第二十一章 文本处理

All Unix-like operating systems rely heavily on text files for several types of data storage. So it makes sense that there are many tools for manipulating text. In this chapter, we will look at programs that are used to “slice and dice” text. In the next chapter, we will look at more text processing, focusing on programs that are used to format text for printing and other kinds of human consumption.

所有类 Unix 的操作系统都非常依赖于被用于几种数据类型存储的文本文件。所以这很有道理， 有许多用于处理文本的工具。在这一章中，我们将看一些被用来“切割”文本的程序。在下一章中， 我们将查看更多的文本处理程序，但主要集中于文本格式化输出程序和其它一些人们需要的工具。

This chapter will revisit some old friends and introduce us to some new ones:

这一章会重新拜访一些老朋友，并且会给我们介绍一些新朋友：

* cat – Concatenate files and print on the standard output
* cat – 连接文件并且打印到标准输出
* sort – Sort lines of text files
* sort – 给文本行排序
* uniq – Report or omit repeated lines
* uniq – 报告或者省略重复行
* cut – Remove sections from each line of files
* cut – 从每行中删除文本区域
* paste – Merge lines of files
* paste – 合并文件文本行
* join – Join lines of two files on a common field
* join – 基于某个共享字段来联合两个文件的文本行
* comm – Compare two sorted files line by line
* comm – 逐行比较两个有序的文件
* diff – Compare files line by line
* diff – 逐行比较文件
* patch – Apply a diff file to an original
* patch – 给原始文件打补丁
* tr – Translate or delete characters
* tr – 翻译或删除字符
* sed – Stream editor for filtering and transforming text
* sed – 用于筛选和转换文本的流编辑器
* aspell – Interactive spell checker
* aspell – 交互式拼写检查器

**文本应用程序**

So far, we have learned a couple of text editors (nano and vim), looked a bunch of configuration files, and have witnessed the output of dozens of commands, all in text. But what else is text used for? For many things, it turns out.

到目前为止，我们已经知道了一对文本编辑器（nano 和 vim），看过一堆配置文件，并且目睹了 许多命令的输出都是文本格式。但是文本还被用来做什么？ 它可以做很多事情。

**文档**

Many people write documents using plain text formats. While it is easy to see how a small text file could be useful for keeping simple notes, it is also possible to write large documents in text format, as well. One popular approach is to write a large document in a text format and then use a markup language to describe the formatting of the finished document. Many scientific papers are written using this method, as Unix-based text processing systems were among the first systems that supported the advanced typographical layout needed by writers in technical disciplines.

许多人使用纯文本格式来编写文档。虽然很容易看到一个小的文本文件对于保存简单的笔记会 很有帮助，但是也有可能用文本格式来编写大的文档。一个流行的方法是先用文本格式来编写一个 大的文档，然后使用一种标记语言来描述已完成文档的格式。许多科学论文就是用这种方法编写的， 因为基于 Unix 的文本处理系统位于支持技术学科作家所需要的高级排版布局的一流系统之列。

**网页**

The world’s most popular type of electronic document is probably the web page. Web pages are text documents that use either HTML (Hypertext Markup Language) or XML (Extensible Markup Language) as markup languages to describe the document’s visual format.

世界上最流行的电子文档类型可能就是网页了。网页是文本文档，它们使用 HTML（超文本标记语言）或者是 XML （可扩展的标记语言）作为标记语言来描述文档的可视格式。

**电子邮件**

Email is an intrinsically text-based medium. Even non-text attachments are converted into a text representation for transmission. We can see this for ourselves by downloading an email message and then viewing it in less. We will see that the message begins with a header that describes the source of the message and the processing it received during its journey, followed by the body of the message with its content.

从本质上来说，email 是一个基于文本的媒介。为了传输，甚至非文本的附件也被转换成文本表示形式。 我们能看到这些，通过下载一个 email 信息，然后用 less 来浏览它。我们将会看到这条信息开始于一个标题， 其描述了信息的来源以及在传输过程中它接受到的处理，然后是信息的正文内容。

**打印输出**

On Unix-like systems, output destined for a printer is sent as plain text or, if the page contains graphics, is converted into a text format page description language known as PostScript, which is then sent to a program that generates the graphic dots to be printed.

在类 Unix 的系统中，输出会以纯文本格式发送到打印机，或者如果页面包含图形，其会被转换成 一种文本格式的页面描述语言，以 PostScript 著称，然后再被发送给一款能产生图形点阵的程序， 最后被打印出来。

**程序源码**

Many of the command line programs found on Unix-like systems were created to support system administration and software development, and text processing programs are no exception. Many of them are designed to solve software development problems. The reason text processing is important to software developers is that all software starts out as text. Source code, the part of the program the programmer actually writes, is always in text format.

在类 Unix 系统中会发现许多命令行程序被用来支持系统管理和软件开发，并且文本处理程序也不例外。 许多文本处理程序被设计用来解决软件开发问题。文本处理对于软件开发者来言至关重要是因为所有的软件 都起始于文本格式。源代码，程序员实际编写的一部分程序，总是文本格式。

**回顾一些老朋友**

Back in Chapter 7 (Redirection), we learned about some commands that are able to accept standard input in addition to command line arguments. We only touched on them briefly then, but now we will take a closer look at how they can be used to perform text processing.

回到第7章（重定向），我们已经知道一些命令除了接受命令行参数之外，还能够接受标准输入。 那时候我们只是简单地介绍了它们，但是现在我们将仔细地看一下它们是怎样被用来执行文本处理的。

**cat**

The cat program has a number of interesting options. Many of them are used to help better visualize text content. One example is the -A option, which is used to display non- printing characters in the text. There are times when we want to know if control characters are embedded in our otherwise visible text. The most common of these are tab characters (as opposed to spaces) and carriage returns, often present as end-of-line characters in MS-DOS style text files. Another common situation is a file containing lines of text with trailing spaces.

这个 cat 程序具有许多有趣的选项。其中许多选项用来帮助更好的可视化文本内容。一个例子是-A 选项， 其用来在文本中显示非打印字符。有些时候我们想知道是否控制字符嵌入到了我们的可见文本中。 最常用的控制字符是 tab 字符（而不是空格）和回车字符，在 MS-DOS 风格的文本文件中回车符经常作为 结束符出现。另一种常见情况是文件中包含末尾带有空格的文本行。

Let’s create a test file using cat as a primitive word processor. To do this, we’ll just enter the command cat (along with specifying a file for redirected output) and type our text, followed by Enter to properly end the line, then Ctrl-d, to indicate to cat that we have reached end-of-file. In this example, we enter a leading tab character and follow the line with some trailing spaces:

让我们创建一个测试文件，用 cat 程序作为一个简单的文字处理器。为此，我们将键入 cat 命令（随后指定了 用于重定向输出的文件），然后输入我们的文本，最后按下 Enter 键来结束这一行，然后按下组合键 Ctrl-d， 来指示 cat 程序，我们已经到达文件末尾了。在这个例子中，我们文本行的开头和末尾分别键入了一个 tab 字符以及一些空格。

[me@linuxbox ~]$ cat > foo.txt

The quick brown fox jumped over the lazy dog.

[me@linuxbox ~]$

Next, we will use cat with the -A option to display the text:

下一步，我们将使用带有-A 选项的 cat 命令来显示这个文本：

[me@linuxbox ~]$ cat -A foo.txt

^IThe quick brown fox jumped over the lazy dog. $

[me@linuxbox ~]$

As we can see in the results, the tab character in our text is represented by ^I. This is a common notation that means “Control-I” which, as it turns out, is the same as a tab character. We also see that a $ appears at the true end of the line, indicating that our text contains trailing spaces.

在输出结果中我们看到，这个 tab 字符在我们的文本中由^I 字符来表示。这是一种常见的表示方法，意思是 “Control-I”，结果证明，它和 tab 字符是一样的。我们也看到一个$字符出现在文本行真正的结尾处， 表明我们的文本包含末尾的空格。

MS-DOS Text Vs. Unix Text

MS-DOS 文本 Vs. Unix 文本

One of the reasons you may want to use cat to look for non-printing characters in text is to spot hidden carriage returns. Where do hidden carriage returns come from? DOS and Windows! Unix and DOS don’t define the end of a line the same way in text files. Unix ends a line with a linefeed character (ASCII 10) while MS-DOS and its derivatives use the sequence carriage return (ASCII 13) and linefeed to terminate each line of text.

可能你想用 cat 程序在文本中查看非打印字符的一个原因是发现隐藏的回车符。那么 隐藏的回车符来自于哪里呢？它们来自于 DOS 和 Windows！Unix 和 DOS 在文本文件中定义每行 结束的方式不相同。Unix 通过一个换行符（ASCII 10）来结束一行，然而 MS-DOS 和它的 衍生品使用回车（ASCII 13）和换行字符序列来终止每个文本行。

There are a several ways to convert files from DOS to Unix format. On many Linux systems, there are programs called dos2unix and unix2dos, which can convert text files to and from DOS format. However, if you don’t have dos2unix on your system, don’t worry. The process of converting text from DOS to Unix format is very simple; it simply involves the removal of the offending carriage returns. That is easily accomplished by a couple of the programs discussed later in this chapter.

有几种方法能够把文件从 DOS 格式转变为 Unix 格式。在许多 Linux 系统中，有两个 程序叫做 dos2unix 和 unix2dos，它们能在两种格式之间转变文本文件。然而，如果你 的系统中没有安装 dos2unix 程序，也不要担心。文件从 DOS 格式转变为 Unix 格式的过程非常 简单；它只简单地涉及到删除违规的回车符。通过随后本章中讨论的一些程序，这个工作很容易 完成。

cat also has options that are used to modify text. The two most prominent are -n, which numbers lines, and -s, which suppresses the output of multiple blank lines. We can demonstrate thusly:

cat 程序也包含用来修改文本的选项。最著名的两个选项是-n，其给文本行添加行号和-s， 禁止输出多个空白行。我们这样来说明：

[me@linuxbox ~]$ cat > foo.txt

The quick brown fox

jumped over the lazy dog.

[me@linuxbox ~]$ cat -ns foo.txt

1 The quick brown fox

2

3 jumped over the lazy dog.

[me@linuxbox ~]$

In this example, we create a new version of our foo.txt test file, which contains two lines of text separated by two blank lines. After processing by cat with the -ns options, the extra blank line is removed and the remaining lines are numbered. While this is not much of a process to perform on text, it is a process.

在这个例子里，我们创建了一个测试文件 foo.txt 的新版本，其包含两行文本，由两个空白行分开。 经由带有-ns 选项的 cat 程序处理之后，多余的空白行被删除，并且对保留的文本行进行编号。 然而这并不是多个进程在操作这个文本，只有一个进程。

**sort**

The sort program sorts the contents of standard input, or one or more files specified on the command line, and sends the results to standard output. Using the same technique that we used with cat, we can demonstrate processing of standard input directly from the keyboard:

这个 sort 程序对标准输入的内容，或命令行中指定的一个或多个文件进行排序，然后把排序 结果发送到标准输出。使用与 cat 命令相同的技巧，我们能够演示如何用 sort 程序来处理标准输入：

[me@linuxbox ~]$ sort > foo.txt

c

b

a

[me@linuxbox ~]$ cat foo.txt

a

b

c

After entering the command, we type the letters “c”, “b”, and “a”, followed once again by Ctrl-d to indicate end-of-file. We then view the resulting file and see that the lines now appear in sorted order.

输入命令之后，我们键入字母“c”，“b”，和“a”，然后再按下 Ctrl-d 组合键来表示文件的结尾。 随后我们查看生成的文件，看到文本行有序地显示。

Since sort can accept multiple files on the command line as arguments, it is possible to merge multiple files into a single sorted whole. For example, if we had three text files and wanted to combine them into a single sorted file, we could do something like this:

因为 sort 程序能接受命令行中的多个文件作为参数，所以有可能把多个文件合并成一个有序的文件。例如， 如果我们有三个文本文件，想要把它们合并为一个有序的文件，我们可以这样做：

sort file1.txt file2.txt file3.txt > final\_sorted\_list.txt

sort has several interesting options. Here is a partial list:

sort 程序有几个有趣的选项。这里只是一部分列表：

|  |  |  |
| --- | --- | --- |
| *Table 21-1: Common sort Options* | | |
| Option | Long Option | Description |
| -b | --ignore-leading-blanks | By default, sorting is performed on the entire line, starting with the first character in the line. This option causes sort to ignore leading spaces in lines and calculates sorting based on the first non-whitespace character on the line. |
| -f | --ignore-case | Makes sorting case insensitive. |
| -n | --numeric-sort | Performs sorting based on the numeric evaluation of a string. Using this option allows sorting to be performed on numeric values rather than alphabetic values. |
| -r | --reverse | Sort in reverse order. Results are in descending rather than ascending order. |
| -k | --key=field1[,field2] | Sort based on a key field located from field1 to field2 rather than the entire line. See discussion below. |
| -m | --merge | Treat each argument as the name of a presorted file. Merge multiple files into a single sorted result without performing any additional sorting. |
| -o | --output=file | Send sorted output to file rather than standard output. |
| -t | --field-separator=char | Define the field separator character. By default fields are separated by spaces or tabs. |

|  |  |  |
| --- | --- | --- |
| *表21-1: 常见的 sort 程序选项* | | |
| 选项 | 长选项 | 描述 |
| -b | --ignore-leading-blanks | 默认情况下，对整行进行排序，从每行的第一个字符开始。这个选项导致 sort 程序忽略 每行开头的空格，从第一个非空白字符开始排序。 |
| -f | --ignore-case | 让排序不区分大小写。 |
| -n | --numeric-sort | 基于字符串的长度来排序。使用此选项允许根据数字值执行排序，而不是字母值。 |
| -r | --reverse | 按相反顺序排序。结果按照降序排列，而不是升序。 |
| -k | --key=field1[,field2] | 对从 field1到 field2之间的字符排序，而不是整个文本行。看下面的讨论。 |
| -m | --merge | 把每个参数看作是一个预先排好序的文件。把多个文件合并成一个排好序的文件，而没有执行额外的排序。 |
| -o | --output=file | 把排好序的输出结果发送到文件，而不是标准输出。 |
| -t | --field-separator=char | 定义域分隔字符。默认情况下，域由空格或制表符分隔。 |

Although most of the options above are pretty self-explanatory, some are not. First, let’s look at the -n option, used for numeric sorting. With this option, it is possible to sort values based on numeric values. We can demonstrate this by sorting the results of the du command to determine the largest users of disk space. Normally, the du command lists the results of a summary in pathname order:

虽然以上大多数选项的含义是不言自喻的，但是有些也不是。首先，让我们看一下 -n 选项，被用做数值排序。 通过这个选项，有可能基于数值进行排序。我们通过对 du 命令的输出结果排序来说明这个选项，du 命令可以 确定最大的磁盘空间用户。通常，这个 du 命令列出的输出结果按照路径名来排序：

[me@linuxbox ~]$ du -s /usr/share/\* | head

252 /usr/share/aclocal

96 /usr/share/acpi-support

8 /usr/share/adduser

196 /usr/share/alacarte

344 /usr/share/alsa

8 /usr/share/alsa-base

12488 /usr/share/anthy

8 /usr/share/apmd

21440 /usr/share/app-install

48 /usr/share/application-registry

In this example, we pipe the results into head to limit the results to the first ten lines. We can produce a numerically sorted list to show the ten largest consumers of space this way:

在这个例子里面，我们把结果管道到 head 命令，把输出结果限制为前 10 行。我们能够产生一个按数值排序的 列表，来显示 10 个最大的空间消费者：

[me@linuxbox ~]$ du -s /usr/share/\* | sort -nr | head

509940 /usr/share/locale-langpack

242660 /usr/share/doc

197560 /usr/share/fonts

179144 /usr/share/gnome

146764 /usr/share/myspell

144304 /usr/share/gimp

135880 /usr/share/dict

76508 /usr/share/icons

68072 /usr/share/apps

62844 /usr/share/foomatic

By using the -nr options, we produce a reverse numerical sort, with the largest values appearing first in the results. This sort works because the numerical values occur at the beginning of each line. But what if we want to sort a list based on some value found within the line? For example, the results of an ls -l:

通过使用此 -nr 选项，我们产生了一个反向的数值排序，最大数值排列在第一位。这种排序起作用是 因为数值出现在每行的开头。但是如果我们想要基于文件行中的某个数值排序，又会怎样呢？ 例如，命令 ls -l 的输出结果：

[me@linuxbox ~]$ ls -l /usr/bin | head

total 152948

-rwxr-xr-x 1 root root 34824 2008-04-04 02:42 [

-rwxr-xr-x 1 root root 101556 2007-11-27 06:08 a2p

...

Ignoring, for the moment, that ls can sort its results by size, we could use sort to sort this list by file size, as well:

此刻，忽略 ls 程序能按照文件大小对输出结果进行排序，我们也能够使用 sort 程序来完成此任务：

[me@linuxbox ~]$ ls -l /usr/bin | sort -nr -k 5 | head

-rwxr-xr-x 1 root root 8234216 2008-04-0717:42 inkscape

-rwxr-xr-x 1 root root 8222692 2008-04-07 17:42 inkview

...

Many uses of sort involve the processing of tabular data, such as the results of the ls command above. If we apply database terminology to the table above, we would say that each row is a record and that each record consists of multiple fields, such as the file attributes, link count, filename, file size and so on. sort is able to process individual fields. In database terms, we are able to specify one or more key fields to use as sort keys. In the example above, we specify the n and r options to perform a reverse numerical sort and specify -k 5 to make sort use the fifth field as the key for sorting.

sort 程序的许多用法都涉及到处理表格数据，例如上面 ls 命令的输出结果。如果我们 把数据库这个术语应用到上面的表格中，我们会说每行是一条记录，并且每条记录由多个字段组成， 例如文件属性，链接数，文件名，文件大小等等。sort 程序能够处理独立的字段。在数据库术语中， 我们能够指定一个或者多个关键字段，来作为排序的关键值。在上面的例子中，我们指定 n 和 r 选项来执行相反的数值排序，并且指定 -k 5，让 sort 程序使用第五字段作为排序的关键值。

The k option is very interesting and has many features, but first we need to talk about how sort defines fields. Let’s consider a very simple text file consisting of a single line containing the author’s name:

这个 k 选项非常有趣，而且还有很多特点，但是首先我们需要讲讲 sort 程序怎样来定义字段。 让我们考虑一个非常简单的文本文件，只有一行包含作者名字的文本。

William Shotts

By default, sort sees this line as having two fields. The first field contains the characters:

默认情况下，sort 程序把此行看作有两个字段。第一个字段包含字符：

“William”

and the second field contains the characters:

和第二个字段包含字符：

“ Shotts”

meaning that whitespace characters (spaces and tabs) are used as delimiters between fields and that the delimiters are included in the field when sorting is performed. Looking again at a line from our ls output, we can see that a line contains eight fields and that the fifth field is the file size:

意味着空白字符（空格和制表符）被当作是字段间的界定符，当执行排序时，界定符会被 包含在字段当中。再看一下 ls 命令的输出，我们看到每行包含八个字段，并且第五个字段是文件大小：

-rwxr-xr-x 1 root root 8234216 2008-04-07 17:42 inkscape

For our next series of experiments, let’s consider the following file containing the history of three popular Linux distributions released from 2006 to 2008. Each line in the file has three fields: the distribution name, version number, and date of release in MM/DD/YYYY format:

让我们考虑用下面的文件，其包含从 2006 年到 2008 年三款流行的 Linux 发行版的发行历史，来做一系列实验。 文件中的每一行都有三个字段：发行版的名称，版本号，和 MM/DD/YYYY 格式的发行日期：

SUSE 10.2 12/07/2006

Fedora 10 11/25/2008

SUSE 11.04 06/19/2008

Ubuntu 8.04 04/24/2008

Fedora 8 11/08/2007

SUSE 10.3 10/04/2007

...

Using a text editor (perhaps vim), we’ll enter this data and name the resulting file distros.txt.

使用一个文本编辑器（可能是 vim），我们将输入这些数据，并把产生的文件命名为 distros.txt。

Next, we’ll try sorting the file and observe the results:

下一步，我们将试着对这个文件进行排序，并观察输出结果：

[me@linuxbox ~]$ sort distros.txt

Fedora 10 11/25/2008

Fedora 5 03/20/2006

Fedora 6 10/24/2006

Fedora 7 05/31/2007

Fedora 8 11/08/2007

...

Well, it mostly worked. The problem occurs in the sorting of the Fedora version numbers. Since a “1” comes before a “5” in the character set, version “10” ends up at the top while version “9” falls to the bottom.

恩，大部分正确。问题出现在 Fedora 的版本号上。因为在字符集中 “1” 出现在 “5” 之前，版本号 “10” 在 最顶端，然而版本号 “9” 却掉到底端。

To fix this problem we are going to have to sort on multiple keys. We want to perform an alphabetic sort on the first field and then a numeric sort on the third field. sort allows multiple instances of the -k option so that multiple sort keys can be specified. In fact, a key may include a range of fields. If no range is specified (as has been the case with our previous examples), sort uses a key that begins with the specified field and extends to the end of the line. Here is the syntax for our multi-key sort:

为了解决这个问题，我们必须依赖多个键值来排序。我们想要对第一个字段执行字母排序，然后对 第三个字段执行数值排序。sort 程序允许多个 -k 选项的实例，所以可以指定多个排序关键值。事实上， 一个关键值可能包括一个字段区域。如果没有指定区域（如同之前的例子），sort 程序会使用一个键值， 其始于指定的字段，一直扩展到行尾。下面是多键值排序的语法：

[me@linuxbox ~]$ sort --key=1,1 --key=2n distros.txt

Fedora 5 03/20/2006

Fedora 6 10/24/2006

Fedora 7 05/31/2007

...

Though we used the long form of the option for clarity, -k 1,1 -k 2n would be exactly equivalent. In the first instance of the key option, we specified a range of fields to include in the first key. Since we wanted to limit the sort to just the first field, we specified 1,1 which means “start at field one and end at field one.” In the second instance, we specified 2n, which means that field two is the sort key and that the sort should be numeric. An option letter may be included at the end of a key specifier to indicate the type of sort to be performed. These option letters are the same as the global options for the sort program: b (ignore leading blanks), n (numeric sort), r (reverse sort), and so on.

虽然为了清晰，我们使用了选项的长格式，但是 -k 1,1 -k 2n 格式是等价的。在第一个 key 选项的实例中， 我们指定了一个字段区域。因为我们只想对第一个字段排序，我们指定了 1,1， 意味着“始于并且结束于第一个字段。”在第二个实例中，我们指定了 2n，意味着第二个字段是排序的键值， 并且按照数值排序。一个选项字母可能被包含在一个键值说明符的末尾，其用来指定排序的种类。这些 选项字母和 sort 程序的全局选项一样：b（忽略开头的空格），n（数值排序），r（逆向排序），等等。

The third field in our list contains a date in an inconvenient format for sorting. On computers, dates are usually formatted in YYYY-MM-DD order to make chronological sorting easy, but ours are in the American format of MM/DD/YYYY. How can we sort this list in chronological order?

我们列表中第三个字段包含的日期格式不利于排序。在计算机中，日期通常设置为 YYYY-MM-DD 格式， 这样使按时间顺序排序变得容易，但是我们的日期为美国格式 MM/DD/YYYY。那么我们怎样能按照 时间顺序来排列这个列表呢？

Fortunately, sort provides a way. The key option allows specification of offsets within fields, so we can define keys within fields:

幸运地是，sort 程序提供了一种方式。这个 key 选项允许在字段中指定偏移量，所以我们能在字段中 定义键值。

[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt

Fedora 10 11/25/2008

Ubuntu 8.10 10/30/2008

SUSE 11.0 06/19/2008

...

By specifying -k 3.7 we instruct sort to use a sort key that begins at the seventh character within the third field, which corresponds to the start of the year. Likewise, we specify -k 3.1 and -k 3.4 to isolate the month and day portions of the date. We also add the n and r options to achieve a reverse numeric sort. The b option is included to suppress the leading spaces (whose numbers vary from line to line, thereby affecting the outcome of the sort) in the date field.

通过指定 -k 3.7，我们指示 sort 程序使用一个排序键值，其始于第三个字段中的第七个字符，对应于 年的开头。同样地，我们指定 -k 3.1和 -k 3.4来分离日期中的月和日。 我们也添加了 n 和 r 选项来实现一个逆向的数值排序。这个 b 选项用来删除日期字段中开头的空格（ 行与行之间的空格数迥异，因此会影响 sort 程序的输出结果）。

Some files don’t use tabs and spaces as field delimiters; for example, the /etc/passwd file:

一些文件不会使用 tabs 和空格做为字段界定符；例如，这个 /etc/passwd 文件：

[me@linuxbox ~]$ head /etc/passwd

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/bin/sh

man:x:6:12:man:/var/cache/man:/bin/sh

lp:x:7:7:lp:/var/spool/lpd:/bin/sh

mail:x:8:8:mail:/var/mail:/bin/sh

news:x:9:9:news:/var/spool/news:/bin/sh

The fields in this file are delimited with colons (:), so how would we sort this file using a key field? sort provides the -t option to define the field separator character. To sort the passwd file on the seventh field (the account’s default shell), we could do this:

这个文件的字段之间通过冒号分隔开，所以我们怎样使用一个 key 字段来排序这个文件？sort 程序提供 了一个 -t 选项来定义分隔符。按照第七个字段（帐户的默认 shell）来排序此 passwd 文件，我们可以这样做：

[me@linuxbox ~]$ sort -t ':' -k 7 /etc/passwd | head

me:x:1001:1001:Myself,,,:/home/me:/bin/bash

root:x:0:0:root:/root:/bin/bash

dhcp:x:101:102::/nonexistent:/bin/false

gdm:x:106:114:Gnome Display Manager:/var/lib/gdm:/bin/false

hplip:x:104:7:HPLIP system user,,,:/var/run/hplip:/bin/false

klog:x:103:104::/home/klog:/bin/false

messagebus:x:108:119::/var/run/dbus:/bin/false

polkituser:x:110:122:PolicyKit,,,:/var/run/PolicyKit:/bin/false

pulse:x:107:116:PulseAudio daemon,,,:/var/run/pulse:/bin/false

By specifying the colon character as the field separator, we can sort on the seventh field.

通过指定冒号字符做为字段分隔符，我们能按照第七个字段来排序。

**uniq**

Compared to sort, the uniq program is a lightweight. uniq performs a seemingly trivial task. When given a sorted file (including standard input), it removes any duplicate lines and sends the results to standard output. It is often used in conjunction with sort to clean the output of duplicates.

与 sort 程序相比，这个 uniq 程序是个轻量级程序。uniq 执行一个看似琐碎的认为。当给定一个 排好序的文件（包括标准输出），uniq 会删除任意重复行，并且把结果发送到标准输出。 它常常和 sort 程序一块使用，来清理重复的输出。

Tip: While uniq is a traditional Unix tool often used with sort, the GNU version of sort supports a -u option, which removes duplicates from the sorted output.

uniq 程序是一个传统的 Unix 工具，经常与 sort 程序一块使用，但是这个 GNU 版本的 sort 程序支持一个 -u 选项，其可以从排好序的输出结果中删除重复行。

Let’s make a text file to try this out:

让我们创建一个文本文件，来实验一下：

[me@linuxbox ~]$ cat > foo.txt

a

b

c

a

b

c

Remember to type Ctrl-d to terminate standard input. Now, if we run uniq on our text file:

记住输入 Ctrl-d 来终止标准输入。现在，如果我们对文本文件执行 uniq 命令：

[me@linuxbox ~]$ uniq foo.txt

a

b

c

a

b

c

the results are no different from our original file; the duplicates were not removed. For uniq to actually do its job, the input must be sorted first:

输出结果与原始文件没有差异；重复行没有被删除。实际上，uniq 程序能完成任务，其输入必须是排好序的数据，

[me@linuxbox ~]$ sort foo.txt | uniq

a

b

c

This is because uniq only removes duplicate lines which are adjacent to each other. uniq has several options. Here are the common ones:

这是因为 uniq 只会删除相邻的重复行。uniq 程序有几个选项。这里是一些常用选项：

|  |  |
| --- | --- |
| *Table 21-2: Common uniq Options* | |
| Option | Description |
| -c | Output a list of duplicate lines preceded by the number of times the line occurs. |
| -d | Only output repeated lines, rather than unique lines. |
| -f n | Ignore n leading fields in each line. Fields are separated by whitespace as they are in sort; however, unlike sort, uniq has no option for setting an alternate field separator. |
| -i | Ignore case during the line comparisons. |
| -s n | Skip (ignore) the leading n characters of each line. |
| -u | Only output unique lines. This is the default. |

|  |  |
| --- | --- |
| *表21-2: 常用的 uniq 选项* | |
| 选项 | 说明 |
| -c | 输出所有的重复行，并且每行开头显示重复的次数。 |
| -d | 只输出重复行，而不是特有的文本行。 |
| -f n | 忽略每行开头的 n 个字段，字段之间由空格分隔，正如 sort 程序中的空格分隔符；然而， 不同于 sort 程序，uniq 没有选项来设置备用的字段分隔符。 |
| -i | 在比较文本行的时候忽略大小写。 |
| -s n | 跳过（忽略）每行开头的 n 个字符。 |
| -u | 只是输出独有的文本行。这是默认的。 |

Here we see uniq used to report the number of duplicates found in our text file, using the -c option:

这里我们看到 uniq 被用来报告文本文件中重复行的次数，使用这个-c 选项：

[me@linuxbox ~]$ sort foo.txt | uniq -c

2 a

2 b

2 c

**切片和切块**

The next three programs we will discuss are used to peel columns of text out of files and recombine them in useful ways.

下面我们将要讨论的三个程序用来从文件中获得文本列，并且以有用的方式重组它们。

**cut**

The cut program is used to extract a section of text from a line and output the extracted section to standard output. It can accept multiple file arguments or input from standard input.

这个 cut 程序被用来从文本行中抽取文本，并把其输出到标准输出。它能够接受多个文件参数或者 标准输入。

Specifying the section of the line to be extracted is somewhat awkward and is specified using the following options:

从文本行中指定要抽取的文本有些麻烦，使用以下选项：

|  |  |
| --- | --- |
| *Table 21-3: cut Selection Options* | |
| Option | Description |
| -c char\_list | Extract the portion of the line defined by char\_list. The list may consist of one or more comma-separated numerical ranges. |
| -f field\_list | Extract one or more fields from the line as defined by field\_list. The list may contain one or more fields or field ranges separated by commas. |
| -d delim\_char | When -f is specified, use delim\_char as the field delimiting character. By default, fields must be separated by a single tab character. |
| --complement | Extract the entire line of text, except for those portions specified by -c and/or -f. |

|  |  |
| --- | --- |
| *表21-3: cut 程序选择项* | |
| 选项 | 说明 |
| -c char\_list | 从文本行中抽取由 char\_list 定义的文本。这个列表可能由一个或多个逗号 分隔开的数值区间组成。 |
| -f field\_list | 从文本行中抽取一个或多个由 field\_list 定义的字段。这个列表可能 包括一个或多个字段，或由逗号分隔开的字段区间。 |
| -d delim\_char | 当指定-f 选项之后，使用 delim\_char 做为字段分隔符。默认情况下， 字段之间必须由单个 tab 字符分隔开。 |
| --complement | 抽取整个文本行，除了那些由-c 和／或-f 选项指定的文本。 |

As we can see, the way cut extracts text is rather inflexible. cut is best used to extract text from files that are produced by other programs, rather than text directly typed by humans. We’ll take a look at our distros.txt file to see if it is “clean” enough to be a good specimen for our cut examples. If we use cat with the -A option, we can see if the file meets our requirements of tab separated fields:

正如我们所看到的，cut 程序抽取文本的方式相当不灵活。cut 命令最好用来从其它程序产生的文件中 抽取文本，而不是从人们直接输入的文本中抽取。我们将会看一下我们的 distros.txt 文件，看看 是否它足够 “整齐” 成为 cut 实例的一个好样本。如果我们使用带有 -A 选项的 cat 命令，我们能查看是否这个 文件符号由 tab 字符分离字段的要求。

[me@linuxbox ~]$ cat -A distros.txt

SUSE^I10.2^I12/07/2006$

Fedora^I10^I11/25/2008$

SUSE^I11.0^I06/19/2008$

Ubuntu^I8.04^I04/24/2008$

Fedora^I8^I11/08/2007$

SUSE^I10.3^I10/04/2007$

Ubuntu^I6.10^I10/26/2006$

Fedora^I7^I05/31/2007$

Ubuntu^I7.10^I10/18/2007$

Ubuntu^I7.04^I04/19/2007$

SUSE^I10.1^I05/11/2006$

Fedora^I6^I10/24/2006$

Fedora^I9^I05/13/2008$

Ubuntu^I6.06^I06/01/2006$

Ubuntu^I8.10^I10/30/2008$

Fedora^I5^I03/20/2006$

It looks good. No embedded spaces, just single tab characters between the fields. Since the file uses tabs rather than spaces, we’ll use the -f option to extract a field:

看起来不错。字段之间仅仅是单个 tab 字符，没有嵌入空格。因为这个文件使用了 tab 而不是空格， 我们将使用 -f 选项来抽取一个字段：

[me@linuxbox ~]$ cut -f 3 distros.txt

12/07/2006

11/25/2008

06/19/2008

04/24/2008

11/08/2007

10/04/2007

10/26/2006

05/31/2007

10/18/2007

04/19/2007

05/11/2006

10/24/2006

05/13/2008

06/01/2006

10/30/2008

03/20/2006

Because our distros file is tab-delimited, it is best to use cut to extract fields rather than characters. This is because when a file is tab-delimited, it is unlikely that each line will contain the same number of characters, which makes calculating character positions within the line difficult or impossible. In our example above, however, we now have extracted a field that luckily contains data of identical length, so we can show how character extraction works by extracting the year from each line:

因为我们的 distros 文件是由 tab 分隔开的，最好用 cut 来抽取字段而不是字符。这是因为一个由 tab 分离的文件， 每行不太可能包含相同的字符数，这就使计算每行中字符的位置变得困难或者是不可能。在以上事例中，然而， 我们已经抽取了一个字段，幸运地是其包含地日期长度相同，所以通过从每行中抽取年份，我们能展示怎样 来抽取字符：

[me@linuxbox ~]$ cut -f 3 distros.txt | cut -c 7-10

2006

2008

2008

2008

2007

2007

2006

2007

2007

2007

2006

2006

2008

2006

2008

2006

By running cut a second time on our list, we are able to extract character positions 7 through 10, which corresponds to the year in our date field. The 7-10 notation is an example of a range. The cut man page contains a complete description of how ranges can be specified.

通过对我们的列表再次运行 cut 命令，我们能够抽取从位置7到10的字符，其对应于日期字段的年份。 这个 7-10 表示法是一个区间的例子。cut 命令手册包含了一个如何指定区间的完整描述。

Expanding Tabs

展开 Tabs

Our distros.txt file is ideally formatted for extracting fields using cut. But what if we wanted a file that could be fully manipulated with cut by characters, rather than fields? This would require us to replace the tab characters within the file with the corresponding number of spaces. Fortunately, the GNU Coreutils package includes a tool for that. Named expand, this program accepts either one or more file arguments or standard input, and outputs the modified text to standard output.

distros.txt 的文件格式很适合使用 cut 程序来抽取字段。但是如果我们想要 cut 程序 按照字符，而不是字段来操作一个文件，那又怎样呢？这要求我们用相应数目的空格来 代替 tab 字符。幸运地是，GNU 的 Coreutils 软件包有一个工具来解决这个问题。这个 程序名为 expand，它既可以接受一个或多个文件参数，也可以接受标准输入，并且把 修改过的文本送到标准输出。

If we process our distros.txt file with expand, we can use the cut -c to extract any range of characters from the file. For example, we could use the following command to extract the year of release from our list, by expanding the file and using cut to extract every character from the twenty-third position to the end of the line:

如果我们通过 expand 来处理 distros.txt 文件，我们能够使用 cut -c 命令来从文件中抽取 任意区间内的字符。例如，我们能够使用以下命令来从列表中抽取发行年份，通过展开 此文件，再使用 cut 命令，来抽取从位置 23 开始到行尾的每一个字符：

**[me@linuxbox ~]$ expand distros.txt | cut -c 23-**

Coreutils also provides the unexpand program to substitute tabs for spaces.

Coreutils 软件包也提供了 unexpand 程序，用 tab 来代替空格。

When working with fields, it is possible to specify a different field delimiter rather than the tab character. Here we will extract the first field from the /etc/passwd file:

当操作字段的时候，有可能指定不同的字段分隔符，而不是 tab 字符。这里我们将会从/etc/passwd 文件中 抽取第一个字段：

[me@linuxbox ~]$ cut -d ':' -f 1 /etc/passwd | head

root

daemon

bin

sys

sync

games

man

lp

mail

news

Using the -d option, we are able to specify the colon character as the field delimiter.

使用-d 选项，我们能够指定冒号做为字段分隔符。

**paste**

The paste command does the opposite of cut. Rather than extracting a column of text from a file, it adds one or more columns of text to a file. It does this by reading multiple files and combining the fields found in each file into a single stream on standard output. Like cut, paste accepts multiple file arguments and/or standard input. To demonstrate how paste operates, we will perform some surgery on our distros.txt file to produce a chronological list of releases.

这个 paste 命令的功能正好与 cut 相反。它会添加一个或多个文本列到文件中，而不是从文件中抽取文本列。 它通过读取多个文件，然后把每个文件中的字段整合成单个文本流，输入到标准输出。类似于 cut 命令， paste 接受多个文件参数和 ／ 或标准输入。为了说明 paste 是怎样工作的，我们将会对 distros.txt 文件 动手术，来产生发行版的年代表。

From our earlier work with sort, we will first produce a list of distros sorted by date and store the result in a file called distros-by-date.txt:

从我们之前使用 sort 的工作中，首先我们将产生一个按照日期排序的发行版列表，并把结果 存储在一个叫做 distros-by-date.txt 的文件中：

[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt > distros-by-date.txt

Next, we will use cut to extract the first two fields from the file (the distro name and version), and store that result in a file named distro-versions.txt:

下一步，我们将会使用 cut 命令从文件中抽取前两个字段（发行版名字和版本号），并把结果存储到 一个名为 distro-versions.txt 的文件中：

[me@linuxbox ~]$ cut -f 1,2 distros-by-date.txt > distros-versions.txt

[me@linuxbox ~]$ head distros-versions.txt

Fedora 10

Ubuntu 8.10

SUSE 11.0

Fedora 9

Ubuntu 8.04

Fedora 8

Ubuntu 7.10

SUSE 10.3

Fedora 7

Ubuntu 7.04

The final piece of preparation is to extract the release dates and store them a file named distro-dates.txt:

最后的准备步骤是抽取发行日期，并把它们存储到一个名为 distro-dates.txt 文件中：

[me@linuxbox ~]$ cut -f 3 distros-by-date.txt > distros-dates.txt

[me@linuxbox ~]$ head distros-dates.txt

11/25/2008

10/30/2008

06/19/2008

05/13/2008

04/24/2008

11/08/2007

10/18/2007

10/04/2007

05/31/2007

04/19/2007

We now have the parts we need. To complete the process, use paste to put the column of dates ahead of the distro names and versions, thus creating a chronological list. This is done simply by using paste and ordering its arguments in the desired arrangement:

现在我们拥有了我们所需要的文本了。为了完成这个过程，使用 paste 命令来把日期列放到发行版名字 和版本号的前面，这样就创建了一个年代列表。通过使用 paste 命令，然后按照期望的顺序来安排它的 参数，就能很容易完成这个任务。

[me@linuxbox ~]$ paste distros-dates.txt distros-versions.txt

11/25/2008 Fedora 10

10/30/2008 Ubuntu 8.10

06/19/2008 SUSE 11.0

05/13/2008 Fedora 9

04/24/2008 Ubuntu 8.04

11/08/2007 Fedora 8

10/18/2007 Ubuntu 7.10

10/04/2007 SUSE 10.3

05/31/2007 Fedora 7

04/19/2007 Ubuntu 7.04

**join**

In some ways, join is like paste in that it adds columns to a file, but it uses a unique way to do it. A join is an operation usually associated with relational databases where data from multiple tables with a shared key field is combined to form a desired result. The join program performs the same operation. It joins data from multiple files based on a shared key field.

在某些方面，join 命令类似于 paste，它会往文件中添加列，但是它使用了独特的方法来完成。 一个 join 操作通常与关系型数据库有关联，在关系型数据库中来自多个享有共同关键域的表格的 数据结合起来，得到一个期望的结果。这个 join 程序执行相同的操作。它把来自于多个基于共享 关键域的文件的数据结合起来。

To see how a join operation is used in a relational database, let’s imagine a very small database consisting of two tables each containing a single record. The first table, called CUSTOMERS, has three fields: a customer number (CUSTNUM), the customer’s first name (FNAME) and the customer’s last name (LNAME):

为了知道在关系数据库中是怎样使用 join 操作的，让我们想象一个很小的数据库，这个数据库由两个 表格组成，每个表格包含一条记录。第一个表格，叫做 CUSTOMERS，有三个数据域：一个客户号（CUSTNUM）， 客户的名字（FNAME）和客户的姓（LNAME）：

CUSTNUM FNAME ME

======== ===== ======

4681934 John Smith

The second table is called ORDERS and contains four fields: an order number (ORDERNUM), the customer number (CUSTNUM), the quantity (QUAN), and the item ordered (ITEM).

第二个表格叫做 ORDERS，其包含四个数据域：订单号（ORDERNUM），客户号（CUSTNUM），数量（QUAN）， 和订购的货品（ITEM）。

ORDERNUM CUSTNUM QUAN ITEM

======== ======= ==== ====

3014953305 4681934 1 Blue Widget

Note that both tables share the field CUSTNUM. This is important, as it allows a relationship between the tables.

注意两个表格共享数据域 CUSTNUM。这很重要，因为它使表格之间建立了联系。

Performing a join operation would allow us to combine the fields in the two tables to achieve a useful result, such as preparing an invoice. Using the matching values in the CUSTNUM fields of both tables, a join operation could produce the following:

执行一个 join 操作将允许我们把两个表格中的数据域结合起来，得到一个有用的结果，例如准备 一张发货单。通过使用两个表格 CUSTNUM 数字域中匹配的数值，一个 join 操作会产生以下结果：

FNAME LNAME QUAN ITEM

===== ===== ==== ====

John Smith 1 Blue Widget

To demonstrate the join program, we’ll need to make a couple of files with a shared key. To do this, we will use our distros-by-date.txt file. From this file, we will construct two additional files, one containing the release date (which will be our shared key for this demonstration) and the release name:

为了说明 join 程序，我们需要创建一对包含共享键值的文件。为此，我们将使用我们的 distros.txt 文件。 从这个文件中，我们将构建额外两个文件，一个包含发行日期（其会成为共享键值）和发行版名称：

[me@linuxbox ~]$ cut -f 1,1 distros-by-date.txt > distros-names.txt

[me@linuxbox ~]$ paste distros-dates.txt distros-names.txt > distros-key-names.txt

[me@linuxbox ~]$ head distros-key-names.txt

11/25/2008 Fedora

10/30/2008 Ubuntu

06/19/2008 SUSE

05/13/2008 Fedora

04/24/2008 Ubuntu

11/08/2007 Fedora

10/18/2007 Ubuntu

10/04/2007 SUSE

05/31/2007 Fedora

04/19/2007 Ubuntu

and the second file, which contains the release dates and the version numbers:

第二个文件包含发行日期和版本号：

[me@linuxbox ~]$ cut -f 2,2 distros-by-date.txt > distros-vernums.txt

[me@linuxbox ~]$ paste distros-dates.txt distros-vernums.txt > distros-key-vernums.txt

[me@linuxbox ~]$ head distros-key-vernums.txt

11/25/2008 10

10/30/2008 8.10

06/19/2008 11.0

05/13/2008 9

04/24/2008 8.04

11/08/2007 8

10/18/2007 7.10

10/04/2007 10.3

05/31/2007 7

04/19/2007 7.04

We now have two files with a shared key (the “release date” field). It is important to point out that the files must be sorted on the key field for join to work properly.

现在我们有两个具有共享键值（ “发行日期” 数据域 ）的文件。有必要指出，为了使 join 命令 能正常工作，所有文件必须按照关键数据域排序。

[me@linuxbox ~]$ join distros-key-names.txt distros-key-vernums.txt | head

11/25/2008 Fedora 10

10/30/2008 Ubuntu 8.10

06/19/2008 SUSE 11.0

05/13/2008 Fedora 9

04/24/2008 Ubuntu 8.04

11/08/2007 Fedora 8

10/18/2007 Ubuntu 7.10

10/04/2007 SUSE 10.3

05/31/2007 Fedora 7

04/19/2007 Ubuntu 7.04

Note also that, by default, join uses whitespace as the input field delimiter and a single space as the output field delimiter. This behavior can be modified by specifying options. See the join man page for details.

也要注意，默认情况下，join 命令使用空白字符做为输入字段的界定符，一个空格作为输出字段 的界定符。这种行为可以通过指定的选项来修改。详细信息，参考 join 命令手册。

**比较文本**

It is often useful to compare versions of text files. For system administrators and software developers, this is particularly important. A system administrator may, for example, need to compare an existing configuration file to a previous version to diagnose a system problem. Likewise, a programmer frequently needs to see what changes have been made to programs over time.

通常比较文本文件的版本很有帮助。对于系统管理员和软件开发者来说，这个尤为重要。 一名系统管理员可能，例如，需要拿现有的配置文件与先前的版本做比较，来诊断一个系统错误。 同样的，一名程序员经常需要查看程序的修改。

**comm**

The comm program compares two text files and displays the lines that are unique to each one and the lines they have in common. To demonstrate, we will create two nearly identical text files using cat:

这个 comm 程序会比较两个文本文件，并且会显示每个文件特有的文本行和共有的文把行。 为了说明问题，通过使用 cat 命令，我们将会创建两个内容几乎相同的文本文件：

[me@linuxbox ~]$ cat > file1.txt

a

b

c

d

[me@linuxbox ~]$ cat > file2.txt

b

c

d

e

Next, we will compare the two files using comm:

下一步，我们将使用 comm 命令来比较这两个文件：

[me@linuxbox ~]$ comm file1.txt file2.txt

a

b

c

d

e

As we can see, comm produces three columns of output. The first column contains lines unique to the first file argument; the second column, the lines unique to the second file argument; the third column contains the lines shared by both files. comm supports options in the form -n where n is either 1, 2 or 3. When used, these options specify which column(s) to suppress. For example, if we only wanted to output the lines shared by both files, we would suppress the output of columns one and two:

正如我们所见到的，comm 命令产生了三列输出。第一列包含第一个文件独有的文本行；第二列， 文本行是第二列独有的；第三列包含两个文件共有的文本行。comm 支持 -n 形式的选项，这里 n 代表 1，2 或 3。这些选项使用的时候，指定了要隐藏的列。例如，如果我们只想输出两个文件共享的文本行， 我们将隐藏第一列和第二列的输出结果：

[me@linuxbox ~]$ comm -12 file1.txt file2.txt

b

c

d

**diff**

Like the comm program, diff is used to detect the differences between files. However, diff is a much more complex tool, supporting many output formats and the ability to process large collections of text files at once. diff is often used by software developers to examine changes between different versions of program source code, and thus has the ability to recursively examine directories of source code often referred to as source trees. One common use for diff is the creation of diff files or patches that are used by programs such as patch (which we’ll discuss shortly) to convert one version of a file (or files) to another version.

类似于 comm 程序，diff 程序被用来监测文件之间的差异。然而，diff 是一款更加复杂的工具，它支持 许多输出格式，并且一次能处理许多文本文件。软件开发员经常使用 diff 程序来检查不同程序源码 版本之间的更改，diff 能够递归地检查源码目录，经常称之为源码树。diff 程序的一个常见用例是 创建 diff 文件或者补丁，它会被其它程序使用，例如 patch 程序（我们一会儿讨论），来把文件 从一个版本转换为另一个版本。

If we use diff to look at our previous example files:

如果我们使用 diff 程序，来查看我们之前的文件实例：

[me@linuxbox ~]$ diff file1.txt file2.txt

1d0

< a

4a4

> e

we see its default style of output: a terse description of the differences between the two files. In the default format, each group of changes is preceded by a change command in the form of **range operation range** to describe the positions and type of changes required to convert the first file to the second file:

我们看到 diff 程序的默认输出风格：对两个文件之间差异的简短描述。在默认格式中， 每组的更改之前都是一个更改命令，其形式为 **range operation range** ， 用来描述要求更改的位置和类型，从而把第一个文件转变为第二个文件：

|  |  |
| --- | --- |
| *Table 21-4: diff Change Commands* | |
| Change | Description |
| r1ar2 | Append the lines at the position r2 in the second file to the position r1 in the first file. |
| r1cr2 | Change (replace) the lines at position r1 with the lines at the position r2 in the second file. |
| r1dr2 | Delete the lines in the first file at position r1, which would have appeared at range r2 in the second file. |

|  |  |
| --- | --- |
| *表21-4: diff 更改命令* | |
| 改变 | 说明 |
| r1ar2 | 把第二个文件中位置 r2 处的文件行添加到第一个文件中的 r1 处。 |
| r1cr2 | 用第二个文件中位置 r2 处的文本行更改（替代）位置 r1 处的文本行。 |
| r1dr2 | 删除第一个文件中位置 r1 处的文本行，这些文本行将会出现在第二个文件中位置 r2 处。 |

In this format, a range is a comma separated list of the starting line and the ending line. While this format is the default (mostly for POSIX compliance and backward compatibility with traditional Unix versions of diff), it is not as widely used as other, optional formats. Two of the more popular formats are the **context format** and the **unified format**.

在这种格式中，一个范围就是由逗号分隔开的开头行和结束行的列表。虽然这种格式是默认情况（主要是 为了服从 POSIX 标准且向后与传统的 Unix diff 命令兼容）， 但是它并不像其它可选格式一样被广泛地使用。最流行的两种格式是上下文模式和统一模式。

When viewed using the **context format** (the -c option), we will see this:

当使用上下文模式（带上 -c 选项），我们将看到这些：

[me@linuxbox ~]$ diff -c file1.txt file2.txt

\*\*\* file1.txt 2008-12-23 06:40:13.000000000 -0500

--- file2.txt 2008-12-23 06:40:34.000000000 -0500

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* 1,4 \*\*\*\*

- a

b

c

d

--- 1,4 ----

b

c

d

+ e

The output begins with the names of the two files and their timestamps. The first file is marked with asterisks and the second file is marked with dashes. Throughout the remainder of the listing, these markers will signify their respective files. Next, we see groups of changes, including the default number of surrounding context lines. In the first group, we see:

这个输出结果以两个文件名和它们的时间戳开头。第一个文件用星号做标记，第二个文件用短横线做标记。 纵观列表的其它部分，这些标记将象征它们各自代表的文件。下一步，我们看到几组修改， 包括默认的周围上下文行数。在第一组中，我们看到：

\*\*\* 1,4 \*\*\*

which indicates lines one through four in the first file. Later we see:

其表示第一个文件中从第一行到第四行的文本行。随后我们看到：

--- 1,4 ---

which indicates lines one through four in the second file. Within a change group, lines begin with one of four indicators:

这表示第二个文件中从第一行到第四行的文本行。在更改组内，文本行以四个指示符之一开头：

|  |  |
| --- | --- |
| *Table 21-5: diff Context Format Change Indicators* | |
| Indicator | Meaning |
| blank | A line shown for context. It does not indicate a difference between the two files. |
| - | A line deleted. This line will appear in the first file but not in the second file. |
| + | A line added. This line will appear in the second file but not in the first file. |
| ! | A line changed. The two versions of the line will be displayed, each in its respective section of the change group. |

|  |  |
| --- | --- |
| *表21-5: diff 上下文模式更改指示符* | |
| 指示符 | 意思 |
| blank | 上下文显示行。它并不表示两个文件之间的差异。 |
| - | 删除行。这一行将会出现在第一个文件中，而不是第二个文件内。 |
| + | 添加行。这一行将会出现在第二个文件内，而不是第一个文件中。 |
| ! | 更改行。将会显示某个文本行的两个版本，每个版本会出现在更改组的各自部分。 |

The **unified format** is similar to the **context format**, but is more concise. It is specified with the -u option:

这个统一模式相似于上下文模式，但是更加简洁。通过 -u 选项来指定它：

[me@linuxbox ~]$ diff -u file1.txt file2.txt

--- file1.txt 2008-12-23 06:40:13.000000000 -0500

+++ file2.txt 2008-12-23 06:40:34.000000000 -0500

@@ -1,4 +1,4 @@

-a

b

c

d

+e

The most notable difference between the context and unified formats is the elimination of the duplicated lines of context, making the results of the unified format shorter than the context format. In our example above, we see file timestamps like those of the context format, followed by the string @@ -1,4 +1,4 @@. This indicates the lines in the first file and the lines in the second file described in the change group. Following this are the lines themselves, with the default three lines of context. Each line starts with one of three possible characters:

上下文模式和统一模式之间最显著的差异就是重复上下文的消除，这就使得统一模式的输出结果要比上下文 模式的输出结果简短。在我们上述实例中，我们看到类似于上下文模式中的文件时间戳，其紧紧跟随字符串 @@ -1,4 +1,4 @@。这行字符串表示了在更改组中描述的第一个文件中的文本行和第二个文件中的文本行。 这行字符串之后就是文本行本身，与三行默认的上下文。每行以可能的三个字符中的一个开头：

|  |  |
| --- | --- |
| *Table 21-6: diff Unified Format Change Indicators* | |
| Character | Meaning |
| blank | This line is shared by both files. |
| - | This line was removed from the first file. |
| + | This line was added to the first file. |

|  |  |
| --- | --- |
| *表21-6: diff 统一模式更改指示符* | |
| 字符 | 意思 |
| 空格 | 两个文件都包含这一行。 |
| - | 在第一个文件中删除这一行。 |
| + | 添加这一行到第一个文件中。 |

**patch**

The patch program is used to apply changes to text files. It accepts output from diff and is generally used to convert older version of files into newer versions. Let’s consider a famous example. The Linux kernel is developed by a large, loosely organized team of contributors who submit a constant stream of small changes to the source code. The Linux kernel consists of several million lines of code, while the changes that are made by one contributor at one time are quite small. It makes no sense for a contributor to send each developer an entire kernel source tree each time a small change is made. Instead, a diff file is submitted. The diff file contains the change from the previous version of the kernel to the new version with the contributor’s changes. The receiver then uses the patch program to apply the change to his own source tree. Using diff/patch offers two significant advantages:

这个 patch 程序被用来把更改应用到文本文件中。它接受从 diff 程序的输出，并且通常被用来 把较老的文件版本转变为较新的文件版本。让我们考虑一个著名的例子。Linux 内核是由一个 大型的，组织松散的贡献者团队开发而成，这些贡献者会提交固定的少量更改到源码包中。 这个 Linux 内核由几百万行代码组成，虽然每个贡献者每次所做的修改相当少。对于一个贡献者 来说，每做一个修改就给每个开发者发送整个的内核源码树，这是没有任何意义的。相反， 提交一个 diff 文件。一个 diff 文件包含先前的内核版本与带有贡献者修改的新版本之间的差异。 然后一个接受者使用 patch 程序，把这些更改应用到他自己的源码树中。使用 diff/patch 组合提供了 两个重大优点：

1. The diff file is very small, compared to the full size of the source tree.
2. The diff file concisely shows the change being made, allowing reviewers of the patch to quickly evaluate it.
3. 一个 diff 文件非常小，与整个源码树的大小相比较而言。
4. 一个 diff 文件简洁地显示了所做的修改，从而允许程序补丁的审阅者能快速地评估它。

Of course, diff/patch will work on any text file, not just source code. It would be equally applicable to configuration files or any other text.

当然，diff/patch 能工作于任何文本文件，不仅仅是源码文件。它同样适用于配置文件或任意其它文本。

To prepare a diff file for use with patch, the GNU documentation (see Further Reading below) suggests using diff as follows:

准备一个 diff 文件供 patch 程序使用，GNU 文档（查看下面的拓展阅读部分）建议这样使用 diff 命令：

diff -Naur old\_file new\_file > diff\_file

Where old\_file and new\_file are either single files or directories containing files. The r option supports recursion of a directory tree.

old\_file 和 new\_file 部分不是单个文件就是包含文件的目录。这个 r 选项支持递归目录树。

Once the diff file has been created, we can apply it to patch the old file into the new file:

一旦创建了 diff 文件，我们就能应用它，把旧文件修补成新文件。

patch < diff\_file

We’ll demonstrate with our test file:

我们将使用测试文件来说明：

[me@linuxbox ~]$ diff -Naur file1.txt file2.txt > patchfile.txt

[me@linuxbox ~]$ patch < patchfile.txt

patching file file1.txt

[me@linuxbox ~]$ cat file1.txt

b

c

d

e

In this example, we created a diff file named patchfile.txt and then used the patch program to apply the patch. Note that we did not have to specify a target file to patch, as the diff file (in unified format) already contains the filenames in the header. Once the patch is applied, we can see that file1.txt now matches file2.txt.

在这个例子中，我们创建了一个名为 patchfile.txt 的 diff 文件，然后使用 patch 程序， 来应用这个补丁。注意我们没有必要指定一个要修补的目标文件，因为 diff 文件（在统一模式中）已经 在标题行中包含了文件名。一旦应用了补丁，我们能看到，现在 file1.txt 与 file2.txt 文件相匹配了。

patch has a large number of options, and there are additional utility programs that can be used to analyze and edit patches.

patch 程序有大量的选项，而且还有额外的实用程序可以被用来分析和编辑补丁。

**运行时编辑**

Our experience with text editors has been largely interactive, meaning that we manually move a cursor around, then type our changes. However, there are non-interactive ways to edit text as well. It’s possible, for example, to apply a set of changes to multiple files with a single command.

我们对于文本编辑器的经验是它们主要是交互式的，意思是我们手动移动光标，然后输入我们的修改。 然而，也有非交互式的方法来编辑文本。有可能，例如，通过单个命令把一系列修改应用到多个文件中。

**tr**

The tr program is used to transliterate characters. We can think of this as a sort of character-based search-and-replace operation. Transliteration is the process of changing characters from one alphabet to another. For example, converting characters from lowercase to uppercase is transliteration. We can perform such a conversion with tr as follows:

这个 tr 程序被用来更改字符。我们可以把它看作是一种基于字符的查找和替换操作。 换字是一种把字符从一个字母转换为另一个字母的过程。例如，把小写字母转换成大写字母就是 换字。我们可以通过 tr 命令来执行这样的转换，如下所示：

[me@linuxbox ~]$ echo "lowercase letters" | tr a-z A-Z

LOWERCASE LETTERS

As we can see, tr operates on standard input, and outputs its results on standard output. tr accepts two arguments: a set of characters to convert from and a corresponding set of characters to convert to. Character sets may be expressed in one of three ways:

正如我们所见，tr 命令操作标准输入，并把结果输出到标准输出。tr 命令接受两个参数：要被转换的字符集以及 相对应的转换后的字符集。字符集可以用三种方式来表示：

1. An enumerated list. For example, ABCDEFGHIJKLMNOPQRSTUVWXYZ
2. A character range. For example, A-Z. Note that this method is sometimes subject to the same issues as other commands, due to the locale collation order, and thus should be used with caution.
3. POSIX character classes. For example, [:upper:].
4. 一个枚举列表。例如， ABCDEFGHIJKLMNOPQRSTUVWXYZ
5. 一个字符域。例如，A-Z 。注意这种方法有时候面临与其它命令相同的问题，归因于 语系的排序规则，因此应该谨慎使用。
6. POSIX 字符类。例如，[:upper:]

In most cases, both character sets should be of equal length; however, it is possible for the first set to be larger than the second, particularly if we wish to convert multiple characters to a single character:

大多数情况下，两个字符集应该长度相同；然而，有可能第一个集合大于第二个，尤其如果我们 想要把多个字符转换为单个字符：

[me@linuxbox ~]$ echo "lowercase letters" | tr [:lower:] A

AAAAAAAAA AAAAAAA

In addition to transliteration, tr allows characters to simply be deleted from the input stream. Earlier in this chapter, we discussed the problem of converting MS-DOS text files to Unix style text. To perform this conversion, carriage return characters need to be removed from the end of each line. This can be performed with tr as follows:

除了换字之外，tr 命令能允许字符从输入流中简单地被删除。在之前的章节中，我们讨论了转换 MS-DOS 文本文件为 Unix 风格文本的问题。为了执行这个转换，每行末尾的回车符需要被删除。 这个可以通过 tr 命令来执行，如下所示：

tr -d '\r' < dos\_file > unix\_file

where dos\_file is the file to be converted and unix\_file is the result. This form of the command uses the escape sequence \r to represent the carriage return character. To see a complete list of the sequences and character classes tr supports, try:

这里的 dos\_file 是需要被转换的文件，unix\_file 是转换后的结果。这种形式的命令使用转义序列 \r 来代表回车符。查看 tr 命令所支持地完整的转义序列和字符类别列表，试试下面的命令：

[me@linuxbox ~]$ tr --help

ROT13: The Not-So-Secret Decoder Ring

ROT13: 不那么秘密的编码环

One amusing use of tr is to perform ROT13 encoding of text. ROT13 is a trivial type of encryption based on a simple substitution cipher. Calling ROT13 “encryption” is being generous; “text obfuscation” is more accurate. It is used sometimes on text to obscure potentially offensive content. The method simply moves each character thirteen places up the alphabet. Since this is half way up the possible twenty-six characters, performing the algorithm a second time on the text restores it to its original form. To perform this encoding with tr:

tr 命令的一个有趣的用法是执行 ROT13文本编码。ROT13是一款微不足道的基于一种简易的替换暗码的 加密类型。把 ROT13称为“加密”是大方的；“文本模糊处理”更准确些。有时候它被用来隐藏文本中潜在的攻击内容。 这个方法就是简单地把每个字符在字母表中向前移动13位。因为移动的位数是可能的26个字符的一半， 所以对文本再次执行这个算法，就恢复到了它最初的形式。通过 tr 命令来执行这种编码：

|  |  |
| --- | --- |
| echo “secret text” | tr a-zA-Z n-za-mN-ZA-M |

frperg grkg

Performing the same procedure a second time results in the translation:

再次执行相同的过程，得到翻译结果：

|  |  |
| --- | --- |
| echo “frperg grkg” | tr a-zA-Z n-za-mN-ZA-M |

secret text

A number of email programs and USENET news readers support ROT13 encoding. Wikipedia contains a good article on the subject:

大量的 email 程序和 USENET 新闻读者都支持 ROT13 编码。Wikipedia 上面有一篇关于这个主题的好文章：

<http://en.wikipedia.org/wiki/ROT13>

tr can perform another trick, too. Using the -s option, tr can “squeeze” (delete) repeated instances of a character:

tr 也可以完成另一个技巧。使用-s 选项，tr 命令能“挤压”（删除）重复的字符实例：

[me@linuxbox ~]$ echo "aaabbbccc" | tr -s ab

abccc

Here we have a string containing repeated characters. By specifying the set “ab” to tr, we eliminate the repeated instances of the letters in the set, while leaving the character that is missing from the set (“c”) unchanged. Note that the repeating characters must be adjoining. If they are not:

这里我们有一个包含重复字符的字符串。通过给 tr 命令指定字符集“ab”，我们能够消除字符集中 字母的重复实例，然而会留下不属于字符集的字符（“c”）无更改。注意重复的字符必须是相邻的。 如果它们不相邻：

[me@linuxbox ~]$ echo "abcabcabc" | tr -s ab

abcabcabc

the squeezing will have no effect.

那么挤压会没有效果。

**sed**

The name sed is short for stream editor. It performs text editing on a stream of text, either a set of specified files or standard input. sed is a powerful and somewhat complex program (there are entire books about it), so we will not cover it completely here.

名字 sed 是 stream editor（流编辑器）的简称。它对文本流进行编辑，要不是一系列指定的文件， 要不就是标准输入。sed 是一款强大的，并且有些复杂的程序（有整本内容都是关于 sed 程序的书籍）， 所以在这里我们不会详尽的讨论它。

In general, the way that sed works is that it is given either a single editing command (on the command line) or the name of a script file containing multiple commands, and it then performs these commands upon each line in the stream of text. Here is a very simple example of sed in action:

总之，sed 的工作方式是要不给出单个编辑命令（在命令行中）要不就是包含多个命令的脚本文件名， 然后它就按行来执行这些命令。这里有一个非常简单的 sed 实例：

[me@linuxbox ~]$ echo "front" | sed 's/front/back/'

back

In this example, we produce a one word stream of text using echo and pipe it into sed. sed, in turn, carries out the instruction s/front/back/ upon the text in the stream and produces the output “back” as a result. We can also recognize this command as resembling the “substitution” (search and replace) command in vi.

在这个例子中，我们使用 echo 命令产生了一个单词的文本流，然后把它管道给 sed 命令。sed，依次， 对流文本执行指令 s/front/back/，随后输出“back”。我们也能够把这个命令认为是相似于 vi 中的“替换” （查找和替代）命令。

Commands in sed begin with a single letter. In the example above, the substitution command is represented by the letter s and is followed by the search and replace strings, separated by the slash character as a delimiter. The choice of the delimiter character is arbitrary. By convention, the slash character is often used, but sed will accept any character that immediately follows the command as the delimiter. We could perform the same command this way:

sed 中的命令开始于单个字符。在上面的例子中，这个替换命令由字母 s 来代表，其后跟着查找 和替代字符串，斜杠字符做为分隔符。分隔符的选择是随意的。按照惯例，经常使用斜杠字符， 但是 sed 将会接受紧随命令之后的任意字符做为分隔符。我们可以按照这种方式来执行相同的命令：

[me@linuxbox ~]$ echo "front" | sed 's\_front\_back\_'

back

By using the underscore character immediately after the command, it becomes the delimiter. The ability to set the delimiter can be used to make commands more readable, as we shall see.

通过紧跟命令之后使用下划线字符，则它变成界定符。sed 可以设置界定符的能力，使命令的可读性更强， 正如我们将看到的.

Most commands in sed may be preceded by an address, which specifies which line(s) of the input stream will be edited. If the address is omitted, then the editing command is carried out on every line in the input stream. The simplest form of address is a line number. We can add one to our example:

sed 中的大多数命令之前都会带有一个地址，其指定了输入流中要被编辑的文本行。如果省略了地址， 然后会对输入流的每一行执行编辑命令。最简单的地址形式是一个行号。我们能够添加一个地址 到我们例子中：

[me@linuxbox ~]$ echo "front" | sed '1s/front/back/'

back

Adding the address 1 to our command causes our substitution to be performed on the first line of our one-line input stream. If we specify another number:

给我们的命令添加地址 1，就导致只对仅有一行文本的输入流的第一行执行替换操作。如果我们指定另一 个数字：

[me@linuxbox ~]$ echo "front" | sed '2s/front/back/'

front

we see that the editing is not carried out, since our input stream does not have a line two. Addresses may be expressed in many ways. Here are the most common:

我们看到没有执行这个编辑命令，因为我们的输入流没有第二行。地址可以用许多方式来表达。这里是 最常用的：

a range of line numbers

|  |  |
| --- | --- |
| *Table 21-7: sed Address Notation* | |
| Address | Description |
| n | A line number where n is a positive integer. |
| $ | The last line. |
| /regexp/ | Lines matching a POSIX basic regular expression. Note that the regular expression is delimited by slash characters. Optionally, the regular expression may be delimited by an alternate character, by specifying the expression with \cregexpc, where c is the alternate character. |
| addr1,addr2 | A range of lines from addr1 to addr2, inclusive. Addresses may be any of the single address forms above. |
| first~step | Match the line represented by the number first, then each subsequent line at step intervals. For example 1~2 refers to each odd numbered line, 5~5 refers to the fifth line and every fifth line thereafter. |
| addr1,+n | Match addr1 and the following n lines. |
| addr! | Match all lines except addr, which may be any of the forms above. |

|  |  |
| --- | --- |
| *表21-7: sed 地址表示法* | |
| 地址 | 说明 |
| n | 行号，n 是一个正整数。 |
| $ | 最后一行。 |
| /regexp/ | 所有匹配一个 POSIX 基本正则表达式的文本行。注意正则表达式通过 斜杠字符界定。选择性地，这个正则表达式可能由一个备用字符界定，通过\cregexpc 来 指定表达式，这里 c 就是一个备用的字符。 |
| addr1,addr2 | 从 addr1 到 addr2 范围内的文本行，包含地址 addr2 在内。地址可能是上述任意 单独的地址形式。 |
| first~step | 匹配由数字 first 代表的文本行，然后随后的每个在 step 间隔处的文本行。例如 1~2 是指每个位于偶数行号的文本行，5~5 则指第五行和之后每五行位置的文本行。 |
| addr1,+n | 匹配地址 addr1 和随后的 n 个文本行。 |
| addr! | 匹配所有的文本行，除了 addr 之外，addr 可能是上述任意的地址形式。 |

We’ll demonstrate different kinds of addresses using the distros.txt file from earlier in this chapter. First, a range of line numbers:

通过使用这一章中早前的 distros.txt 文件，我们将演示不同种类的地址表示法。首先，一系列行号：

[me@linuxbox ~]$ sed -n '1,5p' distros.txt

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

In this example, we print a range of lines, starting with line one and continuing to line five. To do this, we use the p command, which simply causes a matched line to be printed. For this to be effective however, we must include the option -n (the no auto- print option) to cause sed not to print every line by default.

在这个例子中，我们打印出一系列的文本行，开始于第一行，直到第五行。为此，我们使用 p 命令， 其就是简单地把匹配的文本行打印出来。然而为了高效，我们必须包含选项 -n（不自动打印选项）， 让 sed 不要默认地打印每一行。

Next, we’ll try a regular expression:

下一步，我们将试用一下正则表达式：

[me@linuxbox ~]$ sed -n '/SUSE/p' distros.txt

SUSE 10.2 12/07/2006

SUSE 11.0 06/19/2008

SUSE 10.3 10/04/2007

SUSE 10.1 05/11/2006

By including the slash-delimited regular expression /SUSE/, we are able to isolate the lines containing it in much the same manner as grep.

通过包含由斜杠界定的正则表达式 \/SUSE\/，我们能够孤立出包含它的文本行，和 grep 程序的功能 是相同的。

Finally, we’ll try negation by adding an ! to the address:

最后，我们将试着否定上面的操作，通过给这个地址添加一个感叹号：

[me@linuxbox ~]$ sed -n '/SUSE/!p' distros.txt

Fedora 10 11/25/2008

Ubuntu 8.04 04/24/2008

Fedora 8 11/08/2007

Ubuntu 6.10 10/26/2006

Fedora 7 05/31/2007

Ubuntu 7.10 10/18/2007

Ubuntu 7.04 04/19/2007

Fedora 6 10/24/2006

Fedora 9 05/13/2008

Ubuntu 6.06 06/01/2006

Ubuntu 8.10 10/30/2008

Fedora 5 03/20/2006

Here we see the expected result: all of the lines in the file except the ones matched by the regular expression.

这里我们看到期望的结果：输出了文件中所有的文本行，除了那些匹配这个正则表达式的文本行。

So far, we’ve looked at two of the sed editing commands, s and p. Here is a more complete list of the basic editing commands:

目前为止，我们已经知道了两个 sed 的编辑命令，s 和 p。这里是一个更加全面的基本编辑命令列表：

|  |  |
| --- | --- |
| *Table 21-8: sed Basic Editing Commands* | |
| Command | Description |
| = | Output current line number. |
| a | Append text after the current line. |
| d | Delete the current line. |
| i | Insert text in front of the current line. |
| p | Print the current line. By default, sed prints every line and only edits lines that match a specified address within the file. The default behavior can be overridden by specifying the -n option. |
| q | Exit sed without processing any more lines. If the -n option is not specified, output the current line. |
| Q | Exit sed without processing any more lines. |
| s/regexp/replacement/ | Substitute the contents of replacement wherever regexp is found. replacement may include the special character &, which is equivalent to the text matched by regexp. In addition, replacement may include the sequences \1 through \9, which are the contents of the corresponding subexpressions in regexp. For more about this, see the discussion of back references below. After the trailing slash following replacement, an optional flag may be specified to modify the s command’s behavior. |
| y/set1/set2 | Perform transliteration by converting characters from set1 to the corresponding characters in set2. Note that unlike tr, sed requires that both sets be of the same length. |

|  |  |
| --- | --- |
| *表21-8: sed 基本编辑命令* | |
| 命令 | 说明 |
| = | 输出当前的行号。 |
| a | 在当前行之后追加文本。 |
| d | 删除当前行。 |
| i | 在当前行之前插入文本。 |
| p | 打印当前行。默认情况下，sed 程序打印每一行，并且只是编辑文件中匹配 指定地址的文本行。通过指定-n 选项，这个默认的行为能够被忽略。 |
| q | 退出 sed，不再处理更多的文本行。如果不指定-n 选项，输出当前行。 |
| Q | 退出 sed，不再处理更多的文本行。 |
| s/regexp/replacement/ | 只要找到一个 regexp 匹配项，就替换为 replacement 的内容。 replacement 可能包括特殊字符 &，其等价于由 regexp 匹配的文本。另外， replacement 可能包含序列 \1到 \9，其是 regexp 中相对应的子表达式的内容。更多信息，查看 下面 back references 部分的讨论。在 replacement 末尾的斜杠之后，可以指定一个 可选的标志，来修改 s 命令的行为。 |
| y/set1/set2 | 执行字符转写操作，通过把 set1 中的字符转变为相对应的 set2 中的字符。 注意不同于 tr 程序，sed 要求两个字符集合具有相同的长度。 |

The s command is by far the most commonly used editing command. We will demonstrate just some of its power by performing an edit on our distros.txt file. We discussed before how the date field in distros.txt was not in a “computer- friendly” format. While the date is formatted MM/DD/YYYY, it would be better (for ease of sorting) if the format were YYYY-MM-DD. To perform this change on the file by hand would be both time-consuming and error prone, but with sed, this change can be performed in one step:

到目前为止，这个 s 命令是最常使用的编辑命令。我们将仅仅演示一些它的功能，通过编辑我们的 distros.txt 文件。我们以前讨论过 distros.txt 文件中的日期字段不是“友好地计算机”模式。 文件中的日期格式是 MM/DD/YYYY，但如果格式是 YYYY-MM-DD 会更好一些（利于排序）。手动修改 日期格式不仅浪费时间而且易出错，但是有了 sed，只需一步就能完成修改：

[me@linuxbox ~]$ sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/' distros.txt

SUSE 10.2 2006-12-07

Fedora 10 2008-11-25

SUSE 11.0 2008-06-19

Ubuntu 8.04 2008-04-24

Fedora 8 2007-11-08

SUSE 10.3 2007-10-04

Ubuntu 6.10 2006-10-26

Fedora 7 2007-05-31

Ubuntu 7.10 2007-10-18

Ubuntu 7.04 2007-04-19

SUSE 10.1 2006-05-11

Fedora 6 2006-10-24

Fedora 9 2008-05-13

Ubuntu 6.06 2006-06-01

Ubuntu 8.10 2008-10-30

Fedora 5 2006-03-20

Wow! Now that is an ugly looking command. But it works. In just one step, we have changed the date format in our file. It is also a perfect example of why regular expressions are sometimes jokingly referred to as a “write-only” medium. We can write them, but we sometimes cannot read them. Before we are tempted to run away in terror from this command, let’s look at how it was constructed. First, we know that the command will have this basic structure:

哇！这个命令看起来很丑陋。但是它起作用了。仅用一步，我们就更改了文件中的日期格式。 它也是一个关于为什么有时候会开玩笑地把正则表达式称为是“只写”媒介的完美的例子。我们 能写正则表达式，但是有时候我们不能读它们。在我们恐惧地忍不住要逃离此命令之前，让我们看一下 怎样来构建它。首先，我们知道此命令有这样一个基本的结构：

sed 's/regexp/replacement/' distros.txt

Our next step is to figure out a regular expression that will isolate the date. Since it is in MM/DD/YYYY format and appears at the end of the line, we can use an expression like this:

我们下一步是要弄明白一个正则表达式将要孤立出日期。因为日期是 MM/DD/YYYY 格式，并且 出现在文本行的末尾，我们可以使用这样的表达式：

[0-9]{2}/[0-9]{2}/[0-9]{4}$

which matches two digits, a slash, two digits, a slash, four digits, and the end of line. So that takes care of **regexp**, but what about **replacement**? To handle that, we must introduce a new regular expression feature that appears in some applications which use BRE. This feature is called **back references** and works like this: if the sequence \n appears in **replacement** where n is a number from one to nine, the sequence will refer to the corresponding subexpression in the preceding regular expression. To create the subexpressions, we simply enclose them in parentheses like so:

此表达式匹配两位数字，一个斜杠，两位数字，一个斜杠，四位数字，以及行尾。如此关心\_regexp\_， 那么\_replacement\_又怎样呢？为了解决此问题，我们必须介绍一个正则表达式的新功能，它出现 在一些使用 BRE 的应用程序中。这个功能叫做\_逆参照\_，像这样工作：如果序列\n 出现在\_replacement\_中 ，这里 n 是指从 1 到 9 的数字，则这个序列指的是在前面正则表达式中相对应的子表达式。为了 创建这个子表达式，我们简单地把它们用圆括号括起来，像这样：

([0-9]{2})/([0-9]{2})/([0-9]{4})$

We now have three subexpressions. The first contains the month, the second contains the day of the month, and the third contains the year. Now we can construct replacement as follows:

现在我们有了三个子表达式。第一个表达式包含月份，第二个包含某月中的某天，以及第三个包含年份。 现在我们就可以构建\_replacement\_，如下所示：

\3-\1-\2

which gives us the year, a dash, the month, a dash, and the day.

此表达式给出了年份，一个斜杠，月份，一个斜杠，和某天。

Now, our command looks like this:

sed 's/([0-9]{2})/([0-9]{2})/([0-9]{4})$/\3-\1-\2/' distros.txt

We have two remaining problems. The first is that the extra slashes in our regular expression will confuse sed when it tries to interpret the s command. The second is that since sed, by default, accepts only basic regular expressions, several of the characters in our regular expression will be taken as literals, rather than as metacharacters. We can solve both these problems with a liberal application of backslashes to escape the offending characters:

我们还有两个问题。第一个是在我们表达式中额外的斜杠将会迷惑 sed，当 sed 试图解释这个 s 命令 的时候。第二个是因为 sed，默认情况下，只接受基本的正则表达式，在表达式中的几个字符会 被当作文字字面值，而不是元字符。我们能够解决这两个问题，通过反斜杠的自由应用来转义 令人不快的字符：

sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/' distros.txt

And there you have it!

你掌握了吧!

Another feature of the s command is the use of optional flags that may follow the replacement string. The most important of these is the g flag, which instructs sed to apply the search and replace globally to a line, not just to the first instance, which is the default. Here is an example:

s 命令的另一个功能是使用可选标志，其跟随替代字符串。一个最重要的可选标志是 g 标志，其 指示 sed 对某个文本行全范围地执行查找和替代操作，不仅仅是对第一个实例，这是默认行为。 这里有个例子：

[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/'

aaaBbbccc

We see that the replacement was performed, but only to the first instance of the letter “b,” while the remaining instances were left unchanged. By adding the g flag, we are able to change all the instances:

我们看到虽然执行了替换操作，但是只针对第一个字母 “b” 实例，然而剩余的实例没有更改。通过添加 g 标志， 我们能够更改所有的实例：

[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/g'

aaaBBBccc

So far, we have only given sed single commands via the command line. It is also possible to construct more complex commands in a script file using the -f option. To demonstrate, we will use sed with our distros.txt file to build a report. Our report will feature a title at the top, our modified dates, and all the distribution names converted to upper case. To do this, we will need to write a script, so we’ll fire up our text editor and enter the following:

目前为止，通过命令行我们只让 sed 执行单个命令。使用-f 选项，也有可能在一个脚本文件中构建更加复杂的命令。 为了演示，我们将使用 sed 和 distros.txt 文件来生成一个报告。我们的报告以开头标题，修改过的日期，以及 大写的发行版名称为特征。为此，我们需要编写一个脚本，所以我们将打开文本编辑器，然后输入以下文字：

# sed script to produce Linux distributions report

1 i\

\

Linux Distributions Report\

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

y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/

We will save our sed script as distros.sed and run it like this:

我们将把 sed 脚本保存为 distros.sed 文件，然后像这样运行它：

[me@linuxbox ~]$ sed -f distros.sed distros.txt

Linux Distributions Report

SUSE 10.2 2006-12-07

FEDORA 10 2008-11-25

SUSE 11.0 2008-06-19

UBUNTU 8.04 2008-04-24

FEDORA 8 2007-11-08

SUSE 10.3 2007-10-04

UBUNTU 6.10 2006-10-26

FEDORA 7 2007-05-31

UBUNTU 7.10 2007-10-18

UBUNTU 7.04 2007-04-19

SUSE 10.1 2006-05-11

FEDORA 6 2006-10-24

FEDORA 9 2008-05-13

As we can see, our script produces the desired results, but how does is do it? Let’s take another look at our script. We’ll use cat to number the lines:

正如我们所见，我们的脚本文件产生了期望的结果，但是它是如何做到的呢？让我们再看一下我们的脚本文件。 我们将使用 cat 来给每行文本编号：

[me@linuxbox ~]$ cat -n distros.sed

1 # sed script to produce Linux distributions report

2

3 1 i\

4 \

5 Linux Distributions Report\

6

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

8 y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/

Line one of our script is a comment. Like many configuration files and programming languages on Linux systems, comments begin with the # character and are followed by human-readable text. Comments can be placed anywhere in the script (though not within commands themselves) and are helpful to any humans who might need to identify and/or maintain the script.

我们脚本文件的第一行是一条注释。如同 Linux 系统中的许多配置文件和编程语言一样，注释以#字符开始， 然后是人类可读的文本。注释可以被放到脚本中的任意地方（虽然不在命令本身之中），且对任何 可能需要理解和／或维护脚本的人们都很有帮助。

Line two is a blank line. Like comments, blank lines may be added to improve readability.

第二行是一个空行。正如注释一样，添加空白行是为了提高程序的可读性。

Many sed commands support line addresses. These are used to specify which lines of the input are to be acted upon. Line addresses may be expressed as single line numbers, line number ranges, and the special line number “$” which indicates the last line of input.

许多 sed 命令支持行地址。这些行地址被用来指定对输入文本的哪一行执行操作。行地址可能被 表示为单独的行号，行号范围，以及特殊的行号“$”，它表示输入文本的最后一行。

Lines three through six contain text to be inserted at the address 1, the first line of the input. The i command is followed by the sequence backslash-carriage return to produce an escaped carriage return, or what is called a line continuation character. This sequence, which can be used in many circumstances including shell scripts, allows a carriage return to be embedded in a stream of text without signaling the interpreter (in this case sed) that the end of the line has been reached. The i, and likewise, the a (which appends text, rather than inserting it) and c (which replaces text) commands, allow multiple lines of text as long as each line, except the last, ends with a line continuation character. The sixth line of our script is actually the end of our inserted text and ends with a plain carriage return rather than a line continuation character, signaling the end of the i command.

从第三行到第六行所包含地文本要被插入到地址 1 处，也就是输入文本的第一行中。这个 i 命令 之后是反斜杠回车符，来产生一个转义的回车符，或者就是所谓的连行符。这个序列能够 被用在许多环境下，包括 shell 脚本，从而允许把回车符嵌入到文本流中，而没有通知 解释器（在这是指 sed 解释器）已经到达了文本行的末尾。这个 i 命令，同样地，命令 a（追加文本， 而不是插入文本）和 c（取代文本）命令都允许多个文本行，只要每个文本行，除了最后一行，以一个 连行符结束。实际上，脚本的第六行是插入文本的末尾，它以一个普通的回车符结尾，而不是一个 连行符，通知解释器 i 命令结束了。

Note: A line continuation character is formed by a backslash followed immediately by a carriage return. No intermediary spaces are permitted.

注意：一个连行符由一个斜杠字符其后紧跟一个回车符组成。它们之间不允许有空白字符。

Line seven is our search and replace command. Since it is not preceded by an address, each line in the input stream is subject to its action.

第七行是我们的查找和替代命令。因为命令之前没有添加地址，所以输入流中的每一行文本 都得服从它的操作。

Line eight performs transliteration of the lowercase letters into uppercase letters. Note that unlike tr, the y command in sed does not support character ranges (for example, [a-z]), nor does it support POSIX character classes. Again, since the y command is not preceded by an address, it applies to every line in the input stream.

第八行执行小写字母到大写字母的字符替换操作。注意不同于 tr 命令，这个 sed 中的 y 命令不 支持字符区域（例如，[a-z]），也不支持 POSIX 字符集。再说一次，因为 y 命令之前不带地址， 所以它会操作输入流的每一行。

People Who Like sed Also Like…

喜欢 sed 的人们也会喜欢。。。

sed is a very capable program, able to perform fairly complex editing tasks to streams of text. It is most often used for simple one line tasks rather than long scripts. Many users prefer other tools for larger tasks. The most popular of these are awk and perl. These go beyond mere tools, like the programs covered here, and extend into the realm of complete programming languages. perl, in particular, is often used in place of shell scripts for many system management and administration tasks, as well as being a very popular medium for web development. awk is a little more specialized. Its specific strength is its ability to manipulate tabular data. It resembles sed in that awk programs normally process text files line-by-line, using a scheme similar to the sed concept of an address followed by an action. While both awk and perl are outside the scope of this book, they are very good skills for the Linux command line user.

sed 是一款非常强大的程序，它能够针对文本流完成相当复杂的编辑任务。它最常 用于简单的行任务，而不是长长的脚本。许多用户喜欢使用其它工具，来执行较大的工作。 在这些工具中最著名的是 awk 和 perl。它们不仅仅是工具，像这里介绍的程序，且延伸到 完整的编程语言领域。特别是 perl，经常被用来代替 shell 脚本，来完成许多系统管理任务， 同时它也是一款非常流行网络开发语言。awk 更专用一些。其具体优点是其操作表格数据的能力。 awk 程序通常逐行处理文本文件，这点类似于 sed，awk 使用了一种方案，其与 sed 中地址 之后跟随编辑命令的概念相似。虽然关于 awk 和 perl 的内容都超出了本书所讨论的范围， 但是对于 Linux 命令行用户来说，它们都是非常好的技能。

**aspell**

The last tool we will look at is aspell, an interactive spelling checker. The aspell program is the successor to an earlier program named ispell, and can be used, for the most part, as a drop-in replacement. While the aspell program is mostly used by other programs that require spell checking capability, it can also be used very effectively as a stand-alone tool from the command line. It has the ability to intelligently check various type of text files, including HTML documents, C/C++ programs, email messages and other kinds of specialized texts.

我们要查看的最后一个工具是 aspell，一款交互式的拼写检查器。这个 aspell 程序是早先 ispell 程序 的继承者，大多数情况下，它可以被用做一个替代品。虽然 aspell 程序大多被其它需要拼写检查能力的 程序使用，但它也可以作为一个独立的命令行工具使用。它能够智能地检查各种类型的文本文件， 包括 HTML 文件，C/C++ 程序，电子邮件和其它种类的专业文本。

To spell check a text file containing simple prose, it could be used like this:

拼写检查一个包含简单的文本文件，可以这样使用 aspell:

aspell check textfile

where **textfile** is the name of the file to check. As a practical example, let’s create a simple text file named foo.txt containing some deliberate spelling errors:

这里的 textfile 是要检查的文件名。作为一个实际例子，让我们创建一个简单的文本文件，叫做 foo.txt， 包含一些故意的拼写错误：

[me@linuxbox ~]$ cat > foo.txt

The quick brown fox jimped over the laxy dog.

Next we’ll check the file using aspell:

下一步我们将使用 aspell 来检查文件：

[me@linuxbox ~]$ aspell check foo.txt

As aspell is interactive in the check mode, we will see a screen like this:

因为 aspell 在检查模式下是交互的，我们将看到像这样的一个屏幕：

The quick brown fox jimped over the laxy dog.

1)jumped 6)wimped

2)gimped 7)camped

3)comped 8)humped

4)limped 9)impede

5)pimped 0)umped

i)Ignore I)Ignore all

r)Replace R)Replace all

a)Add l)Add Lower

b)Abort x)Exit

?

At the top of the display, we see our text with a suspiciously spelled word highlighted. In the middle, we see ten spelling suggestions numbered zero through nine, followed by a list of other possible actions. Finally, at the very bottom, we see a prompt ready to accept our choice.

在显示屏的顶部，我们看到我们的文本中有一个拼写可疑且高亮显示的单词。在中间部分，我们看到 十个拼写建议，序号从 0 到 9，然后是一系列其它可能的操作。最后，在最底部，我们看到一个提示符， 准备接受我们的选择。

If we press the 1 key, aspell replaces the offending word with the word “jumped” and moves on to the next misspelled word which is “laxy.” If we select the replacement “lazy,” aspell replaces it and terminates. Once aspell has finished, we can examine our file and see that the misspellings have been corrected:

如果我们按下 1 按键，aspell 会用单词 “jumped” 代替错误单词，然后移动到下一个拼写错的单词，就是 “laxy”。如果我们选择替代物 “lazy”，aspell 会替换 “laxy” 并且终止。一旦 aspell 结束操作，我们 可以检查我们的文件，会看到拼写错误的单词已经更正了。

[me@linuxbox ~]$ cat foo.txt

The quick brown fox jumped over the lazy dog.

Unless told otherwise via the command line option –dont-backup, aspell creates a backup file containing the original text by appending the extension .bak to the filename.

除非由命令行选项 --dont-backup 告诉 aspell，否则通过追加扩展名.bak 到文件名中, aspell 会创建一个包含原始文本的备份文件。

Showing off our sed editing prowess, we’ll put our spelling mistakes back in so we can reuse our file:

为了炫耀 sed 的编辑本领，我们将还原拼写错误，从而能够重用我们的文件：

[me@linuxbox ~]$ sed -i 's/lazy/laxy/; s/jumped/jimped/' foo.txt

The sed option -i tells sed to edit the file “in-place,” meaning that rather than sending the edited output to standard output, it will re-write the file with the changes applied. We also see the ability to place more than one editing command on the line by separating them with a semicolon.

这个 sed 选项-i，告诉 sed 在适当位置编辑文件，意思是不要把编辑结果发送到标准输出中。sed 会把更改应用到文件中， 以此重新编写文件。我们也看到可以把多个 sed 编辑命令放在同一行，编辑命令之间由分号分隔开来。

Next, we’ll look at how aspell can handle different kinds of text files. Using a text editor such as vim (the adventurous may want to try sed), we will add some HTML markup to our file:

下一步，我们将看一下 aspell 怎样来解决不同种类的文本文件。使用一个文本编辑器，例如 vim（胆大的人可能想用 sed）， 我们将添加一些 HTML 标志到文件中：

<html>

<head>

<title>Mispelled HTML file</title>

</head>

<body>

<p>The quick brown fox jimped over the laxy dog.</p>

</body>

</html>

Now, if we try to spell check our modified file, we run into a problem. If we do it this way:

现在，如果我们试图拼写检查我们修改的文件，我们会遇到一个问题。如果我们这样做：

[me@linuxbox ~]$ aspell check foo.txt

we’ll get this:

我们会得到这些：

<html>

<head>

<title>Mispelled HTML file</title>

</head>

<body>

<p>The quick brown fox jimped over the laxy dog.</p>

</body>

</html>

1) HTML 4) Hamel

2) ht ml 5) Hamil

3) ht-ml 6) hotel

i) Ignore I) Ignore all

r) Replace R) Replace all

a) Add l) Add Lower

b) Abort x) Exit

?

aspell will see the contents of the HTML tags as misspelled. This problem can be overcome by including the -H (HTML) checking mode option, like this:

aspell 会认为 HTML 标志的内容是拼写错误。通过包含-H（HTML）检查模式选项，这个问题能够 解决，像这样：

[me@linuxbox ~]$ aspell -H check foo.txt

which will result in this:

这会导致这样的结果：

<html>

<head>

<title><b>Mispelled</b> HTML file</title>

</head>

<body>

<p>The quick brown fox jimped over the laxy dog.</p>

</body>

</html>

1) Mi spelled 6) Misapplied

2) Mi-spelled 7) Miscalled

3) Misspelled 8) Respelled

4) Dispelled 9) Misspell

5) Spelled 0) Misled

i) Ignore I) Ignore all

r) Replace R) Replace all

a) Add l) Add Lower

b) Abort x) Exit

?

The HTML is ignored and only the non-markup portions of the file are checked. In this mode, the contents of HTML tags are ignored and not checked for spelling. However, the contents of ALT tags, which benefit from checking, are checked in this mode.

这个 HTML 标志被忽略了，并且只会检查文件中非标志部分的内容。在这种模式下，HTML 标志的 内容被忽略了，不会进行拼写检查。然而，ALT 标志的内容，会被检查。

Note: By default, aspell will ignore URLs and email addresses in text. This behavior can be overridden with command line options. It is also possible to specify which markup tags are checked and skipped. See the aspell man page for details.

注意：默认情况下，aspell 会忽略文本中 URL 和电子邮件地址。通过命令行选项，可以重写此行为。 也有可能指定哪些标志进行检查及跳过。详细内容查看 aspell 命令手册。

**总结归纳**

In this chapter, we have looked at a few of the many command line tools that operate on text. In the next chapter, we will look at several more. Admittedly, it may not seem immediately obvious how or why you might use some of these tools on a day-to-day basis, though we have tried to show some semi-practical examples of their use. We will find in later chapters that these tools form the basis of a tool set that is used to solve a host of practical problems. This will be particularly true when we get into shell scripting, where these tools will really show their worth.

在这一章中，我们已经查看了一些操作文本的命令行工具。在下一章中，我们会再看几个命令行工具。 诚然，看起来不能立即显现出怎样或为什么你可能使用这些工具为日常的基本工具， 虽然我们已经展示了一些半实际的命令用法的例子。我们将在随后的章节中发现这些工具组成 了解决实际问题的基本工具箱。这将是确定无疑的，当我们学习 shell 脚本的时候， 到时候这些工具将真正体现出它们的价值。

**拓展阅读**

The GNU Project website contains many online guides to the tools discussed in this chapter.

GNU 项目网站包含了本章中所讨论工具的许多在线指南。

* From the Coreutils package:
* 来自 Coreutils 软件包：

<http://www.gnu.org/software/coreutils/manual/coreutils.html#Output-of-entire-files>

<http://www.gnu.org/software/coreutils/manual/coreutils.html#Operating-on-sorted-files>

<http://www.gnu.org/software/coreutils/manual/coreutils.html#Operating-on-fields-within-a-line>

<http://www.gnu.org/software/coreutils/manual/coreutils.html#Operating-on-characters>

* From the Diffutils package:
* 来自 Diffutils 软件包：

<http://www.gnu.org/software/diffutils/manual/html_mono/diff.html>

* sed 工具

<http://www.gnu.org/software/sed/manual/sed.html>

* aspell 工具

<http://aspell.net/man-html/index.html>

* There are many other online resources for sed, in particular:
* 尤其对于 sed 工具，还有很多其它的在线资源：

<http://www.grymoire.com/Unix/Sed.html>

<http://sed.sourceforge.net/sed1line.txt>

* Also try googling “sed one liners”, “sed cheat sheets”
* 试试用 google 搜索 “sed one liners”, “sed cheat sheets” 关键字

**友情提示**

There are a few more interesting text manipulation commands worth investigating. Among these are: split (split files into pieces), csplit (split files into pieces based on context), and sdiff (side-by-side merge of file differences.)

有一些更有趣的文本操作命令值得。在它们之间有：split（把文件分割成碎片）， csplit（基于上下文把文件分割成碎片），和 sdiff（并排合并文件差异）。