf
.ta 1.5i 2.75i
\\$3:<tab>\\$2,<tab>\\$1
.Rt
..
```

The first lines are *comments*, so that anyone who looks at these macros will know what they do. The first command line, introduced with the **.de** command, names each macro. These names were selected after checking the file **tmac.s**, which is where the **-ms** macro package is kept, to confirm that they are not used elsewhere. Naturally, using the same macro name in two different places can lead to a great deal of trouble.

The next command, **.nf**, turns off the **nroff**'s normal right justification, which otherwise would smear a table. The **.ta** command sets the tab characters at certain points on the page, measured from the left margin.

The next line gives the order in which the arguments appear. The arguments are separated by tab characters, and punctuation is inserted. The last command, **.Rt**, calls a macro in the file **tmac.s**; this macro resets **nroff** to its normal fill mode and returns the tab settings to normal. Note that these macros can be used only when you also use the **-ms** macro package.

After you have typed the macros into **tmac.lst**, carefully read over what you type to ensure that no there are no errors; if you find any, be sure to correct them. The final step is to move **tmac.lst** into the directory **/usr/lib**, which is where **tmac.s** is also kept.

To test your new macros, type the following text into the file **ex6.r**:

```
The following lists give the personnel who are involved in
this project:
.sp
.LA Ivan Morgan Engineering
.LA Marian Gusman Design
.LA George Meyer Electrical
.LA Catherine Scanlon "Metal Shop"
.LA Fred McElroy Carpentry
.LA Anne Assenmacher "Machine Shop"
.sp
.LB Ivan Morgan Engineering
.LB Marian Gusman Design
.LB George Meyer Electrical
.LB Catherine Scanlon "Metal Shop"
.LB Fred McElroy Carpentry
.LB Anne Assenmacher "Machine Shop"
.sp
.LC Ivan Morgan Engineering
.LC Marian Gusman Design
.LC George Meyer Electrical
.LC Catherine Scanlon "Metal Shop"
.LC Fred McElroy Carpentry
.LC Anne Assenmacher "Machine Shop"
.sp
We expect that they will receive your full cooperation.
```

The same set of names is used three times; the only difference is the macro call employed.

Now, process this file with the following command:

```
nroff -ms -mlst ex6.r >ex6.p
```

As you can see, when you installed **tmac.list** into **/usr/lib**, you could invoke it in the same way that you invoke **tmac.s** with **-ms**.

When you look at the output file **ex6.p**, you should see something that resembles the following:

```
The following lists give the personnel who are involved in this project:

Ivan              Morgan:           Engineering
Marian            Gusman:           Design
George            Meyer:            Electrical
Catherine         Scanlon:          Metal Shop
Fred              McElroy:          Carpentry
Anne              Assenmacher:      Machine Shop

Morgan,           Ivan:             Engineering
Gusman,           Marian:           Design
Meyer,            George:           Electrical
Scanlon,          Catherine:        Metal Shop
McElroy,          Fred:             Carpentry
Assenmacher,      Anne:             Machine Shop

Engineering:      Morgan,           Ivan
```

```
Design:             Gusman,             Marian
Electrical:         Meyer,              George
Metal Shop:         Scanlon,            Catherine
Carpentry:          McElroy,            Fred
Machine Shop:       Assenmacher,        Anne

We expect that they will receive your full cooperation.
```

As you grow proficient in writing **nroff** macros, you will probably find it most efficient to keep special macros in their own files; this will save time by ensuring that **nroff** does not have to process macros that are never called.

Strings

Suppose you are writing a script for **nroff** and, to relieve the tedium, decide to punctuate the text occasionally with a rousing cry of "FOOD FIGHT!!". If you plan to interject this phrase more than a few times in your script, you can take advantage of another labor-saving device, called a *string.* You can use a string name as an abbreviation for a long string of characters you use frequently. Like a macro, a string is a *name* that **nroff** associates with a *definition* that you supply. Wherever you put the name in your text, **nroff** prints the definition. Although macros refer to sets of *commands* that you define, strings refer to strings of *characters* that you define.

You define a string with the *define string* primitive **.ds:**

```
.ds FF "FOOD FIGHT!!"
```

The first field after the **.ds** gives the name of the sting, in this case **FF.** Like a macro name, a string name may be either one or two characters. The second field after the **.ds** gives the definition of the string, in this case

```
"FOOD FIGHT!!"
```

As in this example, you must enclose the definition within quotation marks if it contains spaces.

Be careful whenever you define a macro or a string. If you already have a macro or a string named **X** and you define a new macro or string named **X, nroff** forgets the previous meaning of **X.**

Once you have defined a string, you can refer to it anywhere in your text. The string itself appears in the output text wherever a reference to it appears in the input text. You refer to the string **FF** in the following fashion:

```
\*(FF
```

Use the left parenthesis '(' only when the name of the string is two characters long. If the string name is only a single character, such as **S,** refer to it as follows:

```
\*S
```

As an example, type the following script into **ex7.r**, and process it through **nroff**; do *not* use the **-ms** macro package:

```
.ds FF "FOOD FIGHT!!"
.ds W "WHOOPEE!!"
.ce
From Aristotle's "Poetics"
.br
.sp
A tragedy is the imitation of an action \*(FF
that is serious and also, \*W as having magnitude,
complete in itself, with incidents \*(FF
arousing pity and fear, wherewith to accomplish \*W
\*(FF its purgation of such emotions \*(FF \*(FF.
.bp
```

nroff adjusts the spacings between words in a string but does not hyphenate any word that is within a string. If you use a very short line length, such as two inches, and define a string that includes a three-inch long word, that word would not be hyphenated but would extend past the right-hand margin.

You cannot include a newline character in a string. However, you can spread the definition of a string out over more than one line with the aid of *concealed* newlines (preceded by the backslash character '\'). **nroff** ignores each concealed newline. For example, add the following string to the previous example:

```
.ds PR "GO TEAM \
GO!!!"
```

As you can see, **nroff** ignores concealed newlines anywhere in its input.

Strings Within Strings

You can define a string that has embedded within it a reference to another string. Whenever you refer to the bigger string in your text, **nroff** substitutes the definition of the smaller string for any reference to the smaller string. When you embed strings, though, you should use *two* backslashes to refer to the embedded string, for the same reason that you should use two backslashes to refer to an argument within a macro:

```
.ds S "This string \\*x has embedded \\*y strings"
```

To help understand this better, type following three scripts into your computer and format them with **nroff**. The first script contains proper references to embedded strings (using double backslashes); it works as expected:

```
.ds S "strings \\*X, strings \\*Y, strings \\*Z"
.ds X "here"
.ds Y "there"
.ds Z "everywhere"
\*S
```

The next script contains embedded references that use only single backslashes. Because the embedded strings are defined after the larger string, they are not available when **nroff** defines the larger string, and so the references are ignored:

```
.ds S "strings \*X, strings \*Y, strings \*Z"
.ds X "here"
.ds Y "there"
.ds Z "everywhere"
\*S
```

The third script again contains embedded references using single backslashes. This time, the embedded strings are defined *before* the larger string, and so are available when the larger string is defined:

```
.ds X "here"
.ds Y "there"
.ds Z "everywhere"
.ds S "strings \*X, strings \*Y, strings \*Z"
\*S
```

To avoid unnecessary worry, you should always play it safe and use double backslashes to refer to embedded strings.

Number Registers

You learned in previous sections that **nroff** keeps track of the output page number while it prints its output. You made use of this fact when you created a footer macro that printed page numbers. **nroff** also keeps track of other housekeeping information, such as the current line length, page offset, page length, and vertical position of the last output line. It keeps this information in storage locations called *number registers*.

You can use the *name* of a number register to refer to the number that is stored in it. When you place a reference to a number register in your text, **nroff** substitutes for the name whatever number is currently in the register.

Number register names are one or two characters long, just like macro and string names. You can have a number register with the same name as a string or a macro without confusing **nroff**, even though you cannot give a macro and a string the same name. However, *you* might become confused; **nroff** scripts usually are easier to understand if you keep all macro names, string names, and register names distinct.

Another difference between number registers, macros, and strings is that **nroff** itself does not define any macros or strings (although the **-ms** macro package does), but it does automatically define and update quite a few number registers. You can use these predefined number registers in much the same way that you use registers you define yourself, except that you cannot change their values.

To define a number register, you must specify the *register name* and the *initial value* for the register. The *number register* primitive **.nr** looks like this:

```
.nr X 5
```

Here **X** is the name of the register and **5** is the initial value to store in it. To refer to number register **X** in your text, use **\nX**; if the name is two characters long (for example, **XY)**, use **\n(XY**. This is exactly like the way you refer to a string, except that you use the letter ‘n’ instead of an asterisk ‘*’. When **nroff** sees a reference to number register **X**, it automatically substitutes the value stored in **X**. As you will see shortly, **nroff** can do arithmetic, and learning to use number registers is an important part of learning to take advantage of **nroff**'s arithmetic abilities.

A reference to a number register can occur anywhere a number would normally occur. For example, if you set register **X** to 5, as above, you can set the line length to five inches as follows:

```
.ll \nXi
```

This command is essentially the same as

```
.ll 5i
```

if the current value of register **X** is 5.

A familiar problem arises when you refer to a number register inside a macro or a string definition. If you use just one backslash, **nroff** substitutes the value in the register for the reference when it first processes the macro or string. If you have not yet defined the number register in your script, **nroff** inserts **0** into the macro or string. Normally, you should use a a double backslash, such as **\\nX** or **\\n(XY**, when referring to a number register within a macro or string. Using the double backslash is particularly important if you *change* the value of the register throughout your script, and want the current value to appear in the macro or string each time you call it.

Try typing the following examples into your computer, and processing them with **nroff**. See if you can describe why **nroff** prints what it does in each case. The first example defines a string with a register reference preceded by a single backslash.

```
.ds S "Here is a number \nX"
.nr X 55
\*S
\nX
```

You should see the following output:

```
Here is number 0
55
```

nroff printed what it did because number register X had not yet been defined when it was called in string S; **nroff** therefore erased the reference to X and substituted zero for it. Number register X was then set to 55, which was printed when the register was specifically called later in the script.

The second example is similar, but now the number register is set *before* the string is called:

```
.nr Y 56
.ds T "Here is a number \nY"
\*T
\nY
```

Now the output is

```
Here is a number 56
56
```

The third example uses a double backslash for the register reference.

```
.ds U "Here is a number \\nZ"
.nr Z 57
\*U
.nr Z 58
\*U
```

This script produces the following:

```
Here is a number 57
Here is a number 58
```

The final example uses a single backslash again.

```
.nr W 59
.ds V "Here is a number \nW"
\*V
.nr W 60
\*V
```

The following is produced:

```
Here is a number 59
Here is a number 59
```

The last example illustrates the danger of using a single backslash to refer to a number register within a string definition. You defined the number register **W** before you defined the string **V**, so the value for **W** was available when **nroff** read the definition of **V**. **nroff** substituted the value when it reads the definition; the reference to the number register **W** is no longer there. You then change the value of **W**, but as you see in the next call of **V**, the change does not affect the number that appears in **V**. In contrast to this, notice in the third example that the double backslash in the definition of **U** allows the reference to number register **Z** to remain within the definition of string **U**. Whenever you change the value of **Z** and then call **U**, **nroff** substitutes the new value of **Z** for the reference to **Z** within **U**.

You can also use the **.nr** primitive to increase or decrease the value in a number register. For example, suppose you initially store the value five in **X**:

```
.nr X 5
```

Incrementing and Decrementing

You can change the value of **X** to 9 by adding 4, as follows:

```
.nr X +4
```

You can then change the value of **X** to 7 by subtracting 2:

```
.nr X -2
```

A plus or minus sign before a number on the **.nr** command line tells **nroff** to add or subtract the given amount to or from the value in the register. Because a negative number is always preceded by a minus sign whereas a positive number usually is not preceded by a plus sign, you can use **.nr** to set a register to a positive value in a way that cannot be imitated for negative values. For example, suppose you again start out with number register **X** set to a value of 5:

```
.nr X 5
```

If you immediately follow this with

```
.nr X 7
```

then **nroff** replaces the value of 5 with 7. The second **.nr** does not increase the value of **X** by 7 to produce 12; rather, it wipes out the previous value of 5 and replaces it by the value 7. The command line to increase **X** by 7 is

```
.nr X +7
```

If you again start with a value of 5 in **X** and want to change the value to -4, you cannot use the following command line:

```
.nr X -4
```

nroff interprets this as a command to *decrease* the current value of **X** by 4, which is not what you intended. This command places the value 1 in **X**, since 5-4=1. If **X** initially has a value of 5 and you want to change the value to -4, you could use the command

```
.nr X -9
```

You can also increase or decrease the value of a number register without using **.nr**. If number register **X** currently has the value 10, the reference **\n+X** increases the value in **X** by 1 to 11 and substitutes the new value for the reference. The value in **X** becomes 11; **nroff** replaces the next reference **\nX** by 11, whereas another reference **\n+X** increments the value in **X** to 12 and replaces the reference by 12. Similarly, if number register **XY** currently has the value 15, the reference **\n+(XY** increases the value in **XY** to 16 and replaces the reference by 16.

You can also decrease a register's value. The reference **\n-X** decreases the current value in **X** by 1 and substitutes the new value for the reference. Likewise, the reference **\n-(XY** decreases the current value in **XY** by 1 and substitutes the new value for the reference.

You can change the size of the increment or decrement by means of another option to the **nr** command. If you define **X** with

```
.nr X 1 5
```

then **nroff** sets the value of **X** to 1 and sets the increment value for **X** to 5. The next reference **\n+X** increments the value in **X** from 1 to 6 (the '+' now causes **nroff** to add 5 to the current value of **X** rather than adding 1) and substitutes 6 for the reference. In the same manner, **\n-X** subtracts 5 from the current value of **X** and substitutes the new value for the reference. This is convenient if you plan to repeatedly increment or decrement **X** by the same fixed amount. If you wish to change the size of the increment, simply redefine **X** with another **.nr** that specifies the new initial and increment values. If you define a number register but do not specify an increment value, **nroff** assumes the increment value to be 1.

The following example of a macro illustrates a typical use of a number register and incrementing.

```
.nr W 1                                     \" set W to 1, inc by 1
.ds X "Here's Wrestler No. \\nW,"           \" define string X
.de B                                       \" define macro B
.br
\\*X \\$1!!!                                \" define arg to macro B
.nr b \\n+W                                 \" increment W
..                                          \" end definition
.B "Alex 'Killer' Bovine"                   \" call B with arguments
.B "William 'Crusher' Risible"
.B "Vlad 'the Impaler' Acephalous"
.bp                                         \" force printing of page
```

to produce the following output:

```
Here's Wrestler No. 1, Alex 'Killer' Bovine!!!
Here's Wrestler No. 2, William 'Crusher' Risible!!!
Here's Wrestler No. 3, Vlad 'the Impaler' Acephalous!!!
```

A reference to a number register may appear any place a number can normally appear. For example:

```
.nr X \nY \nZ
```

sets register **X** to the value of register **Y** and sets the increment for **X** to the value of register **Z.**

As mentioned before, **nroff** performs arithmetic. It understands and evaluates properly formed arithmetic expressions involving numbers, references to number registers, the arithmetic operators '+', '-', '*', '/', '%', and parentheses. The first four operators represent addition, subtraction, multiplication, and division. The '%' is the *modulus* or *remainder* operator: the value of 7%3 is 1, which is the remainder when 7 is divided by 3.

One word of caution: **nroff** evaluates expressions from left to right without any preference for performing some operations before others. For example,

```
.nr X 5+4*3/9
```

stores 3 in **X. nroff** does not perform the multiplication and division before the addition, as you might expect.

Another important fact is that number registers hold only integers. If you write

```
.nr X 3.6
```

nroff truncates the value 3.6 and stores 3 in **X**. Also, an assignment such as

```
.nr X 3.9*3.9
```

stores 9 in **X**; **nroff** truncates each factor before it performs the multiplication. The assignment

```
.nr X 0.4*8
```

stores 0 in **X** rather than 3: truncation occurs before **nroff** performs the multiplication rather than after.

A final word of caution: when you use numbers with commands other than **.nr,** the results may *not* be what you expect. **nroff** understands several different units of measurement and converts between units automatically. The next section explains units and conversion in detail.

Units of Measurement

As mentioned above, **nroff** maintains many number registers during processing. For example, it stores the current page length in the register **.l** (Note that the period '.' is actually part of the name of this register.) If you set the line length to five inches with the command

```
.ll 5i
```

nroff stores the length in register **.l** automatically; however, if you print the value in register **.l** by entering **\n(.l**, you find the value is 600. What does this mean?

Many **nroff** commands require that you specify lengths or measurements as arguments. You are already familiar with many of these commands: for example, **.ll, .po, .pl,** and **.lt**. **nroff** accepts various units of measurement, but for purposes of calculation, it converts each into a basic unit called a *machine unit*, which is abbreviated **u**. A machine unit is 1/120 of an inch long. Because one inch is 120 machine units, the length of a five-inch line is 5 times 120, or 600 machine units.

The conversion table for units of measurement is as follows:

inch:	1**i** = 120**u**
vertical line space:	1**v** = 20**u**
centimeter:	1**c** = 47**u**
em:	1**m** = 12**u**
en:	1**n** = 6**u**
pica:	1**P** = 20**u**
point:	1**p** = 1**u**

Most of these are traditional typesetting terms.

As noted briefly earlier, **nroff**'s output actually consists of a sequence of characters. It is useful, though, to think of the output as being "printed" at ten characters per inch (Pica or 10-pitch spacing) and six lines per inch. Many output devices use this spacing. With these assumptions, **5i** is equivalent to five inches of printed output.

Every **nroff** command has a default unit of measurement. For example, the default unit for **.ll** is **m**, whereas the default unit for **.sp** is **v**. If you type

```
.ll 5
```

nroff interprets it not as five inches or five centimeters, but as 5**m**, which it converts to 5 times 12, or 60 machine units (60**u**).

nroff always assumes a unit specification as part of each number and automatically converts each number and its unit specification into machine units. If you append an explicit unit specification to the number, **nroff** uses it; if you do not, **nroff** uses the default unit for the command.

For example, suppose you write the following commands:

```
.nr X 2i
.ll \nX
```

What line length results? The first command stores the number 2 times 120, or 240, in register **X.** The second command is therefore equivalent to typing

```
.ll 240
```

However, the default unit for **.ll** is **m.** Because 1**m** equals 12**u**, **nroff** sets the line length to 12 times 240, or 2,880 machine units. If you wanted a line length of two inches to result from the above commands, you will be unpleasantly surprised, because 2**i** equals only 240**u**. Instead, you should write:

```
.nr X 2i
.ll \nXu
```

By including the **u** in the **.ll** primitive, you do not accidentally multiply your results by 12, as happened earlier.

You should think of the unit specification as a part of a number. Because **nroff** accepts so many different units of measurement, a number without a unit specification is ambiguous. What does '5' mean? Five inches? Centimeters? Ems? **nroff** must know what unit of measurement you are using. If you think of the unit specification as a part of the number, you will have less trouble with potentially mystifying situations like the following. As mentioned, number registers store only integers and **nroff** truncates each number in an arithmetic expression to an integer before evaluating the expression. Therefore, the following stores 0 in register **X**:

```
.nr X 0.4*9
```

But now try the following:

```
.nr X 0.4i
\nX
```

This does not store 0 in **X** like the previous command; it stores 0.4 times 120, or 48 in **X**. The 0.4 is not truncated to 0 here! Truncation occurs *after* conversion to machine units, so **nroff** truncates 0.4**u** in the first example. But the number in the second example is given in inches **i** instead of machine units **u**. **nroff** converts it to **u** before truncating to get an integer.

As another example, the following stores 1 in **X**:

```
.nr X 0.01i
```

nroff converts 0.01 inches to 0.01 times 120, or 1.2**u**, and then truncates 1.2 to 1.

The following command illustrates that **nroff** understands *each* number in an arithmetic expression to have an attached unit specification, whether you supply one or not.

```
.ll 2*8
```

Recall that **nroff** stores the current line length in the register **.l;** if you type

```
\n(.l
```

you will receive the number 2,304. **nroff** interprets the 2 as 2**m** and the 8 as 8**m**, because the default unit for **.ll** is **m.** Then it converts each to machine units and multiplies to give the result: (2*12)*(8*12), or 2,304.

Consider one final example that illustrates the unusual consequences of seemingly innocent assignments. Suppose you set the page offset as follows:

```
.po 8/3
```

nroff stores the current page offset in register **.o**. To see what number it stores there, type

```
\n(.o
```

You see that the page offset is 2. Because the default unit for **.po** is **m,** the calculation is (8*12)/(3*12)=8/3, which **nroff** truncates to 2. Two machine units is equivalent to only 1/60 of an inch. This is not a physically reasonable value for most typewriter-like devices, so a page offset of 0 characters results. On the other hand,

```
.po 8/3u
```

produces a page offset of approximately 1/4 of an inch.

Conditional Input

Now that you have been introduced to number registers, you can use them in conjunction with powerful *conditional commands* to create more elaborate **nroff** scripts.

To see how conditional statements help you construct an **nroff** script, consider again the problem of creating header and footer macros. Earlier, you constructed macros that skipped space at the top of the page and printed the page number at the bottom of each page.

Suppose, however, that you are formatting a paper that has a title. You want to print the page number for page 1 at the bottom of the page, and to print the rest of the page numbers at the top of the page. Both the header and the footer need some kind of conditional mechanism to perform differently on the first page than on subsequent pages. On page 1, the header should skip to where the title will be printed; on other pages, the header should print the page number. On page 1, the footer should print the page number; on other pages, the footer should leave a block of blank space at the bottom of the page.

To execute commands conditionally, use the *if/else* commands **.ie** and **.el**, which are demonstrated in the following example. Note that the formation '', which is used with the **.tl** command, represents two apostrophes, *not* a quotation mark.

```
.de HD            \" define header
.ie \\n%=1 .A
.el .B            \" else do B
..
.de A             \" define macro A
.sp |1.0i         \" space down to 1 inches from top of page
..
.de B             \" define macro B
'sp 2v            \" skip 2 spaces
.tl ''- % -''     \" print page no.
'sp |1.0i         \" skip to 1 inch from top of page
..
```

```
.de FO              \" define footer
.ie \\n%=1 .C       \" if page no. is 1 then do C
.el .D              \" else do D
..
.de C               \" define macro C
'sp |-4v            \" move to 4 in. above bottom of page
.tl ''- % -''       \" print page no.
'bp                 \" begin new page
..
.de D               \" define macro D
'bp                 \" begin new page
..
```

As you can see, the **.ie** and **.el** commands always occur in pairs. **.ie** consists of three parts: the command name **.ie**, then a *condition* that **nroff** tests, followed by a *command* **nroff** performs if the condition is true. If the condition on the **.ie** command line is not true, **nroff** performs the command on the **.el** line instead.

In the example, each conditional invokes a macro on the command line. Actually, the conditional can specify *input* text rather than the command after the condition. If you want to execute several commands or include several text lines conditionally, enclose the lines with the special sequences '\{' and '\}'.

Note, too, that one other new element was introduced in the construction of these macros. Some of the **.sp** commands have a vertical bar immediately in front of the measurement; for example,

```
.sp |1.0i
```

Normally, when **nroff** sees a command like **.sp 1.0i,** it moves down one inch on the output page. The movement is relative to where **nroff** happens to be on the output page when it received the request. The vertical bar tells **nroff** that the following measurement is an *absolute* measurement, measuring either from the top of the page (if positive) or from the bottom of the page (if negative). Therefore,

```
.sp |1.0i
```

tells **nroff** to move to one inch from the top of the page;

```
.sp |(-4v)
```

tells it to move to four vertical spaces from the bottom of the page.

The **.if** primitive is formed exactly like **.ie**. Unlike **.ie**, which must always be used with **.el**, the **.if** command may be used by itself. If the condition on the **.if** line is true, **nroff** performs the command that follows the condition; if the condition is false, it ignores the command altogether.

This chapter ends with two substantial examples that incorporate most of what you have studied so far. To illustrate the use of conditionals, the first example begins each even paragraph of output with the phrase **Even Paragraph:** and begins each odd paragraph with the phrase **Odd Paragraph:**. Type this into the file **ex8.r**, and process it through **nroff** without using the **-ms** macro package, and as before, there is no need to copy the comments:

```
.wh 0 HD            \" set header trap
.wh -2i FO          \" set footer trap
.nr EO 1            \" set EO register to 1
.po 2i              \" page offset 2 inches
.pl 6i              \" page length 6 inches
.lt 4i              \" title length 4 inches
.ll 4i              \" line length 4 inches
.de HD              \" define header
'sp |(1i-1v)        \" space down to 1 inch minus 1 line
.tl ''\\*(WS''      \" set WS macro in title
'sp |1.5i           \" space down to 1.5 inches
..
.de FO              \" define footer
'sp |(3i+3v)        \" space down to 3 inches plus 3 lines
.tl ''- % -''       \" set page number in footer
'bp                 \" begin new page
..
.ds WS "From the Devil\'s Dictionary"
```

```
.                   \" define string WS
.de PP              \" define paragraph macro
.ie \\n(EO=0 .EP    \" if EO = 0 (even) then do EP
.el .OP             \" else do OP
..
.de EP              \" define EP (even paragraph)
.br
.nr EO 1            \" set register EO to 1
.sp 1v              \" skip 1 line
.ll 4i              \" set line length to 4 inches
.lt 4i              \" set title length to 4 inches
\\*E                \" insert string E
..
.ds E "Even Paragraph:"
.                   \" define string E
.de OP              \" define macro OP (odd paragraph)
.br
.nr EO 0            \" set register EO to 0
.sp 1v
.ll 3i              \" set line length to 3 inches
.lt 3i              \" set title length to 3 inches
\\*O                \" insert string O
..
.ds O "Odd Paragraph:"
.                   \" define string O
.PP
Debt, n. An ingenious substitute for the whip
and chain of the slave-driver.
.PP
Bore, n. One who talks when you wish him to listen.
.PP
Brandy, n. A cordial composed of one part
thunder-and lightning, one part remorse, two parts
bloody murder, one part death-hell-and-the-grave,
and four parts clarified Satan.
.PP
Responsibility, n. A detachable burden easily
shifted onto the shoulders of God, Fate, Fortune,
Luck, or one's neighbor.
```

This example uses an "even/odd" register called **EO** to determine whether you are beginning an even or an odd paragraph. To distinguish between even and odd paragraphs, it uses a line length of four inches for even paragraphs and one of of three inches for odd paragraphs. It changes the title length with each paragraph, so **nroff** centers the page number with respect to whichever kind of paragraph happens to occur at the bottom of a page.

The final example illustrates a loop constructed with the **if/else** commands. The first paragraph is six inches long with no page offset; each succeeding paragraph is one inch shorter with a page offset one inch longer. The line length of the sixth paragraph is one inch; the next paragraph renews the cycle with a six-inch line length. Type this into file **ex9.r**, and process it as you did the above example:

```
.nr PO 0 1          \" set register PO to 0, increment by 1
.de PP              \" define paragraph macro
.ie \\n(PO=6 .A     \" if register PO=6 then do A
.el .B              \" else do B
..
.de A               \" define macro A
.br
.nr PO 0            \" set register PO to 0
.nr LL 6-\\n(PO     \" set register LL to 6 minus PO
.ll \\n(LLi         \" set line length to LL inches
.po \\n(POi         \" set page offset to PO inches
.nr PO \\n+(PO      \" increment register PO
```

```
.sp               \" skip a space
..
.de B             \" define macro B
.br
.nr LL 6-\\n(PO   \" set LL to 6 minus PO
.ll \\n(LLi       \" set line length to LL inches
.po \\n(POi       \" set page offset to PO inches
.nr PO \\n+(PO    \" increment register PO
.sp               \" skip a space
..
.PP
Future, n. That period of time in which our affairs prosper,
our friends are true, and our happiness is assured.
.PP
Gallows, n. A stage for the performance of miracle plays, in
which the leading actor is translated into heaven.
.PP
Geneaology, n. An account of one’s descent from an ancestor
who did not particularly care to trace his own.
.PP
Guillotine, n. A machine which makes a Frenchman shrug
his shoulders with good reason.
.PP
History, n. An account most false, of events
most unimportant, which are brought about by
rulers mostly knaves, and soldiers mostly fools.
.PP
Idiot, n. A member of a large and powerful tribe
whose influence in human affairs has always been
dominant and controlling.
.PP
Kiss, n. A word invented by the poets as a rhyme
for "bliss".
```

You should try this example to see verify that “loop” works as advertised.

Environments and Diversions

Another aspect of **nroff**’s power is the ability to shift from one *environment* to another.

The **nroff** *environment* is the overall manner in which **nroff** processes your input text. The environment’s definition includes such aspects as line length, fill and adjust modes, and indentation.

nroff allows you to define three independent environments, called **0**, **1**, and **2**. In each, you can set as you wish such parameters as line length, filling, adjustment, and indentation. You can call a different environment with the **.ev** command; the parameters you define for the new environment control text processing until you change them within the present environment or shift to another environment.

Not all **nroff** parameters change when you switch to a new environment. For example, different environments do *not* have independent page offsets; the **.po** command affects all environments. Parameters that may be set to different values in different environments are *environmental parameters*; parameters that cannot be switched according to environment, like page offset, are *global parameters*. Macro and string definitions are global.

When you first call **nroff,** you are by default in environment 0. In all the examples used in this tutorial thus far, everything happened in environment 0. The following example illustrates how to switch back and forth between environments. Type in the following **ex10.r** and process it to see the output as you go along.

```
.po 1i            \" set global page offset to 1 inch
.ll 4i            \" set line length in ev 0 to 4 inches
.de PP            \" define paragraph macro
.sp
.ti 0.5i          \" indent first line 1/2 inch
..
.PP
The heart of the righteous studieth to answer,
```

```
but the mouth of the wicked poureth out evil things.
.br
.ev 1             \" switch to environment 1
.ll 3i            \" set line length to 3 inches
.ls 2             \" line spacing now double space
.PP
A froward man soweth strife, and a whisperer
separateth chief friends.
.br
.ev               \" return to previous ev (0)
.PP
It is naught, it is naught, sayeth the buyer;
but when he is gone his way, then he boasteth.
.br
.ev 1             \" switch to ev 1
.PP
Wealth maketh many friends; but the poor is separated
from his neighbors.
.br
.ev               \" return to ev 0
```

The first **.ll** command sets a line length of four inches in environment 0. After defining the paragraph macro **.PP** and an initial paragraph in environment 0, you switched to environment 1 with the command

```
.ev 1
```

You now enter a new environment. If you do not explicitly set environmental parameters, such as line length, **nroff** automatically uses default values for them. **nroff** assigns the same default values in environments 1 and 2 as it does in environment 0.

The line length in environment 1 is set to three inches with the output text double-spaced. The *line space* primitive

```
.ls 2
```

leaves one blank line between each output line. Thus, paragraphs processed in environment 0 have four-inch single-spaced lines, whereas paragraphs processed in environment 1 have three-inch double-spaced lines.

The example used the command line

```
.ev
```

without an argument to *leave* environment 1. This leaves environment 1 and restores (or "pops") the previous environment — in this case, environment 0. The next time you enter environment 1, you will not need to set the line length to three inches again: the value stays in effect in environment 1 until you specifically change it. The same is true of all environmental parameters.

To understand how **nroff** switches between environments, imagine that you have a set of plates, each marked with either a **0**, a **1**, or a **2**. You have as many plates of each type as you wish. You stack the plates on a table; the top plate represents your current environment. You begin with a '0' plate on the table, to represent the initial environment when you enter **nroff**.

Switching to environment 1 with the command **.ev 1** corresponds to placing a '1' plate on top of the '0' plate. You can again change the stack of two plates either by placing a new plate on top of the stack, or by removing the top plate from the stack: the former corresponds to calling a new environment, whereas the latter corresponds to restoring the previous environment with the command line **.ev**.

Because you can have as many plates of each type as you wish, you can call environment 1, then call environment 2, then restore environment 1, then call environment 0, and so on. The command **.ev** *N*, where *N* is 0, 1, or 2, places (or "pushes") a plate onto the stack; the command **.ev** removes (or "pops") the top plate from the stack.

To illustrate this, add the following text to the end of the previous example. Use a piece of paper and pencil to keep track of how the **.ev** commands add or remove environments. Because the line lengths are different in each environment, it should be easy to tell in which environment **nroff** has processed each paragraph:

```
.ev 2             \" introduce environment 2
.ll 5i            \" set line length
.in 1i            \" set indentation
```

```
.PP                  \" paragraph in ev 2
A poor man that oppresseth the poor is like
a sweeping rain which leaveth no food.
.br
.ev 0                \" push ev 0
.PP
As a roaring lion, and a ranging bear; so is
a wicked ruler over the poor people.
.br
.ev 1                \" push ev 1
.PP
Wrath is cruel, and anger is outrageous;
but who is able to stand before envy?
.br
.ev 2                \" push ev 2
.PP
A good name is rather to be chosen than
great riches; and loving favour rather than
silver and gold.
.br
.ev 0                \" push ev 0
.PP
Pride goeth before destruction, and an haughty
spirit before a fall.
.br
.ev                  \" return to ev 2
.ev                  \" return to ev 1
.PP
He that answereth a matter before he heareth it,
it is folly and shame unto him.
.br
.ev                  \" return to ev 0
.ev                  \" return to ev 2
.PP
A merry heart doeth good like a medicine, but a
broken spirit drieth the bones.
.br
```

Buffers

Earlier, it was shown that **nroff** uses a buffer to assemble words from its input into output lines. Actually, each environment has its own buffer. Switching to a new environment does *not* cause a break. Suppose you are currently in environment 1 with an unfinished line in the buffer. When you give the command **.ev 2**, the unfinished line remains undisturbed in the environment 1 buffer until you return to environment 1. Text you process in the meantime in environment 2 or in environment 0 has no effect on the partial line in the environment 1 buffer, because **nroff** assembles text processed in other environments in different buffers.

In the following example, you process some text in environment 0 and then switch to environment 2. Any partial line collected in environment 0 when you switch to environment 2 waits patiently in the buffer until you return to environment 0 and issue the break command to flush the buffer. You then return to environment 2 and flush any partially filled line left when you restored environment 0. Enter the following into the file **ex11.r** and process it through **nroff**:

```
.ll 3i               \" set line length in ev 0
.po 2i               \" set page offset in ev 0
This is environment 0.
.ev 2                \" introduce ev 2
This is environment 2
.br                  \" flush ev 2 buffer
.ev                  \" return to ev 0
.br                  \" flush ev 0 buffer
```

As you can see, the order of the two sentences is reversed from the way you entered them. If you were to delete the **.br** commands after the texts in **ex10.r**, the output would be very badly affected.

Headers and Footers

A common use of environment switching is for the creation of header and footer macros. As the following example demonstrates, the length of title set by the **.lt** command is an environmental parameter. The following constructs header and footer macros that print strings of asterisks in the margins above and below the text; type it into your computer as **ex12.r**:

```
.wh 0 HD            \" set header trap
.wh |2.5i FO        \" set footer trap
.de HD              \" define header macro
.ev 1               \" define ev 1
.lt 5i              \" set title length to 5 inches
'sp 3v              \" move down three spaces
.tl '****''****'    \" define header title
'sp 2v              \" skip two more spaces
.ev                 \" pop environment
..
.de FO              \" define footer macro
'sp 2
.ev 1               \" push ev 1
.tl '****'%'****'   \" define footer title
.ev                 \" pop environment
'bp                 \" begin new page
..
.ll 4i              \" set line length in ev 0
.pl 3i              \" set page length
.in 1i              \" set indentation
.po 2i              \" set page offset
.de PP              \" define paragraph macro
.sp 1
.ti 0.5i            \" indent 1st line 1/2 inch
..
.PP
When in the course of human events ...
```

The following section explains why header and footer macros often use a different environment.

More About Fonts

As earlier described in some detail, **nroff** output includes representations for **boldface** and *italic* characters, in addition to the normal Roman characters. The visual appearance of boldface and italic characters depends on the device you use to print your **nroff** output.

If you want a word or a phrase to appear in boldface, enclose the word or phrase between **\fB** and **\fR**:

```
The last word of this sentence appears in \fBboldface\fR.
```

The sequence **\fB** tells **nroff** to print in boldface, whereas the sequence **\fR** tells **nroff** to return to the Roman font. Italics are used in a similar manner:

```
An entire phrase \fIappears in italics\fR.
```

To print more than a few words in a different font, you should use the *font* command **.ft:**

```
.ft I
Here is text you want to
appear in italics.
.ft R
```

The initial **.ft I** switches the print to italic font, and the concluding **.ft R** returns it to Roman font. As you might have suspected, the command **.ft B** switches to boldface.

You have two additional options when you use the **.ft** primitive. The command **.ft P** returns to the *previous* font. You can use **.ft P** within a macro or a string to return to the previous output font, even though you do not know which font was previously in effect. You can also use the sequence **\fP** to return to the previous font. The **.ft** primitive without an argument tells **nroff** to return to the Roman font.

In scripts that frequently change fonts, you should switch to a new environment for header and footer macros, in order to protect their font settings.

Diversions

The *diversion* is a powerful feature that allows you to suspend outputting lines until **nroff** has collected all of a block of text. For example, suppose you use **nroff** to format a chapter of a book. The chapter includes footnotes at various places in the text; you want **nroff** to collect these footnotes and print at the end of the chapter. To do this, **nroff** must store the processed footnote text somewhere until the end of the chapter, when you want it printed. Where do you store the text until the time comes for it to appear? To handle situations like this, **nroff** provides a *diversion* mechanism: you can *divert* text into temporary storage within a macro.

Diversion normally involves passing to a new environment to process the footnote without causing a break in the main environment. When the text of the diversion ends, **nroff** returns to the main environment, again without causing a break, and continues processing just as if the text of the note had never been in the input.

However, before you attempt to write a footnote macro, type the following text into the file **ex13.r**, and process it with **nroff**. This example illustrates the basic features of diversion. The example exchanges two paragraphs of text, so that **nroff** prints the second before the first.

```
.di X              \" divert the following to macro X
.sp
A soft answer turneth away wrath:
but grievous words stir up anger.
.br                \" send last line of paragraph to X
.di                \" end diversion
.sp
He that is slow to anger is better than the
mighty; and he that ruleth his spirit than he
that taketh a city.
.br
.sp
.X                 \" print the paragraph diverted to X
```

The new command here is the *divert* primitive **.di**. The command **.di X** tells **nroff** to divert the text that follows into macro **X**; the matching **.di** with no argument marks the end of the diversion.

The break is necessary before the end of the diversion because **nroff** diverts *processed* text into the macro. Without the break, **nroff** would not divert any partially filled line in its buffer to **X**; the last few words of diverted text might not form a complete line in the buffer, so **nroff** might not divert them. However, if you break the input before you end the diversion, **nroff** will also divert those last few words.

As you saw earlier, the **.br** command must be used to flush that environment's buffer before switching environments.

The next example, **ex14.r**, illustrates a similar point.

```
.br                \" clear buffer
testword           \" put 'testword' into buffer
.di X              \" divert to X
Piracy, n. Commerce without its folly-swaddles,
just as God made it.
.br                \" divert last line
.di                \" end diversion
.X                 \" print text in X
```

Here **nroff** diverts **testword** into **X** along with the text between **.di X** and **.di**. Why did this happen? The command **.di X** does *not* cause a break. Because you did not pass to a new environment in this example before you diverted, **nroff** formed the diversion text in the same buffer in which it stored **testword**. You did not break the input, so **nroff** appended the diverted text to **testword**.

To make sure **nroff** diverts only text between **.di X** and **.di** into **X**, do one of the following: If you want to process the diverted text within the current environment, empty the buffer by inserting the **.br** command before you start the diversion. If you switch to a new environment before starting the diversion, flush the buffer for the new environment before you begin to process diverted text.

Diverting processed text into a macro that already holds material will erase whatever had already been stored there. In some cases, such as the footnote example, you need to append information into the same macro. The *divert and append* variation **.da** of the diversion construction allows you to do so. The following example, **ex15.r**, demonstrates this command:

```
.ll 3i            \" set line length
.po 2i            \" set page offset
.de PP            \" define paragraph macro
.br
.sp 1
.ti 0.5i          \" indent first line 1/2 inch
..
.di X             \" divert the following into X
.PP
Litigation, n. A machine which you go into as a pig
and come out of as a sausage.
.br
.di               \" end diversion
.X                \" print what is in X
.br
.da X             \" divert and append material into X
.PP
Inventor, n. A person who makes an ingenious arrangement
of wheels, levers and springs, and believes it
civilization.
.br
.di               \" end diversion
.X                \" print what is now in X
```

In this example, you first diverted a single paragraph into the macro **X**. **nroff** stored in **X** the *processed* paragraph; in other words, the command line **.PP** is *not* stored in **X**; its *output* is. When you invoke **X** with the command line **.X**, **nroff** takes the processed text in **X** as input. To **nroff,** there is no difference between processed text and unprocessed text as input: it processes the contents of **X** in the current environment, just like any other text. Therefore, **nroff** processes diverted text *twice*: first when it stores the text within the macro, and again when you invoke the macro.

The fact that **nroff** processes diverted text twice can cause problems if you are not careful. Fortunately, nothing strange happens in the example above. You store a processed paragraph with lines three inches long in **X**. When you invoke **X**, the line length is three inches. Because each line in **X** is already exactly three inches long, nothing happens to it when reprocessed; the layout of the output paragraph is unchanged.

But now, consider what happens in the following example, **ex16.r**:

```
.ll 3i            \" set line length
.po 2i            \" set page offset
.de PP            \" define PP macro
.sp 1
.ti 0.5i          \" indent first line 1/2 inch
..
.di X             \" divert following into X
.ev 2             \" push environment 2
.ll 4i            \" set line length to 4 inches
.PP
Justice, n. A commodity which in a more or less
adulterated condition, the State sells to the
citizen as a reward for his allegiance, taxes
and personal service.
.br
.ev               \" pop environment (return to ev 0)
.di               \" end diversion
.X
```

A pargraph processed in environment 0 in this example has three-inch lines; you want your diverted paragraph to have four-inch lines. However, when you print the diverted paragraph with the command line **.X**, what happened?

nroff did *not* print four-inch lines. The four-line text lines set in environment 2 were reprocessed into three-inch lines when the diversion macro is called in environment 0.

There are two ways to prevent such disasters. First, if you wish to invoke **X** in the main environment, use no-fill mode:

```
.nf                \" begin no-fill mode
.X
.fi                \" return to fill mode
```

In no-fill mode, **nroff** outputs lines of input exactly as it receives them, so it keeps four-inch lines four inches long and does not change the format of the diverted text. The second strategy is to return to environment 2 and then invoke **X**; again, the format of the diverted paragraph does not change, because the line length in environment 2 is four inches.

```
.ev 2              \" push environment 2
.X
.ev                \" restore original environment
```

A Footnote Macro

The footnote macro that follows does not print notes at the bottom of each page; rather, it prints everything at the end of the chapter. In the processed text, number register **Fn** is used to keep track of the footnote number; the footnote number will be printed in square brackets where the footnote originally appeared in the text.

Type this macro into the file **ex17.r**. If you wish to use it in your text processing, transfer it to the directory **/usr/lib** under the name **tmac.fn**. Then, whenever you wish to use this macro, be sure to include the option

```
-mfn
```

when you invoke **nroff**:

```
.de FN             \" define macro FN
[\\n+(Fn]          \" print footnote no. in main text
.ev 1              \" push environment 1
.da Z              \" divert and append following into Z
.sp
\\n(Fn. \\$2, \\fI\\$1\\fR,
    \\$3, \\$4.    \" format & print footnote in Z
.br                \" flush diversion buffer
.di                \" end diversion
.ev                \" pop environment (return to ev 0)
..
```

Note that requests to change fonts are preceded by double backslashes, because they are within a macro. The change to the italic font prints the first macro argument, which should be the title of the work, in italics. Number register **Fn** contains the number of the last footnote; you should initialize it with the command

```
.nr Fn 0 1
```

As shown above, each footnote entry in your text should have four arguments. In your input text, each footnote will look like this:

```
.FN "Personal narrative of a pilgrimage to\
El-Medinah and Mecca" "Richard F. Burton"\
London 1856.
```

When you print the diversion **.Z** at the end of the chapter, each footnote will be laid out as follows:

8.   Richard F. Burton,
        Personal narrative of a pilgrimage to
        El_Medinah and Mecca,
        London, 1856.

Command Line Options

In the previous sections, you learned how to control **nroff** by including *commands* in the input along with the *text*. You can also supply information in another way: on the command line you type to call **nroff**. Unlike the commands discussed above, this information is *not* part of the script you input into **nroff**.

You already know about some simple **nroff** command lines. For example, the command

```
nroff
```

forces **nroff** to accept input from the keyboard (sometimes called the *standard input)* and print output on the terminal (the *standard output).* Type **<ctrl-D>** (that is, hold down the **ctrl** key and type **D**) to exit from **nroff** if it is reading input from your terminal.

The command line

```
nroff script1.r
```

forces **nroff** to take accept input from the file **script1.r** instead of from your terminal, while the command

```
nroff -ms script.r
```

processes **script1.r** with the **-ms** macro package. You can also redirect **nroff** output to another file **target:**

```
nroff -ms script1.r >target
```

The general form of the **nroff** command line is:

nroff *[option ...] [file ...]*

This means that the command line consists of the **nroff** command, followed by zero or more *option*s, followed by zero or more *file*s. **nroff** processes each named *file* and prints the result on the standard output (the terminal, unless redirected). If no *file* argument is given, as in the first example above, **nroff** reads from the standard input.

Each *option* on the command line must begin with a hyphen '-' to distinguish it from a *file* specification. Using **nroff** with the **-ms** macro package is one example of entering an option. In general, the **-m** option takes the form

-**m***name*

which means the option consists of the characters **-m** immediately followed by a *name.* This tells **nroff** to process the macro package found in the COHERENT file

/usr/lib/tmac.*name*

For example, the **ms** macro package discussed in chapter 2 is in the file **/usr/lib/tmac.s,** whereashe **man** macro package used for the **man** command and to process manual pages is in the file **/usr/lib/tmac.an.**

Any macro packages that you customize for your own use should be stored in the directory **/usr/lib** under such a name if you wish to use them with the **-m***name* option.

The **-i** option tells **nroff** to read input from the standard input after processing each given *file.* This allows you to supply additional input interactively from your terminal.

The **-x** option tells **nroff** not to move to the bottom of the last output page when done. This is especially useful if you want to see the output on the screen of a CRT terminal.

The -**n***N* option sets the page number of the first output page to the number *N,* rather than starting at page 1. This is useful for processing large documents with input text in several files which **nroff** processes separately.

The -**r***XN* option sets the value of number register *X* to *N.* This option lets you initialize number registers when you invoke **nroff**.

The COHERENT system provides many useful features which can be helpful while you are using **nroff.** In particular, you can use a number of special characters. The *stop-output* and *start-output* characters, usually **<ctrl-S>** and **<ctrl-Q>,** stop and restart output on your terminal. The *interrupt* character, usually **<ctrl-C>**, interrupts program execution; you can use it to stop an **nroff** command if you typed the command line incorrectly. The *kill* character, usually **<ctrl-\>**, also terminates program execution. Some COHERENT systems use different characters than those mentioned above; consult *Using the COHERENT System* for details.

For Further Information

The Lexicon entry for **nroff** summarizes its primitives, dedicated number registers, escape sequences, and command-line options.

For example, this manual was typeset by COHERENT **troff**. The program **troff** also performs text formatting. Unlike **nroff**, however, **troff** produces proportionally spaced output that can be printed on printers that support the Hewlett-Packard Page Control Language (including the LaserJet and DeskJet families of printers) or on any printer that implements the PostScript page-control language. This manual (including the positioning of the fancy capital letters and ornaments) was typeset by **troff** under the COHERENT system. See the Lexicon entry for **troff** for details on how to use this command.

The Lexicon also has entries for two macros packages that are included with the COHERENT system: **man** which produces manual pages similar to those that appear in the Lexicon; and **ms**, which performs formatting somewhat similar to that seen in this tutorial. You will find that these two packages already perform practically all of the formatting tasks that you will commonly need to do.

The error messages generated by **nroff** are given in the appendix at the rear of this manual.