

续表

| 命令 | 代表的含义 |
|----------------------------------|---|
| ls | 列出远程系统上的目录列表 |
| lcd Desktop | 切换至本地系统的 ~/Desktop 目录。本例中，ftp 程序是在 home (~) 目录下启动的，此命令行将工作目录切换至 ~/Desktop 下 |
| get ubuntu-8.04-desktop-i386.iso | 告诉远程系统将 ubuntu-8.04-desktop-i386.iso 映像文件发送给本地系统。由于本地系统的工作目录已经切换至 ~/Desktop 下，映像文件也会下载到该目录下 |
| bye | 注销登录远程服务器并且结束 ftp 程序。也可以使用 quit 或 exit 命令代替 |

在提示符 ftp>后面输入 help 会显示 ftp 所支持的命令列表。在已被授予足够权限的服务器上使用 ftp 命令，可以执行许多常见的文件管理任务。虽然这很笨拙，但这不失为一种办法。

16.2.2 lftp——更好的 ftp（文件传输协议）

ftp 并不是唯一的命令行 FTP 客户端。事实上，有很多这样的命令行。其中更好用也更受欢迎的一个就是由 Alexander Lukyanov 编写的 lftp 命令，它与传统的 ftp 程序功能类似但却有很多额外的便利功能，包括多协议支持（HTTP）、下载失败时自动重新尝试、后台进程支持、Tab 键完成文件名输入等许多其他的功能。

16.2.3 wget——非交互式网络下载工具

wget 是另一个用于文件下载的命令行程序。该命令既可以用于从网站上下载内容也可以用于从 FTP 站点下载，单个文件、多个文件甚至整个网站都可以被下载。下例演示的就是用 wget 命令下载网站 <http://www.linuxcommand.org/> 第一页内容。

```
[me@linuxbox ~]$ wget http://linuxcommand.org/index.php
--11:02:51-- http://linuxcommand.org/index.php
              => `index.php'
Resolving linuxcommand.org... 66.35.250.210
Connecting to linuxcommand.org|66.35.250.210|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

[=>                                         ] 3,120          --.-K/s
11:02:51 (161.75 MB/s) - `index.php' saved [3120]
```

wget 命令的许多参数选项支持递归下载、后台文件下载（允许下线的情况下继续下载）以及继续下载部分被下载的文件等操作。这些特点都清楚的写在该命令的 better-than-average（优于平均水平） man 手册页中。

16.3 与远程主机的安全通信

多年以前，类 UNIX 操作系统就可以通过网络进行远程操控。早期，在互联网还未普及的时候，登录远程主机有两个很受欢迎的命令——rlogin 和 telnet 命令。但是它们与 ftp 命令有着相同的致命缺点，即所有通信信息（包括用户名和密码）都是以明文的方式传输的，所以它们并不适用于互联网时代。

16.3.1 ssh——安全登录远程计算机

为了解决明文传送的问题，一个叫做 SSH（Secure Shell 的缩写）的新协议应运而生。SSH 协议解决了与远程主机进行安全通信的两个基本问题：第一，该协议能验证远程主机的身份是否真实，从而避免中间人攻击；第二，该协议将本机与远程主机之间的通信内容全部加密。

SSH 协议包括两个部分：一个是运行在远程主机上的 SSH 服务端，用来监听端口 22 上可能过来的连接请求；另一个是本地系统上的 SSH 客户端，用来与远程服务器进行通信。

多数 Linux 发行版都采用 BSD 项目的 openSSH（SSH 的免费开源实现）方法实现 SSH。有些发行版如 Red Hat 会默认包含客户端包和服务端包，而有的版本如 Ubuntu 则仅仅提供客户端包。系统想要接收远程连接，就必须安装、配置以及运行 OpenSSH-server 软件包，并且必须允许 TCP 端口 22 上进来的网络连接（当服务器正在运行防火墙或是在防火墙后面时）。

注意

如果没有可以连接的远程系统但却要尝试本例，可以在系统已安装了 OpenSSH-server 软件包的基础上将远程主机名设为本地主机名。这样，机器便会与自身建立网络连接。

ssh 命令作为 SSH 客户端程序用于建立与远程 SSH 服务器之间的通信再合适不过了。如下便是使用 ssh 客户端程序来建立与远程主机 remote-sys 的连接的例子。

```
[me@linuxbox ~]$ ssh remote-sys
The authenticity of host 'remote-sys (192.168.1.4)' can't be established.
RSA key fingerprint is 41:ed:7a:df:23:19:bf:3c:a5:17:bc:61:b3:7f:d9:bb.
Are you sure you want to continue connecting (yes/no)?
```

第一次尝试连接的时候，由于 ssh 程序从来没有接触过此远程主机，所以会跳出一条“不能确定远程主机真实性”的消息。当出现这条警告消息的时候输入 yes 接受远程主机的身份，一旦建立了连接，会提示用户输入密码。

```
Warning: Permanently added 'remote-sys,192.168.1.4' (RSA) to the list of known hosts.
me@remote-sys's password:
```

密码输入正确后，远程系统的 shell 提示符便出现了。

```
Last login: Tue Aug 30 13:00:48 2011
[me@remote-sys ~]$
```

远程 shell 对话将一直开启着，直到用户在该对话框中输入 exit 命令断开与远程系统的连接。连接一旦断开后，本地 shell 会话恢复，本地 shell 提示符又重新出现。

使用非本地系统上使用的用户名也可以登录远程系统。例如，当本地用户 me 在远程系统上有一个 bob 账户，me 用户就可以用下面的命令登录远程系统上的 bob 账户。

```
[me@linuxbox ~]$ ssh bob@remote-sys
bob@remote-sys's password:
Last login: Tue Aug 30 13:03:21 2011
[bob@remote-sys ~]$
```

正如前面所说，ssh 命令会验证远程主机的真实性。如果远程主机没有成功验证，就会跳出下面的警告信息。

```
[me@linuxbox ~]$ ssh remote-sys
@@@@@@@@@@@WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
41:ed:7a:df:23:19:bf:3c:a5:17:bc:61:b3:7f:d9:bb.
Please contact your system administrator.
Add correct host key in /home/me/.ssh/known_hosts to get rid of this message.
Offending key in /home/me/.ssh/known_hosts:1
RSA host key for remote-sys has changed and you have requested strict
checking.
Host key verification failed.
```

一般有两个原因导致此警告信息：第一，有攻击者正在尝试中间人攻击，

这种情况很少见，因为大家都知道 ssh 会提醒用户发生了这样的攻击；第二，也是更有可能的原因便是远程系统在某种程度上改变了，例如，远程主机重新安装了操作系统或者 SSH 服务端。然而，从安全性上考虑，不应该因为发生第一种情况的可能性小就直接忽略，该警告出现时仍然要向远程系统的管理者核对。

确定该警告是由良性原因导致后，在客户端修正问题就安全了，解决方法就是用某种文本编辑器（也许是 vim）从`~/.ssh/known_hosts` 文件中移除过时的密钥。警告当中包含如下这句话。

Offending key in /home/me/.ssh/known_hosts:1

这意味着`known_hosts` 文件的第一行包含了相悖的密钥。将该行删除后，ssh 程序就能重新获取远程系统新的身份验证凭据了。

ssh 命令除了能开启远程系统上的 shell 会话外，还能在远程系统上执行单个简单命令。例如，我们可以在`remote-sys` 远程主机上执行`free` 命令并将其结果直接输出到本地系统上。

```
[me@linuxbox ~]$ ssh remote-sys free
me@twin4's password:
      total    used     free   shared  buffers   cached
Mem:       775536   507184   268352        0   110068   154596
-/+ buffers/cache:   242520   533016
Swap:      1572856        0   1572856
[me@linuxbox ~]$
```

该特性可以有更有趣的用途，比如在远程系统上执行`ls` 命令后直接将运行结果输出到本地系统的文件中。

```
[me@linuxbox ~]$ ssh remote-sys 'ls *' > dirlist.txt
me@twin4's password:
[me@linuxbox ~]$
```

请注意命令行中的单引号，之所以使用单引号是因为我们并不希望本该在远程主机上进行的路径扩展在本地系统上进行。同样，如果我们还希望执行结果能直接输出到远程系统的文件中，就应该将重定向符号和文件名一起置于单引号中。

```
[me@linuxbox ~]$ ssh remote-sys 'ls * > dirlist.txt'
```

SSH 的隧道技术

通过 SSH 与远程主机建立连接后，一个本地与远程系统之间的加密隧道就被建立了。通常，该隧道用于将在本地系统输入的命令安全地传送给远程

系统并将结果安全地传送回来。除了这样的基本功能外，SSH 协议还可以在本地与远程系统之间建立某种虚拟专用网络（VPN，Virtual Private Network），从而实现多种网络通信经此加密隧道传送。

也许，这一特性最常见的用途就是允许 X Window 系统之间的通信。在一个运行 X Window 服务器（也就是使用图形用户接口的机器）的系统上，可以远程启动和运行远程主机上的 X Window（图形化应用）客户端程序并将其图像化效果显示在本地系统上。这个过程操作起来很容易，假定我们正在操作一台运行 X Window 服务器的叫做 linuxbox 的本地主机，并打算在 remote-sys 远程主机上远程运行 xload 程序，还要在本地主机上看到该程序运行后的图形化效果，示例如下。

```
[me@linuxbox ~]$ ssh -X remote-sys
me@remote-sys's password:
Last login: Mon Sep 05 13:23:11 2011
[me@remote-sys ~]$ xload
```

xload 命令在远程系统上运行后，其图形窗口就会出现在本地系统上。然而对于某些系统，可能需要用 -Y 参数选项而不是像上面所用的 -X 选项完成该操作。

16.3.2 scp 和 sftp——安全传输文件

OpenSSH 软件包包含了两个使用 SSH 加密隧道进行网络间文件复制的程序，scp（secure copy 的缩写）便是其中之一。该命令与普通的文件复制命令 cp 类似，而它们之间最大的差别在于 scp 命令的源或目的地路径前面多个远程主机名和冒号。下面的例子实现了从 remote-sys 远程系统的 home 目录中将 document.txt 的文件复制到本地系统当前工作目录下的操作。

```
[me@linuxbox ~]$ scp remote-sys:document.txt .
me@remote-sys's password:
document.txt                                100% 5581      5.5KB/s   00:00
[me@linuxbox ~]$
```

与 ssh 命令一样，如果不是用本地系统的用户名登录远程系统，那么就需在远程主机名前添加将要登录的远程系统的账户名，示例如下。

```
[me@linuxbox ~]$ scp bob@remote-sys:document.txt .
```

另外一个 SSH 文件复制程序是 sftp。顾名思义，它是 ftp 程序的安全版本。sftp 与我们先前使用的 ftp 程序功能极为相似，只是 sftp 是用 SSH 加密隧道传输

信息而不是以明文方式传输。sftp 相比传统的 ftp 而言，还有一个重要的优点，就是它并不需要远程主机上运行 FTP 服务器，仅仅需要 SSH 服务器。这就意味着任何与 SSH 客户端连接的远程机器都可以当作 FTP 服务器使用，下面就是一个简单的会话实例。

```
[me@linuxbox ~]$ sftp remote-sys
Connecting to remote-sys...
me@remote-sys's password:
sftp> ls
ubuntu-8.04-desktop-i386.iso
sftp> lcd Desktop
sftp> get ubuntu-8.04-desktop-i386.iso
Fetching /home/me/ubuntu-8.04-desktop-i386.iso to ubuntu-8.04-desktop-i386.iso

/home/me/ubuntu-8.04-desktop-i386.iso 100% 699MB 7.4MB/s 01:35
sftp> bye
```

注意

Linux 发行版中许多图形文件管理器都支持 SFTP 协议。不管系统使用的是 Nautilus (GNOME) 图形界面还是 Konqueror (KDE) 图形界面，都可以在地址栏中输入以 sftp://开头的 URI，并对存储在运行 SSH 服务器的远程系统上的文件进行操作。

Windows 的 SSH 客户端

假使你正在操作一台 Windows 系统的计算机，但需要登录 Linux 服务器进行一些实际的操作。该怎么办？当然是为你的 Windows 计算机配置一个 SSH 客户端程序！有很多这样的工具。最受欢迎的可能要算 Simon Tatham 及其团队开发的 PuTTY 程序了。PuTTY 程序会显示一个终端窗口，并允许 Windows 用户在远程主机中打开一个 SSH（或 Telnet）会话，该程序也提供与 scp 和 sftp 命令功能类似的命令。

PuTTY 程序可以从 <http://www.chiark.greenend.org.uk/~sgtatham/putty/> 网站上下载。

第 17 章

文件搜索

已经学习 Linux 这么长时间了，相信大家有一点已经很清楚，就是 Linux 系统含有非常多的文件！这就自然产生一个问题，“我们应该怎样查找东西？”。虽然我们已经知道，Linux 文件系统良好的组织结构，源自于类 UNIX 的操作系统代代传承的习俗，但是仅文件数量就会引起可怕的问题。

本章我们主要介绍两个用于在 Linux 系统中搜索文件的工具。

- **locate:** 通过文件名查找文件。
- **find:** 在文件系统目录框架中查找文件。

同时，我们也会介绍一个通常与文件搜索命令一起使用、处理搜索结果文件列表的命令。

- **xargs:** 从标准输入中建立、执行命令行。

此外，我们还介绍了两个辅助工具。

- **touch:** 更改文件的日期时间。

- stat: 显示文件或文件系统的状态。

17.1 locate——较简单的方式查找文件

locate 命令通过快速搜索数据库，以寻找路径名与给定子字符串相匹配的文件，同时输出所有匹配结果。例如，假定查找名称以 zip 字符串开头的程序，由于查找的是程序文件，所以可以认为包含所要查找的程序的目录名应以 bin/结尾。因此，可以尝试下面的命令行。

```
[me@linuxbox ~]$ locate bin/zip
```

locate 程序将搜索该路径名数据库，并输出文件名包含字符串 bin/zip 的所有文件。

```
/usr/bin/zip  
/usr/bin/zipcloak  
/usr/bin/zipgrep  
/usr/bin/zipinfo  
/usr/bin/zipnote  
/usr/bin/zipsplit
```

有时搜索需求并不是这么简单，这时便可以用 locate 命令结合其他诸如 grep 这样的工具实现一些更有趣的搜索。

```
[me@linuxbox ~]$ locate zip | grep bin  
/bin/bunzip2  
/bin/bzip2  
/bin/bzip2recover  
/bin/gunzip  
/bin/gzip  
/usr/bin/funzip  
/usr/bin/gpg-zip  
/usr/bin/preunzip  
/usr/bin/prezip  
/usr/bin/prezip-bin  
/usr/bin/unzip  
/usr/bin/unzipsfx  
/usr/bin/zip  
/usr/bin/zipcloak  
/usr/bin/zipgrep  
/usr/bin/zipinfo  
/usr/bin/zipnote  
/usr/bin/zipsplit
```

locate 程序已经使用了很长时间，因此出现了多种衍生体。slocate 和 mlocate 是现代 Linux 发行版本中最常见的两个衍生体，而它们通常都是由名为 locate 的符号链接访问。不同版本的 locate 有一些相同的选项设置，而有些版本则包括

正则表达式匹配（将会在第 19 章涉及）和通配符支持等。我们可以查看 locate 的 man 手册页确定系统安装的是哪个版本的 locate。

locate 的搜索数据库从何而来

你也许曾注意过，有些 Linux 版本，系统刚刚安装好时 locate 命令并不能工作，但是如果第二天再尝试，就会发现它又能正常工作，这到底是怎么回事呢？其实，是因为 locate 的搜索数据库由另外一个叫做 updatedb 的程序创建，通常该程序作为一个 cron 任务定期执行。所谓 cron 任务就是指定期由 cron 守护进程执行的任务，多数装有 locate 命令的系统每天执行一次 updatedb 命令。由此可见，locate 的搜索数据库并不是持续更新的，所以 locate 命令查找不到非常新的文件。解决方法就是切换为超级用户，在提示框下手动运行 updatedb 程序。

17.2 find——较复杂的方式查找文件

locate 程序查找文件仅仅是依据文件名，而 find 程序则是依据文件的各种属性在既定的目录（及其子目录）里查找。本章将会花大量的篇幅介绍 find 命令的用法，因为它有很多有趣的特性会在后面有关编程概念的章节中多次讲到。

find 最简单的用法就是用户给定一个或是更多目录名作为其搜索范围。下面就用 find 命令列出当前系统主目录（~）下的文件列表清单。

```
[me@linuxbox ~]$ find ~
```

对于一些比较活跃的用户，一般系统内文件会比较多，使得上述命令行输出的列表肯定很长。不过，列表信息是以标准形式输出的，所以可以直接将此输出结果作为其他程序的输入。如下就是用 wc 程序计算 find 命令搜索到的文件的总量。

```
[me@linuxbox ~]$ find ~ | wc -l  
47068
```

大家忙碌起来吧！find 命令的美妙之处就是可以用来搜索符合特定要求的文件，它通过综合应用 test 选项、action 选项以及 options 选项（看起来有点奇怪）实现高级文件搜索。下面让我们首先了解 test 选项。

17.2.1 test 选项

假定我们想要查找的是目录文件，我们可以添加下面的 test 选项达到此目的。

```
[me@linuxbox ~]$ find ~ -type d | wc -l
1695
```

添加 test 参数-type d 可以将搜索范围限制为目录，而下面例子中使用-type f 则表示只对普通文件进行搜索。

```
[me@linuxbox ~]$ find ~ -type f | wc -l
38737
```

表 17-1 列出了 find 命令支持的常用文件类型。

表 17-1 find 支持搜索的文件类型

| 文件类型 | 描述 |
|------|--------|
| b | 块设备文件 |
| c | 字符设备文件 |
| d | 目录 |
| f | 普通文件 |
| l | 符号链接 |

另外我们还可以通过添加其他的 test 项参数实现依据文件大小和文件名的搜索。如下命令行就是用来查找所有符合*.JPG 通配符格式以及大小超过 1MB 的普通文件。

```
[me@linuxbox ~]$ find ~ -type f -name "*.JPG" -size +1M | wc -l
840
```

本例中添加的-name "*.JPG" 的 test 选项表示查找的是符合.JPG 通配符格式的文件。注意，这里将通配符扩在双引号中是为了避免 shell 路径名扩展。另外添加的-size +1M test 选项，前面的加号表示查找的文件大小比给定的数值 1M 大。若字符串前面是减号则代表要比给定数值小，没有符号则表示与给定值完全相等。末尾的 M 是计量单位 MB（兆字节）的简写，表 17-2 中列出了每个字母与特定计量单位之间的对应关系。

表 17-2 find 支持的计量单位

| 字母 | 单位 |
|----|-----------------------|
| b | 512 字节的块（没有具体说明时的默认值） |
| c | 字节 |
| w | 两个字节的字 |

续表

| 字母 | 单位 |
|----|-----------------------------|
| k | KB (每单位包含 1,024 字节) |
| M | MB (每单位包含 1,048,576 字节) |
| G | GB (每单位包含 1,073,741,824 字节) |

find 命令支持多种 test 参数，表 17-3 概括了一些常见的参数。注意，前面所讲述的“+”和“-”号的用法适用于所有用到数值参数的情况。

表 17-3 find 命令的 test 项参数

| test 参数 | 描述 |
|----------------|---|
| -cmin n | 匹配 n 分钟前改变状态（内容或属性）的文件或目录。如果不到 n 分钟，就用-n，如果超过 n 分钟，就用+n |
| -cnewer file | 匹配内容或属性的修改时间比文件 file 更晚的文件或目录 |
| -ctime n | 匹配系统中 n*24 小时前文件状态被改变（内容、属性、访问权限等）的文件或目录 |
| -empty | 匹配空文件及空目录 |
| -group name | 匹配属于 name 组的文件或目录。name 可以描述为组名，也可以描述为该组的 ID 号 |
| -iname pattern | 与-name test 项功能类似只是不区分大小写 |
| -inum n | 匹配索引节点是 n 的文件。该 test 选项有助于查找某个特定索引节点上的所有硬件连接 |
| -mmin n | 匹配 n 分钟前内容被修改的文件或目录 |
| -mtime n | 匹配 n*24 小时前只有内容被更改的文件或目录 |
| -name pattern | 匹配有特定通配符模式的文件或目录 |
| -newer file | 匹配内容的修改时间比 file 文件更近的文件或目录。这在编写 shell 脚本进行文件备份的时候非常有用。每次创建备份时，更新某个文件（比如日志），然后用 find+此参数选项来确定上一次更新后哪个文件改变了 |
| -nouser | 匹配不属于有效用户的文件或目录。该 test 可以用来查找那些属于已删除账户的文件，也可以用来检测攻击者的活动 |
| -nogroup | 匹配不属于有效组的文件或目录 |
| -perm mode | 寻找访问权限与既定模式匹配的文件或目录。既定模式可以以八进制或符号的形式表示 |
| -samefile name | 与-inum test 选项类似。匹配与 file 文件用相同的 inode 号的文件 |
| -size n | 匹配 n 大小的文件 |
| -type c | 匹配 c 类型的文件 |
| -user name | 匹配属于 name 用户的文件和目录。name 可以描述为用户名也可以描述为该组的 ID 号 |

以上并不是 test 参数的完整列表，要了解其他相关内容可以查看 find 的 man 手册页。

操作符

即使拥有了 find 命令提供的所有 test 参数，我们仍然会需要一个更好的工具来描述 test 参数之间的逻辑关系。例如，如果我们需要确定某目录下是否所有的文件和子目录都有安全的访问权限，该怎么办？原则上就是去查找那些访问权限不是 0600 的文件和访问权限不是 0700 的子目录。幸运的是，find 命令的 test 选项可以结合逻辑操作从而建立具有复杂逻辑关系的匹配条件。我们可以用下面的命令行来满足上述 find 命令的匹配搜索。

```
[me@linuxbox ~]$ find ~ \(\ -type f -not -perm 0600 \) -or \(\ -type d -not -perm 0700 \)
```

这些都是什么？看起来太别扭了！不过，一旦你开始了解逻辑操作符，就会发现它们并没有那么复杂了（见表 17-4）。

表 17-4 find 命令的逻辑操作符

| 操作符 | 功能描述 |
|------------|--|
| -and（与操作） | 查找使该操作符两边的检验条件都是真的匹配文件。有时直接缩写成-a。注意如果两个检测条件之间没有显式的显示操作符，and 就是默认的逻辑关系 |
| -or（或操作） | 查找使该操作符任何一边的检测条件为真的匹配文件。有时直接缩写成-o |
| -not（-非操作） | 查找使该操作符后面的检测条件为假的匹配文件。有时直接缩写成-! |
| 0（括号操作） | 多个检测条件和逻辑操作符一起组成更长的表达式，而()操作就是用来区分逻辑表达式优先权的。默认的情况下，find 命令从左向右运算逻辑值。然而有时为了获得想要的结果必须扰乱默认的执行顺序，即便不需要，将一串字符串表达式括起来对提高命令的可读性也很有帮助。请注意，括号字符在 shell 环境中有特殊意义，所以必须将它们在命令行中用引号引起来，这样才能作为 find 的参数传递。通常用反斜杠来避免这样的问题 |

对照这张逻辑操作符的列表，重新剖析上述的 find 命令行。从最全局的角度看，所有检测条件由一个 or (or) 操作符分成了两组。

(表达式1) -or (表达式2)

这样做是有原因的，因为我们查找的是具有某种权限设置的文件和另外一种权限设置的目录。那既然要同时查找文件和目录，怎么不是用 and 而是用 or？因为 find 命令在浏览扫描所有的文件和目录时，会判断每一个文件或目录是否匹配该 test 项检测条件。而我们的目标是具有不安全访问权限的文件或是具有不安全访问权限的目录，都知道匹配者不可能既是文件又是目录，所以不能用

and 逻辑关系。由此，可将表达式扩充如下。

```
(file with bad perms) -or (directory with bad perms)
```

接下来的问题就是如何判断文件或是目录具有“危险”权限，我们该怎么做呢？事实上，我们并不需要直接去寻找“危险”权限的文件或是目录，而是查找那些具有“‘不’好”权限的文件或目录，因为我们知道什么是“好的权限”。对于文件来说，权限是 0600 表示“好”（安全）的，而对于目录，“好”的权限应该是 0700。于是，判断具有“不好”访问权限的文件的表达式如下。

```
-type f -and -not -perms 0600
```

同样，判断具有“不好”访问权限的目录的表达式如下。

```
-type d -and -not -perms 0700
```

正如表 17-4 中提到的，and 操作是默认的，所以可以安全地移除。把如上表达式都整理到一起以下。

```
find ~ (-type f -not -perms 0600) -or (-type d -not -perms 0700)
```

然而，由于括号在 shell 环境下有特殊含义，所以我们必须对它们进行转义以防 shell 试图编译它们。在每个括号前加上反斜杠便可解决此问题。

逻辑运算符的另外一个特性也很值得大家了解，有如下两个被逻辑操作符分开的表达式。

```
expr1 -operator expr2
```

在任何情况下，表达式 expr1 都会被执行，而中间的操作符将决定表达式 expr2 是否被执行。表 17-5 列出了具体执行情况。

表 17-5 find 命令的 and/or 逻辑运算

| 表达式 expr1 的结果 | 逻辑操作 | 表达式 expr2 执行情况... |
|---------------|------|-------------------|
| 真 | and | 总是执行 |
| 假 | and | 不执行 |
| 真 | or | 不执行 |
| 假 | or | 总是执行 |

为什么会出现这样的情况？主要还是为了提高效率，以-and 逻辑运算为例，很明显表达式 expr1 -and expr2 的值在 expr1 为假的情况下不可能为真，所以就没有必要再去运算表达式 expr2 了。同样，对于表达式 expr1 -or expr2，在表达式 expr1 为真的情况下其逻辑值显然为真，于是就没有必要再运算表达式 expr2 了。

由此达到效率提高的目的。但为什么这一性质这么重要？这是因为这将直接影响下一步 actions 选项的动作行为，下面就来介绍 actions 选项。

17.2.2 action 选项

行动起来吧！前面 find 命令已经查找到所需要的文件，但是我们真正想做的是处理这些已查找到的文件。幸运的是，find 命令允许直接对搜索结果执行动作。

预定义动作

对搜索到的文件进行操作，即可以用诸多现成的预定义动作指令，也可以使用用户自定义的动作。首先来看一些预定义动作，见表 17-6。

表 17-6 预定义的 find 命令操作

| 动作 | 功能描述 |
|---------|---|
| -delete | 删除匹配文件 |
| -ls | 对匹配文件执行 ls 操作，以标准格式输出其文件名以及所要求的其他信息 |
| -print | 将匹配的文件的全路径以标准形式输出。当没有指定任何具体操作时，该操作是默认操作 |
| -quit | 一旦匹配成功便退出 |

与 test 参数选项相比，actions 参数选项数量更多，可以参考 find 的 man 手册页获取更全面信息。

本章开头所举的第一个例子如下。

find ~

此命令行产生了一个包含当前系统主 (~) 目录中所有文件和子目录的列表。该列表之所以会在屏幕上显示出来，是因为在没有指定其他操作的情况下，-print 操作是默认的。因此，上述命令行等效于如下形式的命令行。

find ~ -print

当然也可以使用 find 命令删除满足特定条件的文件。示例如下，此命令行用于删除.BAK（这种文件一般是用来指定备份文件的）后缀的文件。

find ~ -type f -name '*.BAK' -delete

本例中，用户主目录及其子目录下的每个文件都被搜索了一遍匹配文件名以.BAK 结尾的文件。一旦被找到，则直接删除。

注意

毫无疑问，在使用`-delete`操作时你一定要格外小心。最好先用`-print`操作确认搜索结果后再执行`-delete`删除命令。

在我们继续讲下一个知识点前，先来补充介绍一下逻辑运算是如何影响`find`的`action`操作的。仔细揣摩下面的命令。

```
find ~ -type f -name '*.BAK' -print
```

正如我们所知道的，该命令行用来查找所有文件名以.BAK结尾的普通文件（`-type f`）并且以标准形式（`-print`）输出每个匹配文件的相关路径名。然而，该命令行之所以照这样的方式执行是由每个`test`选项和`action`选项之间的逻辑关系决定的。记住，每个`test`选项和`action`选项之间默认的逻辑关系是与（`and`）逻辑。下面的命令行逻辑关系能看得更清楚些。

```
find ~ -type f -and -name '*.BAK' -and -print
```

如上便是完整的表达式，而表 17-7 便列出了逻辑运算符是如何影响`action`操作的。

表 17-7 逻辑运算符的影响

| 检测条件/执行操作 | 在该情况下执行操作 |
|----------------------------|--|
| <code>-print</code> | <code>-type f</code> 和 <code>-name '*.BAK'</code> 条件都匹配时，也就是文件是普通文件并且文件名也是以.BAK结尾时 |
| <code>-name '*.BAK'</code> | <code>-type f</code> 项匹配，也就是说文件是普通文件时 |
| <code>-type f</code> | 总是执行，因为是与逻辑关系中的第一个操作数 |

`test`选项与`action`选项之间的逻辑关系决定了它们的执行情况，所以`test`选项和`action`选项的顺序很重要。例如，如果重新排列这些`test`选项和`action`选项，并将`-print`操作作为逻辑运算的第一个操作数，那么命令行的运行结果将会有很大的不同。

```
find ~ -print -and -type f -and -name '*.BAK'
```

此命令行会把每个文件显示出来（因为`-print`操作运算值总是为真），然后再对文件类型以及特定的文件扩展名进行匹配检查。

用户自定义操作

除了已有的预定义操作命令，同样也可以任意调用用户想要执行的操作命令。传统的方法就是像以下命令行使用`-exec`操作。

```
-exec command {} ;
```

该格式中的 *command* 表示要执行的操作命令名，{}花括号代表的是当前路径，而分号作为必需的分隔符表示命令结束。使用-exec 完成-delete 操作示例如下。

```
-exec rm '{}' ';'
```

同样，由于括号和分号字符在 shell 环境下有特殊含义，所以在输入命令行时，要将它们用引号包起来或者用转义符隔开。

当然，交互式地执行用户自定义操作也不是不可能。通过使用-ok 操作取代原来的-exec 操作，每一次指定命令执行之前系统都会询问用户。

```
find - -type f -name 'foo*' -ok ls -l '{}' ';' 
< ls ... /home/me/bin/foo > ? y
-rwxr-xr-x 1 me   me 224 2011-10-29 18:44 /home/me/bin/foo
< ls ... /home/me/foo.txt > ? y
-rw-r--r-- 1 me   me 0 2012-09-19 12:53 /home/me/foo.txt
```

上例中，查找文件名以 foo 字符串开始的文件，并且每次找到匹配文件后执行 ls -l 命令。-ok 操作会在 ls 命令执行之前询问用户是否执行。

提高效率

当使用-exec 操作时，每次查找到匹配文件后都会调用执行一次指定命令。但有时用户更希望只调用一次命令就完成对所有匹配文件的操作。例如，多数人可能更喜欢这样的命令执行方式。

```
ls -l file1
ls -l file2
```

而不是以下这样的方式。

```
ls -l file1 file2
```

第一种方式只需要执行命令一次而第二种方式则要多次重复执行。实现这样的一次操作有两种方法：一种方式比较传统，使用外部命令 xargs；另一种则是使用 find 本身自带的新特性。首先介绍下第二种方法。

通过将命令行末尾的分号改为加号，便可将 find 命令所搜索到的匹配结果作为指定命令的输入，从而一次完成对所有文件的操作。回到刚才的例子。

```
find - -type f -name 'foo*' -exec ls -l '{}' '+' 
-rwxr-xr-x 1 me   me 224 2011-10-29 18:44 /home/me/bin/foo
-rw-r--r-- 1 me   me 0 2012-09-19 12:53 /home/me/foo.txt
```

每次找到匹配文件后就执行一次 ls 命令。将上述命令行改成下面的命令行。

```
find ~ -type f -name 'foo*' -exec ls -l '{}' +
-rwxr-xr-x 1 me me 224 2011-10-29 18:44 /home/me/bin/foo
-rw-r--r-- 1 me me 0 2012-09-19 12:53 /home/me/foo.txt
```

我们也能得到相同的结果，但是系统整体只执行一次 ls 命令。

同样我们可以使用 xargs 命令获得相同的效果，xargs 处理标准输入信息并将其转变为某指定命令的输入参数列表。结合前面的实例，我们可以这样使用 xargs 命令。

```
find ~ -type f -name 'foo*' -print | xargs ls -l
-rwxr-xr-x 1 me me 224 2011-10-29 18:44 /home/me/bin/foo
-rw-r--r-- 1 me me 0 2012-09-19 12:53 /home/me/foo.txt
```

该命令行中，find 命令的执行结果直接作为 xargs 输入，xargs 反过来将其转换成了 ls 命令的输入参数列表，最后执行 ls 操作。

注意

虽然一个命令行中可允许输入的参数有很多，但这并不表示可以无限输入，也存在命令行过长而使得 shell 编辑器无法承受的情况。如果命令行中包含的输入参数太多而超过了系统支持的最大长度，xargs 只会尽可能对最大数量的参数执行指定操作，并不断重复这一过程直到所有标准输入全部处理完毕。在 xargs 命令后面添加--show-limits 选项，即可知道命令行最大能承受的参数数量。

处理有趣的文件名

类 UNIX 系统允许文件名里面有嵌入的空格（甚至换行符！），这会给像 xargs 这些为其他程序创建参数列表的命令带来一些问题。因为内嵌的空格可能会被当做分隔符，而要执行的操作命令可能会把空格隔开的单词当做不同的输入参数来处理。为了解决这一问题，find 和 xargs 命令允许使用空字符串作为参数之间的分隔符。在 ASCII 码中，空字符串是由数字 0 代表的字符表示（而空格符在 ASCII 码中是由数字 32 代替）。find 命令提供-print0 这一 action 选项来产生以空字符串作为各参数之间分隔符的输出结果，而 xargs 命令则有--null 参数选项支持 xargs 接收以空字符串为参数分隔符的输入。示例如下。

```
find ~ -iname '*.*' -print0 | xargs --null ls -l
```

使用这一特性，可以确保所有的文件包括那些文件名中包含内嵌空格的文件也能得到正确的处理。

17.2.3 返回到 playground 文件夹

现在可以实际应用 find 命令了。首先让我们创建一个包含很多子目录及文件的 playground 文件夹平台。

```
[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}
[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-{A..Z}
```

不得不惊叹于 Linux 命令行的威力！简单的两行命令，就创建了一个包含 100 个子目录的 playground 文件夹，并且每个子目录中又包含 26 个空文件。你可以试试用图形用户界面 GUI 能否这么快得达到效果！

我们用来创造这个奇迹的方法包含一个熟悉的命令 mkdir、一个奇异的 shell 花括号扩展以及一个新命令 touch。mkdir 命令结合-p 选项(-p 选项使 mkdir 命令按指定的路径创建父目录)的同时用花括号扩展，便完成了 100 个目录的创建。

touch 命令一般用于设定或是更新文件的修改时间。然而，当文件名参数是一个不存在的文件时，那么该命令就会创建一个空文件。

playground 文件夹里，总共创建了 100 个叫做 file-A 的文件。现在，我们可以查找它们。

```
[me@linuxbox ~]$ find playground -type f -name 'file-A'
```

请注意，与 ls 命令不同，find 命令不会产生有排列顺序的结果，其输出顺序是由在存储设备中的布局决定的。下面的命令行验证了该文件夹确实有 100 个 file-A 文件。

```
[me@linuxbox ~]$ find playground -type f -name 'file-A' | wc -l
100
```

下面来看一个根据文件的修改时间查找文件的例子，这在创建备份文件以及按时间顺序排列文件时非常有用。首先需要创建一个用作比较修改时间的参照文件。

```
[me@linuxbox ~]$ touch playground/timestamp
```

该命令行创建了一个名为 timestamp 的空文件，并将当前时刻设为该文件的修改时间。我们可以使用另外一个便捷的命令 stat 来检验执行效果，stat 命令可以说是 ls 的增强版，该命令会将系统所掌握文件的所有信息及属性全部显示出来。

```
[me@linuxbox ~]$ stat playground/timestamp
  File: `playground/timestamp'
  Size: 0          Blocks: 0      IO Block: 4096 regular empty file
Device: 803h/2051d Inode: 14265061 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1001/ me)  Gid: ( 1001/ me)
Access: 2012-10-08 15:15:39.000000000 -0400
Modify: 2012-10-08 15:15:39.000000000 -0400
Change: 2012-10-08 15:15:39.000000000 -0400
```

当我们再一次对此文件执行 touch 命令并用 stat 命令检验时，会发现文件的时间得到了更新。

```
[me@linuxbox ~]$ touch playground/timestamp
[me@linuxbox ~]$ stat playground/timestamp
  File: `playground/timestamp'
  Size: 0          Blocks: 0      IO Block: 4096 regular empty file
Device: 803h/2051d Inode: 14265061 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1001/ me)  Gid: ( 1001/ me)
Access: 2012-10-08 15:23:33.000000000 -0400
Modify: 2012-10-08 15:23:33.000000000 -0400
Change: 2012-10-08 15:23:33.000000000 -0400
```

接下来，我们便可以用 find 命令更新 playground 文件夹里的一些文件。

```
[me@linuxbox ~]$ find playground -type f -name 'file-B' -exec touch '{}' '+';
```

该命令行更新了 playground 文件夹里面叫做 file-B 的所有文件。下面我们通过比较参照文件 timestamp 与其他文件的修改时间，使用 find 命令查找刚刚被更新的文件。

```
[me@linuxbox ~]$ find playground -type f -newer playground/timestamp
```

命令行的运行结果包含 100 个文件名为 file-B 的文件。由于我们是在对 timestamp 文件执行了 touch 命令之后，才对 playground 文件夹中对名为 file-B 的所有文件执行了 touch 操作，所以它们现在要比 timestamp 文件新，从而我们可选用-newer test 选项来查找。

最后，回顾之前讨论的查找不安全访问权限文件的例子，并将其用于 playground 目录。

```
[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 \| ) -or \( -type d
-not -perm 0700 \| )
```

该命令行列出了 playground 目录下的所有的 100 个子目录以及 2600 个文件（再加上 timestamp 文件和 playground 自身，总共 2702 个），之所以全部列出是因为 playground 里面没有一个文件满足“安全访问权限”的要求。运用前面所学 find 命令的 operator 选项和 action 选项的知识，我们可以在命令后面增加 action

参数选项来改变 playground 目录下文件或目录的访问权限。

```
[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 -exec chmod 0600
'{}' ';' \) -or \( -type d -not -perm 0700 -exec chmod 0700 '{}' ';' \)
```

依据日常经验，大家可能会觉得用两条命令——分别针对目录和文件，比用这样一个长而复杂的命令容易得多。不过知道这一知识点总比不知道好，此处的重点是，要了解如何结合使用操作符选项与行为选项，来执行一些有用的任务。

17.2.4 option 选项

最后，我们谈一下 find 命令的 option 选项。option 选项用于控制 find 命令的搜索范围。在构成 find 命令的表达式时，它们可能包含在其他测试选项或行为选项之中。表 17-8 列出了最常用的 option 选项。

表 17-8 find 命令的 option 选项

| 选项 | 描述 |
|------------------|---|
| -depth | 引导 find 程序处理目录前先处理目录内文件。当指定-delete 操作时，该参数选项会自动调用 |
| -maxdepth levels | 当执行测试条件行为时，设置 find 程序陷入目录数的最大级别数 |
| -mindepth levels | 在应用测试条件和行为之前，设置 find 程序陷入目录数的最小级别数 |
| -mount | 引导 find 不去遍历挂载在其他文件系统上的目录 |
| -noleaf | 指导 find 程序不要基于“正在搜索类 UNIX 文件系统”的假设来优化它的搜索。当扫描 DOS/Windows 文件系统和 CD 时，会用到该选项 |

第 18 章

归档和备份

维护系统数据安全是计算机系统管理者的基本任务之一，及时创建系统文件的备份文件是维护系统数据安全的一种常用方法。即便对于非系统管理员，经常创建备份文件或是在设备之间、文件夹之间移动大文件集通常都是非常有益的。

本章会介绍一些用于管理文件集合的常用命令。

文件压缩程序：

- **gzip**: 压缩和解压缩文件工具。
- **bzip2**: 块排序文件压缩工具。

文件归档程序：

- **tar**: 磁带归档工具。
- **zip**: 打包和压缩文件。

文件同步程序：

- **rsync**: 远程文件和目录的同步。

18.1 文件压缩

在计算领域的发展历史中，人们一直在努力实现以最小的可利用空间存储最多的数据，其中可利用空间包括内存、存储设备或者网络带宽。许多如今认为理所当然的数据服务，比如便携式音乐播放器、高清电视和宽带互联网等之所以能够存在，都应归功于有效数据压缩技术。

数据压缩是一个移除数据冗余信息的过程。如下例，假设有一张 100×100 像素的全黑图像文件，就数据存储而言（假设每个像素占用 24 位，也就是 3 个字节），该图像也会占用 30,000 ($100 \times 100 \times 3 = 30,000$) 字节的存储量。

一幅只有一种颜色的图像包含了完全冗余的数据，我们要是聪明的话，编码该图像数据时可以直接简单地描述成有 30,000 个字节的黑色像素。因此，无需存储一个包含 30,000 字节的 0（图像文件里面，黑色通常用零表示）的数据块，而是将这些数据压缩成数字 30,000 和一个 0 来表示。这种数据压缩技术，称为游程编码（run-length encoding），它是最基本的一种压缩技术。现今的压缩技术则更先进、更复杂，但基本目标一直是消除冗余数据信息。

压缩算法（压缩采用的数学方法）一般分为两大类：无损压缩与有损压缩。无损压缩保留原文件中的所有数据，也就是说这种方式的压缩文件还原时，还原后的文件与原文件完全一致。而有损压缩，在压缩时为了实现更大程度的压缩而删除了某些数据信息，有损压缩文件还原时，与原文件并不是完全吻合，但是与原文件差别并不大。JPEG（图像压缩技术）和 MP3（音频压缩技术）技术是典型的有损压缩实例。下面的讨论中，仅仅涉及无损压缩，因为计算机上的大多数数据无法容忍任何数据损失。

18.1.1 gzip——文件压缩与解压缩

`gzip` 命令用于压缩一个或更多文件。执行命令后，原文件会被其压缩文件取代。与之相反，`gunzip` 命令则将压缩文件还原为原文件。示例如下。

```
[me@linuxbox ~]$ ls -l /etc > foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me   me  15738 2012-10-14 07:15 foo.txt
[me@linuxbox ~]$ gzip foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me   me   3230 2012-10-14 07:15 foo.txt.gz
[me@linuxbox ~]$ gunzip foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me   me  15738 2012-10-14 07:15 foo.txt
```

面本例中，我们首先创建了一个名为 `foo.txt` 的文本文件，其内容为某目录所含文件的列表清单，然后运行 `gzip` 命令。于是，压缩后的文件 `foo.txt.gz` 便取代了原文件。`foo.*` 表达式的文件，我们可以看到原文件已被其压缩文件取代，并且压缩后的文件大小差不多才是原文件的 1/5。此外，我们还可以看出，压缩后的文件与原文件有着相同的权限和时间戳。

接着，我们运用 `gunzip` 命令进行解压缩，如此该压缩文件又被原始文件取代，而且权限和时间戳仍然保持一致。

`gzip` 有许多选项，表 18-1 列出了一些。

表 18-1 `gzip` 的选项

| 选项 | 功能描述 |
|---------|---|
| -c | 将输出内容写到标准输出端口并且保持原有文件。也可以用 <code>--stdout</code> 或是 <code>--to-stdout</code> 替代 |
| -d | 解压缩。加上此选项， <code>gzip</code> 命令便类似于 <code>gunzip</code> 。也可以用 <code>--decompress</code> 或 <code>--uncompress</code> 替代 |
| -f | 强制压缩，即便原文件的压缩版本已经存在了。也可以用 <code>--force</code> 替代 |
| -h | 显示有用信息。也可以用 <code>--help</code> 替代 |
| -l | 列出所有压缩文件的压缩统计。也可以用 <code>--list</code> 替代 |
| -r | 如果该命令行的操作参数中有一个或是多个是目录，那么递归压缩包含在目录中的文件。也可以用 <code>--recursive</code> 替代 |
| -t | 检验压缩文件的完整性。也可以用 <code>--test</code> 替代 |
| -v | 在压缩时显示详细信息。也可以用 <code>--verbose</code> 替代 |
| -number | 设定压缩级别。number 是 1（速度最快，压缩比最小）~9（速度最慢，压缩比最大）范围中的一个整数。当然 1~9 的数值亦可以分别描述为 <code>--fast</code> 和 <code>--best</code> 。 <code>gzip</code> 默认的压缩级别是 6 |

回顾前面的例子

```
[me@linuxbox ~]$ gzip foo.txt
[me@linuxbox ~]$ gzip -tv foo.txt.gz
foo.txt.gz:      OK
[me@linuxbox ~]$ gzip -d foo.txt.gz
```

此例中，首先，我们将压缩文件 `foo.txt.gz` 取代了原文件 `foo.txt`。接着，我们运用 `-t`、`-v` 选项检查压缩文件的完整性。最后，解压缩该文件为原来的形式。

借助标准输入输出，`gzip` 有很多有趣的用法。

```
[me@linuxbox ~]$ ls -l /etc | gzip > foo.txt.gz
```

此命令创建了一个目录列表的压缩版本。

`gunzip` 命令用于解压 `gzip` 的压缩文件，并且默认解压缩后缀为“.gz”的文件，所以，我们没有必要明确指定，只要指定名与已存在的非压缩文件名不冲突就可以了。

```
[me@linuxbox ~]$ gunzip foo.txt
```

如果只是希望查看某个压缩文本文件的内容，可以直接输入下面的命令行：

```
[me@linuxbox ~]$ gunzip -c foo.txt | less
```

或者，利用 `zcat` 命令联合 `gzip` 一起，其效果等同于带有-c 选项的 `gunzip`。`zcat` 的功能与 `cat` 命令相同，只是它的操作对象是压缩文件。用 `zcat` 命令处理 `gzip` 压缩文件的示例如下。

```
[me@linuxbox ~]$ zcat foo.txt.gz | less
```

注意

同样也有 `zless` 命令，它与前面所讲的 `less` 的管道功能相同。

18.1.2 bzip2——牺牲速度以换取高质量的数据压缩

`bzip2` 程序由 Julian Seward 开发，与 `gzip` 命令功能相仿，但使用不同的压缩算法。该算法具有高质量的数据压缩能力，但却降低了压缩速度。多数情况下，其用法与 `gzip` 类似，只是用 `bzip2` 压缩后的文件以.bz2 为后缀。

```
[me@linuxbox ~]$ ls -l /etc > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-r--r-- 1 me me 15738 2012-10-17 13:51 foo.txt
[me@linuxbox ~]$ bzip2 foo.txt
[me@linuxbox ~]$ ls -l foo.txt.bz2
-rw-r--r-- 1 me me 2792 2012-10-17 13:51 foo.txt.bz2
[me@linuxbox ~]$ bunzip2 foo.txt.bz2
```

由此例可以看出，`bzip2` 用法与 `gzip` 类似，前面所讨论的 `gzip` 的所有选项（除了-r 选项），`bzip2` 都支持。然而，要注意的是，两者的压缩级别选项(-number)有些许不同。与此同时，解压缩 `bzip2` 压缩文件的专用工具是 `bunzip2` 和 `bzcat` 命令。

`bzip2` 还配有专门的 `bzip2recover` 命令，该命令用于恢复损坏的.bz2 文件。

不要强制压缩

有时，笔者看到有人试图压缩已经用有效压缩算法压缩过的文件，他们通常会这么做。

```
$ gzip picture.jpg
```

不可以这么做，这只是在浪费时间和空间而已！如果对一个已压缩文件再次进行压缩，实际上只会导致文件变大。因为所有的压缩技术会在压缩文件前首先增加一些开销，以描述本次压缩。如果试图压缩一个已无冗余信息的文件，那么此次压缩不会腾出任何空间来抵消增加的开销。

18.2 文件归档

归档是与压缩操作配合使用的一个常用文件管理任务。归档是一个聚集众多文件并将它们组合为一个大文件的过程，它通常作为系统备份的一部分，而且通常也用于将旧数据从某个系统移到某些长期存储设备的情况下。

18.2.1 tar——磁带归档工具

`tar` 命令是类 UNIX 系统中用于归档文件的经典工具。`tar` 是 `tape archive` 的缩写，由此可见，该命令最初的作用就是磁带备份。虽然该命令仍可用于传统的磁带备份，但同样也可用于其他存储设备。大家肯定经常看到文件名以`.tar` 和`.tgz` 结尾的文件，它们分别是用普通的`tar` 命令归档的文件和用`gzip` 归档的文件。`tar` 归档文件可以由许多独立的文件、一个或多个目录层次或者两者的混合组合而成，其用法如下。

```
tar mode[options] pathname...
```

其中的 `mode` 是指表 18-2 中（此列表只列出了部分模式，想获得全部信息可以查看`tar` 的 man 手册页）列出的操作模式的一种。

表 18-2 `tar` 命令的操作模式

| 模式 | 描述 |
|----|------------------|
| c | 创建文件和/或目录列表的归档文件 |
| x | 从归档文件中提取文件 |
| t | 在归档文件末尾追加指定路径名 |
| r | 列出归档文件的内容 |

tar 命令在选项的表达方式上有点奇怪，所以，我们需要举一些例子以说明其是如何工作的。首先，重新创建一个上一章中所建的 playground 文件夹。

```
[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}
[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-{A..Z}
```

下面，用 tar 命令为整个 playground 文件夹创建一个归档文件。

```
[me@linuxbox ~]$ tar cf playground.tar playground
```

该命令行创建了一个叫做 playground.tar 的 tar 归档文件，该归档文件包含了 playground 文件夹的整个目录结构。从命令行中我们可以看到，tar 命令的操作模式 c 参数和用于指定归档文件名的 f 参数可以直接连着写在一起而中间不需要连字符隔开。然而，请注意，mode 参数必须在任何选项之前指定。

下面的命令行用于列出归档文件的内容，可以用于查看已经备份了哪些文件。

```
[me@linuxbox ~]$ tar tf playground.tar
```

如若想获取更详细的信息，可以增加-v（详细信息）选项。

```
[me@linuxbox ~]$ tar tvf playground.tar
```

现在，将 playground 文件夹中的内容提取到一个新的位置。首先，创建一个名为 foo 的新文件夹，然后切换工作目录，再提取该归档文件。

```
[me@linuxbox ~]$ mkdir foo
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ tar xf ../playground.tar
[me@linuxbox foo]$ ls
playground
```

查看~/foo/playground 目录下的内容，便会发现该归档文件已经成功提取，并且是原文件的精确复制。但是存在一个问题，除非是以超级用户的名义执行该命令，不然，从归档文件中提取出来的文件和目录的所有权属于执行归档操作的用户而不是文件的原始作者。

tar 命令处理档案文件路径名的方式也很有趣，其默认的路径名是相对路径而不是绝对路径，tar 命令创建归档文件时会简单地通过移除路径名前面的斜杠来实现相对路径。作为演示，下面会重新创建一个归档文件，此次明确指定一个绝对路径。

```
[me@linuxbox foo]$ cd
[me@linuxbox ~]$ tar cf playground2.tar ~/playground
```

记住，当按下 Enter 键时，上面命令行中输入的目录~/playground 会扩展为 /home/me/playground，也就是绝对路径。接下来，我们按照前面的步骤从归档文件中提取文件，注意观察所发生的变化。

```
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ tar xf ../playground2.tar
[me@linuxbox foo]$ ls
home playground
[me@linuxbox foo]$ ls home
me
[me@linuxbox foo]$ ls home/me
playground
```

此刻，我们便会发现当解压第二个归档文件时，在当前工作目录~/foo 下重新创建了一个 home/me/playground 目录，而不是在认定的绝对路径根目录下创建的。这样的工作方式看起来很奇怪，但是却更有用，因为如此可以将归档文件解压缩到任何目录下而不用被迫解压到原目录下，使用-v 选项重复操作此实例可以详细地了解其运行情况。

下面让我们看一个假想的但很实用的 tar 命令应用实例。假设我们需要将一个系统上的主目录及其内容复制到另外一个系统上，并且具备用于实现这一转移过程的 USB 大硬盘。在现代 Linux 系统中，此硬盘会自动挂载到/media 目录下。再假设 USB 硬盘连接系统后，设备名为 BigDisk。接着用 tar 进行文件归档，示例如下。

```
[me@linuxbox ~]$ sudo tar cf /media/BigDisk/home.tar /home
```

tar 归档的文件写入硬盘后，我们将硬盘卸载，再将其与另外一台计算机连接。同样，此硬盘挂载在了 /media/BigDisk 目录下。那么如何解压缩该归档文件，示例如下。

```
[me@linuxbox2 ~]$ cd /
[me@linuxbox2 /]$ sudo tar xf /media/BigDisk/home.tar
```

这里需要重点注意的是，我们首先要将工作目录改为根目录，以便文件解压缩在根目录下，因为归档文件中的所有文件采用的都是相对路径。

当从归档文件中提取文件时，可以限制只提取某些文件。例如，如果希望从归档文件中只提取单个文件，可以用如下命令行。

```
tar xf archive.tar pathname
```

在命令后面添加要提取的文件的路径名，可以确保 tar 只恢复指定文件，而且可以指定多个路径名。注意，指定的路径名必须是存储在归档文件中的完整、

准确的相对路径。在指定路径名时，通常不支持通配符。但是，GNU 版本的 tar 命令（在 Linux 发行版本中该版本的 tar 最常见）通过使用--wildcards 选项而支持通配符。下面就是一个利用前面的 playground.tar 归档文件实践通配符的例子。

```
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ tar xf ../playground2.tar --wildcards 'home/me/playground/
dir-*/file-A'
```

此命令行只会提取那些路径名与通配符 dir-* 匹配的文件。

tar 命令创建归档文件时通常辅助以 find 命令。首先使用 find 命令查找到需要被归档的文件，然后使用 tar 对这些文件进行归档，实例如下。

```
[me@linuxbox ~]$ find playground -name 'file-A' -exec tar rf playground.tar '{'
}' '+'
```

上例，使用 find 命令匹配 playground 文件夹中叫做 file-A 的文件，然后借助-exec 操作选项，启动 tar 的附加模式 r 将匹配文件添加到归档文件 playground.tar 中。

tar 命令结合 find 命令很适合创建目录树以及整个系统的增量备份，使用 find 命令找到那些在时间戳文件之后创建的文件，便可以创建一份只包含上一次归档之后创建的文件的归档文件，当然假定该时间戳文件是在每一个归档文件创建之后就立刻更新。

tar 命令还可以利用标准输入输出。下面就是一个综合例子。

```
[me@linuxbox foo]$ cd
[me@linuxbox ~]$ find playground -name 'file-A' | tar cf - --files-from=- | gzip
> playground.tgz
```

本例中，先用 find 命令搜索得到匹配文件列表，然后将匹配文件再送至 tar 命令处理。如果文件名前面明确指定有连字符“-”，那就意味着这是标准输入输出的文件（顺便讲一下，使用“-”代表标准“输入/输出”的惯例，其他许多程序也都采用）。--files-from 选项（也可以简写成-T）则指定了 tar 命令从文件中而不是从命令行中读取文件路径名列表。最后，tar 命令归档后的文件再送至 gzip 进行压缩，由此得到压缩归档文件 playground.tgz。后缀.tgz 已经惯例性成为经 gzip 压缩的 tar 归档文件名的后缀，当然，我们有时也用.tar.gz 作后缀。

虽然可以从外部使用 gzip 命令创建压缩归档文件，但现代 GNU 版本的 tar 命令则提供 gzip +z 选项和 bzip2+j 选项直接实现这一功能。以前面的例子为例，可以将命令行简化为以下命令行。

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar czf playground.tgz -T -
```

当然，如果我们想要创建一个 bzip2 压缩的归档文件，可以这么做。

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar cjf playground.tbz -T -
```

通过简单地将压缩选项从 z 变为 j（并将输出文件后缀改为.tbz 以显示是 bzip2 压缩的文件），即可实现 bzip2 式的压缩归档文件。

利用 tar 命令在系统之间传输网络文件，是 tar 另外一个利用标准输入输出的有趣用法。假设，有两台类 UNIX 系统的计算机正在运行，并且都安装了 tar 命令和 ssh 命令，于是，我们便可以将远程系统（本例中远程系统主机名叫做 remote-sys）中的某目录转移到本地系统。

```
[me@linuxbox ~]$ mkdir remote-stuff
[me@linuxbox ~]$ cd remote-stuff
[me@linuxbox remote-stuff]$ ssh remote-sys 'tar cf - Documents' | tar xf -
me@remote-sys's password:
[me@linuxbox remote-stuff]$ ls
Documents
```

上例中，名为 Documents 的目录从 remote-sys 的远程系统复制到本地系统上的 remote-stuff 的文件目录里。这是如何实现的呢？首先，用 ssh 程序在远程系统上启动 tar 命令，此时可能会联想到前面所讲的 ssh 具有在联网机器上运行远程程序并将结果显示在本地系统的能力，也就是远程系统的标准输出送至本地系统显示。于是，可以利用这一特性，我们可以将 tar 命令创建的归档文件（用 c 模式创建的）送至标准输出而不是直接输出文件（-f 选项），然后通过 ssh 建立的加密隧道将该归档文件送至本地系统。在本地系统上，我们再执行 tar 命令提取（x 模式）标准输入的归档文件（同样用 f 选项加上连字符作为参数）。

18.2.2 zip——打包压缩文件

zip 程序既是文件压缩工具也是文件归档工具。Windows 用户肯定很熟悉这种文件格式，因为其读写的是.zip 后缀的文件。然而，Linux 系统中，gzip 才是主要的压缩指令，而 bzip2 仅次之。Linux 用户主要使用 zip 程序与 Windows 系统交换文件，而不是将其用于压缩或是归档文件。

zip 最基本的调用方式如下。

```
zip options zipfile file...
```

例如，创建一个 playground 的 zip 归档文件，可以输入下面的命令行。

```
[me@linuxbox ~]$ zip -r playground.zip playground
```

此例中，如果不加-r 选项递归的话，只会保留 playground 这个目录而不包括目录中内容。虽然程序会自动默认添加后缀.zip，但为了以示清晰，最好还是在命令行中添加文件后缀。

zip 归档文件创建的过程中，zip 通常会显示如下的一系列信息。

```
adding: playground/dir-020/file-Z (stored 0%)
adding: playground/dir-020/file-Y (stored 0%)
adding: playground/dir-020/file-X (stored 0%)
adding: playground/dir-087/ (stored 0%)
adding: playground/dir-087/file-S (stored 0%)
```

这些信息显示的是每个新添归档文件的状态。zip 使用两种存储方式向归档文件中添加文件。第一，不对文件进行压缩直接存储，如本例；第二，缩小文件大小，即对文件进行压缩后存储。紧随存储方法之后显示的数值表示的是实现的压缩比。由于使用的 playground 文件夹是空文件夹，所以并没有对其内容进行压缩。

利用 unzip，我们可以直接提取 zip 文件中的内容。

```
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ unzip ../playground.zip
```

关于 zip，有一点需要注意（与 tar 命令相比），即如果指定的归档文件已经存在，那么 zip 仅仅只会更新而不会取而代之。这意味着原来存在的归档文件会保留下，只是增加了一些新文件，原有匹配文件则被替换。

通过给 unzip 指定提取的文件名，我们可以选择性地从 zip 归档文件中提取文件。

```
[me@linuxbox ~]$ unzip -l playground.zip playground/dir-087/file-Z
Archive: ./playground.zip
      Length      Date    Time     Name
      -----      ----   ----
          0  10-05-12 09:25  playground/dir-087/file-Z
      -----
          0                  1 file
[me@linuxbox ~]$ cd foo
[me@linuxbox foo]$ unzip ../playground.zip playground/dir-087/file-Z
Archive: ../playground.zip
replace playground/dir-087/file-Z? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
extracting: playground/dir-087/file-Z
```

使用-l 选项，unzip 只会列出归档文件的内容而不会从中提取文件。如果没有指定任何文件，unzip 将会提取归档文件中的所有文件，我们可以增加-v 选项得到更详细的列表。注意当提取的文件与已存在文件冲突时，原文件被取代之

前会提示用户是否执行此替换操作。

与 tar 命令类似，zip 命令也可以利用标准输入输出，尽管此用法在某种程度上来说作用并不大。我们也可以用-@选项将多个文件送至 zip 进行压缩。

```
[me@linuxbox foo]$ cd
[me@linuxbox ~]$ find playground -name "file-A" | zip -@ file-A.zip
```

本例中，我们利用 find 命令产生一个匹配-name 项的“file-A”文件列表，然后将结果直接作为 zip 命令的输入，从而得到了一个包含选定文件的归档文件 file-A.zip。

zip 同样可以将结果送至标准输出，但是由于只有极少的命令能够利用其输出结果，所以这种用法具有局限性。不幸的是，unzip 程序不支持标准输入，所以 zip 和 unzip 不能像 tar 命令一样一起用于处理网络文件。

然而，zip 命令支持标准输入，所以可以用于压缩其他程序的输出结果。

```
[me@linuxbox ~]$ ls -l /etc/ | zip ls-etc.zip -
adding: - (deflated 80%)
```

本例中，我们将 ls 的输出结果列表直接送给 zip。与 tar 命令一样，zip 会默认末尾的连字符代表输入的文件是标准输入。

当指定-p 选项后，unzip 命令便将其输出结果以标准形式输出。

```
[me@linuxbox ~]$ unzip -p ls-etc.zip | less
```

到目前为止，本章只涉及了 zip 和 unzip 命令的一些基本用法，它们其实还有很多选项，尽管有些只适用于其他系统的特定平台，但是它们使用起来很灵活。zip 和 unzip 的 man 手册页都很全面，且包含了很多有用的例子。

18.3 同步文件和目录

将一个或多个目录与本地系统（通常是某种可移动存储设备）或是远程系统上其他的目录保持同步，是维护系统备份文件的常用方法。例如，本地系统上有一个正在开发的网站备份，用户通常会在远程 Web 服务器上进行“实时”备份以实现同步更新。

18.3.1 rsync——远程文件、目录的同步

针对类 UNIX 系统，完成这一同步任务最合适的工具当属 rsync。该命令通

过运用 rsync 远程更新协议，同步本地系统与远程系统上的目录，该协议允许 rsync 命令快速检测到本地和远程系统上两个目录之间的不同，从而以最少数量的复制动作以完成两个目录之间的同步。因此，rsync 命令与其他复制命令相比，显得既快又经济。

rsync 命令调用方式如下。

```
rsync options source destination
```

这里的 source 和 destination 是下列选项之一：

- 一个本地文件或目录；
- 一个远程文件或目录，形式为 [user@]host:path；
- 一个远程 rsync 服务器，由 rsync://[user@]host[:port]/path 指定。

请注意，source 和 destination 中必须有一个本地文件，因为 rsync 不支持远程系统与远程系统之间的复制。

我们在本地文件上实践 rsync 命令，首先应清空 foo 目录。

```
[me@linuxbox ~]$ rm -rf foo/*
```

接着，同步 playground 目录和它在 foo 目录中的相应副本：

```
[me@linuxbox ~]$ rsync -av playground foo
```

此命令行中，我们运用了-a（用于归档——进行递归归档并保留文件属性）选项和-v（详细输出）选项在 foo 目录中生成了 playground 目录的镜像备份。此命令行运行过程中，我们会看到一系列文件和目录被复制。最后，显示如下的汇总信息，表示文件复制的总量。

```
sent 135759 bytes received 57870 bytes 387258.00 bytes/sec
total size is 3230 speedup is 0.02
```

再次运行该命令行，会得到不同的结果。

```
[me@linuxbox ~]$ rsync -av playground foo
building file list ... done

sent 22635 bytes received 20 bytes 45310.00 bytes/sec
total size is 3230 speedup is 0.14
```

请注意，此时并不会列出文件列表。因为 rsync 目录检测出~/playground 和 ~/foo/playground 两个文件夹之间并没有区别，因此不需要进行任何复制操作。如果 playground 目录中的某个文件被修改了，那么 rsync 会检测到该变化并且只

复制这个刚刚更新的文件。

```
[me@linuxbox ~]$ touch playground/dir-099/file-Z
[me@linuxbox ~]$ rsync -av playground foo
building file list ... done
playground/dir-099/file-Z
sent 22685 bytes received 42 bytes 45454.00 bytes/sec
total size is 3230 speedup is 0.14
```

下面举一个实际的例子，回想前面讲解 tar 命令时所举的虚拟外部硬盘的例子。当该设备与系统连接时，再次挂载在/media/BigDisk 目录下，就可以进行系统备份了。首先，我们在外部硬盘上创建一个/backup 的目录，然后使用 rsync 命令将系统中最重要的内容复制到该外部设备。

```
[me@linuxbox ~]$ mkdir /media/BigDisk/backup
[me@linuxbox ~]$ sudo rsync -av --delete /etc /home /usr/local /media/BigDisk/
backup
```

在本例中，系统中的/etc、/home 和/usr/local 目录成功备份到了虚拟存储设备中，同时添加了-delete 选项以移除那些残留于备份设备中而源设备中已经不存在的文件（这一步骤在第一次备份时无关紧要，但在后续的复制操作中会起作用）。重复插入该外围设备并运行 rsync 命令，对于小型系统来说，这是一个持续备份的好（虽然不是完美的）方法。当然，如果此处使用别名会更方便。于是，我们可以定义一个别名，并将其添加到.bashrc 文件中，以提供该备份功能。

```
alias backup='sudo rsync -av --delete /etc /home /usr/local /media/BigDisk/backup'
```

现在所要做的就是插入外围设备，运行别名 backup 即可完成上述备份操作。

18.3.2 在网络上使用 rsync 命令

通过网络复制文件是 rsync 用法的另一个美妙之处。毕竟，rsync 命令名中的 r 其实指的是 remote（远程）。远程复制可以由以下两种方法中的任一种实现。

方法之一是针对于已安装了 rsync 命令以及诸如 ssh 等远程 shell 程序的系统。假定本地网络有另外一个具有足够可利用硬盘空间的系统，同时希望利用远程系统而非外部设备进行备份操作。假使远程系统已经有一个用于存放备份文件的/backup 目录，那么我们便可以直接运行下面的命令。

```
[me@linuxbox ~]$ sudo rsync -av --delete --rsh=ssh /etc /home /usr/local remote-
sys:/backup
```

此处命令行改动了两个地方。第一，增加了--rsh=ssh 选项，该选项告诉 rsync

使用 ssh 命令作为其远程 shell 命令。只有这样，我们才可以通过 SSH 的加密隧道安全地从本地系统向远程主机传输数据。第二，在 destination 路径名前指定了远程主机名（本例中的远程主机名是 remote-sys）。

方法之二，使用 rsync 服务器同步网络文件，通过配置 rsync 运行一个守护进程监听进来的同步请求。这种方法通常用于远程系统的镜像备份。例如，Red Hat 软件为发行其 Fedora 系统，需要维持一个正在开发的大的软件包库。对于软件测试员来说，在发行版的测试阶段创建这个大集合的备份是很重要的，因为库中的文件会频繁变动（每天会有多次改动），所以通过周期性的同步来维持本地文件镜像要比批量复制软件库更可取。Georgia Tech 就维护了其中一个库，可以使用本地复制工具 rsync 以及 Georgia Tech 的 rsync 服务器来创建该库的镜像备份。

```
[me@linuxbox ~]$ mkdir fedora-devel
[me@linuxbox ~]$ rsync -av -delete rsync://rsync.gtlb.gatech.edu/fedora-
linux-core/development/i386/os fedora-devel
```

本例中，使用 rsync 远程服务的 URI 是由协议（rsync://）、远程主机名（rsync.gtlb.gatech.edu）和库的路径名组成。

第 19 章

正则表达式

在下面几章，我们会讨论一些用于文本操作的工具。我们已经知道，文本数据在类 UNIX 系统中（比如 Linux）扮演着非常重要的角色。但是，在领略这些工具强大的功能前，我们还是先看一下经常与这些工具的复杂用法相关联的技术——正则表达式。

前面我们已经接触过命令行提供的许多特性和工具，并且也遇到过一些相当神秘的 shell 特性及命令，比如 shell 扩展和引用、键盘快捷键和命令历史记录等，更不用提 vi 编辑器了。正则表达式也延续了这种传统，而且可以说是众多特性中最神秘的一个（该说法应该会持有争议）。当然，并不是说这些特性不值得大家花时间去学习。恰恰相反，熟练掌握这些用法会给人意想不到的效果，尽管它们的全部价值可能不会立即体现出来。

19.1 什么是正则表达式

简单地说，正则表达式是一种符号表示法，用于识别文本模式。在某种程

度上，它们类似于匹配文件和路径名时使用的 shell 通配符，但其用途更广泛。许多命令行工具和大多数编程语言都支持正则表达式，以此来解决文本操作方面的问题。然而，在不同的工具，以及不同的编程语言之间，正则表达式都会略有不同，这让事情进一步麻烦起来。方便起见，我们将正则表达式的讨论限定在 POSIX 标准中（它涵盖了大多数命令行工具），与许多编程语言（最著名的 Perl）不同，这些编程语言使用的符号集要更多一些。

19.2 grep——文本搜索

我们用来处理正则表达式的主要程序是 grep。grep 名字源于 “global regular expression print”，由此也可以看到，grep 与正则表达式有关。实际上，grep 搜索文本文件中与指定正则表达式匹配的行，并将结果送至标准输出。

目前为止，我们已经利用 grep 搜索了固定的字符串，如下所示：

```
[me@linuxbox ~]$ ls /usr/bin | grep zip
```

该命令行的作用是列出 /usr/bin 目录下文件名包含 zip 字符串的所有文件。

grep 程序按照如下方式接受选项和参数。

`grep [options] regex [file...]`

其中字符串 *regex* 代表的是某个正则表达式。

表 19-1 列出了 grep 常用的选项。

表 19-1 grep 选项

| 选项 | 功能描述 |
|----|--|
| -i | 忽略大小写。不区分大写和小写字符，也可以用--ignore-case 指定 |
| -v | 不匹配。正常情况下，grep 会输出匹配行，而该选项可使 grep 输出不包含匹配项的所有行。也可以用--invert-match 指定 |
| -c | 输出匹配项数目（如果有-v 选项，那就输出不匹配项的数目）而不是直接输出匹配行自身。也可以用--count 指定 |
| -l | 输出匹配项文件名而不是直接输出匹配行自身。也可以用--files-with-matches 指定 |
| -L | 与-l 选项类似，但输出的是不包含匹配项的文件名。也可以用--files-without-match 指定 |
| -n | 在每个匹配行前面加上该行在文件内的行号。也可以用--line-number 指定 |
| -h | 进行多文件搜索时，抑制文件名输出。也可以用--no-filename 指定 |

为了更为全面地了解 grep，我们创建几个文本文件来进行搜索。

```
[me@linuxbox ~]$ ls /bin > dirlist-bin.txt
[me@linuxbox ~]$ ls /usr/bin > dirlist/usr-bin.txt
[me@linuxbox ~]$ ls /sbin > dirlist-sbin.txt
[me@linuxbox ~]$ ls /usr/sbin > dirlist/usr-sbin.txt
[me@linuxbox ~]$ ls dirlist*.txt
dirlist-bin.txt  dirlist-sbin.txt  dirlist/usr-sbin.txt
dirlist/usr-bin.txt
```

我们可以对文件列表执行简单搜索，如下所示。

```
[me@linuxbox ~]$ grep bzip dirlist*.txt
dirlist-bin.txt:bzip2
dirlist-bin.txt:bzip2recover
```

本例中，`grep` 命令会搜索所有的文件，以查找字符串 `bzip`，并找到了两个匹配项，而且这两个匹配项都在文件 `dirlist-bin.txt` 里。如果我们只对包含匹配项的文件感兴趣而不是对匹配项本身感兴趣，可以指定`-l` 选项。

```
[me@linuxbox ~]$ grep -l bzip dirlist*.txt
dirlist-bin.txt
```

相反，如果那只想查看那些不包含匹配项的文件，则可以用如下命令行。

```
[me@linuxbox ~]$ grep -L bzip dirlist*.txt
dirlist-sbin.txt
dirlist/usr-bin.txt
dirlist/usr-sbin.txt
```

19.3 元字符和文字

虽然看起来不是很明显，但 `grep` 搜索一直都在使用正则表达式，尽管那些例子都很简单。正则表达式 `bzip` 用于匹配文本中至少包含 4 个字符、存在连续的按 `b,z,i,p` 顺序组成的字符串的行。字符串 `bzip` 中的字符都是文字字符(literal character)，即它们只能与自身进行匹配。除了文字字符，正则表达式还可以包含用于指定更为复杂的匹配的元字符。正则表达式的元字符包括以下字符。

`^ $. [] { } - ? * + () | \`

其他所有字符则被当作文字字符，但是在极少数的情况下，反斜杠字符用来创建元序列，以及用来对元字符进行转义，使其成为文字字符，而再被解释为元字符。

注意

可以看到，当 shell 在执行扩展时，许多正则表达式的元字符在 shell 中具有特殊的含义。所以，在命令行中输入包含元字符的正则表达式时，应把这些元字符用引号括起来以避免不必要的 shell 扩展。

19.4 任意字符

接下来讨论的第一个元字符是“点”字符或者句点字符，该字符用于匹配任意字符。如果将其加进某个正则表达式中，它将会在对应位置匹配任意字符。下面就是一个应用实例。

```
[me@linuxbox ~]$ grep -h '.zip' dirlist*.txt
bunzip2
bzip2
bzip2recover
gunzip
gzip
funzip
gpg-zip
preunzip
prezip
prezip-bin
unzip
unzipsfx
```

上述命令行，搜索到了所有匹配正则表达式.zip 的命令行。但其输出结果有一些有趣的地方，比如说输出中并没有包含 zip 程序，这是因为正则表达式中的“.”元字符将匹配长度增加到了 4 个字符。而“zip”只包含了 3 个字符，所以不匹配。同样，如果列表中某个文件包含了文件扩展名“.zip”，那么该文件也会被认为是匹配文件，因为文件扩展名中的“.”符号也被当作“任意字符”处理了。

19.5 锚

插入符 (^) 和美元符号 (\$) 在正则表达式被当做锚，也就是说正则表达式只与行的开头 (^) 或是末尾 (\$) 的内容进行匹配比较。

```
[me@linuxbox ~]$ grep -h '^zip' dirlist*.txt
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
[me@linuxbox ~]$ grep -h 'zip$' dirlist*.txt
gunzip
gzip
funzip
gpg-zip
preunzip
prezip
```

```
unzip
zip
[me@linuxbox ~]$ grep -h '^zip$' dirlist*.txt
zip
```

上例中搜索的是行开头、行末尾都有字符串“zip”（例如 zip 自成一行）的文件。请注意，正则表达式“`^$`”（行开头和行末尾之间没有字符）将会匹配空行。

纵横填字字谜助手

我的妻子很喜欢玩纵横填字字谜这个游戏，并且有时她会拿一个具体问题向我寻求帮助。诸如“有一个 5 个字母的单词，它的第三个字母是 j，最后一个字母是 r，请问这是什么单词？”等等，这类问题不得不让我思考。

你们知道 Linux 系统本身自带一个字典吗？查看 `/usr/share/dict` 目录，就会发现一个或是多个这样的字典。字典中的文件包含的是一个常常的单词列表，每个单词占一行，以字母表的顺序排列。在我自己的系统上，单词文件中包含了超过 98,500 个单词。可以输入如下命令行，得到上述字谜问题的可能答案：

```
[me@linuxbox ~]$ grep -i '^..j.r$' /usr/share/dict/words
Major
major
```

利用这样的正则表达式，便可以找到字典中所有匹配的单词，即满足字符串长度为 5、第三个字母是 j 以及末尾字母是 r 的所有单词。

19.6 中括号表达式和字符类

中括号除了可以用于匹配正则表达式中给定位置的任意字符外，还可以用于匹配指定字符集中的单个字符。借助于中括号，我们可以指定要匹配的字符集（也包括那些可能会被解释为元字符的字符）。如下命令行则利用了一个两个字母组成的字符集，用于匹配包含 bzip 或 gzip 字符串的文本行。

```
[me@linuxbox ~]$ grep -h '[bg]zip' dirlist*.txt
bzip2
bzip2recover
gzip
```

一个字符集可以包含任意数目的字符，并且当元字符放置到中括号中时，会失去它们的特殊含义。然而，在两种情况下，则会在中括号中使用元字符，并且会有不同的含义。第一个就是插入符（`^`），它在中括号内使用表示否定；

另外一个是连字符 (-)，它表示字符范围。

19.6.1 否定

如果中括号内的第一个字符是插入符 (^)，那么剩下的字符则被当作不应该在指定位置出现的字符集。作为演示，我们对前面的例子稍作修改。

```
[me@linuxbox ~]$ grep -h '[^bg]zip' dirlist*.txt
bunzip2
gunzip
funzip
gpg-zip
preunzip
prezip
prezip-bin
unzip
unzipsfx
```

通过使用否定操作，我们可以得到那些包含 zip 字符串但 zip 前面既不是 b 也不是 g 的所有程序。请注意，此时 zip 命令仍然没有出现在结果列表中，由此可见否定，字符集仍然需要在指定位置有对应字符，只不过这个字符不是否定字符集中的成员而已。

插入符号 “^” 只有是中括号表达式中的第一个字符时才会被当作否定符，如果不是第一个，“^” 将会丧失其特殊含义而成为普通字符。

19.6.2 传统字符范围

如果我们希望建立一个正则表达式，用于查找文件名以大写字母开头的文件，可以用下面的命令行。

```
[me@linuxbox ~]$ grep -h '^[ABCDEFGHIJKLMNOPQRSTUVWXYZ]' dirlist*.txt
```

这仅仅是将 26 个大写字母写入中括号的小事，但是要输入 26 个字母实在有点麻烦，我们可以用下面的简单方法完成。

```
[me@linuxbox ~]$ grep -h '^[A-Z]' dirlist*.txt
MAKEDEV
ControlPanel
GET
HEAD
POST
X
X11
Xorg
MAKEFLOPPIES
NetworkManager
NetworkManagerDispatcher
```

通过使用三个字符表示的字符范围，我们可以缩写这 26 个字母。能够按照这种方式表达的任何字符范围可以包含多个范围，比如下面这个表达式可以匹配以字母和数字开头的所有文件名。

```
[me@linuxbox ~]$ grep -h '^[A-Za-z0-9]' dirlist*.txt
```

在字符范围内，可以看到连字符有了特殊的用法，那么如何在中括号中真正包括一个连字符字符？可将连字符作为中括号内的第一个字符，示例如下。

```
[me@linuxbox ~]$ grep -h '[A-Z]' dirlist*.txt
```

此命令行匹配的是文件名中包含一个大写字母的文件。而下面的命令行

```
[me@linuxbox ~]$ grep -h '[-AZ]' dirlist*.txt
```

匹配的则是文件名包含连字符、大写字母 A 或大写字母 Z 的文件。

19.6.3 POSIX 字符类

传统的字符范围表示方法很容易理解，而且能够有效、快速地指定字符集。但不足之处在于，它并不是所有情况都适用。虽然到目前为止，在使用 grep 命令时还没有遇到过任何问题，但是在其他程序中则可能会遇到问题。

在第 4 章，我们讨论了如何使用通配符来执行路径名扩展。在该讨论中我们提到，字符范围的用法几乎与其在正则表达式中一致，现在问题出现了。

```
[me@linuxbox ~]$ ls /usr/sbin/[ABCDEFIGHIJKLMNOPQRSTUVWXYZ]*  
/usr/sbin/MAKEFLOPPIES  
/usr/sbin/NetworkManagerDispatcher  
/usr/sbin/NetworkManager
```

Linux 发行版本不同，上述命令行得到的结果可能会不同，甚至有可能是空列表。本例中的列表来自于 Ubuntu 系统。该命令行得到了预期效果——只有以大写字母开头的文件列表。但是，如果我们使用下面的命令行，便会得到完全不同的结果（只显示了输出结果的一部分）。

```
[me@linuxbox ~]$ ls /usr/sbin/[A-Z]*  
/usr/sbin/biosdecode  
/usr/sbin/chat  
/usr/sbin/chpasswd  
/usr/sbin/chpasswd  
/usr/sbin/chroot  
/usr/sbin/cleanup-info  
/usr/sbin/complain  
/usr/sbin/console-kit-daemon
```

为什么会出现这样的差异？说来话长，简单解释如下。

在 UNIX 开发初期，它只识别 ASCII 字符，而正是这一特性导致了上面的差异。在 ASCII 码中，前 32 个字符（第 0~31 字符）都是控制字符（像 Tab 键、空格键以及 Enter 键等），后 32 个字符（第 32~63）包含可打印字符，包括大多数的标点符号以及数字 0~9，接下来的 32 个（第 64~95）包含大写字母和一些标点符号，最后的 31 个（第 96~127）则包含小写字母以及更多的标点符号。基于这样的安排，使用 ASCII 的系统使用了下面这种排序：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

这与通常的字典顺序不一样，字典中字母的顺序表通常如下。

```
aAbBcCdDeEfFgGhHjIiJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
```

随着 UNIX 在美国以外国家的普及，人们越来越希望计算机能支持美式英语中找不到的字符。于是，ASCII 字符表也得以扩展，开始使用 8 位二进制来表示，这也就增加了第 188~255 的字符，兼容了更多的语言。为了支持这种功能，POSIX 标准引入了域（locale）的概念，它通过不停调整以选择特定的位置所需要的字符集。我们可以使用下面的命令行查看系统的语言设置。

```
[me@linuxbox ~]$ echo $LANG
en_US.UTF-8
```

有了这个设置，POSIX 兼容的应用程序使用的便是字典中的字母排列顺序，而不是用 ASCII 码中的字符排列顺序。这样，便解释了上面命令行的诡异行为。A~Z 的字符范围，用字典中的顺序诠释时，包括了字母表中除了小写字母 a 的所有字母，因此使用命令行 ls /usr/sbin/[A-Z]* 才会出现全然不同的结果。

为了解决这一问题，POSIX 标准包含了许多标准字符类，这些字符类提供了一些有用的字符范围，见表 19-2。

表 19-2 POSIX 字符类

| 字符类 | 描述 |
|-----------|-------------------------------------|
| [:alnum:] | 字母字符和数字字符；在 ASCII 码中，与[A-Za-z0-9]等效 |
| [:word:] | 基本与[:alnum:]一样，只是多了一个下划线字符(_) |
| [:alpha:] | 字母字符；在 ASCII 中，等效于[A-Za-z] |
| [:blank:] | 包括空格和制表符 |
| [:cntrl:] | ASCII 控制码；包括 ASCII 字符 0~31 以及 127 |
| [:digit:] | 数字 0~9 |
| [:graph:] | 可见字符；在 ASCII 中，包括字符 33~126 |

续表

| 字符类 | 描述 |
|-----------|---|
| [:lower:] | 小写字母 |
| [:punct:] | 标点符号字符；在 ASCII 中，与[-!"#\$%&'()*+,.:/;<=>?@[\\"\\`{ }~]等效 |
| [:print:] | 可打印字符；包括[:graph:]中的所有字符再加上空格字符 |
| [:space:] | 空白字符如空格符、制表符、回车符、换行符、垂直制表符以及换页符。在 ASCII 中，等效为[\t\r\n\v\f] |
| [:upper:] | 大写字母 |
| [xdigit:] | 用于表示十六进制的字符；在 ASCII 中，与[0-9A-Fa-f]等效 |

当然，即便是有了这么多字符类，仍然没有比较方便的方法表示部分范围，如[A-M]。

使用字符类，我们可以重复上述大写字母的例子，并得到改善的输出结果。

```
[me@linuxbox ~]$ ls /usr/sbin/[[:upper:]]*
/usr/sbin/MAKEFLOPPIES
/usr/sbin/NetworkManagerDispatcher
/usr/sbin/NetworkManager
```

然而，请记住，上述并不是一个正则表达式的示例，它其实是 shell 路径名扩展的一个例子。在此处提及，主要是因为这两种用法都支持 POSIX 字符类。

恢复为传统的排列顺序

你可以设置自己的系统采用传统的（ASCII）字符顺序，方法就是改变 LANG 环境变量的值。LANG 变量包含语言的名称以及该语言环境中使用的字符集，该参数值在安装 Linux 系统选择安装语言时就已经设定。

要查看环境设置，使用下面的 locale 命令。

```
[me@linuxbox ~]$ locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

将环境设置为使用传统的 UNIX 行为，可将 LANG 变量值设为 POSIX：

```
[me@linuxbox ~]$ export LANG=POSIX
```

请注意，这样的改变将会导致系统使用美式英语（更精确地说，是 ASCII 码格式）的字符集，所以在进行此改变之前要三思。

如果想永久性维持该变化，可以在系统.bashrc 文件中添加下面的命令行。

```
export LANG=POSIX
```

19.7 POSIX 基本正则表达式和扩展正则表达式的比较

在读者正觉得正则表达式已经复杂得不能再复杂时，又会发现 POSIX 规范将正则表达式的实现方法分为了两种：基本正则表达式（BRE）和扩展正则表达式（ERE）。到目前为止，我们所讨论的正则表达式的所有特性，都得到了兼容 POSIX 的应用程序的支持，并且都是以 BRE 的方式实现。grep 命令就是这样的一个例子。

BRE 和 ERE 到底有什么区别？其实仅仅是元字符的不同！在 BRE 方式中，只承认^、\$、.、[、]、*这些是元字符，所有其他的字符都被识别为文字字符。而 ERE 中，则添加了(、)、{、}、?、+|、等元字符（及其相关功能）。

然而（也是有趣的部分），只有在用反斜杠进行转义的情况下，字符(、)、{、}才会在 BRE 被当作元字符处理，而 ERE 中，任何元符号前面加上反斜杠反而会使其被当作文字字符来处理。

由于下面要讨论的特性是 ERE 的一部分，所以需要使用不一样的 grep。传统上，这是由 egrep 程序来执行的，但是 GNU 版本的 grep 可以运用-E 选项以支持 ERE 方式。

POSIX

在 20 世纪 80 年代，UNIX 成为一款非常受欢迎的商业操作系统，但是直到 1988 年，UNIX 世界仍然一片混乱。许多电脑制造商从 UNIX 的创造者 AT&T 获得了 UNIX 源代码授权，并且都发行了不同版本的操作系统产品。然而，制造商在努力追求产品差异化的同时，每个制造商都增加了自己专用的变化以及扩展，这就渐渐限制了软件的兼容性。由于一直由专有厂商销售，所以每一个厂商都想要尽办法锁定它们的客户群。UNIX 史上的这段黑暗时期被大家称之为“割据时代”。

接着，我们进入了 IEEE (Institute of Electrical and Electronics Engineers) 时代。在 20 世纪 80 年代中期，IEEE 开始开发一套规范 UNIX 和类 UNIX 系统工作方式的标准。这些标准，官方名称是 IEEE 1003，定义了应用程序接口

(API)、shell 以及一些实用程序，它们可以在标准类 UNIX 系统中找到。该标准由 Richard Stallman 提议命名为 POSIX，它是 Portable Operating System Interface（末尾增加 X 只是为了更流畅）的缩写，后来被 IEEE 采纳。

19.8 或选项

我们将要讨论的第一个扩展正则表达式的特性是或选项 (alternation)，它是用于匹配表达式集的工具。括号表达式可以从指定字符集中匹配单一字符，而或选项则用于从字符串集或正则表达式集中寻找匹配项。

以下便利用 grep 结合 echo 作为演示实例。首先，我们进行一个简单的字符串匹配。

```
[me@linuxbox ~]$ echo "AAA" | grep AAA
AAA
[me@linuxbox ~]$ echo "BBB" | grep AAA
[me@linuxbox ~]$
```

这是一个非常直白的例子，将 echo 的输出结果送至 grep 进行匹配搜索。如果匹配成功，结果便输出打印出来；如无匹配项，则无结果输出。

现在添加或选项，它用元字符 “|” 表示。

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB'
AAA
[me@linuxbox ~]$ echo "BBB" | grep -E 'AAA|BBB'
BBB
[me@linuxbox ~]$ echo "CCC" | grep -E 'AAA|BBB'
[me@linuxbox ~]$
```

这里出现了'AAA|BBB'正则表达式，此表达式的含义是“匹配字符串 AAA 或者匹配字符串 BBB”。请注意，由于此处使用的是扩展特性，所以 grep 增加了-E 选项（虽然可以使用 egrep 命令来代替），并且将正则表达式用引号引起起来以防止 shell 将元字符 “|” 当作管道操作符来处理。另外，或选项并不局限于两种选择，还可以有更多的选择项。

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB|CCC'
AAA
```

为了将或选项可与其他正则表达式符号结合使用，我们可以用 “()” 将或选项的所有元素与其他符号隔开。

```
[me@linuxbox ~]$ grep -Eh '^bz|gz|zip)' dirlist*.txt
```

以上表达式的含义是匹配文件名以 bz、gz 或是 zip 开头的文件。如果不使用括号“()”，该正则表达式的含义就完全不同，其匹配的便是文件名以 bz 开头或者是包含 gz 和 zip 的文件。

```
[me@linuxbox ~]$ grep -Eh '^bz|gz|zip' dirlist*.txt
```

19.9 限定符

扩展正则表达式（ERE）提供多种方法指定某元素匹配的次数。

19.9.1 ? ——匹配某元素 0 次或 1 次

该限定符实际上意味着“前面的元素可选”。比如，如果我们想检查某电话号码的有效性。所谓电话号码有效，指的是电话号码必须是下面两种形式 *(nnm)nnn-nnnn* 和 *nnn nnn-nnnn* 中的一种，其中 n 是数值。于是，我们可以构造如下所示的正则表达式。

```
^\(?[0-9][0-9][0-9]\)\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$
```

此表达式中，括号字符的后面增加了“？”符号以表示括号字符只能匹配一次或零次。同样，由于括号字符在 ERE 中通常是元字符，所以其前面加上了反斜杠告诉 shell 此括号为文字字符。

示例如下。

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)\)? [0-9][0-9][0-9]$'
(555) 123-4567
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)\)? [0-9][0-9][0-9]-[0-9][0-9][0-9]$'
555 123-4567
[me@linuxbox ~]$ echo "AAA 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)\)? [0-9][0-9][0-9]-[0-9][0-9][0-9]$'
[me@linuxbox ~]$
```

由此可以看出，该表达式匹配了上述两种形式的电话号码，但不匹配那些包含非数字字符的号码。

19.9.2 *——匹配某元素多次或零次

与“？”元字符类似，“*”用于表示一个可选择的条目。然而，与“?”不同，该条目可以多次出现，而不仅仅是一次。例如，如果我们想知道一串字符是否是一句话，也就是说，这串字符是否以大写字母开头而以句号结束，并且中间

内容是任意数目的大小写字母和空格，那么要匹配这种非常粗糙的句子定义，可以用如下正则表达式。

```
[[[:upper:]][[:upper:][:lower:]]*].
```

该表达式包含了三个条目：包含[:upper:]字符类的中括号表达式、包含[:upper:]和[:lower:]两个字符类以及一个空格的中括号表达式、用反斜杠转义过的圆点符号。第二个条目后面紧跟着“*”元字符，所以只要句子的第一个字母是大写字母，后面不管会出现多少数目的大小写字母都无关紧要。

```
[me@linuxbox ~]$ echo "This works." | grep -E '[[[:upper:]][[:upper:][:lower:]]*].'
This works.
[me@linuxbox ~]$ echo "This Works." | grep -E '[[[:upper:]][[:upper:][:lower:]]*].'
This Works.
[me@linuxbox ~]$ echo "this does not" | grep -E '[[[:upper:]][[:upper:][:lower:]]*].'
[me@linuxbox ~]$
```

该表达式匹配前两个测试语句，但是不匹配第三个，原因是它的首字母不是大写并且末尾没有句号。

19.9.3 +——匹配某元素一次或多次

“+”元字符与“*”非常类似，只是“+”要求置于其前面的元素至少出现一次。示例如下，该正则表达式用于匹配由单个空格分隔的一个或者多个字母字符组成的行。

```
^([[[:alpha:]]+ ?)+$
```

示例如下。

```
[me@linuxbox ~]$ echo "This that" | grep -E '^([[[:alpha:]]+ ?)+$'
This that
[me@linuxbox ~]$ echo "a b c" | grep -E '^([[[:alpha:]]+ ?)+$'
a b c
[me@linuxbox ~]$ echo "a b 9" | grep -E '^([[[:alpha:]]+ ?)+$'
[me@linuxbox ~]$ echo "abc d" | grep -E '^([[[:alpha:]]+ ?)+$'
[me@linuxbox ~]$
```

我们可以看到此表达式并不匹配“a b 9”这一行，因为该行包含了非字母字符“9”，同样也不匹配“abc d”这一行，因为c和d之间被多个空格符分开了。

19.9.4 {}——以指定次数匹配某元素

“{”和“}”元字符用于描述最小和最大次数的需求匹配。可以通过4种方

法来指定，见表 19-3。

表 19-3 指定匹配次数

| 指定项 | 含义 |
|-------|-------------------------|
| {n} | 前面的元素恰好出现 n 次则匹配 |
| {n,m} | 前面的元素出现的次数在 n~m 次之间时则匹配 |
| {n,} | 前面的元素出现次数超过 n 次则匹配 |
| {,m} | 前面的元素出现次数不超过 m 次则匹配 |

回到前面电话号码的例子，我们可以运用此处讲到的指定重复次数的办法将原来的正则表达式

“^(\(?[0-9][0-9][0-9]\)\)? [0-9][0-9][0-9]-[0-9][0-9][0-9]\$”

简化为

“^(\?[0-9]{3}\)\)? [0-9]{3}-[0-9]{4}\$”

示例如下：

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^(\?[0-9]{3}\)\)? [0-9]{3}-[0-9]{4}$'
(555) 123-4567
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^(\?[0-9]{3}\)\)? [0-9]{3}-[0-9]{4}$'
555 123-4567
[me@linuxbox ~]$ echo "5555 123-4567" | grep -E '^(\?[0-9]{3}\)\)? [0-9]{3}-[0-9]{4}$'
[me@linuxbox ~]$
```

结果表明，修改后的表达式不管有无括号都可验证数字的有效性，并剔除那些格式不正确的数字。

19.10 正则表达式的应用

让我们回顾一些前面已经用过的命令，观察它们如何使用正则表达式。

19.10.1 用 grep 命令验证号码簿

前面的例子中，我们只验证了一个电话号码的有效性，而检验一个号码列表往往才是实际需求，所以我们先创建一个号码列表。于是，我们在命令行中输入一些“神奇的咒语”以创建号码列表，之所以称其为“神奇”是因为里面多数命令到目前为止本书还未涉及到。不过不用担心，我们将在下面的章节中

会详细讲述。下面的代码便是神奇咒语的内容。

```
[me@linuxbox ~]$ for i in {1..10}; do echo "(${RANDOM:0:3}) ${RANDOM:0:3}-$${RANDOM:0:4}" >> phonelist.txt; done
```

该命令行会产生一个包含 10 个电话号码的名为 *phonelist.txt* 的文件。每次重复该命令行，该列表就会添加 10 个号码。我们也可以通过更改命令行前端的数字 10 来指定创建更多或更少的电话号码。然而，如果检查文件内容，我们会发现如下问题。

```
[me@linuxbox ~]$ cat phonelist.txt
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
```

其中有部分数字是畸形的，而这正好满足我们的练习要求，因为接下来就是要用 *grep* 判断它们的有效性。

进行有效验证，可对文件内容进行扫描以搜索无效数字，并将结果显示出来。

```
[me@linuxbox ~]$ grep -Ev '^(([0-9]{3}\)) [0-9]{3}-[0-9]{4}$' phonelist.txt
(292) 108-518
(129) 44-1379
[me@linuxbox ~]$
```

此处我们使用了 *-v* 选项输出相反结果，也就是输出列表中不符合指定表达式的那些行。此表达式本身在每行末尾使用了锚元字符，从而确保每一个数字末尾没有多余的字符。另外，本表达式要求有效的号码中必须要有括号，与前面所举的号码例子有稍许不同。

19.10.2 用 *find* 查找奇怪文件名的文件

find 命令的 *test* 选项可以用正则表达式表示。运用正则表达式时，*find* 和 *grep* 有一点不同，*grep* 命令是搜索那些只包含与指定表达式匹配的字符串的行，而 *find* 则要求文件名与指定表达式完全一致。下面的例子中，我们将利用 *find* 命令结合正则表达式来查找文件名中不包含如下所示集合中的任一字符的文件。

```
[-./0-9a-zA-Z]
```

用此表达式进行搜索，将会输出文件名中包含内嵌空格以及其他潜在不规范字符的文件。

```
[me@linuxbox ~]$ find . -regex '.*[^-.-/0-9a-zA-Z].*'
```

由于要求整个路径名与正则表达式的描述完全一致，所以表达式的两端增加了“`.*`”以匹配 0 个或多个字符。在表达式的中间部分，我们使用了一个否定的中括号表达式，其中包含了可接受的路径名字符集。

19.10.3 用 locate 查找文件

`locate` 命令既支持基本正则表达式（`--regexp` 选项），也支持扩展正则表达式（`--regex` 选项）。利用 `locate`，可以完成之前对 `dirlist` 文件所做的许多操作。

```
[me@linuxbox ~]$ locate --regex 'bin/(bz|gz|zip)'
/bin/bzcat
/bin/bzcmp
/bin/bzdiff
/bin/bzegrep
/bin/bzexe
/bin/bzfgrep
/bin/bzgrep
/bin/bzip2
/bin/bzip2recover
/bin/bzless
/bin/bzmore
/bin/gzexe
/bin/gzip
/usr/bin/zip
/usr/bin/zipcloak
/usr/bin/zipgrep
/usr/bin/zipinfo
/usr/bin/zipnote
/usr/bin/zipsplit
```

利用或选项，我们搜索到了那些路径名中包含 `bin/bz`、`bin/gz` 或 `/bin/zip` 等字符串的文件。

19.10.4 利用 less 和 vim 命令搜索文本

`less` 和 `vim` 采用同样的方法进行文本搜索。按下 “/” 键后输入正则表达式，即开始进行搜索。我们可以利用 `less` 查看 `phonelist.txt` 文件的内容：

```
[me@linuxbox ~]$ less phonelist.txt
```

然后查找有效的表达式。

```
(232) 298-2265
(624) 381-1078
```

```
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
-
-
-
/^([0-9]{3}\) [0-9]{3}-[0-9]{4}$
```

less 会以高亮的方式显示匹配字符传，这样就很容易找到那些无效的号码。

```
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
-
-
-
(END)
```

另一方面，vim 也支持基本正则表达式，所以，我们可以用下面的搜索表达式。

```
/([0-9]\{3\}) [0-9]\{3\}-[0-9]\{4\}
```

可以看到，表达式基本一样。然而，在扩展表达式中被当作元字符的许多字符在，在基本表达式中则被看作文字字符，只有使用反斜杠转义后才会被当作元字符。匹配项是否以高亮形式显示，则取决于自己系统上 vim 的设置。如果没有，请尝试在命令模式中输入 `hlsearch` 以激活高亮搜索。

注意

由于 Linux 发行版本的不同，vim 可能不支持文本高亮搜索。尤其是 Ubuntu，其默认提供了一个 vim 的精简版本。在这样的系统上，可能会需要用软件包管理器来安装一个较完整的 vim 版本。

19.11 本章结尾语

本章介绍了正则表达式的很多用法。当然，如果用它们进行一些额外的应用搜索，你会发现正则表达式有更多的用途。我们可以通过查看 man 手册页获

取更详细的内容。

```
[me@linuxbox ~]$ cd /usr/share/man/man1  
[me@linuxbox man1]$ zgrep -El 'regex|regular expression' *.gz
```

`zgrep` 程序比 `grep` 在前端多了一个字母 `z`，由此便可以对压缩文件进行搜索。本例中，我们在 `man` 文件通常存放的目录下对压缩的第一部分 `man` 手册页进行了搜索。该命令行的输出结果是一列包含字符串 `regex` 或 `regular expression` 的文件。可以看到，正则表达式可用于大量应用程序中。

本章未谈及基本正则表达式的另外一个特性——后向引用（*back reference*），该特性将在下一章中进行详解。

第 20 章

文 本 处 理

由于所有类 UNIX 操作系统都严重依赖于文本文件来进行某些数据类型的存储，所以需要有很多可以进行文本操作的工具。本章主要介绍一些与“切割”文本有关的命令，第 21 章会进一步探讨文本处理工具，并重点讲解那些用于格式化文本输出以及其他满足人类需求的程序。

本章首先会回顾之前讲过的一些命令，然后讲解一些新的命令。

- **cat:** 连接文件并打印到标准输出。
- **sort:** 对文本行排序。
- **uniq:** 报告并省略重复行。
- **cut:** 从每一行中移除文本区域。
- **paste:** 合并文件文本行。
- **join:** 基于某个共享字段来联合两个文件的文本行。

- **comm:** 逐行比较两个已经排好序的文件。
- **diff:** 逐行比较文件。
- **patch:** 对原文件打补丁。
- **tr:** 转换或删除字符。
- **sed:** 用于过滤和转换文本的流编辑器。
- **aspel:** 交互式拼写检查器。

20.1 文本应用程序

到目前为止，我们总共介绍了两种文本编辑器（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。那么，除了这些，文本操作还有哪些用途？事实证明，它还有很大用途。

20.1.1 文件

许多人都采用纯文本格式编辑文件。虽然大家都知道用一些较小的文本文件进行一些简单的笔记很方便、很实用，但同样，我们也可以用文本格式编辑较大的文档。有一种常用的方法，即首先在文本编辑器中编辑大型文档的内容，然后使用标记语言描述文件格式。许多科学论文就是用这种方式撰写的，因为基于 UNIX 的文本处理系统是最早一批支持先进排版布局的，而这些先进的排版技术又正是技术领域的专家们所需要的。

20.1.2 网页

网页可以说是世界上最常见的电子文档。网页属于文本文档，一般使用 HTML（Hypertext Markup Language）或者 XML（eXtensible Markup Language）等标记语言描述内容的可视化形式。

20.1.3 电子邮件

电子邮件本质上是一种基于文本的媒介，即便是非文本附件，在传输的时候也会被转成文本格式。读者可以亲自验证这一点，首先下载一个电子邮件信息并用 less 命令查看其内容，就会发现其内容包含一个 header 开头，其描述的是此封邮件的来源及其在传输过程中所经历的处理，然后才是邮件信

息的正文。

20.1.4 打印机输出

在类 UNIX 系统中，准备向打印机传送的信息是以纯文本格式传送的。如果该页包含图像，则将其转换成 PostScript 文本格式页面描述语言后再送至指定程序以打印图像像素。

20.1.5 程序源代码

类 UNIX 系统中的许多命令行程序都是为了支持系统管理和软件开发而编写的，文本处理程序也不例外。它们中多数是为解决软件开发问题而设计的。文本处理对软件开发者如此重要，是因为所有的软件都是从文本开始，程序员实际所编写的源代码，也总是以文本的形式编辑。

20.2 温故以求新

在第 6 章，我们学习了一些既支持命令行参数输入也支持标准输入的命令。不过当时只是泛泛而谈，现在我们将详细讨论这些命令如何用于文本处理。

20.2.1 cat——进行文件之间的拼接并且输出到标准输出

cat 命令有许多有趣的参数选项，而其中多数则是用于提高文本内容的可视化效果。-A 选项就是一个例子，它用于显示文本中的非打印字符。例如，用户有时会想知道可见文本中是否嵌入了控制字符，其中最为常见的就是制表符（而不是空格）以及回车符，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。

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，只需要输入 cat 命令（随后指定了用于重定向输出的文件）再输入文本内容，按 Enter 键结束行输入，最后按下 Ctrl-D 告诉 cat 到达文件末尾。下例中，我们输入了一个以 Tab 制表符开头、空格符结尾的文本行。

```
[me@linuxbox ~]$ cat > foo.txt
The quick brown fox jumped over the lazy dog.
[me@linuxbox ~]$
```

下面，我们利用带有-A 选项的 cat 命令显示文本内容：

```
[me@linuxbox ~]$ cat -A foo.txt
^IThe quick brown fox jumped over the lazy dog. $
[me@linuxbox ~]$
```

输出结果表明，文本中的 Tab 制表符由符号“^I”表示。这是一种常见的表示方法，意思是“Ctrl-I”，结果证明，它等同于 Tab 制表符。同时，在文件末尾出现的“\$”符说明行末尾存在空格。

MS-DOS 文本与 UNIX 文本的比较

我们利用 cat 查找文本中的非打印字符，原因之一是 cat 可以发现隐藏的回车符。而这些隐藏的回车符来自哪里呢？当然是 DOS 和 Windows！UNIX 和 DOS 在文本文件中定义每行结束的方式并不相同，UNIX 以换行符（ASCII 码 10）作为行末尾，而 MS-DOS 系统及其衍生系统则使用回车符（ASCII 码 13）和换行符共同作为行末尾。

有几种方法可以将文件从 DOS 格式转化为 UNIX 格式。许多 Linux 系统，都自带 dos2UNIX 和 UNIX2dos 程序，它们用于 DOS 和 UNIX 格式之间的相互转换。然而，即便系统中没有 dos2UNIX 程序也没关系，DOS 格式转换为 UNIX 格式的过程非常简单，只要将多余的回车符删除就可以，此过程可以用本章后续所讲的一些程序很简单地实现。

cat 也有很多用于修改文本的参数选项。最著名的两个选项：-n，对行编号；-s，禁止输出多个空白行。示例如下。

```
[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 ~]$
```

本例中，我们创建了一个 foo.txt 测试文件的新版本，该文件内容为两个文本行，并以空白行隔开。用 cat 加-ns 选项对其操作后，多余的空白行便被移除，并对剩余的行进行了编号。然而这并不是多个进程在操作这个文本，只有一个进程。

20.2.2 sort——对文本行进行排序

sort 是一个排序程序，它的操作对象为标准输入或是命令行中指定的一个或

多个文件后将结果送至标准输出。与 cat 用法类似，如下所示，我们将直接使用键盘演示标准输入内容的处理过程。

```
[me@linuxbox ~]$ sort > foo.txt
c
b
a
[me@linuxbox ~]$ cat foo.txt
a
b
c
```

输入 sort 命令后，输入字母 c、b、a，最后按下 Ctrl-D 结束输入。然后查看处理结果，会发现这些行都以排好的顺序出现。

由于 sort 命令允许多个文件作为其输入参数，所以可以将多个文件融合为一个已排序的整体文件。例如，我们有三个文本文件，并期望将它们拼接为一个已排序的整体文件。我们可以用下面的命令行去执行。

```
sort file1.txt file2.txt file3.txt > final_sorted_list.txt
```

sort 有一些有趣的选项。表 20-1 列出了部分。

表 20-1 常见的 sort 选项

| 选项 | 全局选项表示 | 描述 |
|----|---|---|
| -b | --ignore-leading-blanks | 默认情况下，整个行都会进行排序操作，也就是从行的第一个字符开始。添加该选项后，sort 会忽略行开头的空格，并且从第一个非空白字符开始排序 |
| -f | --ignore-case | 排序时不区分字符大小写 |
| -n | --numeric-sort | 基于字符串的长度进行排序。该选项使得文件按数值顺序而不是按字母表顺序进行排序 |
| -r | --reverse | 逆序排序。输出结果按照降序排列而不是升序 |
| -k | --key= <i>field1</i> [, <i>field2</i>] | 对 <i>field1</i> 与 <i>field2</i> 之间的字符排序，而不是整个文本行 |
| -m | --merge | 将每个输入参数当作已排好序的文件名。将多个文件合并为一个排好序的文件，而不执行额外的排序操作 |
| -o | --output= <i>file</i> | 将排序结果输出到文件而不是标准输出 |
| -t | --field-separator= <i>char</i> | 定义字段分隔符。默认情况下，字段是由空格或制表符分隔的 |

尽管以上多数选项的作用都易从其字面意义中看出，但也有一些例外，用于数值排序的 -n 选项就是一个例子。该参数选项可使 sort 根据数值进行排序。作为演示，我们可将 du 命令的输出结果进行排序，以确定最大的硬盘空间用户。正常情况下，du 命令会列出一个以路径名顺序排列的列表。

```
[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
```

本例中，我们把结果管道到 head 命令，把输出结果限制为只显示前 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
```

通过使用-nr 参数选项，我们便可以产生一个逆向的数值排序，它使得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一行的开头。但是如果我们要基于文本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 2012-04-04 02:42 [REDACTED]
-rwxr-xr-x 1 root  root   101556 2011-11-27 06:08 a2p
-rwxr-xr-x 1 root  root   13036 2012-02-27 08:22 aconnect
-rwxr-xr-x 1 root  root   10552 2011-08-15 10:34 acpi
-rwxr-xr-x 1 root  root    3800 2012-04-14 03:51 acpi_fakekey
-rwxr-xr-x 1 root  root    7536 2012-04-19 00:19 acpi_listen
-rwxr-xr-x 1 root  root    3576 2012-04-29 07:57 addpart
-rwxr-xr-x 1 root  root   20808 2012-01-03 18:02 addr2line
-rwxr-xr-x 1 root  root  489704 2012-10-09 17:02 adept_batch
```

此刻，忽略 ls 命令自有的根据文件大小排序的功能，而用 sort 程序依据文件大小进行排序。

```
[me@linuxbox ~]$ ls -l /usr/bin | sort -nr -k 5 | head
-rwxr-xr-x 1 root  root  8234216 2012-04-07 17:42 inkscape
-rwxr-xr-x 1 root  root  8222692 2012-04-07 17:42 inkview
-rwxr-xr-x 1 root  root  3746508 2012-03-07 23:45 gimp-2.4
-rwxr-xr-x 1 root  root  3654020 2012-08-26 16:16 quanta
-rwxr-xr-x 1 root  root  2928760 2012-09-10 14:31 gdbtui
```

```
-rwxr-xr-x 1 root  root  2928756 2012-09-10 14:31 gdb
-rwxr-xr-x 1 root  root  2602236 2012-10-10 12:56 net
-rwxr-xr-x 1 root  root  2304684 2012-10-10 12:56 rpcclient
-rwxr-xr-x 1 root  root  2241832 2012-04-04 05:56 aptitude
-rwxr-xr-x 1 root  root  2202476 2012-10-10 12:56 smbcacls
```

sort 的许多用法都与表格数据处理有关，比如上面 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，我们会说每一行就是一项记录，而每一个记录又包含多个字段，诸如文件属性、链接数、文件名、文件大小等。sort 能够处理独立的字段，在数据库术语中，我们可以指定一个或多个关键字段作为排序的关键值。在上面的例子中，指定了 n 和 r 选项进行数值的逆序排序，并指定-k 5 让 sort 程序使用第 5 个字段作为排序的关键值。

k 这个参数选项非常有趣，并且有很多特性，但是首先我们需要了解 sort 是如何定义字段的。让我们考虑一个非常简单的文本文件，它只有一行，并且该行只包含了该作者的名字。

```
William Shotts
```

默认情况下，sort 程序会把该行看作有两个字段。第一个字段包含“William”字符串，第二个字段则是“Shotts”。这意味着空白字符（空格和制表符）用作字段之间的界定符，并且在排序时，这些界定符是包括在字段中的。

重新回到前面的 ls 例子，我们可以看到 ls 的输出行包含 8 个字段，并且第 5 个字段指的是文件大小。

```
-rwxr-xr-x.1 root  root  8234216 2012-04-07 17:42 inkscape
```

让我们考虑用下面的文件。该文件包含从 2006 年～2008 年三款流行的 Linux 发行版的发行历史。文件每行都有 3 个字段：发行版本名、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 | 11.0 | 06/19/2008 |
| Ubuntu | 8.04 | 04/24/2008 |
| Fedora | 8 | 11/08/2007 |
| 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 |
| SUSE | 10.1 | 05/11/2006 |
| 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 |

使用文本编辑器（可能是 vim），输入此数据并将其保存为 *distros.txt*。

接下来，我们试着对该文件进行排序并观察其输出结果。

```
[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
Fedora       9      05/13/2008
SUSE        10.1    05/11/2006
SUSE        10.2    12/07/2006
SUSE        10.3    10/04/2007
SUSE        11.0    06/19/2008
Ubuntu      6.06    06/01/2006
Ubuntu      6.10    10/26/2006
Ubuntu      7.04    04/19/2007
Ubuntu      7.10    10/18/2007
Ubuntu      8.04    04/24/2008
Ubuntu      8.10    10/30/2008
```

嗯，大部分正确。Fedora 的版本号排序时却出现了问题。因为字符集中，字符 1 是在字符 5 前面的，所以导致版本 10 位于第一行而版本 9 却在最后。

为了解决这个问题，我们必须依据多个键值进行排序。首先我们对第一个字段进行字母排序，然后再对第三字段进行数值排序。`sort` 支持`-k` 选项的多个实例，所以可以指定多个排序键值。事实上，一个键值可能是一个字段范围，如果没有指定任何范围（如前面所举的例子），`sort` 会使用一个键值，该键值始于指定的字段，一直扩展到行尾。

如下便是采用多键值进行排序的语法。

```
[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
Fedora      8      11/08/2007
Fedora      9      05/13/2008
Fedora     10      11/25/2008
SUSE        10.1    05/11/2006
SUSE        10.2    12/07/2006
SUSE        10.3    10/04/2007
SUSE        11.0    06/19/2008
Ubuntu      6.06    06/01/2006
Ubuntu      6.10    10/26/2006
Ubuntu      7.04    04/19/2007
Ubuntu      7.10    10/18/2007
Ubuntu      8.04    04/24/2008
Ubuntu      8.10    10/30/2008
```

虽然为了清晰，我们使用了选项的长格式，但是`-k1、-k 2n` 格式是等价的。

在第一个 key 选项的实例中，指定了一个字段范围。因为我们只想对第一个字段排序，所以指定了“1, 1”，它意味着“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，表示“第二个字段是排序的键值，并且按照数值排序”。一个选项字母可能包含在一个键值说明符的末尾，用来指定排序的种类。这些选项字母与 sort 命令的全局选项一样：b（忽略开头空白字符）、n（数值排序）、r（逆序排序）等。

以上列表的第三个字段包含的日期形式并不利于排序。计算机中，日期通常以 YYYY-MM-DD 的形式存储，以方便按时间顺序排序，但该文本中的时间则是以美国形式 MM/DD/YYYY 存储。那么，该如何对其进行时间排序呢？

幸好 sort 提供了一种解决方法。sort 的 key 选项允许在字段中指定偏移，所以我们可以在字段内定义键值。

```
[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
Fedora      9      05/13/2008
Ubuntu      8.04   04/24/2008
Fedora      8      11/08/2007
Ubuntu      7.10   10/18/2007
SUSE        10.3   10/04/2007
Fedora      7      05/31/2007
Ubuntu      7.04   04/19/2007
SUSE        10.2   12/07/2006
Ubuntu      6.10   10/26/2006
Fedora      6      10/24/2006
Ubuntu      6.06   06/01/2006
SUSE        10.1   05/11/2006
Fedora      5      03/20/2006
```

通过指定-k 3.7，我们告诉 sort 从第三个字段的第 7 个字符开始排序，也就是从年份开始排序。同样，指定-k 3.1 和-k 3.4 选项以区分日期中的月和日，另外我们利用了 n、r 选项进行逆序数值排序。同时添加的 b 选项用来删除日期字段中开头的空格（行与行之间的空格字符数量不同，因此会影响排序结果）。

有些文件并不是使用制表符或空格符作为字段定界符，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
```

该文件的字段之间以冒号（:）作为分界符，那么该如何利用关键字段对此文件进行排序呢？sort 提供了-t 选项定义字段分隔符，根据 passwd 文件的第 7 个字段内容进行排序（用户默认的 shell 环境），如下面的命令行。

```
[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
```

指定冒号字符作为字段分隔符，便实现了依据第 7 个字段进行排序的目的。

20.2.3 uniq——通知或省略重复的行

与 sort 相比，uniq 算是一个轻量级命令。uniq 执行的是一个看似简单的任务，给定一个已排好序的文件（包括标准输入）后，uniq 会删除任何重复的行并将结果输出到标准输出中。它通常与 sort 结合使用以删除 sort 输出内容中重复的行。

注意

虽然 uniq 是一个与 sort 一起使用的传统的 UNIX 工具，但是 GNU 版本的 sort 同样支持-u 选项，以用于移除 sort 输出内容中的重复行。

创建一个文本文件以验证此特性。

```
[me@linuxbox ~]$ cat > foo.txt
a
b
c
a
b
c
```

不要忘了按下 Ctrl-D 以结束标准输入。此刻，如果运行 uniq，文件内容并没有很大改动，重复的行也并没有被删除。

```
[me@linuxbox ~]$ uniq foo.txt
a
b
c
a
```

b
c

uniq 只有对已经排好序的文本才有作用。

```
[me@linuxbox ~]$ sort foo.txt | uniq
a
b
c
```

这是因为 uniq 只能移除相邻的重复行。

当然, uniq 有许多选项, 表 20-2 列出了常用的一些。

表 20-2 常见的 uniq 选项

| 选项 | 功能描述 |
|------|---|
| -c | 输出重复行列表, 并且重复行前面加上其出现的次数 |
| -d | 只输出重复行, 而不包括单独行 |
| -f n | 忽略每行前 n 个字段。字段之间以空格分开, 这与 sort 类似, 但与 sort 不同的是, uniq 没有提供参数设置可选择的字段分隔符 |
| -i | 行与行之间比较时忽略大小写 |
| -s n | 跳过(忽略)每行的前 n 个字符 |
| -u | 仅输出不重复的行。该选项是默认的 |

使用 uniq 的-c 选项, 可输出文本中重复行的数量, 示例如下。

```
[me@linuxbox ~]$ sort foo.txt | uniq -c
2 a
2 b
2 c
```

20.3 切片和切块

下面讨论的 3 个命令, 它们的作用是剥离文本文件的列, 并将它们以期望的方式重组。

20.3.1 cut——删除文本行中的部分内容

cut 命令用于从文本行中提取一段文字并将其输出至标准输出。它可以接受多个文件和标准输入作为输入参数

指定所要提取的内容, 实现起来确实有点别扭, 表 20-3 列出的是指定时要用到的选项。

表 20-3 cut 选择选项

| 选项 | 功能描述 |
|---------------|--|
| -c char_list | 从文本行中提取 char_list 定义的部分内容。此列表可能会包含一个或更多冒号分开的数值范围 |
| -f field_list | 从文本行中提取 field_list 定义的一个或多个字段。该列表可能会包含由冒号分隔的一个、多个字段或字段范围 |
| -d delim_char | 指定-f 选项后，使用 delim_char 作为字段分界符。默认时，字段必须以单个 Tab 制表符隔开 |
| --complement | 从文本中提取整行，除了那些由-c 和/或-f 指定的部分 |

正如大家所看到的，cut 提取文本的方式非常不灵活。cut 适合从其他命令的输出结果中提取文本内容，而不是直接从输入文本中提取。我们可以判断下面的 distros.txt 文件是否达到了 cut 的提取要求，利用 cat 的-A 选项，可以检查该文件是否用 Tab 作为字段分隔符的。

```
[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$
```

看起来不错——没有内嵌空格，字段之间仅仅只有单个的制表符。鉴于文件使用的是制表符而不是空格，所以可以使用-f 选项提取字段内容

```
[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

由于 *distros* 文件是以制表符作为分界符的，所以用 cut 提取字段而不是字符再合适不过。这是因为用 Tab 作为分界符的文件，一般每行不会包含相同的字符数，所以计算字符在行内的位置很困难或是根本不可能。然而在上例中，我们已经提取好了包含相同长度数据的字段，从而可以拿此字段作为字符提取实例。

```
[me@linuxbox ~]$ cut -f 3 distros.txt | cut -c 7-10
2006
2008
2008
2008
2007
2007
2006
2007
2007
2006
2006
2008
2006
2008
2008
2006
2006
```

对输出结果再进行一次 cut 操作，便可以将字段中与年份相对应的第 7 至第 10 个字符提取出来，命令行中的符号“7-10”指范围。cut 的 man 手册页包含了关于范围指定的完整描述。

当处理字段时，我们可以指定非 Tab 字符作为分界符。如下所示的例子演示的是从 /etc/passwd 文件中提取了每行的第一个字段。

```
[me@linuxbox ~]$ cut -d ':' -f 1 /etc/passwd | head
root
daemon
bin
sys
sync
games
man
lp
mail
news
```

此处我们还使用了 -d 选项指定冒号作为字段的分界符。

扩展制表符 Tab

上例中的 *distros.txt* 文件已经被格式化为 cut 提取最理想的形式。但是如

果想要一个可以完全由 `cut` 借助字符，而非字段执行提取操作的文件，该怎么办呢？这就需要我们将文件中的 Tab 符号置换成等长的空格符号。幸运的是，GNU 的 `coreutils` 软件包提供了 `expand` 这样一个工具，该命令的输入参数既可以是标准输入也可以是一至多个文件，改动后的文本会以标准形式输出。

如果事先用 `expand` 处理 `distros.txt` 文件，便可以直接用 `cut -c` 从文件中提取任意范围的字符。示例如下，通过 `expand` 扩展该文件后，使用 `cut` 提取文本行中从第 23 个到末尾的字符，这样，我们便单独提取出了发行年份这一列表栏。

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

另外，`coreutils` 软件包同样也提供了 `unexpand` 命令，从而用空格取代了制表符。

20.3.2 `paste`——合并文本行

`paste` 命令是 `cut` 的逆操作，它不是从文本文件中提取列信息，而是向文件中增加一个或是更多的文本列。该命令读取多个文件并将每个文件中提取出的字段结合为一个整体的标准输出流。与 `cut` 类似，`paste` 也可以接受多个文件输入参数和标准输入。至于 `paste` 是如何运行的，我们可以通过下面的例子进行了解。如下所示，我们描述了一个使用 `paste` 对 `distros.txt` 文件按照发行版本的时间顺序而排序的例子。

首先，我们用前面所学的 `sort` 命令，得到一个依据日期进行排序的 `distros` 列表，并将输出结果储存于文件 `distros-by-date.txt` 中。

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

接下来，我们使用 `cut` 提取文件中前两个字段（`distros` 的名字和发行版本），并将结果存于文件 `distro-versions.txt` 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
```

最后一步准备工作就是提取发行版本日期，并将结果存于 `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
```

此刻，万事俱备，只欠东风。作为整个例子的最后一步，我们使用 `paste` 将提取的日期这一列内容置于 `distro` 中操作系统名称和发行版本号这两列之前，于是便生成了一个按时间顺序排列的发行版本列表。这一过程只是简单地使用 `paste` 命令来将各参数按照指定顺序进行排列。

```
[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
12/07/2006    SUSE        10.2
10/26/2006    Ubuntu       6.10
10/24/2006    Fedora      6
06/01/2006    Ubuntu       6.06
05/11/2006    SUSE        10.1
03/20/2006    Fedora      5
```

20.3.3 join——连接两文件中具有相同字段的行

从某种程度上来说，`join` 与 `paste` 类似，因为它也是向文件增加列信息，只是实现方式有些许不同。“`join`”操作通常与“关联数据库”联系在一起，它在关联数据库中把共享关键字段多个表格的数据组合成一个期望结果。“`join`”是一个基于共享关键字段将多个文件的数据拼接在一起的操作。

至于“`join`”命令在关联数据库中是如何操作的，我想大家一定想知道。示例如下，假定有一个比较小的包含两个表格的数据库，并且每个表格都只包含一项纪录。第一个表格，叫做 `CUSTOMERS`，它有三个字段，分别是顾客号码（`CUSTNUM`），顾客的姓（`FNAME`）以及顾客的名（`LNAME`）。

| CUSTNUM | FNAME | LNAME |
|---------|-------|-------|
| ===== | ===== | ===== |
| 4681934 | John | Smith |

第二个表格叫做 ORDERS，包含 4 个字段，它们是订单号 (ORDERNUM)、顾客号码 (CUSTNUM)、数量 (QUAN) 以及订购内容 (ITEM)。

| ORDERNUM | CUSTNUM | QUAN | ITEM |
|------------|---------|------|-------------|
| ===== | ===== | ==== | ==== |
| 3014953305 | 4681934 | 1 | Blue Widget |

请注意，两个表格拥有公共字段 CUSTNUM。这很重要，因为这就是两个表格之间的联系。

join 操作完成了两个表格中的字段结合，从而得到了有用的输出结果，诸如用于准备一张发票。利用两个表格中 CUSTNUM 字段的共有匹配值，进行 join 操作后便会输出以下结果。

| FNAME | LNAME | QUAN | ITEM |
|-------|-------|------|-------------|
| ===== | ===== | ==== | ==== |
| John | Smith | 1 | Blue Widget |

做为演示，我们需要创建两个具有共有字段的文件，于是便可以使用 *distros-by-date.txt* 文件。利用该文件，可以生成两个附属文件，其中一个文件包含的内容是发行时间（作为本例中的共有字段）和发行版本名。

```
[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
```

第二个文件的内容则包含发行时间和发行版本号。

```
[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 |

此刻，两个具有公共字段（“发行时间”作为共有字段）的文件便准备妥当。此处需要重点强调的是，文件必须事先依据共有关键字段排好序，因为只有这样 join 才能正常工作。

```
[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
```

同样请注意，默认情况下，join 会把空格当作输入字段的分界符，而以单个空格作为输出字段的分界符，当然我们也可以通过指定参数选项改变这一默认属性。我们可以查看 join 的 man 手册页获取更详细信息。

20.4 文本比较

比较文本文件的版本号通常很有用，尤其对系统管理者以及软件开发者而言。例如，一个系统的管理者可能需要将已存在的配置文件与原先的配置文件相比较，以检查出系统漏洞，与此类似，程序员也会经常需要查看程序代码所经历的变化。

20.4.1 comm——逐行比较两个已排序文件

comm 命令一般用于文本文件之间的比较，显示两文件中相异的行以及相同的行。作为演示，我们首先利用 cat 生成两个近乎一样的文本文件。

```
[me@linuxbox ~]$ cat > file1.txt
a
b
c
d
[me@linuxbox ~]$ cat > file2.txt
b
```

```
c  
d  
e
```

接下来，便利用 comm 比较两个文件的差异。

```
[me@linuxbox ~]$ comm file1.txt file2.txt
a
b
c
d
e
```

从以上结果可以看出，comm 输出了三列内容。第一列显示的是第一个文件中独有的行，第二列显示的是第二个参数文件中独有的行，第三列显示的则是两个文件所共有的行。comm 还支持-n 形式的参数选项，此处的 n 可以是 1、2 或者 3，使用时，它表示省略第 n 列的内容。例如，如果只想显示两个文件的共有行，便可以省略第 1 列和第 2 列的内容，示例如下。

```
[me@linuxbox ~]$ comm -12 file1.txt file2.txt
b
c
d
```

20.4.2 diff——逐行比较文件

与 comm 命令类似，diff 用于检测文件之间的不同。然而，diff 比 comm 更复杂，它支持多种输出形式，并且具备一次性处理大文件集的能力。diff 通常被软件开发者用于检查不同版本的源代码之间的差异，因为它能够递归检查源代码目录（通常称为源树）。diff 的常见用法就是创建 diff 文件和补丁，它们可以为诸如 patch（后面将会深入讨论）这样的命令所用，从而实现一个版本的文件更新为另一个版本。

将 diff 运用于前面例子中使用到的文件，我们会发现其默认形式的输出结果其实是对两者文件差异的一个简洁描述。

```
[me@linuxbox ~]$ diff file1.txt file2.txt
1d0
< a
4a4
> e
```

默认形式中，每一组改动的前面都有一个以“范围 执行操作 范围”形式 (*range operation range*) 表示的改变操作命令（见表 20-4），该命令会告诉程序对第一个文件的某个位置进行某种改变，便可实现与第二个文件内容一致。

表 20-4 diff 改变命令

| 改变操作 | 功能描述 |
|-------|---|
| r1ar2 | 将第二个文件中 r2 位置的行添加到第一个文件中的位置 r1 处 |
| r1cr2 | 用第二个文件 r2 处的行替代第一个文件 r1 处的行 |
| r1dr2 | 删除第一个文件 r1 处的行，并且删除的内容作为第二个文件 r2 行范围的内容 |

此格式中，范围 range 一般是由冒号隔开的起始行和末尾行组成。虽然，此格式是默认的（大多数情况下是为了兼容 POSIX 的同时向后兼容传统 UNIX 版本的 diff），但它并没有其他格式的应用广泛，上下文格式和统一格式才是比较普遍使用的格式。

上下文格式的输出结果如下。

```
[me@linuxbox ~]$ diff -c file1.txt file2.txt
*** file1.txt      2012-12-23 06:40:13.000000000 -0500
--- file2.txt      2012-12-23 06:40:34.000000000 -0500
*****
*** 1,4 ****
- a
b
c
d
--- 1,4 ----
b

c
d
+ e
```

该结果以两个文件的名字和时间信息开头，第一个文件用星号表示，第二个文件用破折号表示。输出结果的其余部分出现的星号和破折号则分别表示各自所代表的文件。其他的内容便是两个文件之间的差异组，包括文本的默认行号。第一组差异，以*** 1,4 ****开头，表示第一个文件中的第 1 行至第 4 行；第二组便以--- 1,4 ----开头，表示第二个文件的第 1 行至第 4 行。每个差异组的行都是以如下四个标识符之一开头，见表 20-5。

表 20-5 diff 上下文格式差异标识符

| 标识符 | 含义 |
|-----|--|
| (无) | 该行表示上下文文本。表示两个文件共有的行 |
| - | 缺少的行。指此行内容只在第一个文件中出现，第二个文件中则没有 |
| + | 多余的行。此行内容只有第二个文件才有，第一个文件则没有 |
| ! | 改变的行。两个版本的行内容都会显示出来，每一个都各自出现在差异组中相应的部分 |

统一格式与上下文格式相似但是更简明，此格式用-u 选项指定。

```
[me@linuxbox ~]$ diff -u file1.txt file2.txt
--- file1.txt      2012-12-23 06:40:13.000000000 -0500
+++ file2.txt      2012-12-23 06:40:34.000000000 -0500
@@ -1,4 +1,4 @@
-a
 b
 c
 d
+e
```

上下文格式和统一格式之间最显著的区别就是，统一格式下没有重复的文本行，这使得统一格式的输出结果比上下文格式的精简。上例中，也输出了上下文格式中出现的文件时间信息，并且后面紧跟着“@@ -1,4 +1,4 @@”字符串，它表示差异组中描述的两个文件各自的行范围。此字符串之后便是行本身，其中包含默认的三行文本内容。正如表 20-6 所示，每一行都以下面的三个可能的标示符开头。

表 20-6 diff 统一格式的差异标示符

| 字符 | 含义 |
|-----|-----------------------|
| (无) | 两个文件共有的行 |
| - | 相对于第二个文件而言，第一个文件中没有的行 |
| + | 第一个文件多余的行 |

20.4.3 patch——对原文件进行 diff 操作

patch 命令用于更新文本文件。它利用 diff 命令的输出结果将较旧版本的文件升级成较新版本。下面看一个众所周知的例子，Linux 内核是由一个很大的、组织松散的志愿者团队开发的，其源代码处于持续不断更新中。Linux 内核包含几百万行代码，所以相比而言，某位开发成员每次所做的改变是如此微不足道。因此，对于每位开发者来说，每对代码改动一次就得向其他开发者发送整个内核源代码树是多么不切实际。事实上，一般只要发送 diff 补丁文件即可，diff 补丁的内容是内核从较旧版本转变为较新版本所经历的改变，接收者然后使用 patch 命令将这些改变应用于其自身的源代码树。diff/patch 有两个重要的优点。

- 与源代码树的大小相比，diff 文件很小。
- diff 文件非常简洁地描述了文件所做的改变，便于补丁的接收者快速对其进行评价。

当然，diff/patch 不仅仅局限于源代码，它适用于任何文本文件。因此，它同样适用于配置文件以及其他文本文件。

生成供 patch 使用的 diff 文件，GNU 文件系统建议采用如下方式使用 diff。

```
diff -Naur old_file new_file > diff_file
```

此处 *old_file* 和 *new_file* 既可以是单独的文件也可以是包含文件的目录，使用-r 参数选项则是为了进行递归目录树搜索。

一旦创建了 diff 文件，便可以将其用于修补原文件 *old_file*，从而升级为新文件 *new_file*：

```
patch < diff_file
```

以前面的测试文件为例：

```
[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
```

本例中，我们创建了一个叫做 *patchfile.txt* 的 diff 文件，然后使用 patch 命令进行修补。请注意该命令行中，并没有给 patch 命令指定目标文件，因为 diff 文件（统一形式）已经在页眉中包含了文件名。可以发现，进行修补后，*file1.txt* 便与 *file2.txt* 一致了。

patch 有很多参数选项，而且有很多额外的工具命令可以分析并编辑这些补丁文件。

20.5 非交互式文本编辑

之前所描述的文本编辑大多都是交互式的，也就是说只要手动移动鼠标然后输入需要进行的改变，然而，也可以用非交互的方式进行文本编辑。例如，可以只用一个简单的命令一次性更改多个文件。

20.5.1 tr——替换或删除字符

tr 是替换字符命令，可以将其看作一种基于字符的查找和替换操作。所谓替换，实际是指将字符从一个字母更换为其他字母。例如，将小写字母转变为

大写字母。如下便是一个利用 tr 进行大小写字母替换的例子。

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

该输出结果表明，tr 可对标准输入进行操作并且将结果以标准形式输出。tr 有两个参数：等待转换的字符集和与之相对应的替换字符集。字符集的表示方法可以是下面三种方式中的任一种。

- 枚举列表；例如，ABCDEFIGHJKLMNOPQRSTUVWXYZ。
- 字符范围；例如，A-Z。请注意，这种方法有时会与其他命令一样受限于同一个问题（由于不同系统的排序顺序），因此使用时要小心。
- POSIX 字符类；例如，[:upper:]。

多数情况下，这两个字符集应该是同等长度；然而，第一个字符集比第二个字符集长也是可能的，如下便是一个将多个字符替换为单个字符的例子。

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

除了替换，tr 还可以直接从输入流中删除字符。本章前篇，讨论了将 MS-DOS 类型的文本文件向 UNIX 类型转换的问题。要进行这样的转换，需要移除每行末尾的回车符，如此便可以用 tr 命令解决，如下所示。

```
tr -d '\r' < dos_file > unix_file
```

这里的 dos_file 是待转换的文件，而 UNIX_file 则是转换的结果文件。此命令形式使用了转义字符\r 代替回车符。如若想了解 tr 支持的所有转义符号和字符类，请查看 tr 的帮助手册。

```
[me@linuxbox ~]$ tr --help
```

ROT13：保密性不强的编码环

tr 的一个有趣用法就是可以进行 ROT13 文本编码。ROT13 是一种基于简单替换算法的加密方式。ROT13 这个名字有点抽象，“文本模糊”应该更具体一点。在文本中应用该加密方式，有时是为了隐藏潜在的攻击性内容。该方法仅仅是将每个字母在字母表中向上移动了 13 位，由于正好移动了字母表中字母总数 26 的一半，所以再执行一次此算法文本便可以恢复原样。示例如下，使用 tr 进行编码。

```
echo "secret text" | tr a-zA-Z n-za-mN-zA-M
Trperg grkg
```

再进行一次同样的步骤即得到了译码结果：

```
echo "frperg grkg" | tr a-zA-Z n-za-mN-ZA-M
secret text
```

许多 email 程序以及 Usenet 新闻阅读器都支持 ROT13 编码。维基百科上有一篇关于该主题的很好的文章，大家有兴趣的可以看一看，网址为：<http://en.wikipedia.org/wiki/ROT13>。

tr 还有另外一个奇妙的用法。使用-s 选项，tr 可以“挤兑”（删除）重复出现的字符，示例如下。

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

本例字符串中含有重复字符，通过给 tr 指定 ab 字符集，便消除了该字符集中重复的“a”和“b”字母，而字母 c 并没有改变，因为 tr 设定的字符集中并没有包含 c。请注意重复的字符必须是毗邻的，否则该挤兑操作将不起作用。

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

20.5.2 sed——用于文本过滤和转换的流编辑器

sed 是 *stream editor*（流式编辑器）的缩写，它可以对文本流、指定文件集或标准输入进行文本编辑。sed 功能非常强大，并且从某种程度上来说，它还是一个比较复杂的命令（有整本书的内容对它进行了讲解），所以本书无法涉及其所有用法。

sed 的用法，总得来说，首先给定 sed 某个简单的编辑命令（在文本行中）或是包含多个命令的脚本文件名，然后 sed 便对文本流的内容执行给定的编辑命令。下面是一个非常简单 sed 应用实例。

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

该例先利用 echo 生成了只包含一个单词的文本流，然后将该文本流交给 sed 处理，而 sed 则对文本流执行 s/front/back/ 指令，最后输出“back”作为运行结果。所以可以认为，本例中的 sed 与 vi 中的替代（查找与替换）命令相似。

sed 中的命令总是以单个字母开头。上例中，替换命令便由字母 s 代替，其后紧跟替换字符，替换字符由作为分界符的斜线字符分开。分界符的选择是随

意的，习惯上一般使用斜线，但是 sed 支持任意字符作为分界符，sed 会默认紧跟在 sed 的命令之后的字符为分界符。下面的命令行能得到相同的效果。

```
[me@linuxbox ~]$ echo "front" | sed 's_front_back_'
back
```

由于下划线是紧跟在命令字符“s”之后的，所以它便是分界符。由此可见，这种自动设定分界符的能力增强了命令行的可读性。

sed 中的多数命令允许在其前添加一个地址，该地址用来指定输入流的哪一行被编辑。如果该地址省略了，便会默认对输入流的每一行执行该编辑命令。最简单的地址形式就是一个行号。例如，在上例中增加一个“1”，如下所示。

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

增加的“1”表示此替换操作只对输入流的第一行起作用。当然也可以指定其他行号，如下所示。

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

结果显示此替换操作并未执行，这是因为该输入流中并没有第二行。地址可以有多种方式表达，表 20-7 列出了最常用的几个。

表 20-7 sed 的地址表达法

| 地址 | 功能说明 |
|-------------|---|
| N | n 是正整数表示行号 |
| \$ | 最后一行 |
| /regexp/ | 用 POSIX 基本正则表达式描述的行。请注意，这里的正则表达式用的是斜线作为分界符，当然，也可以自己选择分界符，只要用\cregexp{c} 选项指定即可，这里的 c 就是用于取代斜杠的分界符 |
| addr1,addr2 | 行范围，表示从 addr1 至 addr2 的所有行。地址可以是上面所述形式的任何一种 |
| first~step | 代表行号从 first 行开始，以 step 为间隔的所有行。例如，1~2 是指所有奇数行，而 5~5 是指第 5 行和以及随后的所有是 5 的倍数的行 |
| addr1,+n | addr1 行及其之后的 n 行 |
| addr! | 除了 addr 行之外的所有行，addr 可以用上面的任何一种形式表达 |

接下来，我们会用本章前面所使用的 *distros.txt* 文件演示多种不同形式的地址表达。首先，用行号范围的表达方式如下所示。

```
[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 |

此例显示了 *distros.txt* 文件中第 1 行到第 5 行的内容，利用了 p 命令输出指定匹配行的内容，从而完成上述操作。然而，要想得到正确结果，就必须添加选项-n（不会自动打印选项）以防 sed 会默认输出每一行的内容。

下面尝试使用正则表达式。

| |
|--|
| [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 |

此处用的是以斜杠隔开的正则表达式/SUSE/，查找包含 SUSE 字符串的文本行，该用法与 grep 的用法类似。

最后，尝试在地址前添加表示否定意义的感叹号（!），用法如下。

| |
|---|
| [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 |

这样我们得到了理想输出结果，即除了那些与正则表达式匹配的行，其他所有行都显示出来了。

到目前为止，我们已经介绍了 sed 的两个编辑命令——s 和 p，表 20-8 是一张更完整的基本编辑指令表。

表 20-8 sed 基本编辑指令

| 命令 | 功能描述 |
|----|-----------|
| = | 输出当前行号 |
| a | 在当前行后附加文本 |
| d | 删除当前行 |
| i | 在当前行前输入文本 |

续表

| 命令 | 功能描述 |
|-----------------------|--|
| p | 打印当前行。默认情况下，sed 会输出每一行并且只编辑文件内那些匹配指定地址的行。当指定-n 选项时，默认操作会被覆盖 |
| q | 退出 sed 不再处理其他行。如果没有指定-n 选项，就会输出当前行 |
| Q | 直接退出 sed 不再处理行 |
| s/regexp/replacement/ | 将 regexp 的内容替换为 replacement 代表的内容。replacement 可能会包含特殊字符&，它代表的其实就是 regexp 所表示内容。除此之外，replacement 也可能包含\1 到\9 的序列，它们代表的是 regexp 中相应位置的描述内容。学习接下来关于回参考的讨论可以了解更多这方面的知识。跟在 replacement 后面的反斜杠，可以指定一个可选择的标志以修改 s 命令的行为 |
| y/set1/set2 | 将字符集 set1 转换为字符集 set2。请注意，与 tr 不同，sed 要求这两个字符集等长 |

s 命令是目前为止使用最普遍的编辑命令。接下来，我们通过编辑 *distros.txt* 文件来演示其强大功能的一小部分。之前已经讨论过 *distros.txt* 文件中的时间字段并不是以“计算机友好”的形式存储，因为此时间形式是 MM/DD/YYYY，YYYY-MM-DD 这样的形式会方便很多（更容易排序）。但如果手动更改文件，不仅浪费时间而且容易出错，而 sed 可以一步完成这样的操作。

```
[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
```

哇塞！这个命令行看起来还真复杂，但它确实有效。只需一步，就改变了文件中的日期形式。此例充分解释了为什么正则表达式有时被开玩笑的称为“只写”介质。因为我们可以编写正则表达式，但有时却读不懂它们。在被该命令吓跑之前，这我们还是先来分析它的各组成部分的含义。首先，sed 命令有其基本结构，如下所示。

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

接下来我们需要理解将日期分隔开来的正则表达式。由于它是以 MM/DD/YYYY 的形式存在，并且出现在行末尾，所以使用如下的表达式。

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

该表达式的匹配格式：两位数字、斜杠、两位数字、斜杠、4位数字以及行尾标志。所以这代表了 *regexp* 表达式的形式，但是怎么处理 *replacement* 的表达式？解决这一问题，我们必须引进正则表达式的一个新特性，该特性一般存在于那些使用 BRE 的应用中。此特性称为回参考，并且工作方式类似于此，即如果 *replacement* 中出现了 \n 转义字符，并且这里的 n 是 1-9 之间的任意数字，那么此转义字符就是指前面正则表达式中与之相对应的子表达式。如何创建该表达式，可以简单地将其括于括号中，如下所示。

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

现在，我们便有了三个子表达式。第一个的内容是月份，第二个内容是具体的日，第三个内容则是指年份。于是便可用如下命令行构建替换字符串。

```
\3-\1-\2
```

该表达式表示的顺序如下：年份、斜杠、月份、斜杠和具体的日。

于是，整个命令行如下。

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

但是仍有两个遗留问题：第一，当 sed 试图编译 s 命令时，正则表达式中多余的斜杠会令 sed 混淆；第二，sed 默认情况下只接受基本正则表达式，所以正则表达式中的部分元字符会被当成文字字符。我们可以利用反斜杠来避免这些冒犯字符，从而一次性解决这两个问题。

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

这样便大功告成了！

s 命令的另外一个特点，就是替换字符串后面可以紧跟可选择标志符。其中最重要的标志符就是 g，该标志告诉 sed 对每行的所有匹配项进行替换操作，而不是默认的只替换第一个匹配项。

示例如下。

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

我们可以看到执行了替换操作，但只是对第一个字母 b 有效，剩下的字母并没有改变。通过增加 g 标志符，便可以对所有的 b 进行替换操作。

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

到目前为止，我们只使用了命令行方式向 sed 传送操作命令，其实也可以用-f 选项建立更复杂的命令脚本文件。作为演示实例，我们运用 *distros.txt* 文件结合 sed 创建一个报告。该报告的构成有顶端标题、修改时间和大写字母组成的所有发行版本名。进行这些操作之前，我们需要编写一个脚本文件，所以启动文本编辑器并输入如下内容。

```
# 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/ABCDEFGHIJKLMNPQRSTUVWXYZ/
```

将此 sed 脚本保存为 *distros.sed*，并且照如下方式运行。

```
[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 |
| UBUNTU | 6.06 | 2006-06-01 |
| UBUNTU | 8.10 | 2008-10-30 |
| FEDORA | 5 | 2006-03-20 |

正如读者所看到的，输出显示了理想结果，但是它到底是如何工作的呢？让我们再次查看脚本文件，并使用 cat 将行号都标注出来。

```
[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/ABCDEFGHIJKLMNPQRSTUVWXYZ/
```

第一行只是一个声明。与 Linux 系统中的许多配置文件和编程语言一样，声明一般都是以“#”符号开头。剩下的则是一些人类可理解的文本。声明可以放于脚本文件的任何位置（只要不在命令行中），并且对于任何一个需要验证或是维护该脚本的人都很有用。

第二行是空白行。与声明一样，空白行也是为了增加可读性。

多数 sed 命令都支持行地址，这些行地址用于指定哪些输入行执行指定操作。行地址可以用简单的行号来描述，也可以用行号范围以及“\$”表示，“\$”是一个表示文本最后一行的特殊符号。

第 3~6 行包含的则是要插入文本中第一行的内容。i 命令后面紧跟转义回车符，转义回车符由反斜杠和回车符组成，亦称为行继续符。此种先反斜杠后回车符的顺序，可以用于包括 shell 脚本在内的许多场合，可确保文本流中嵌入回车符但不会告诉编译器（在 sed 这个例子中）已经到了行末尾。i 命令、a 命令（追加文本）以及 c 命令（替换文本）能作用于多个文本行，只要除了最后一行的每行都以行继续符结尾。该脚本文件的第 6 行确实是所输入文本的末尾行，并且标志 i 命令结尾的符号是一个简单的回车符而不再是行继续符。

注意

行继续符是由反斜杠后紧跟回车符组成，两者之间不容许有任何空格。

第 7 行是查找和替换命令。由于该命令前并未指定地址，所以该命令将对输入流的每一行进行操作。

第 8 行则实现了小写字母向大写字母的转变。注意，与 tr 不同，sed 中的 y 命令并不支持字符范围（例如[a-z]），也不支持 POSIX 字符类。同样，由于 y 命令前并没有指定地址，所以将对输入流的每一行执行操作。

偏爱 sed 的人同样会喜欢……

sed 是一个功能非常强大的程序，它可以对文本流执行非常复杂的编辑任务。它通常用于简单的单命令行任务而不是长脚本。对于更大些的任务，许多用户则偏向于其他工具。其中最受欢迎的有 awk 和 perl。这两个命令已经超出了本书所谈论的命令范畴，并且延伸到了完整的编程语言领域。尤其是 perl，通常取代 shell 脚本用于执行系统管理和管理任务，同时也是一个非常受欢迎的 web 开发介质。而 awk 则更专业化，数据处理是其强项。它与 sed

的相似之处就是 awk 命令通常也是逐行处理文本文件，并且使用方法也继承了 sed 的地址后面加上执行操作这一个概念。虽然 awk 和 perl 都不在本书的讨论范围中，但对于 Linux 命令行用户来说它们仍然是非常有用的工具。

20.5.3 aspell——交互式拼写检查工具

最后一个所要讨论的文本工具就是 aspell，它是交互式的拼写检查工具。aspell 命令继承的是早期的 ispell 命令，并且多数情况下，可以直接取代 ispell。虽然 aspell 命令通常为那些需要进行拼写检查的程序所用，但它同样可以作为一个独立于命令行的工具发挥其效用。它可以智能地检查不同类型文本文件的错误，包括 HTML 文件、C/C++ 程序、email 消息以及其他专业的文本文件。

检查一篇简单散文的拼写错误，可以用如下方式使用 aspell。

```
aspell check testfile
```

此处的 testfile 是要进行检查的文件名。作为实例进行讲解，下面创建了一个简单的 foo.txt 文本文件，它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.
```

接下来使用 aspell 检查文件中的拼写错误。

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

由于 aspell 在检验模式下是与用户交互的，所以我们会看到如下的显示界面。

```
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 |

?

显示内容的顶部，被怀疑错误的字符是以高亮的形式显示的。中间部分，有 10 个标号从 0~9 的替换拼写建议以及其他可能的动作选项。最后，末端有一个提示框供用户进行操作选择。

假定我们输入 1，则 aspell 会将错误的单词用 jumped 取代并且继续处理下一个错误单词 laxy。如果选择替代单词 lazy，aspell 便执行此替换操作然后终止程序。aspell 命令检查结束后，可以再次查看文件，会发现那些拼写错误的单词已经改正过来，如下所示。

```
[me@linuxbox ~]$ cat foo.txt
The quick brown fox jumped over the lazy dog.
```

除非额外指定了命令行选项--dont-backup，不然 aspell 将会创建一个包含原文本内容的备份文件，此备份文件文件名则由原文件名加上后缀.bak 组成。

sed 其实有更强大的编辑功能，恢复 *foo.txt* 文件中原有的拼写错误以便再次利用：

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

sed 选项-i 告诉 sed “原地” 编辑文件，这表示 sed 不会将编辑结果送至标准输出，而是将改变后的文本重新写入文件中。同样可以看出，一个命令行中可以输入多个编辑命令，只要用分号将它们隔开即可。

接下来就来讨论一下 aspell 如何处理不同类型的文件。使用诸如 vim 的文本编辑器（可以挑战性地尝试使用 sed），给文件增加一些 HTML 语言，如下所示。

```
<html>
  <head>
    <title>Misplaced HTML file</title>
  </head>
  <body>
    <p>The quick brown fox jimped over the laxy dog.</p>
  </body>
</html>
```

现在，如果试图检查修改后该文件的拼写错误，便会遇到一个问题。照如下方式输入命令行。

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

便会得到如下输出结果。

```
<html>
  <head>
    <title>Misplaced HTML file</title>
  </head>
  <body>
    <p>The quick brown fox jimped over the laxy dog.</p>
  </body>
</html>
```

```

2) ht ml
3) ht-ml
i) Ignore
r) Replace
a) Add
b) Abort
5) Hamil
6) hotel
I) Ignore all
R) Replace all
l) Add Lower
x) Exit

```

?

aspell 会认为 HTML 标签的所有内容是拼写错误。该问题可以通过增加-H (HTML) 模式选项克服，示例如下。

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

可以得到如下结果：

```

<html>
  <head>
    <title>Misplaced HTML file</title>
  </head>
  <body>
    <p>The quick brown fox jimped over the laxy dog.</p>
  </body>
</html>

```

```

1) Mi spelled
2) Mi-spelled
3) Misspelled
4) Dispelled
5) Spelled
i) Ignore
r) Replace
a) Add
b) Abort
6) Misapplied
7) Miscalled
8) Respelled
9) Misspell
0) Misled
I) Ignore all
R) Replace all
l) Add Lower
x) Exit

```

?

使用-H 选项后，HTML 语言部分就被忽略了，只有那些非 HTML 标签部分才会被检查。这种模式下，HTML 标签内容被忽略了并且不会进行拼写检查。然而，Alt 标签的内容，在这种模式下则是要检查的。

注意

默认情况下，aspell 会忽略文本中的 URL 和 email 地址，该行为可以通过设置命令行选项覆盖。同样也可以指定哪些标签内容需要被检查，而哪些可跳过检查。具体内容，可以查看 aspell 的 man 手册页。

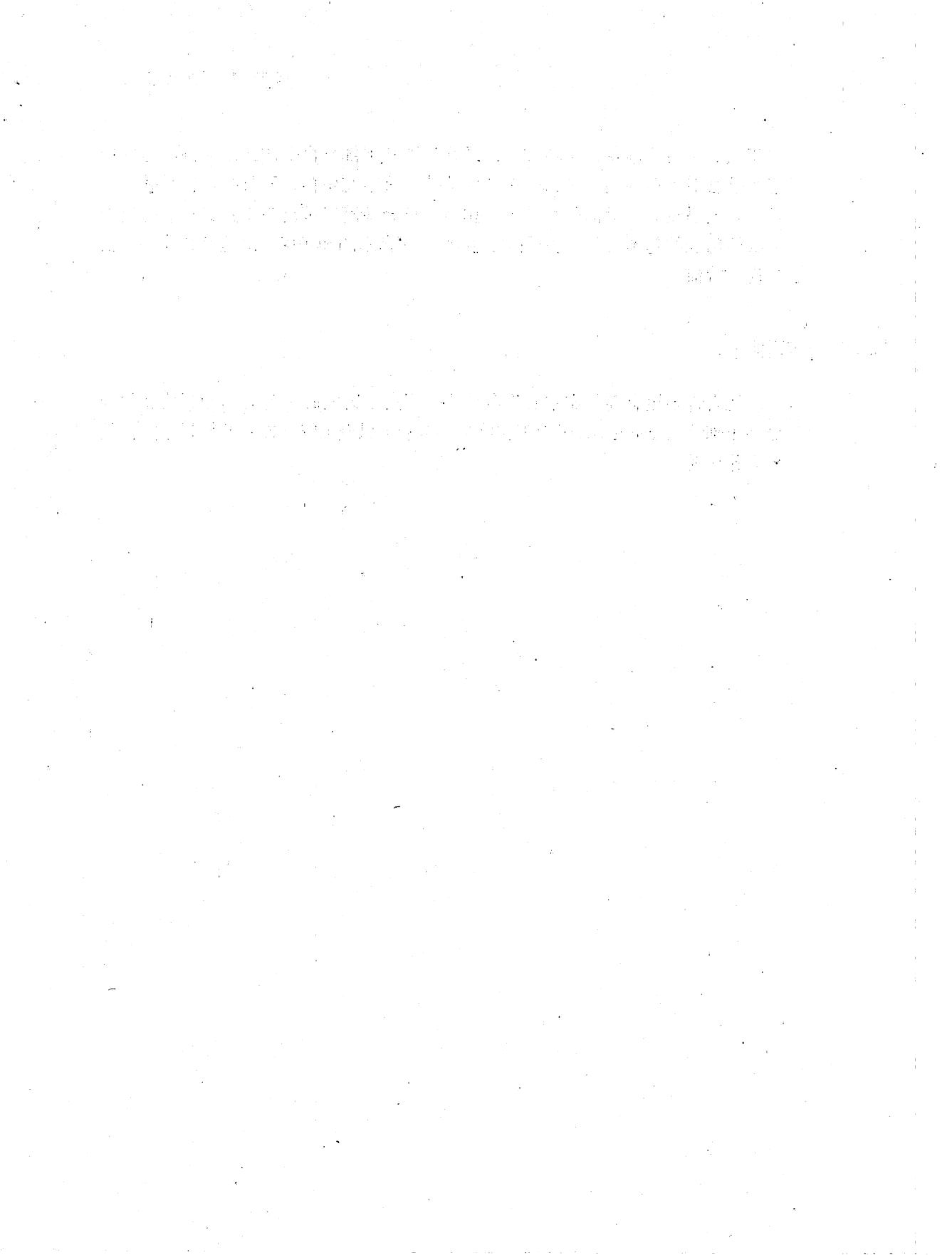
20.6 本章结尾语

本章主要介绍了一些用于文本编辑的命令行工具，下一章会介绍更多的这

类工具。不得不承认，虽然文中已经列举了其中部分命令的应用实例，但你仍然可能会对如何使用以及为什么使用这些工具存在疑问，而且答案似乎并不是那么显而易见。在后面的章节中，相信大家会逐渐发现这些工具其实是解决一组实际问题的基本工具，在后面接触到 shell 脚本的知识时，这些工具才会真正体现其价值。

20.7 附加项

还有许多更有趣的文本操作命令值得探索。这些命令包括 `split`（将文件分成多个部分）、`csplit`（基于上下文将文件分块）以及 `sdiff`（左右并排显示文件差异并作比较）。



第 21 章

格式化输出

本章继续讨论与文本相关的工具，重点讲一些用于格式化文本输出而非改变文本自身内容的命令。这些命令通常用于文本的打印，而“打印”这一主题将在下一章介绍。本章将要讨论的命令如下所示。

- `nl`: 对行进行标号。
- `fold`: 设定文本行长度。
- `fmt`: 简单的文本格式化工具。
- `pr`: 格式化打印文本。
- `printf`: 格式化并打印数据。
- `groff`: 文档格式化系统。

21.1 简单的格式化工具

首先让我们看一些简单的格式化工具，它们多数都是“单目的”程序，而且一般执行一些不复杂的操作，它们一般用于一些小的任务，并作为管道传输和脚本的一部分。

21.1.1 nl——对行进行标号

nl 命令是一个非常神秘的工具，用于完成一个非常简单的任务：对行进行编号。就其最简单用法，与 **cat -n** 很相似。

```
[me@linuxbox ~]$ nl distros.txt | head
 1 SUSE      10.2   12/07/2006
 2 Fedora    10      11/25/2008
 3 SUSE      11.0   06/19/2008
 4 Ubuntu    8.04   04/24/2008
 5 Fedora    8      11/08/2007
 6 SUSE      10.3   10/04/2007
 7 Ubuntu    6.10   10/26/2006
 8 Fedora    7      05/31/2007
 9 Ubuntu    7.10   10/18/2007
10 Ubuntu    7.04   04/19/2007
```

和 **cat** 命令一样，**nl** 既支持多个文件名作为命令行参数，也支持标准输入。然而，**nl** 可以进行多种复杂的编号，因为它有许多参数选项，且支持原始形式的标记。

nl 进行标号时支持一个叫做逻辑页的概念，所以它可以重置（重新开始）数值序列。通过合理运用参数选项，**nl** 可以设置起始编号为特定的值，并在有限的范围内设置其格式。逻辑页可以进一步分解为逻辑页标题、正文和页脚。在每一个部分中，行号都可以重置并/或分配不同的风格。如果 **nl** 的输入参数是多个文件，那么 **nl** 会把它们当作一个文本流整体。文本流中的每一个部分都由一些看起来非常奇怪的标记来区别，表 21-1 列出了部分标记。

表 21-1 **nl** 标记

| 标记 | 含义 |
|--------|---------|
| \:\:\: | 逻辑页页眉开头 |
| \:\: | 逻辑页正文开头 |
| \: | 逻辑页页脚开头 |

表 21-1 中的每一个标记元素在一行中只允许出现一次。每次处理完一个标

记元素后，`nl`便将其从文本流中删除。

表 21-2 列出了 nl 的常用选项。

表 21-2 常用的 nl 选项

| 选项 | 含义 |
|-----------|---|
| -b style | 按照 style 格式对正文进行编号，这里的 style 是下面类型中的一个 <ul style="list-style-type: none">• a 对每行编号• t 仅仅对非空白行编号，此选项是默认的• n 不对任何行进行编号• pregexp 只对与基本正则表达式匹配的行进行编号 |
| -f style | 以 style 的格式对页脚进行编号。默认选项是 n (无) |
| -h style | 以 style 的格式对标题进行编号。默认选项是 n (无) |
| -i number | 设置页编号的步进值为 number。默认值为 1 |
| -n format | 设置编号格式为 format，此处的 format 可以是如下表示中的一种 <ul style="list-style-type: none">• ln 左对齐，无缩进• rn 右对齐，无缩进。这是默认选项• rz 右对齐，有缩进 |
| -p | 在每个逻辑页的开始不再进行页编码重置 |
| -s string | 在每行行号后面增加 string 作为分隔符。默认的情况下是一个简单的 tab 制表符 |
| -v number | 将每个逻辑页的第一个行号设为 number。默认是 1 |
| -w width | 设置行号字段的宽度为 width。默认值是 6 |

不得不承认，用户可能并不会那么频繁地进行行编号，但是，用户可以利用 `nl` 结合其他工具进行更复杂的任务。基于上一章的基础，我们生成一个 Linux 发行版本的报告。由于我们会使用 `nl`，报告中最好包含标题/正文/页脚等标记。我们可以用上章提到的 `sed` 脚本添加这些标记，用文本编辑器对 `sed` 脚本文件做如下改动，并将其存于 `distros-nl.sed` 文件中。

该脚本文件完成了向原报告中插入 nl 逻辑页标记以及在报告末尾添加页脚内容的任务。请注意，输入标记时必须使用双反斜杠，否则 sed 会将它们解释为转义字符。

接下来，我们便可以结合 sort、sed 和 nl 创建一个增强版的报告。

```
[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 |
| 16 Ubuntu | 8.10 | 2008-10-30 |

End Of Report

多个命令进行管道传输得到了需求结果。首先，我们根据发行版名称和发行时间（字段 1 和字段 2）进行排序；然后用 sed 处理排序结果，添加了报告的页眉（包括 nl 的逻辑页标记）和页脚；最后，我们执行了 nl 命令。nl 在默认情况下，只会对属于逻辑页正文部分的文本流进行行编号。

我们可以重复执行 nl，并尝试不同的参数选项。如下所示列举了一些有趣的参数。

nl -n rz

以及

nl -w 3 -s ' '

21.1.2 fold——将文本中的行长度设定为指定长度

fold 是一个将文本行以指定长度分解的操作。与其他命令类似，fold 支持一

个或多个文本文件或是标准输入作为输入参数。向 fold 输入一个简单的文本流，便可了解其工作方式。

```
[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.
```

这样，我们便能明白 fold 到底完成了什么操作。echo 命令输出的文本被指定了-w 选项的 fold 分解成了片段。本例中，指定了行的宽度为 12 个字符。如果没有指定行宽，则默认是 80 个字符宽。请注意，fold 在断行时并不会考虑单词边界。而增加-s 选项，可使 fold 在到达 width 字符数前的最后一个有效空格处将原文本行断开，示例如下。

```
[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.
```

21.1.3 fmt——简单的文本格式化工具

fmt 命令同样会折叠文本，另外还包括更多其他功能。它既可以处理文件也可以处理标准输入，并对文本流进行段落格式化。就其基本功能而言，它可以在保留空白行和缩进的同时对文本行进行填充和连接。

作为演示的文本内容，不如从 fmt 的帮助手册页中复制一些内容吧。

```
'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.
```

将这段文字复制到文本编辑器中，并将其存为 *fmt-info.txt* 文本文件。现在，

假定我们需要重新格式化该文本，以满足每行 50 个字符宽的规则。那么我们可以输入 fmt 结合-w 选项完成这样的格式化。

```
[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.
```

这个输出结果还真是奇怪。也许，实际上我们应该认真地阅读一遍下面的文字，因为它解释了事情发生的原委。

默认情况下，空白行、单词之间的空格和缩进都保留在输出结果中；不同缩进量的连续输入行并不进行拼接；制表符会在输入中扩展并直接输出。

所以，fmt 保留了第一行的缩进。幸运的是，fmt 提供了一个参数选项以修正这一问题。

```
[me@linuxbox ~]$ fmt -cw 50 fmt-info.txt
`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.
```

这样一来，输出结果看起来顺眼了很多。由此可见，通过增加-c 选项，我

们便得到了理想输出结果。

`fmt` 有一些有趣的选项，见表 21-3。

表 21-3 `fmt` 选项

| 选项 | 功能描述 |
|------------------------|---|
| <code>-c</code> | 在“冠边缘”模式下运行。此模式保留段落前两行的缩进，随后的行都与第二行的缩进对齐 |
| <code>-p string</code> | 只格式化以前缀字符串 <i>string</i> 开头的行。格式化后， <i>string</i> 的内容仍然会作为每一个格式化行的前缀。该选项可以用于格式化内容是源代码的文本，例如，任何以“#”号作为声明开头的编程语言或是配置文件都可以通过指定 <code>-p '#'</code> 选项进行格式化，指定该选项后可保证只格式化声明中的内容。具体的可见后续例子 |
| <code>-s</code> | “仅截断行”模式。在此模式下，将会只根据指定的列宽截断行。而短行并不会与其他行结合。此模式适用于格式化文本但不需要行拼接的场合，比如代码等 |
| <code>-u</code> | 字符间隔统一。采取传统的“打字机风格”模式格式化文本，这意味着字符之间间隔一个空格字符，句子之间间隔两个空格字符。该模式对于删除齐行非常有用，所谓齐行就是指文本行被强迫与左右边缘对齐 |
| <code>-w width</code> | 格式化文本使每行文本不超过 <i>width</i> 个字符，默认值是 75。请注意， <code>fmt</code> 格式化文本时往往由于要保持行平衡而使得行实际宽度比指定宽度略小 |

`-p` 选项尤为有趣，通过它，我们可以选择性地格式化文件内容，前提是要是要格式化的文本行都以相同的字符序列开头。许多编程语言都使用 hash 符号 (#) 作为注释的开始，因此可以用此选项只格式化注释文本。下面创建一个有注释的类似于程序的文本文件。

```
[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.
```

我们的示例文本包含了以“#”（“#”号后紧跟一个空格）字符串开头的注释以及“代码”行（虽然并不是真正意义上的代码）。现在，我们使用 `fmt` 格式化该注释内容而不改动代码。

```
[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.
```

请注意，毗邻的注释行已经拼接起来，但是保留了空白行以及不是以指定前缀开头的文本行。

21.1.4 pr——格式化打印文本

Pr 命令用于给文本标页码。打印文本时，通常希望将输出内容分成几页，并且每页的顶部和底部都留出几行空白行，这些空白行可以用于插入页眉和页脚。

示例如下，该命令行将 *distros.txt* 文件格式化为一系列非常短的页（只显示了前两页）。

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

| | | |
|------------------|-------------|--------|
| 2012-12-11 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 |

| | | |
|------------------|-------------|--------|
| 2012-12-11 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 |

上例中，结合了-l 选项（页长）以及-w 选项（页宽）定义了一“页”内容包含 15 行，每行包含 65 个字符。**pr** 对 *distros.txt* 文件的内容进行分页，页与页之间则用几行空白行隔开，并且创建了一个包含文件修改时间、文件名以及页码的默认页眉。**pr** 命令有很多选项用于控制页面布局，详见第 22 章。

21.1.5 printf——格式化并打印数据

与本章中涉及的其他命令不一样，**printf** 命令并不适用于管道传输（也就是说它不支持标准输入），而且在命令行应用中它也不常见（多应用于脚本文件）。

那么它到底为什么会这么重要？因为它有如此广泛的应用范围。

`printf`（短语 `print formatted` 的缩写）起初是为 C 语言开发的，后来许多编程语言也都实现了这一功能，也包括 `shell` 环境。事实上，在 `bash` 中，`printf` 是内置的。

`printf` 的用法如下。

```
printf "format" arguments
```

该命令行给出了一个包含格式说明的字符串，然后将该格式应用于 `arguments` 所代表的输入内容，最后格式化结果送至标准输出。如下就是一个简单例子。

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

该格式化字符串可以包含文字文本（如“`I formatted the string`”）、转义字符（如`\n`，即换行符）以及以%开头的表示转换规格的字符序列。上例中，转换规格`%s`用于格式化字符串 `foo` 并将其结果输出。再看下面一个例子。

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

以上可以看出，`%s` 所代表的转换规格被字符串 `foo` 取代。`s` 表示格式化字符串数据，其他类型的数据则用其他指定字符表示。表 21-4 列出了常用的数据类型。

表 21-4 常用 `printf` 数据类型指定符

| 指定符 | 说明 |
|-----|---------------------------------------|
| D | 将一个数字格式化为有符号的十进制表示形式 |
| F | 格式化数字并以浮点数的格式输出 |
| O | 将一个整数格式化为八进制格式的整数 |
| s | 格式化字符串 |
| X | 将一个整数格式化为十六进制的数，并且在使用字母时，用小写字母 a~f 表示 |
| X | 与 X 类似，只是字母用大写字母表示 |
| % | 打印文字符号“%”（例如，指定“%%”） |

下例中，利用字符串“380”演示了每个转换说明符的效果：

```
[me@linuxbox ~]$ printf "%d, %f, %o, %s, %x, %X\n" 380 380 380 380 380 380
380, 380.000000, 574, 380, 17c, 17C
```

本例指定了 6 个转换说明符，并向 printf 提供了 6 个输入参数。得到了 6 种转换说明符的输出效果。

转换说明符也可以通过增加一些可选组件以对输出效果进行调整。一个完整的转换规格可能会包含如下内容。

`%[flags][width][.precision]conversion_specification`

当使用多个可选组件时，这些组件必须按照上面的顺序才能被正确编译。表 21-5 给出了每个组件的说明。

表 21-5 printf 转换规范的组成部分

| 组件 | 功能描述 |
|------------------|---|
| | 总共有 5 个不同的 flag |
| <i>flags</i> | <ul style="list-style-type: none"> # 使用替代格式输出。这取决于数据类型。对于 o (八进制数) 类转换，输出结果以 0 开头。对于 x 和 X 类转换，输出则分别以 0x 和 0X 开头 0 (零) 用 0 填充输出。这代表着字段前会填充 0，如 000380 - (破折号) 输出左对齐。默认情况下，printf 是右对齐输出的 (空格) 为正数产生一个前导空格 + (加号) 正数符号。默认情况下，printf 只会输出负数的符号 |
| <i>width</i> | 一个数字，该数字指定了最小字段宽度 |
| <i>precision</i> | 对于浮点数，便是指定小数点后的小数精确度。对于字符串转换，precision 则指定了输出字符的个数 |

表 21-6 列出了一些不同格式化实例。

表 21-6 print 转换规范实例

| 参数 | 格式 | 转换结果 | 说明 |
|---------------|-----------|---------------|--|
| 380 | “%d” | 380 | 对整数进行简单的格式化 |
| 380 | “%#x” | 0x17c | 使用替代格式标记将整数格式化为一个十六进制数 |
| 380 | “%05d” | 00380 | 将整数格式化为至少 5 个字符宽度的字段，不足位数可在前面填充 0 |
| 380 | “%05.5f” | 380.00000 | 将数字格式化为精确到小数点后 5 位的浮点数，不足位数用 0 填充。由于指定的最小字段宽度 (5) 要比格式化后实际的数值位数小，所以此处并没有进行填充 |
| 380 | “%010.5f” | 0380.00000 | 把最小字段宽度增加为 10，并且 0 填充可见 |
| 380 | “%+d” | +380 | +标记符表示此数是正数 |
| 380 | “%-d” | 380 | -标记表示左对齐 |
| abcdefghijklm | “%5s” | abcdefghijklm | 用最小的字段宽度来最小化字符串 |
| abcdefghijklm | “%.5s” | abcde | 根据字符串的精确位数截断字符串 |

同样，printf 通常用于脚本文件的表格数据格式化操作，而并不会直接应用于命令行。不过，我们仍然可以用其解决各种各样的格式化问题。首先，我们可以利用 printf 输出一些由制表符隔开的字段。

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

插入\t（Tab 的转义符），便获得了理想结果。接下来，利用其输出一些格式整齐的数字。

```
[me@linuxbox ~]$ printf "Line: %05d %15.3f Result: %+15d\n" 1071 3.14156295
32589
Line: 01071           3.142 Result:      +32589
```

此例充分显示了最小字段宽度对字段间距的影响。那么如何格式化一个小网页呢？

```
[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>" "Page Title" "Page Content"
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <p>Page Content</p>
  </body>
</html>
```

21.2 文档格式化系统

到目前为止，本章只介绍了一些简单的文本格式化工具。这些工具对于一些小的、简单的任务非常有用，那对于一些比较大的任务呢？系统为用户提供了多种编辑多类型文件的工具，特别是科学类和学术类的出版物，这也是 UNIX 成为一个在科学和技术领域比较受欢迎的操作系统的原因之一（还有一个原因是 UNIX 提供了强大的多任务、多用户的软件开发环境）。事实上，正如下面的 GNU 文档所描述的，文档编辑对 UNIX 的开发很重要。

第一个版本的 UNIX 系统是在贝尔实验室的 PDP-7 上开发的。1971 年，开发者们想要一台 PDP-11 以对操作系统进行进一步的研究。为了充分证明这一系统值得花费这么多资金，开发者们指出会为 AT&T 专利部门完成一种文件格式化系统。J.F. Ossanna 是该格式化程序首任作者，另外，此程序其实是 McIlroy 编写的 roff 程序的重新实现。

21.2.1 roff 和 TeX 家族

当今主宰文档格式化领域的主要有两大家族，它们是从原始 roff 程序延伸而来的 nroff 和 troff 和基于 Donald Knuth 的 TeX（发音为 ‘tek’）排版系统。自然，TeX 中间偏低的字母 “E” 不可忽略。

roff 这个名字来源于 “I'll run off a copy for you.” 中的短语 “*run off*”。nroff 程序格式化后的文档一般输出至使用等宽字体的设备，诸如字符终端、类打字机风格的打印机等。在该程序风靡期间，它几乎覆盖了所有与计算机相连的打印设备。后来的 troff 程序则用于格式化排字机等设备输出的文档，这些设备通常输出 “camera-ready” 格式的文本用于商业印刷。如今，多数电脑打印机都能模拟排字机的输出。roff 系列还包含了一些用于处理其他文件类型的程序，这些程序包括 eqn（针对数学等式）以及 tbl（针对表格）等。

TeX 系统（其稳定版本）在 1989 年首次成形，并且一定程度上，取代了 troff 而成为排字机输出文档的格式化工具。本书不讨论 TeX，不仅是因为其比较复杂（有好几本单独讲述 TeX 的书籍），还因为多数现代 Linux 系统并不会默认安装它。

注意

对于那些对安装 TEX 有兴趣的读者，可以学习 texlive 软件包以及 LyX 图形编辑器的相关知识，texlive 软件包在绝大多数发行版本库中都能找到。

21.2.2 groff——文档格式化系统

groff 其实是 GNU 实现方式的 troff 系列程序集，它还包含一个用于模拟 nroff 及其他 roff 家族系列的程序功能的脚本。

虽然 roff 及其衍生体都是用来创建格式化文件的，但是它们格式化的方式对于现代用户来说却非常陌生。如今多数文档都是用文字处理器编辑的，并且这些文字处理器可以一步完成文档的内容编辑和布局格式安排。图形化文字处理器出现之前，文档编辑通常包括两个步骤——使用文本编辑器编辑内容和使用诸如 troff 这样的程序进行格式化，格式化程序的指令都已经通过标记语言嵌入到文本中。现代的网页制作过程与此过程相似，因为网页也是先通过某种文本编辑器编辑，然后使用由网络浏览器提供的 HTML 标记语言来描述最后的网页布局的。

当然，本章并不打算全面讨论 groff，因为它的许多标记语言元素都与一些非常诡秘的排版细节相关。相反，大家还是应该把精力集中到仍然广泛应用的“宏包”上。这些“宏包”将许多低级命令压缩成一个很小的高级命令集，从而

使得 groff 使用起来容易得多。

耽误一点时间，大家先来看一看下面这个不起眼的手册页，它是`/usr/share/man` 目录下的一个 gzip 压缩的文本文件。查看其解压缩后内容，可以用下面的命令行（`ls` 的 man 手册页在第一部分有说明）。

```
[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
[\"OPTION\fR]... [\"FILE\fR]...
.SH DESCRIPTION
.\" Add any additional description here
.PP
```

与其正常的 man 手册页进行比较分析，就可以发现标记语言及其输出结果之间的相关性。

| | |
|---------------------------------|---------------|
| [me@linuxbox ~]\$ man ls head | |
| LS(1) | User Commands |

| | | |
|------------------------------|--|-------|
| NAME | | LS(1) |
| ls - list directory contents | | |

| | | |
|--------------------------|--|--|
| SYNOPSIS | | |
| ls [OPTION]... [FILE]... | | |

这真的很有趣，因为此 man 手册页是由 groff 使用 mandoc 的“宏包”进行浏览的。事实上，我们还可以用如下管道传输模拟 man 命令。

| | |
|---|---------------|
| [me@linuxbox ~]\$ zcat /usr/share/man/man1/ls.1.gz groff -mandoc -T ascii | |
| head | |
| LS(1) | User Commands |

| | | |
|------------------------------|--|-------|
| NAME | | LS(1) |
| ls - list directory contents | | |

| | | |
|--------------------------|--|--|
| SYNOPSIS | | |
| ls [OPTION]... [FILE]... | | |

此处，我们利用了 groff 程序，并设置参数指定为 mandoc “宏包”以及 ASCII 的输出格式。groff 可以输出多种不同格式的结果，如果未指定输出格式，那么 PostScript 便是其默认输出格式。

| | |
|---|--|
| [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 2 13:44:37 2012
%%DocumentNeededResources: font Times-Roman

%%%+ font Times-Bold
%%%+ font Times-Italic
%%DocumentSuppliedResources: procset grops 1.18 1
%%Pages: 4
%%PageOrder: Ascend
%%Orientation: Portrait
```

PostScript 是一种页面描述语言，一般用于描述送至类排字机设备打印的页面内容。我们可以提取 groff 命令的输出结果并将其存于文件中（假设使用的是备有 Desktop 目录的图形化桌面系统）。

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

桌面上应该会出现一个表示输出文件的图标，双击该图标启动页面浏览器，浏览器便以其所默认的形式（见图 21-1）显示该文件内容。

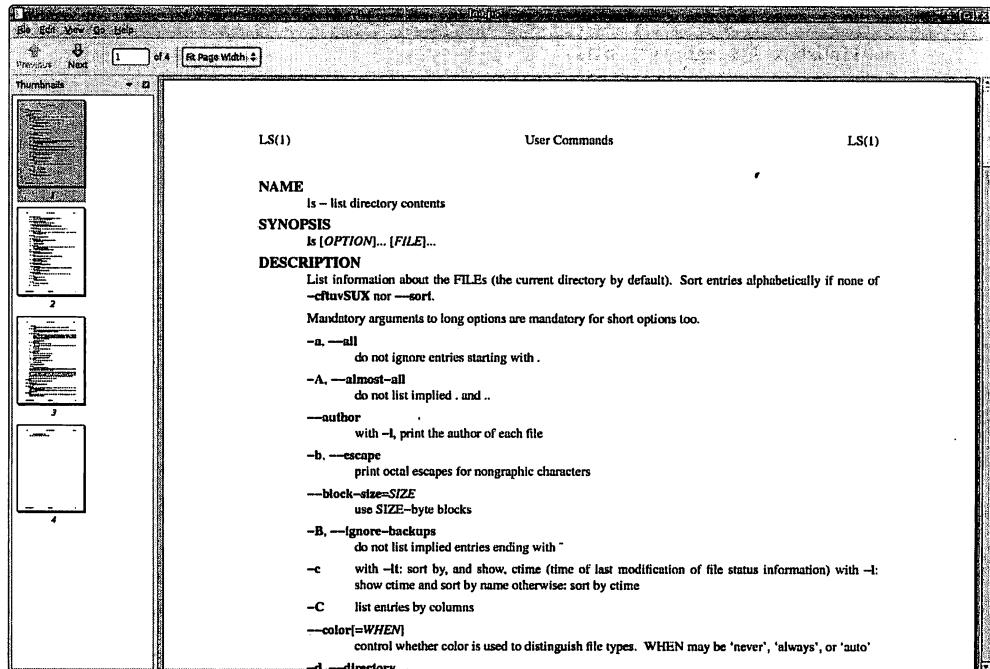


图 21-1 用 GNOME 中的页面浏览器查看 PostScript 格式的输出内容

我们得到的是一个排版整齐的 ls 的 man 手册页！事实上，我们可以用下面的 ps2pdf 命令将 PostScript 格式的文件转化为 PDF (*Portable Document Format*, 可移植文档格式) 文件，示例如下。