第二十二章 格式化输出

In this chapter, we continue our look at text related tools, focusing on programs that are used to format text output, rather than changing the text itself. These tools are often used to prepare text for eventual printing, a subject that we will cover in the next chapter. The programs that we will cover in this chapter include:

在这章中，我们继续着手于文本相关的工具，关注那些用来格式化输出的程序，而不是改变文本自身。 这些工具通常让文本准备就绪打印，这是我们在下一章会提到的。我们在这章中会提到的工具有：

* nl – Number lines
* nl – 添加行号
* fold – Wrap each line to a specified length
* fold – 限制文件列宽
* fmt – A simple text formatter
* fmt – 一个简单的文本格式转换器
* pr – Prepare text for printing
* pr – 让文本为打印做好准备
* printf – Format and print data
* printf – 格式化数据并打印出来
* groff – A document formatting system
* groff – 一个文件格式系统

**简单的格式化工具**

We’ll look at some of the simple formatting tools first. These are mostly single purpose programs, and a bit unsophisticated in what they do, but they can be used for small tasks and as parts of pipelines and scripts.

我们将先着眼于一些简单的格式工具。他们都是功能单一的程序，并且做法有一点单纯， 但是他们能被用于小任务并且作为脚本和管道的一部分 。

**nl - 添加行号**

The nl program is a rather arcane tool used to perform a simple task. It numbers lines. In its simplest use, it resembles cat -n:

nl 程序是一个相当神秘的工具，用作一个简单的任务。它添加文件的行数。在它最简单的用途中，它相当于 cat -n:

[me@linuxbox ~]$ nl distros.txt | head

Like cat, nl can accept either multiple files as command line arguments, or standard input. However, nl has a number of options and supports a primitive form of markup to allow more complex kinds of numbering.

像 cat，nl 既能接受多个文件作为命令行参数，也能标准输出。然而，nl 有一个相当数量的选项并支持一个简单的标记方式去允许更多复杂的方式的计算。

nl supports a concept called “logical pages” when numbering. This allows nl to reset (start over) the numerical sequence when numbering. Using options, it is possible to set the starting number to a specific value and, to a limited extent, its format. A logical page is further broken down into a header, body, and footer. Within each of these sections, line numbering may be reset and/or be assigned a different style. If nl is given multiple files, it treats them as a single stream of text. Sections in the text stream are indicated by the presence of some rather odd-looking markup added to the text:

nl 在计算文件行数的时候支持一个叫“逻辑页面”的概念 。这允许nl在计算的时候去重设（再一次开始）可数的序列。用到那些选项 的时候，可以设置一个特殊的开始值，并且在某个可限定的程度上还能设置它的格式。一个逻辑页面被进一步分为 header,body 和 footer 这样的元素。在每一个部分中，数行数可以被重设，并且/或被设置成另外一个格式。如果nl同时处理多个文件，它会把他们当成一个单一的 文本流。文本流中的部分被一些相当古怪的标记的存在加进了文本：

|  |  |
| --- | --- |
| *Table 22-1: nl Markup* | |
| MarkUp | Meaning |
| \:\:\: | Start of logical page header |
| \:\: | Start of logical page body |
| \: | Start of logical page footer |

|  |  |
| --- | --- |
| *Table 22-1: nl 标记* | |
| 标记 | 含义 |
| \:\:\: | 逻辑页页眉开始处 |
| \:\: | 逻辑页主体开始处 |
| \: | 逻辑页页脚开始处 |

Each of the above markup elements must appear alone on its own line. After processing a markup element, nl deletes it from the text stream.

每一个上述的标记元素肯定在自己的行中独自出现。在处理完一个标记元素之后，nl 把它从文本流中删除。

Here are the common options for nl:

这里有一些常用的 nl 选项：

|  |  |
| --- | --- |
| *Table 22-2: Common nl Options* | |
| Option | Meaning |
| -b style | Set body numbering to style, where style is one of the following:  a = number all lines  t = number only non-blank lines. This is the default.  n = none  pregexp = number only lines matching basic regular expression regexp. |
| -f style | Set footer numbering to style. Default is n (none). |
| -h style | Set header numbering to style. Default is n (none). |
| -i number | Set page numbering increment to number. Default is one. |
| -n format | Sets numbering format to format, where format is:  ln = left justified, without leading zeros.  rn = right justified, without leading zeros. This is the default.  rz = right justified, with leading zeros. |
| -p | Do not reset page numbering at the beginning of each logical page. |
| -s string | Add string to the end of each line number to create a separator.Default is a single tab character. |
| -v number | Set first line number of each logical page to number. Default is one. |
| -w width | Set width of the line number field to width. Default is six. |

|  |  |
| --- | --- |
| *表格 22-2: 常用 nl 选项* | |
| 选项 | 含义 |
| -b style | 把 body 按被要求方式数行，可以是以下方式：  a = 数所有行  t = 数非空行。这是默认设置。  n = 无  pregexp = 只数那些匹配了正则表达式的行 |
| -f style | 将 footer 按被要求设置数。默认是无 |
| -h style | 将 header 按被要求设置数。默认是 |
| -i number | 将页面增加量设置为数字。默认是一。 |
| -n format | 设置数数的格式，格式可以是：  ln = 左偏，没有前导零。  rn = 右偏，没有前导零。  rz = 右偏，有前导零。 |
| -p | 不要在没一个逻辑页面的开始重设页面数。 |
| -s string | 在没一个行的末尾加字符作分割符号。默认是单个的 tab。 |
| -v number | 将每一个逻辑页面的第一行设置成数字。默认是一。 |
| -w width | 将行数的宽度设置，默认是六。 |

Admittedly, we probably won’t be numbering lines that often, but we can use nl to look at how we can combine multiple tools to perform more complex tasks. We will build on our work in the previous chapter to produce a Linux distributions report. Since we will be using nl, it will be useful to include its header/body/footer markup. To do this, we will add it to the sed script from the last chapter. Using our text editor, we will change the script as follows and save it as distros-nl.sed:

坦诚的说，我们大概不会那么频繁地去数行数，但是我们能用 nl 去查看我们怎么将多个工具结合在一个去完成更复杂的任务。 我们将在之前章节的基础上做一个 Linux 发行版的报告。因为我们将使用 nl，包含它的 header/body/footer 标记将会十分有用。 我们将把它加到上一章的 sed 脚本来做这个。使用我们的文本编辑器，我们将脚本改成一下并且把它保存成 distros-nl.sed:

# sed script to produce Linux distributions report

1 i\

\\:\\:\\:\

\

Linux Distributions Report\

\

Name

Ver. Released\

----

---- --------\

\\:\\:

s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/

$ i\

\\:\

\

End Of Report

The script now inserts the nl logical page markup and adds a footer at the end of the report. Note that we had to double up the backslashes in our markup, because they are normally interpreted as an escape character by sed.

这个脚本现在加入了 nl 的逻辑页面标记并且在报告的最后加了一个 footer。记得我们在我们的标记中必须两次使用反斜杠， 因为他们通常被 sed 解释成一个转义字符。

Next, we’ll produce our enhanced report by combining sort, sed, and nl:

下一步，我们将结合 sort, sed, nl 来生成我们改进的报告：

[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-nl.sed | nl

Linux Distributions Report

Name Ver. Released

---- ---- --------

1 Fedora 5 2006-03-20

2 Fedora 6 2006-10-24

3 Fedora 7 2007-05-31

4 Fedora 8 2007-11-08

5 Fedora 9 2008-05-13

6 Fedora 10 2008-11-25

7 SUSE 10.1 2006-05-11

8 SUSE 10.2 2006-12-07

9 SUSE 10.3 2007-10-04

10 SUSE 11.0 2008-06-19

11 Ubuntu 6.06 2006-06-01

12 Ubuntu 6.10 2006-10-26

13 Ubuntu 7.04 2007-04-19

14 Ubuntu 7.10 2007-10-18

15 Ubuntu 8.04 2008-04-24

End Of Report

Our report is the result of our pipeline of commands. First, we sort the list by distribution name and version (fields one and two), then we process the results with sed, adding the report header (including the logical page markup for nl) and footer. Finally, we process the result with nl, which, by default, only numbers the lines of the text stream that belong to the body section of the logical page.

我们的报告是一串命令的结果，首先，我们给名单按发行版本和版本号（表格1和2处）进行排序，然后我们用 sed 生产结果， 增加了 header（包括了为 nl 增加的逻辑页面标记）和 footer。最后，我们按默认用 nl 生成了结果，只数了属于逻辑页面的 body 部分的 文本流的行数。

We can repeat the command and experiment with different options for nl. Some interesting ones are:

我们能够重复命令并且实验不同的 nl 选项。一些有趣的方式：

nl -n rz

and

和

nl -w 3 -s ' '

**fold - 限制文件行宽**

Folding is the process of breaking lines of text at a specified width. Like our other commands, fold accepts either one or more text files or standard input. If we send fold a simple stream of text, we can see how it works:

折叠是将文本的行限制到特定的宽的过程。像我们的其他命令，fold 接受一个或多个文件及标准输入。如果我们将 一个简单的文本流 fold，我们可以看到它工作的方式：

[me@linuxbox ~]$ echo "The quick brown fox jumped over the lazy dog."

| fold -w 12

The quick br

own fox jump

ed over the

lazy dog.

Here we see fold in action. The text sent by the echo command is broken into segments specified by the -w option. In this example, we specify a line width of twelve characters. If no width is specified, the default is eighty characters. Notice how the lines are broken regardless of word boundaries. The addition of the -s option will cause fold to break the line at the last available space before the line width is reached:

这里我们看到了 fold 的行为。这个用 echo 命令发送的文本用 -w 选项分解成块。在这个例子中，我们设定了行宽为12个字符。 如果没有字符设置，默认是80。注意到文本行不会因为单词边界而不会被分解。增加的 -s 选项将让 fold 分解到最后可用的空白 字符，即会考虑单词边界。

[me@linuxbox ~]$ echo "The quick brown fox jumped over the lazy dog."

| fold -w 12 -s

The quick

brown fox

jumped over

the lazy

dog.

**fmt - 一个简单的文本格式器**

The fmt program also folds text, plus a lot more. It accepts either files or standard input and performs paragraph formatting on the text stream. Basically, it fills and joins lines in text while preserving blank lines and indentation.

fmt 程序同样折叠文本，外加很多功能。它接受文本或标准输入并且在文本流上呈现照片转换。基础来说，他填补并且将文本粘帖在 一起并且保留了空白符和缩进。

To demonstrate, we’ll need some text. Let’s lift some from the fmt info page:

为了解释，我们将需要一些文本。让我们抄一些 fmt 主页上的东西吧：

‘fmt’ reads from the specified FILE arguments (or standard input if

none are given), and writes to standard output.

By default, blank lines, spaces between words, and indentation are

preserved in the output; successive input lines with different

indentation are not joined; tabs are expanded on input and introduced on

output.

‘fmt’ prefers breaking lines at the end of a sentence, and tries to

avoid line breaks after the first word of a sentence or before the last

word of a sentence. A "sentence break" is defined as either the end of

a paragraph or a word ending in any of ‘.?!’, followed by two spaces or

end of line, ignoring any intervening parentheses or quotes. Like TeX,

‘fmt’ reads entire “paragraphs” before choosing line breaks; the

algorithm is a variant of that given by Donald E. Knuth and Michael F.

Plass in “Breaking Paragraphs Into Lines”, ‘Software—Practice &

Experience’ 11, 11 (November 1981), 1119–1184.

We’ll copy this text into our text editor and save the file as fmt-info.txt. Now, let’s say we wanted to reformat this text to fit a fifty character wide column. We could do this by processing the file with fmt and the -w option:

我们将把这段文本复制进我们的文本编辑器并且保存文件名为 fmt-info.txt。现在，让我们重新格式这个文本并且让它成为一个50 个字符宽的项目。我们能用 -w 选项对文件进行处理：

[me@linuxbox ~]$ fmt -w 50 fmt-info.txt | head

'fmt' reads from the specified FILE arguments

(or standard input if

none are given), and writes to standard output.

By default, blank lines, spaces between words,

and indentation are

preserved in the output; successive input lines

with different indentation are not joined; tabs

are expanded on input and introduced on output.

Well, that’s an awkward result. Perhaps we should actually read this text, since it explains what’s going on:

好，这真是一个奇怪的结果。大概我们应该认真的阅读这段文本，因为它恰好解释了发生了什么：

“By default, blank lines, spaces between words, and indentation are preserved in the output; successive input lines with different indentation are not joined; tabs are expanded on input and introduced on output.”

默认情况下，输出会保留空行，单词之间的空格，和缩进；持续输入的具有不同缩进的文本行不会连接在一起；tab 字符在输入时会展开，输出时复原 。

So, fmt is preserving the indentation of the first line. Fortunately, fmt provides an option to correct this:

所以，fmt 会保留第一行的缩进。幸运的是，fmt 提供了一个选项来更正这种行为：

Much better. By adding the -c option, we now have the desired result.

好多了。通过添加 -c 选项，现在我们得到了所期望的结果。

fmt has some interesting options:

fmt 有一些有意思的选项：

The -p option is particularly interesting. With it, we can format selected portions of a file, provided that the lines to be formatted all begin with the same sequence of characters. Many programming languages use the pound sign (#) to indicate the beginning of a comment and thus can be formatted using this option. Let’s create a file that simulates a program that uses comments:

这个 -p 选项尤为有趣。通过它，我们可以格式文件选中的部分，通过在开头使用一样的符号。 很多编程语言使用锚标记（#）去提醒注释的开始，而且它可以通过这个选项来被格式。让我们创建一个有用到注释的程序。

[me@linuxbox ~]$ cat > fmt-code.txt

# This file contains code with comments.

# This line is a comment.

# Followed by another comment line.

# And another.

This, on the other hand, is a line of code.

And another line of code.

And another.

Our sample file contains comments which begin the string “# “ (a # followed by a space) and lines of “code” which do not. Now, using fmt, we can format the comments and leave the code untouched:

我们的示例文件包含了用 “#” 开始的注释（一个 # 后跟着一个空白符）和代码。现在，使用 fmt，我们能格式注释并且 不让代码被触及。

[me@linuxbox ~]$ fmt -w 50 -p '# ' fmt-code.txt

# This file contains code with comments.

# This line is a comment. Followed by another

# comment line. And another.

This, on the other hand, is a line of code.

And another line of code.

And another.

Notice that the adjoining comment lines are joined, while the blank lines and the lines that do not begin with the specified prefix are preserved.

注意相邻的注释行被合并了，空行和非注释行被保留了。

**pr – 格式化打印文本**

The pr program is used to paginate text. When printing text, it is often desirable to separate the pages of output with several lines of whitespace, to provide a top and bottom margin for each page. Further, this whitespace can be used to insert a header and footer on each page.

pr 程序用来把文本分页。当打印文本的时候，经常希望用几个空行把输出的页面

We’ll demonstrate pr by formatting our distros.txt file into a series of very short pages (only the first two pages are shown):

[me@linuxbox ~]$ pr -l 15 -w 65 distros.txt

2008-12-11 18:27 distros.txt Page 1

SUSE 10.2 12/07/2006

Fedora 10 11/25/2008

SUSE 11.0 06/19/2008

Ubuntu 8.04 04/24/2008

Fedora 8 11/08/2007

2008-12-11 18:27 distros.txt Page 2

SUSE 10.3 10/04/2007

Ubuntu 6.10 10/26/2006

Fedora 7 05/31/2007

Ubuntu 7.10 10/18/2007

Ubuntu 7.04 04/19/2007

In this example, we employ the -l option (for page length) and the -w option (page width) to define a “page” that is 65 columns wide and 15 lines long. pr paginates the contents of the distros.txt file, separates each page with several lines of whitespace and creates a default header containing the file modification time, filename, and page number. The pr program provides many options to control page layout. We’ll take a look at more of them in the next chapter.

**printf – Format And Print Data**

Unlike the other commands in this chapter, the printf command is not used for pipelines (it does not accept standard input) nor does it find frequent application directly on the command line (it’s mostly used in scripts). So why is it important? Because it is so widely used.

printf (from the phrase “print formatted”) was originally developed for the C programming language and has been implemented in many programming languages including the shell. In fact, in bash, printf is a builtin. printf works like this:

printf “format” arguments

The command is given a string containing a format description which is then applied to a list of arguments. The formatted result is sent to standard output. Here is a trivial example:

[me@linuxbox ~]$ printf "I formatted the string: %s\n" foo

I formatted the string: foo

The format string may contain literal text (like “I formatted the string:”), escape sequences (such as \n, a newline character), and sequences beginning with the % character, which are called conversion specifications. In the example above, the conversion specification %s is used to format the string “foo” and place it in the command’s output. Here it is again:

[me@linuxbox ~]$ printf "I formatted '%s' as a string.\n" foo

I formatted 'foo' as a string.

As we can see, the %s conversion specification is replaced by the string “foo” in the command’s output. The s conversion is used to format string data. There are other specifiers for other kinds of data. This table lists the commonly used data types:

|  |  |
| --- | --- |
| *Table 22-5: printf Conversion Specification Components* | |
| Component | Description |
| d | Format a number as a signed decimal integer. |
| f | Format and output a floating point number. |
| o | Format an integer as an octal number. |
| s | Format a string. |
| x | Format an integer as a hexadecimal number using lowercase a-f where needed. |
| X | Same as x but use uppercase letters. |
| % | Print a literal % symbol (i.e., specify “%%”) |

We’ll demonstrate the effect each of the conversion specifiers on the string “380”:

[me@linuxbox ~]$ printf "%d, %f, %o, %s, %x, %X\n" 380 380 380 380

380 380

380, 380.000000, 574, 380, 17c, 17C

Since we specified six conversion specifiers, we must also supply six arguments for printf to process. The six results show the effect of each specifier. Several optional components may be added to the conversion specifier to adjust its output. A complete conversion specification may consist of the following:

%[flags][width][.precision]conversion\_specification

Multiple optional components, when used, must appear in the order specified above to be properly interpreted. Here is a description of each:

|  |  |
| --- | --- |
| *Table 22-5: printf Conversion Specification Components* | |
| Component | Description |
| flags | There are five different flags:  # – Use the “alternate format” for output. This varies by data type. For o (octal number) conversion, the output is prefixed with 0. For x and X (hexadecimal number) conversions, the output is prefixed with 0x or 0X respectively.  0–(zero) Pad the output with zeros. This means that the field will be filled with leading zeros, as in “000380”.  - – (dash) Left-align the output. By default, printf right-aligns output.  ‘ ’ – (space) Produce a leading space for positive numbers.  + – (plus sign) Sign positive numbers. By default, printf only signs negative numbers. |
| width | A number specifying the minimum field width. |
| .precision | For floating point numbers, specify the number of digits of precision to be output after the decimal point. For string conversion, precision specifies the number of characters to output. |

Here are some examples of different formats in action:

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 22-6: print Conversion Specification Examples* | | | |
| Argument | Format | Result | Notes |
| 380 | "%d" | 380 | Simple formatting of an integer. |
| 380 | "%#x" | 0x17c | Integer formatted as a hexadecimal number using the “alternate format” flag. |
| 380 | "%05d" | 00380 | Integer formatted with leading zeros (padding) and a minimum field width of five characters. |
| 380 | "%05.5f" | 380.00000 | Number formatted as a floating point number with padding and five decimal places of precision. Since the specified minimum field width (5) is less than the actual width of the formatted number, the padding has no effect. |
| 380 | "%010.5f" | 0380.00000 | By increasing the minimum field width to 10 the padding is now visible. |
| 380 | "%+d" | +380 | The + flag signs a positive number. |
| 380 | "%-d" | 380 | The - flag left aligns the formatting. |
| abcdefghijk | "%5s" | abcedfghijk | A string formatted with a minimum field width. |
| abcdefghijk | "%d" | abcde | By applying precision to a string, it is truncated. |

Again, printf is used mostly in scripts where it is employed to format tabular data, rather than on the command line directly. But we can still show how it can be used to solve various formatting problems. First, let’s output some fields separated by tab characters:

[me@linuxbox ~]$ printf "%s\t%s\t%s\n" str1 str2 str3

str1 str2 str3

By inserting \t (the escape sequence for a tab), we achieve the desired effect. Next, some numbers with neat formatting:

[me@linuxbox ~]$ printf "Line: %05d %15.3f Result: %+15d\n" 1071

3.14156295 32589

Line: 01071 3.142 Result: +32589

This shows the effect of minimum field width on the spacing of the fields. Or how about formatting a tiny web page:

[me@linuxbox ~]$ printf "<html>\n\t<head>\n\t\t<title>%s</title>\n

\t</head>\n\t<body>\n\t\t<p>%s</p>\n\t</body>\n</html>\n" "Page Tit

le" "Page Content"

<html>

<head>

<title>Page Title</title>

</head>

<body>

<p>Page Content</p>

</body>

</html>

**Document Formatting Systems**

So far, we have examined the simple text-formatting tools. These are good for small, sim-

ple tasks, but what about larger jobs? One of the reasons that Unix became a popular operating system among technical and scientific users (aside from providing a powerful multitasking, multiuser environment for all kinds of software development) is that it offered tools that could be used to produce many types of documents, particularly scientific and academic publications. In fact, as the GNU documentation describes, document preparation was instrumental to the development of Unix:

The first version of UNIX was developed on a PDP-7 which was sitting around Bell Labs. In 1971 the developers wanted to get a PDP-11 for further work on the operating system. In order to justify the cost for this system, they proposed that they would implement a document formatting system for the AT&T patents division. This first formatting program was a reimplementation of McIllroy’s `roff’, written by J. F. Ossanna.

Two main families of document formatters dominate the field: those descended from the original roff program, including nroff and troff, and those based on Donald Knuth’s TEX (pronounced “tek”) typesetting system. And yes, the dropped “E” in the middle is part of its name.

The name “roff” is derived from the term “run off” as in, “I’ll run off a copy for you.” The nroff program is used to format documents for output to devices that use monospaced fonts, such as character terminals and typewriter-style printers. At the time of its introduction, this included nearly all printing devices attached to computers. The later troff program formats documents for output on typesetters, devices used to produce “camera-ready” type for commercial printing. Most computer printers today are able to simulate the output of typesetters. The roff family also includes some other programs that are used to prepare portions of documents. These include eqn (for mathematical equations) and tbl (for tables).

The TEX system (in stable form) first appeared in 1989 and has, to some degree, displaced troff as the tool of choice for typesetter output. We won’t be covering TEX here, due both to its complexity (there are entire books about it) and to the fact that it is not installed by default on most modern Linux systems.

Tip: For those interested in installing TEX, check out the texlive package which can be found in most distribution repositories, and the LyX graphical content editor.

**groff**

groff is a suite of programs containing the GNU implementation of troff. It also includes a script that is used to emulate nroff and the rest of the roff family as well.

While roff and its descendants are used to make formatted documents, they do it in a way that is rather foreign to modern users. Most documents today are produced using word processors that are able to perform both the composition and layout of a document in a single step. Prior to the advent of the graphical word processor, documents were often produced in a two-step process involving the use of a text editor to perform composition, and a processor, such as troff, to apply the formatting. Instructions for the formatting program were embedded into the composed text through the use of a markup language. The modern analog for such a process is the web page, which is composed using a text editor of some kind and then rendered by a web browser using HTML as the markup language to describe the final page layout.

We’re not going to cover groff in its entirety, as many elements of its markup language deal with rather arcane details of typography. Instead we will concentrate on one of its macro packages that remains in wide use. These macro packages condense many of its low-level commands into a smaller set of high-level commands that make using groff much easier.

For a moment, let’s consider the humble man page. It lives in the /usr/share/man directory as a gzip compressed text file. If we were to examine its uncompressed contents, we would see the following (the man page for ls in section 1 is shown):

[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | head

.\" DO NOT MODIFY THIS FILE! It was generated by help2man 1.35.

.TH LS "1" "April 2008" "GNU coreutils 6.10" "User Commands"

.SH NAME

ls \- list directory contents

.SH SYNOPSIS

.B ls

[\fIOPTION\fR]... [\fIFILE\fR]...

.SH DESCRIPTION

.\" Add any additional description here

.PP

Compared to the man page in its normal presentation, we can begin to see a correlation between the markup language and its results:

[me@linuxbox ~]$ man ls | head

LS(1) User Commands LS(1)

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

The reason this is of interest is that man pages are rendered by groff, using the mandoc macro package. In fact, we can simulate the man command with the following pipeline:

[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc -T

ascii | head

LS(1) User Commands LS(1)

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

Here we use the groff program with the options set to specify the mandoc macro package and the output driver for ASCII. groff can produce output in several formats. If no format is specified, PostScript is output by default:

[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc |

head

%!PS-Adobe-3.0

%%Creator: groff version 1.18.1

%%CreationDate: Thu Feb 5 13:44:37 2009

%%DocumentNeededResources: font Times-Roman

%%+ font Times-Bold

%%+ font Times-Italic

%%DocumentSuppliedResources: procset grops 1.18 1

%%Pages: 4

%%PageOrder: Ascend

%%Orientation: Portrait

We briefly mentioned PostScript in the previous chapter, and will again in the next chapter. PostScript is a page description language that is used to describe the contents of a printed page to a typesetter-like device. If we take the output of our command and store it to a file (assuming that we are using a graphical desktop with a Desktop directory):

[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc >

~/Desktop/foo.ps

An icon for the output file should appear on the desktop. By double-clicking the icon, a page viewer should start up and reveal the file in its rendered form:

Figure 4: Viewing PostScript Output With A Page Viewer In GNOME

What we see is a nicely typeset man page for ls! In fact, it’s possible to convert the Post- Script file into a PDF (Portable Document Format) file with this command:

[me@linuxbox ~]$ ps2pdf ~/Desktop/foo.ps ~/Desktop/ls.pdf

The ps2pdf program is part of the ghostscript package, which is installed on most Linux systems that support printing.

Tip: Linux systems often include many command line programs for file format conversion. They are often named using the convention of format2format. Try using the command

ls /usr/bin/\*[[:alpha:]]2[[:alpha:]]\*

to identify them. Also try searching for programs named formattoformat.

For our last exercise with groff, we will revisit our old friend distros.txt once more. This time, we will use the tbl program which is used to format tables to typeset our list of Linux distributions. To do this, we are going to use our earlier sed script to add markup to a text stream that we will feed to groff.

First, we need to modify our sed script to add the necessary requests that tbl requires. Using a text editor, we will change distros.sed to the following:

# sed script to produce Linux distributions report

1 i\

.TS\

center box;\

cb s s\

cb cb cb\

l n c.\

Linux Distributions Report\

=\

Name Version Released\

\_

s/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/

$ a\

.TE

Note that for the script to work properly, care must been taken to see that the words “Name Version Released” are separated by tabs, not spaces. We’ll save the resulting file as distros-tbl.sed. tbl uses the .TS and .TE requests to start and end the table. The rows following the .TS request define global properties of the table which, for our example, are centered horizontally on the page and surrounded by a box. The remaining lines of the definition describe the layout of each table row. Now, if we run our reportgenerating pipeline again with the new sed script, we’ll get the following :

[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-tbl

.sed | groff -t -T ascii 2>/dev/null

+------------------------------+

| Linux Distributions Report |

+------------------------------+

| Name Version Released |

+------------------------------+

|Fedora 5 2006-03-20 |

|Fedora 6 2006-10-24 |

|Fedora 7 2007-05-31 |

|Fedora 8 2007-11-08 |

|Fedora 9 2008-05-13 |

|Fedora 10 2008-11-25 |

|SUSE 10.1 2006-05-11 |

|SUSE 10.2 2006-12-07 |

|SUSE 10.3 2007-10-04 |

|SUSE 11.0 2008-06-19 |

|Ubuntu 6.06 2006-06-01 |

|Ubuntu 6.10 2006-10-26 |

|Ubuntu 7.04 2007-04-19 |

|Ubuntu 7.10 2007-10-18 |

|Ubuntu 8.04 2008-04-24 |

|Ubuntu 8.10 2008-10-30 |

+------------------------------+

Adding the -t option to groff instructs it to pre-process the text stream with tbl. Likewise, the -T option is used to output to ASCII rather than the default output medium, PostScript.

The format of the output is the best we can expect if we are limited to the capabilities of a terminal screen or typewriter-style printer. If we specify PostScript output and graphically view the resulting output, we get a much more satisfying result:

[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-tbl

.sed | groff -t > ~/Desktop/foo.ps

Figure 5: Viewing The Finished Table

**Summing Up**

Given that text is so central to the character of Unix-like operating systems, it makes sense that there would be many tools that are used to manipulate and format text. As we have seen, there are! The simple formatting tools like fmt and pr will find many uses in scripts that produce short documents, while groff (and friends) can be used to write books. We may never write a technical paper using command line tools (though there are many people who do!), but it’s good to know that we could.

**Further Reading**

* groff User’s Guide

<http://www.gnu.org/software/groff/manual/>

* Writing Papers With nroff Using -me:

<http://docs.freebsd.org/44doc/usd/19.memacros/paper.pdf>

* -me Reference Manual:

<http://docs.freebsd.org/44doc/usd/20.meref/paper.pdf>

* Tbl – A Program To Format Tables:

<http://plan9.bell-labs.com/10thEdMan/tbl.pdf>

* And, of course, try the following articles at Wikipedia:

<http://en.wikipedia.org/wiki/TeX>

<http://en.wikipedia.org/wiki/Donald_Knuth>

<http://en.wikipedia.org/wiki/Typesetting>