

第 9 章

权 限

传统的 UNIX 操作系统与那些传统的 MS-DOS 操作系统不同，区别在于它们不仅是多重任务处理（multitasking）系统，而且还是多用户（multiuser）系统。

确切地说，这意味着什么呢？这意味着同一时间内可以有多个用户使用同一台计算机。虽然一台标准的计算机可能只包含一个键盘和一台显示器，但是它仍然可以同时被一个以上的用户使用。例如，如果计算机连接到一个网络或者互联网中，远程用户可以通过 ssh（安全 shell）登录并且操作这台计算机。事实上，远程用户可以执行图形化应用程序，而且图形化的输出结果将会出现在远程显示器上。X 窗口系统把这个作为基本设计理念的一部分，并支持这种功能。

Linux 的多用户功能并不是最近的“创新”，而是深嵌在操作系统设计理念中的一个特色功能。想想 UNIX 系统诞生时的背景环境，该功能的出现有着重大的意义。很多年前，在计算机“个人化”之前，计算机普遍体积大，价格昂

贵，而且都是集中控制的。例如，一个典型的校园计算机系统，是由一台放置在某建筑物中的大型中央计算机以及遍布校园的各台终端机组成的，每台终端机都连接到中央计算机上。这台中央计算机可以同时支持很多用户。

为了保证多用户功能实际可用，系统特别设计了一种方案来保护当前用户不受其他用户操作的影响。毕竟，一个用户的操作不能导致计算机崩溃，一个用户的操作界面也不能显示属于另一个用户的文件。

本章将介绍系统安全的基础知识以及如下命令的使用。

- **id:** 显示用户身份标识。
- **chmod:** 更改文件的模式。
- **umask:** 设置文件的默认权限。
- **su:** 以另一个用户的身份运行 shell。
- **sudo:** 以另一个用户的身份来执行命令。
- **chown:** 更改文件所有者。
- **chgrp:** 更改文件所属群组。
- **passwd:** 更改用户密码。

9.1 所有者、组成员和其他所有用户

我们在第 4 章讲解文件系统时，当试图查看类似/etc/shadow 的文件时，会遇到下面的问题。

```
[me@linuxbox ~]$ file /etc/shadow
/etc/shadow: regular file, no read permission
[me@linuxbox ~]$ less /etc/shadow
/etc/shadow: Permission denied
```

产生这种错误信息的原因是，作为一个普通用户，没有读取这个文件的权限。

在 UNIX 安全模型中，一个用户可以拥有 (own) 文件和目录。当一个用户拥有一个文件或者目录时，它将对该文件或目录的访问权限拥有控制权。反过来，用户又归属于一个群组 (group)，该群组由一个或者多个用户组成，组中用户对文件和目录的访问权限由其所有者授予。除了可以授予群组访问权限之外，文件所有者也可以授予所有用户一些访问权限，在 UNIX 术语中，所有用户是指整个世界 (world)。使用 id 命令可以获得用户身份标识的相关信息，如

下所示。

```
[me@linuxbox ~]$ id  
uid=500(me) gid=500(me) groups=500(me)
```

查看 id 命令的输出结果。在创建用户账户的时候，用户将被分配一个称为用户 ID (user ID) 或者 uid 的号码。为了符合人们的使用习惯，用户 ID 与用户名一一映射。同时用户将被分配一个有效组 ID (primary group ID) 或者称为 gid，而且该用户也可以归属于其他的群组。前面的例子是在 Fedora 系统中运行的结果。在其他系统中，比如 Ubuntu 系统，输出结果可能会有一些不同。

```
[me@linuxbox ~]$ id  
uid=1000(me) gid=1000(me)  
groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(  
plugdev),108(lpadmin),114(admin),1000(me)
```

我们可以发现，两个系统中用户的 uid 和 gid 号码是不同的。原因很简单，因为在 Fedora 系统中，普通用户账户是从 500 开始编号的，而在 Ubuntu 系统中则是从 1000 开始编号。同时我们也可以发现，Ubuntu 系统中的用户归属于更多的群组。这和 Ubuntu 系统管理系统设备和服务权限的方式有关。

那么这些信息从何而来呢？类似于 Linux 系统中的很多情况，这些信息来源于一系列的文本文件。用户账户定义在文件/etc/passwd 中，用户组定义在文件/etc/group 文件中。在创建用户账户和群组时，这些文件随着文件/etc/shadow 的变动而修改，文件/etc/shadow 中保存了用户的密码信息。对于每一个用户账户，文件/etc/passwd 中都定义了对应用户的用户（登录）名、uid、gid、账户的真实姓名、主目录以及登录 shell 信息。如果查看文件/etc/passwd 和文件/etc/group 的内容，那么你将会发现除了普通用户账户信息之外，文件中还有对应于超级用户（uid 为 0）和其他不同种类的系统用户的账户信息。

在第 10 章中我们介绍进程时，你将会发现这些其他的“用户”中有一些实际上是相当忙碌的。

许多类 UNIX 系统会把普通用户分配到一个公共的群组中（比如，users），然而现在的 Linux 操作则是创建一个独一无二的，只有一个用户的群组，而且组名和用户的名字相同。这使得特定类型的权限分配变得更加容易。

9.2 读取、写入和执行

对文件和目录的访问权限是按照读访问、写访问以及执行访问来定义的。

当我们查看 ls 命令的输出结果时，可以得到一些线索，了解其实现方式。

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me me 0 2012-03-06 14:52 foo.txt
```

列在输出结果中的前 10 个字符表示的是文件属性（file attribute，见图 9-1）。其中第一个字符表示文件类型（file type）。表 9-1 列出了最可能见到的文件类型（还有其他的不常见类型）。

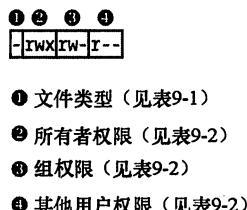


图 9-1 文件属性的分类

表 9-1 文件类型

属性	文件类型
-	普通文件
d	目录文件
l	符号链接。注意对于符号链接文件，剩下的文件属性始终是 rwxrwxrwx，它是一个伪属性值。符号链接指向的文件的属性才是真正文件的属性
c	字符设备文件。该文件类型表示以字节流形式处理数据的设备，如终端或调制解调器
b	块设备文件。该文件类型表示以数据块方式处理数据的设备，如硬盘驱动或者光盘驱动

文件属性中剩下的 9 个字符称为文件模式（file mode），分别表示文件所有者、文件所属群组以及其他所有用户对该文件的读取、写入和执行权限。

分别设置 r、w 和 x 的模式属性将会对文件和目录带来不同的影响，如表 9-2 所示。

表 9-2 权限属性

属性	文件	目录
r	允许打开和读取文件	如果设置了执行权限，那么允许列出目录下的内容
w	允许写入或者截短文件；如果也设置了执行权限，那么目录中的文件允许被创建、被删除以及被重命名	但是该权限不允许重命名或者删除文件。是否能重命名和删除文件由目录权限决定

续表

属性	文件	目录
x	允许把文件当作程序一样来执行。用脚本语言写的程序文件必须被设置为可读，以便能被执行	允许进入目录下，例如 cd directory

表 9-3 给出了一些文件属性设置的例子。

表 9-3 权限属性实例

文件属性	含义
-rwx-----	普通文件，文件所有者具有读取、写入和执行权限。组成员和其他所有用户都没有任何访问权限
-rw-----	普通文件，文件所有者具有读取和写入权限。组成员和其他所有用户都没有任何访问权限
-rwxr--r--	普通文件，文件所有者具有读取和写入权限。文件所有者所在群组的成员可以读取该文件。该文件对于所有用户来说都是可读的
-rwxr-xr-x	普通文件，文件所有者具有读取、写入和执行权限。其他所有用户也可以读取和执行该文件
-rw-rw----	普通文件，只有文件所有者和文件所有者所在群组的成员具有读取和执行权限
Lrwxrwxrwx	符号链接。所有的符号链接文件显示的都是“伪”权限属性，真正的权限属性由符号链接指向的实际文件决定
drwxrwx---	目录文件。文件所有者和所有者所在群组的成员都可以进入该目录，而且可以创建、重命名和删除该目录下的文件
drwxr-x---	目录文件。文件所有者可以进入该目录，而且可以创建、重命名和删除该目录下的文件。所有者所在群组的成员可以进入该目录，但是不能创建、重命名和删除该目录下的文件

9.2.1 chmod——更改文件模式

我们可以使用 chmod 命令来更改文件或者目录的模式（权限）。需要注意的是只有文件所有者和超级用户才可以更改文件或者目录的模式。chmod 命令支持两种不同的改变文件模式的方式——八进制数字表示法和符号表示法。首先我们来学习八进制数字表示法。

八进制数字表示法

八进制表示法指的是使用八进制数字来设置所期望的权限模式。因为每个八进制数字对应着 3 个二进制数字，所以这种对应关系正好可以和用来存储文件模式的结构方式一一映射。表 9-4 形象地说明了这一点。

表 9-4 以二进制和八进制方式表示文件模式

八进制	二进制	文件模式
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

八进制到底是什么

八进制 (octal, 以 8 为基数) 和十六进制 (hexadecimal, 以 16 为基数) 都是数字系统，它们经常用来表示计算机中的数字。由于人生来就有十个手指 (至少大多数人都是如此)，所以采用以 10 为基数的数字系统来计数。而另一方面，计算机生来只有一个手指，因此它们采用二进制 (以 2 为基数) 来完成所有的计数。它们的数字系统只有两个数字：0 和 1。所以在二进制中，以这种方式计数：0、1、10、11、100、101、110、111、1000、1001、1010、1011...

八进制使用数字 0~7 来计数，即 0、1、2、3、4、5、6、7、10、11、12、13、14、15、16、17、20、21...

十六进制使用数字 0~9，加上字母 A~F 来计数，即 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F、10、11、12、13...

二进制出现的意义可以理解 (由于计算机生来只有一个手指)，但是八进制和十六进制又有什么用处呢？答案是为了给人们提供方便。许多时候，计算机中的小部分数据以位模式 (bit pattern) 来表示。以 RGB 颜色为例子来说明。在大多数的计算机显示器中，每个像素由三种颜色组件组成：8 位红色、8 位绿色以及 8 位蓝色。一种漂亮的中蓝色由一组 24 位数字来表示，即 010000110110111111001101。

没有人愿意整天都读写这种类型的数字。这里使用另一种数字系统将更简单。十六进制中的一个数字代表二进制中的四个数字。八进制中的一个数字代表二进制中的三个数字。因此，这 24 位中蓝色二进制数将可以压缩成一个 6 位十六进制数：436FCD。由于十六进制的数字和二进制的位“排列整齐”，所以该颜色中的红色对应 43，绿色对应 6F，蓝色对应 CD。

目前十六进制 (经常称为 hex) 比八进制的使用更普遍，但是八进制数字

可以用来表示 3 位二进制数的功能是非常有用的，接下来我们将很快将可以看到这一点。

通过使用 3 位八进制数字，我们可以分别设置文件所有者、组成员和其他所有用户（world）的文件模式。

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me 0 2012-03-06 14:52 foo.txt
[me@linuxbox ~]$ chmod 600 foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw----- 1 me    me 0 2012-03-06 14:52 foo.txt
```

通过传递参数 600，我们可以设置文件所有者具有读写权限，而取消组用户和其他所有用户（world）的所有权限。虽然看起来，要记住八进制和二进制之间的映射关系好象不是那么简单，但是实际上，常用的也就只有这几个而已：7（rwx）、6（rw-）、5（r-x）、4（r--）和 0（---）。

符号表示法

chmod 命令支持一种符号表示法来指定文件模式。该符号表示法分为三部分：更改会影响谁、要执行哪个操作以及要设置哪种权限。可以通过字符 u、g、o 和 a 的组合来指定要影响的对象，如表 9-5 所示。

表 9-5 chmod 命令符号表示法

符号	含义
u	user 的简写，表示文件或者目录的所有者
g	文件所属群组
o	others 的简写，表示其他所有用户
a	all 的简写，是‘u’、‘g’和‘o’三者的组合

如果没有指定字符，则假定使用 all。操作符“+”表示添加一种权限，“-”表示删除一种权限，“=”表示只有指定的权限可用，其他所有的权限都被删除。

权限由字符“r”、“w”和“x”来指定。表 9-6 列出了一些符号表示法的实例。

表 9-6 chmod 命令符号表示法实例

符号	含义
u+x	为文件所有者添加可执行权限
u-x	删除文件所有者的可执行权限
+x	为文件所有者、所属群组和其他所有用户添加可执行权限，等价于 a+x

续表

符号	含义
o-rw	除了文件所有者和所属群组之外，删除其他所有用户的读写权限
go=rw	除了文件所有者之外，设置所属群组和其他所有用户具有读写权限。如果所属群组或者其他所有用户之前已经具有可执行权限，那么删除他们的可执行权限
u+x, go=rx	为文件所有者添加可执行权限，同时设置所属群组和其他所有用户具有读权限和可执行权限。指定多种权限时，需用逗号分隔

有些人喜欢使用八进制表示法，而有些人则真的很喜欢用符号表示法。符号表示法的优点在于允许设置单个属性，而不影响其他的任何属性。

我们可以查看 `chmod` 命令的帮助页面，以获取更多的细节内容和选项信息。关于`--recursive` 选项，我们需要注意，它对文件和目录都起作用，所以该选项并不如想象中的那么有用，因为用户很少会想要给文件和目录设置相同的权限。

9.2.2 采用 GUI 设置文件模式

现在，我们已经知道了如何设置文件和目录的权限，这样就可以更好地理解 GUI 中的设置权限对话框了。在 Nautilus (GNOME 桌面系统) 和 Konqueror (KDE 桌面系统) 中，右击文件或者目录图标都将会弹出一个属性对话框。图 9-2 是 KDE 3.5 环境下运行的一个例子。

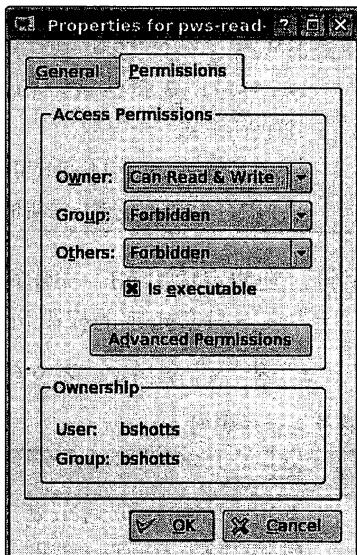


图 9-2 KDE 3.5 运行环境下文件属性对话框

可以看到，这个对话框中可以设置文件所有者、用户组和其他所有用户的权限。在 KDE 运行环境下，单击该对话框中的“Advanced Permissions（高级权限）”按钮，将弹出另一个对话框，在这个对话框中允许单独设置各个模式属性。另一种易于理解的实现方式就是使用命令行！

9.2.3 umask——设置默认权限

umask 命令控制着创建文件时指定给文件的默认权限。它使用八进制表示法来表示从文件模式属性中删除一个位掩码。

参见下面的例子：

```
[me@linuxbox ~]$ rm -f foo.txt
[me@linuxbox ~]$ umask
0002
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me    me    0 2012-03-06 14:53 foo.txt
```

首先，删除 foo.txt 文件存在的所有副本，以保证一切都是重新开始。下一步，运行不带任何参数的 umask 命令，查看当前掩码值，得到的值是 0002（0022 是另一个常用默认值），它是掩码的八进制表示形式。接着创建文件 foo.txt 的一个新实例，查看该文件的权限。

可以发现，文件所有者和组都获得了读写权限，而其他所有用户则只获得读权限。其他所有用户没有写权限的原因在于掩码值。重复执行该实例，不过这次是自己设置掩码值。

```
[me@linuxbox ~]$ rm foo.txt
[me@linuxbox ~]$ umask 0000
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-rw- 1 me    me    0 2012-03-06 14:58 foo.txt
```

在设置掩码为 0000（实际上是关闭该功能）时，可以看到其他所有用户也拥有写权限了。为了理解它是如何实现的，再来看看八进制数。如果把该掩码展开成二进制形式，然后再与属性进行对比，那么就能明白是怎么回事了。

原始文件模式	---	rw-	rw-	rw-
掩码	000	000	000	010
结果	---	rw-	rw-	r--

先忽略掉掩码中前面的 0（稍后再看），观察掩码中出现 1 的地方，将会发

现 1 的位置对应的属性被删除——在这个例子中对应的是其他所有用户的写权限。这就是掩码的操作方式。掩码的二进制数值中每个出现 1 的位置，其对应的属性都被取消。如果设置掩码值为 0022，那么具体操作如下。

原始文件模式	---	rW-	rW-	rW-
掩码	000	000	000	010
结果	---	rW-	rW-	r-

同样地，二进制数值中 1 出现的位置，其对应的属性都被取消。再试一下其他的掩码值（尝试一些带数字 7 的），以熟悉掩码的操作方式。记得每次操作完之后清理文件，并把掩码值还原到默认值。

```
[me@linuxbox ~]$ rm foo.txt; umask 0002
```

大多数情况下，你并不需要修改掩码值，系统提供的默认掩码值就很好了。然而，在一些高安全级别的环境下，你则需要控制掩码值。

一些特殊权限

虽然通常看到的八进制权限掩码都是用三位数字表示的，但是，确切地说，从技术层面上来看，它是用四位数字来表示的。为什么呢？因为，除了读取、写入和执行权限之外，还有一些其他的较少用到的权限设置。

其中之一是 `setuid` 位（八进制表示为 4000）。当把它应用到一个可执行文件时，有效用户 ID 将从实际用户 ID（实际运行该程序的用户）设置成该程序所有者的 ID。大多数情况下，该权限设置通常应用于一些由超级用户所拥有的程序。当普通用户运行一个具有“`setuid root`”（已设置 `setuid` 位，由 root 用户所有）属性的程序时，该程序将以超级用户的权限来执行。这使得该程序可以访问一些普通用户通常禁止访问的文件和目录。很明显，这会带来安全方面的问题，因此允许设置 `setuid` 位的程序个数必须控制在绝对小的范围内。

第二个是 `setgid` 位（八进制表示为 2000）。类似于 `setuid` 位，它会把有效组 ID 从该用户的实际组 ID 更改为该文件所有者的组 ID。如果对一个目录设置 `setgid` 位，那么在该目录下新创建的文件将由该目录所在组所有，而不是由文件创建者所在组所有。当一个公共组下的成员需要访问共享目录下的所有文件时，设置 `setgid` 位将很有用，并不需要关注文件所有者所在的有效组。

第三个是 `sticky` 位（八进制表示为 1000）。它是从传统 UNIX 中继承下来的，可以标记一个可执行文件为“不可交换的”。在 Linux 中，会忽略文件的

sticky 位，但是如果对一个目录设置 sticky 位，那么将能阻止用户删除或者重命名文件，除非用户是这个目录的所有者、文件所有者或者是超级用户。它常用来控制对共享目录（比如，/tmp）的访问。

这里有一些使用 chmod 命令和符号表示法来设置这些特殊权限的实例。

首先，授予程序 setuid 权限：

```
chmod u+s 程序名
```

下一步，授予目录 setgid 权限：

```
chmod g+s 目录
```

最后，授予目录 sticky 位权限：

```
chmod +t 目录
```

使用 ls 命令可以查看这些特殊权限的设置结果。首先，设置了 setuid 位的程序：

```
-rwsr-xr-x
```

具有 setgid 属性的目录：

```
drwxrwsr-x
```

设置了 sticky 位的目录：

```
drwxrwxrwt
```

9.3 更改身份

在很多时候，我们会发现可以拥有另一个用户的身份是很有必要的。我们经常会需要获得超级用户的特权来执行一些管理任务，但是也可以“变成”另一个普通用户来执行这些任务，就好像是在测试一个账户。有三种方法用来转换身份，具体如下。

- 注销系统并以其他用户的身份重新登录系统。
- 使用 su 命令。
- 使用 sudo 命令。

因为大家都知道如何操作第一种方法，而且它不如其他两种方法来得方便，所以这里跳过第一种方法。在 shell 会话状态下，使用 su 命令将允许你假定为另一个用户的身份，既可以以这个用户的 ID 来启动一个新的 shell 会话，也可以以这个

用户的身份来发布一个命令。使用 `sudo` 命令将允许管理者创建一个称为 `/etc/sudoer` 的配置文件，并且定义一些特定的命令，这些命令只有被赋予为假定身份的特定用户才允许执行。选择使用哪个命令在很大程度上取决于使用的 Linux 发行版本。有些发行版本可能对两个命令都支持，但是它的系统配置可能只是偏向于其中一个。首先我们来介绍 `su` 命令。

9.3.1 su——以其他用户和组 ID 的身份来运行 shell

`su` 命令用来以另一个用户的身份来启动 shell。该命令的一般形式如下。

```
su [-l] [user]
```

如果包含 “`-l`” 选项，那么得到的 shell 会话界面将是用于指定用户的登录 shell (login shell) 界面。这就意味着，该指定用户的运行环境将被加载，而且其工作目录也将更改为该指定用户的主目录。这也常常是我们想要得到的结果。如果没有指定用户，那么默认假定为超级用户。需要注意的是，`-l` 可以缩写为`-`，而且这一形式经常被使用。我们可以通过以下的操作来以超级用户的身份启动 shell。

```
[me@linuxbox ~]$ su -
Password:
[root@linuxbox ~]#
```

在输入 `su` 命令后，系统会提示输入该超级用户的密码。如果密码输入正确，那么将会出现新的 shell 提示符，该提示符表示该 shell 将拥有超级用户的特权（提示符的末尾字符是`#`，而不是`$`），而且当前的工作目录现在也是用于超级用户的主目录（通常情况下为`/root`）。一旦进入了这个新的 shell 环境，我们就可以以超级用户的身份执行命令了。在使用结束时，输入 `exit`，将返回到之前的 shell 环境。

```
[root@linuxbox ~]# exit
[me@linuxbox ~]$
```

我们也可以使用 `su` 命令执行单个命令，而不需要开启一个新的交互式命令界面，操作方式如下。

```
Su -c 'command'
```

使用这种格式，单个命令行将被传递到一个新的 shell 环境下进行执行。这里需要用单引号把命令行引起来，这一点很重要。因为该命令扩展并不希望在当前 shell 环境下执行，而是希望在新的 shell 环境下执行。

```
[me@linuxbox ~]$ su -c 'ls -l /root/*'
Password:
-rw----- 1 root root      754 2011-08-11 03:19 /root/anaconda-ks.cfg

/root/Mail:
total 0
[me@linuxbox ~]$
```

9.3.2 sudo——以另一个用户的身份执行命令

`sudo` 命令在很多方面都类似于 `su` 命令，但是它另外还有一些重要的功能。管理者可以通过配置 `sudo` 命令，使系统以一种可控的方式，允许一个普通用户以一个不同的用户身份（通常是超级用户）执行命令。在特定情况下，用户可能被限制为只能执行一条或者几条特定的命令，而对其他命令没有执行权限。另一个重要的区别在于，使用 `sudo` 命令并不需要输入超级用户的密码。使用 `sudo` 命令时，用户只需要输入自己的密码来进行认证。比如说，配置 `sudo` 命令来允许普通用户运行一个虚构的备份程序（称为 `backup_script`），这个程序需要超级用户的权限。

通过 `sudo` 命令，该程序将会以如下的方式运行。

```
[me@linuxbox ~]$ sudo backup_script
Password:
System Backup Starting...
```

在输入 `sudo` 命令后，系统将提示输入用户自己的密码（而不是超级用户的密码），而且一旦认证通过，指定的命令就将被执行。`su` 命令和 `sudo` 命令之间的一个重要区别在于 `sudo` 命令并不需要启动一个新的 shell 环境，而且也不需要加载另一个用户的运行环境。这意味着，使用 `sudo` 命令的时候并不需要用单引号把命令行引起来。需要注意的是，我们可以通过指定不同的选项来改变命令执行的效果。查看 `sudo` 命令的帮助页面可以获得更多的细节内容。

要想知道 `sudo` 命令可以授予哪些权限，可以使用`-l` 选项来查看，具体如下。

```
[me@linuxbox ~]$ sudo -l
User me may run the following commands on this host:
(ALL) ALL
```

Ubuntu 与 sudo

普通用户经常会遇到这样的一个问题，即如何完成某些特定的、需要超级用户权限才能完成的任务。这些任务包括安装和更新软件、编辑系统配置文件和访问设备等。在 Windows 世界中，这些任务通常是通过授予用户管理

员权限来完成的。然而，这也会使得用户执行的程序具有相同的功能。大多数情况下，这正是用户所期望的结果，但是这样也会使得类似病毒这样的恶意软件（malware）可以随意地操作计算机。

在 UNIX 世界中，由于 UNIX 多用户的传统特性，普通用户和管理者之间一直都存在着更大的差别。UNIX 采用的方法是只有在需要的时候才允许授予超级用户的特权，通常使用 su 命令和 sudo 命令来实现这一操作。

几年前，大多数的 Linux 发布版本都依靠 su 命令来达成这一目的。su 命令并不需要 sudo 命令所要求的配置环境，而且按照 UNIX 的传统，su 命令会拥有一个 root 账户。但是，这将会产生一个问题，即用户在不是很必要的情况下也会试图以 root 用户的身份来进行操作。事实上，一些用户专门以 root 用户的身份来操作系统，从而摆脱那些烦人的“permission denied（权限被拒绝）”信息，这就使得 Linux 系统的安全级别降低到和 Windows 系统的安全级别一样。因此，使用 su 命令并不是一个好主意。

在推出 Ubuntu 的时候，Ubuntu 的创造者采取了一个不同的策略。默认情况下，Ubuntu 不允许用户以 root 账户的身份登录（因为不能成功为 root 账户设置密码），取而代之的是使用 sudo 命令来授予超级用户的特权。最初的用户账户可以通过 sudo 命令来获得超级用户的全部权限，后面的用户账户也可以被授予相似的权限。

9.3.3 chown——更改文件所有者和所属群组

chown 命令用来更改文件或者目录的所有者和所属群组。使用这个命令需要超级用户的权限。chown 命令的语法格式如下。

```
chown [owner][:[group]] file ...
```

chown 命令更改的是文件所有者还是文件所属群组，或者对两者都更改，取决于该命令的第一个参数。表 9-7 列出了一些实例。

表 9-7 chown 命令参数实例

参数	结果
bob	把文件所有者从当前所有者更改为用户 bob
bob:users	把文件所有者从当前所有者更改为用户 bob，并把文件所属群组更改为 users 组
:admins	把文件所属群组更改为 admins 组，文件所有者不变
bob:	把文件所有者从当前所有者更改为用户 bob，并把文件所属群组更改为用户 bob 登录系统时所属的组

假设有两个用户——拥有超级用户权限的 janet 和没有该权限的 tony。用户 janet 想要从她的主目录复制一个文件到用户 tony 的主目录中。因为用户 janet 希望 tony 能够编辑该文件，所以 janet 把该文件的所有者从 janet 更改为 tony，具体操作如下。

```
[janet@linuxbox ~]$ sudo cp myfile.txt ~tony
Password:
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt
-rw-r--r-- 1 root root 8031 2012-03-20 14:30 /home/tony/myfile.txt
[janet@linuxbox ~]$ sudo chown tony: ~tony/myfile.txt
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt
-rw-r--r-- 1 tony tony 8031 2012-03-20 14:30 /home/tony/myfile.txt
```

这里我们可以看到，用户 janet 首先把文件从她的目录复制到用户 tony 的主目录中。接下来，janet 把该文件的所有者从 root（使用 sudo 命令的结果）更改为 tony。通过在第一个参数末尾加上冒号，janet 也把文件的所属群组更改成了 tony 登录系统时所属的组，该组名碰巧也叫 tony。

值得注意的是，在 janet 第一次使用了 sudo 命令之后，系统为什么没有提示她输入她的密码呢？这是因为，在大多数的配置环境下，sudo 命令会“信任”用户几分钟（直到计时结束）。

9.3.4 chgrp——更改文件所属群组

在更早的 UNIX 版本中，chown 命令只能更改文件的所有者，而不能改变文件所属群组。为了达到这个目的，我们可以使用一个独立的命令 chgrp 来实现。该命令除了限制多一点之外，和 chown 命令的使用方式几乎相同。

9.4 权限的使用

既然我们已经学习了权限是如何工作的，那么现在是时候学以致用了。接下来让我们看一个常见问题的解决方案——创建一个共享目录。假设有两个用户，分别命名为 bill 和 karen。他们都有音乐 CD 集，并想要创建一个共享目录，在该目录下他们各自以 Ogg Vorbis 格式或者 MP3 格式来存储音乐文件。用户 bill 通过 sudo 命令获得了超级用户的访问权限。

第一件需要做的事情，就是创建一个以 bill 和 karen 为成员的组。通过使用 GNOME 的图形化用户管理工具，bill 创建了一个组，命名为 music，并且把用户 bill 和 karen 添加到该组中，如图 9-3 所示。

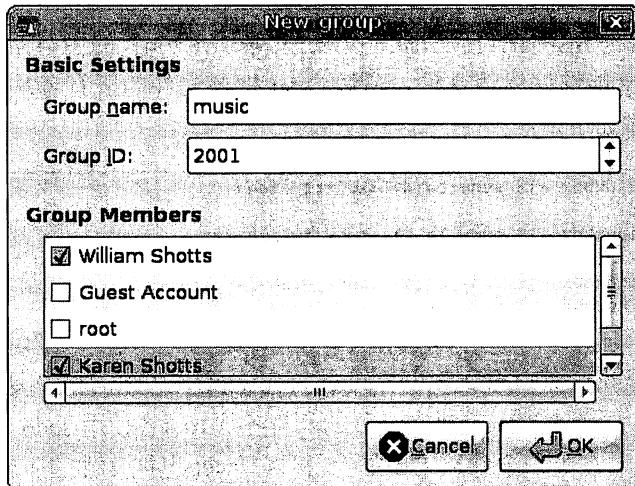


图 9-3 使用 GNOME 创建一个新组

接下来，bill 创建了存储音乐文件的目录。

```
[bill@linuxbox ~]$ sudo mkdir /usr/local/share/Music
Password:
```

因为 bill 正在操作的对象是他的主目录之外的文件，所以他需要拥有超级用户的权限。新创建的目录具有以下的所有权和权限。

```
[bill@linuxbox ~]$ ls -ld /usr/local/share/Music
drwxr-xr-x 2 root root 4096 2012-03-21 18:05 /usr/local/share/Music
```

可以看到，这个目录由 root 用户所有，而且权限值为 755。要使得该目录可共享，bill 需要更改该目录的所属群组，而且该组需要具有写入权限。

```
[bill@linuxbox ~]$ sudo chown :music /usr/local/share/Music
[bill@linuxbox ~]$ sudo chmod 775 /usr/local/share/Music
[bill@linuxbox ~]$ ls -ld /usr/local/share/Music
drwxrwxr-x 2 root music 4096 2012-03-21 18:05 /usr/local/share/Music
```

做所有的这些事情有什么意义呢？它意味着当前我们已经有了 /usr/local/share/Music 目录，该目录由 root 用户所有，而且 music 组拥有该目录的读写权限。music 组的成员为 bill 和 karen，因此 bill 和 karen 都可以在目录 /usr/local/share/Music 下创建文件。其他用户可以列出该目录下的内容，但是不能在该目录下创建文件。

但是我们仍然有一个问题。在当前的权限下，在 Music 目录下创建的文件和目录拥有用户 bill 和 karen 的常规权限。

```
[bill@linuxbox ~]$ > /usr/local/share/Music/test_file
[bill@linuxbox ~]$ ls -l /usr/local/share/Music
-rw-r--r-- 1 bill bill 0 2012-03-24 20:03 test_file
```

实际上，这会产生两个问题。第一个问题是，系统中的默认掩码是 0022，这将不会允许组成员对属于组内其他成员的文件执行写入操作。如果共享目录中只包含文件，那么这倒不是问题，但是因为该目录下将存放音乐文件，而音乐文件一般都是按照艺术家和唱片集的层次结构来组织分类的，所以组成员需要拥有能在同组其他成员创建的目录下创建文件和目录的权限。需要把 bill 和 karen 使用的掩码值更改成 0002。

第二个问题是，由成员创建的每一个文件和目录都将被设置为归属于该用户的有效组，而不是归属于 music 组。可以通过对该目录设置 setgid 位来修复这个问题。

```
[bill@linuxbox ~]$ sudo chmod g+s /usr/local/share/Music
[bill@linuxbox ~]$ ls -ld /usr/local/share/Music
drwxrwsr-x 2 root music 4096 2012-03-24 20:03 /usr/local/share/Music
```

现在来测试一下，看看是否新的权限修复了这个问题。bill 把他的掩码值设置为 0002，删除了之前的测试文件，又创建了一个新的测试文件和目录。

```
[bill@linuxbox ~]$ umask 0002
[bill@linuxbox ~]$ rm /usr/local/share/Music/test_file
[bill@linuxbox ~]$ > /usr/local/share/Music/test_file
[bill@linuxbox ~]$ mkdir /usr/local/share/Music/test_dir
[bill@linuxbox ~]$ ls -l /usr/local/share/Music
drwxrwsr-x 2 bill music 4096 2012-03-24 20:24 test_dir
-rw-rw-r-- 1 bill music 0 2012-03-24 20:22 test_file
[bill@linuxbox ~]$
```

当前创建的文件和目录都具有正确的权限，允许 music 组内的所有成员在 Music 目录下创建文件和目录。

最后剩下的一个问题是在于 umask 命令的。umask 命令设置的掩码值只能在当前 shell 会话中生效，在当前 shell 会话结束后，则必须重新设置。在第 11 章中，我们将介绍如何使得对 umask 命令掩码值的更改永久生效。

9.5 更改用户密码

本章的最后一个主题就是用户如何为自己设置密码（如果拥有超级用户权限，那么也可以为其他的用户设置密码）。使用 passwd 命令，可以设置或者更改密码。该命令的语法格式如下。

```
Passwd [user]
```

如果要更改的是用户自己的密码，那么只需要输入 passwd 命令。接下来 shell 将会提示用户输入旧密码和新密码。

```
[me@linuxbox ~]$ passwd  
(current) UNIX password:  
New UNIX password:
```

passwd 命令会试着强迫用户使用“强”密码。也就是说，它会拒绝接受太短的密码、与之前的密码相似的密码、字典中的单词作为密码或者是太容易猜到的密码。

```
[me@linuxbox ~]$ passwd  
(current) UNIX password:  
New UNIX password:  
BAD PASSWORD: is too similar to the old one  
New UNIX password:  
BAD PASSWORD: it is WAY too short  
New UNIX password:  
BAD PASSWORD: it is based on a dictionary word
```

如果你具有超级用户的权限，那么可以通过指定一个用户名作为 passwd 命令的参数来为另一个用户设置密码。对于超级用户，还可以使用该命令的其他选项来设置账户锁定、密码失效等功能。你可以查看 passwd 命令的帮助页面获取更多的细节内容。

第 10 章

进 程

现代操作系统通常都支持多重任务处理（multitasking）。多重任务处理是指系统通过快速切换运行中的程序来实现多任务的同时执行。Linux 内核通过使用进程来管理多重任务。进程是 Linux 用来安排不同程序等待 CPU 调度的一种组织方式。

有时候计算机运行速度会变得很慢，或者应用程序会停止响应。本章将介绍命令行中可用来查看程序当前运行情况以及终止运行异常的进程的一些工具。

本章将介绍以下命令。

- **ps:** 显示当前所有进程的运行情况。
- **top:** 实时显示当前所有任务的资源占用情况。
- **jobs:** 列出所有活动作业的状态信息。
- **bg:** 设置在后台中运行作业。

- **fg:** 设置在前台中运行作业。
- **kill:** 发送信号给某个进程。
- **killall:** 杀死指定名字的进程。
- **shutdown:** 关机或者重启系统。

10.1 进程如何工作

系统启动时，内核先把它的一些程序初始化为进程，然后运行一个称为 init 的程序。init 程序将依次运行一系列称为脚本初始化 (init script) 的 shell 脚本 (放在/etc 目录下)，这些脚本将会启动所有的系统服务。其中的很多服务都是通过守护程序 (daemon program) 来实现的。而后台程序只是呆在后台做它们自己的事情，并且没有用户界面。因此，即使没有用户登录，系统也在忙于执行一些例行程序。

一个程序的运行可以触发其他程序的运行，在进程系统中这种情况被表述为父进程创建子进程。

内核会保存每个进程的信息以便确保任务有序进行。比如，每个进程将被分配一个称为进程 ID (PID, process ID) 的号码。进程 ID 是按递增的顺序来分配的，init 进程的 PID 始终为 1。内核也记录分配给每个进程的内存信息以及用来恢复运行的进程就绪信息。和文件系统类似，进程系统中也存在所有者、用户 ID、有效用户 ID 等。

10.1.1 使用 ps 命令查看进程信息

用来查看进程信息的命令中 (有多个)，使用最普遍的就是 ps 命令。ps 命令有很多选项，其中最简单的使用格式如下所示。

```
[me@linuxbox ~]$ ps
  PID TTY      TIME CMD
 5198 pts/1    00:00:00 bash
10129 pts/1    00:00:00 ps
```

这个例子的输出结果列出了两个进程：进程 5198 和进程 10129，它们分别对应 bash 命令和 ps 命令。我们可以发现，默认情况下，ps 命令输出的信息并不是很多，只是输出和当前终端会话相关的进程信息。为了获得更多的信息，我们需要添加一些选项，但是在介绍这个之前，让我们先看看 ps 命令输出的其

他字段信息。TTY 是 *teletype*（电传打字机）的缩写，代表了进程的控制终端（controlling terminal）。UNIX 在这里也显示了进程的运行时间，TIME 字段表示了进程消耗的 CPU 时间总和。可以看出，这两个进程都没有使计算机变得忙碌。

如果在 ps 命令后添加一个选项，那么我们将得到反映系统运行情况的更大视图界面，如下所示。

```
[me@linuxbox ~]$ ps x
  PID TTY      STAT   TIME COMMAND
 2799 ?        Ssl    0:00 /usr/libexec/bonobo-activation-server -ac
 2820 ?        Sl     0:01 /usr/libexec/evolution-data-server-1.10 --
15647 ?        Ss    0:00 /bin/sh /usr/bin/startkde
15751 ?        Ss    0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --
15754 ?        S      0:00 /usr/bin/dbus-launch --exit-with-session
15755 ?        Ss    0:01 /bin/dbus-daemon --fork --print-pid 4 -pr
15774 ?        Ss    0:02 /usr/bin/gpg-agent -s -daemon
15793 ?        S      0:00 start_kdeinit --new-startup +kcminit_start
15794 ?        Ss    0:00 kdeinit Running...
15797 ?        S      0:00 dcopserver -nosid
```

and many more...

添加 x 选项（注意这里没有前置的连字符）将告知 ps 命令显示所有的进程，而不需要关注它们是由哪个终端（如果有其他的情况）所控制的。TTY 列中出现的“？”表示没有控制终端，使用这个选项可以查看所有进程的列表信息。

由于系统中运行着大量的进程，所以 ps 命令将会输出一个长列表。把 ps 命令的输出作为 less 命令输入的方法通常很管用，它可以更方便地查看显示结果。有些选项组合也会产生很长的输出行，因此最大化终端仿真窗口也是一个好主意。

输出结果中添加了一个命名为 STAT 的新列。STAT 是 state 的缩写，显示的是进程的当前状态，如表 10-1 所示。

表 10-1 进程状态

状态	含义
R	运行状态。进程正在运行或者准备运行
S	睡眠状态。进程不在运行，而是在等待某事件发生，如键盘输入或者收到网络报文
D	不可中断的睡眠状态。进程在等待 I/O 操作，如硬盘驱动
T	暂停状态。进程被指示暂停（后续还可继续运行）
Z	无效或者“僵尸”进程。子进程被终止，但是还没有被其父进程彻底释放掉
<	高优先级进程。进程可以被赋予更多的重要性，分配更多的 CPU 时间。进程的这一特性称为优先级（niceness）。高优先级的进程被说成较不友好，是因为它将消耗更多的 CPU 时间，这样留给其他进程的 CPU 时间就会变少
N	低优先级进程。低优先级进程（友好进程，a nice process）只有在其他更高优先级的进程使用完处理器后才能够获得使用处理器的时间

这些进程状态的后面可以带其他的字符来表示不同的特殊进程特性。你可以查看 ps 命令的帮助页面来获取更多的详细信息。

另一个常用的选项组合是 aux (不带前置连字符)，它将输出更多的信息，如下所示。

```
[me@linuxbox ~]$ ps aux
USER     PID %CPU %MEM    VSZ   RSS TTY STAT START   TIME COMMAND
root      1  0.0  0.0  2136  644 ?  Ss   Mar05  0:31 init
root      2  0.0  0.0     0    0 ?  S<  Mar05  0:00 [kt]
root      3  0.0  0.0     0    0 ?  S<  Mar05  0:00 [mi]
root      4  0.0  0.0     0    0 ?  S<  Mar05  0:00 [ks]
root      5  0.0  0.0     0    0 ?  S<  Mar05  0:06 [wa]
root      6  0.0  0.0     0    0 ?  S<  Mar05  0:36 [ev]
root      7  0.0  0.0     0    0 ?  S<  Mar05  0:00 [kh]
```

and many more...

该选项组合将会显示属于每个用户的进程信息，使用这些选项时不带前置连字符将使得命令以“BSD 模式 (BSD-style)”运行。ps 命令的 Linux 版本可以模拟多种 UNIX 版本中 ps 程序的运行方式，使用这些选项将显示更多列的信息，具体如表 10-2 所示。

表 10-2 BSD 模式下 ps 命令输出的列标题

标题	含义
USER	用户 ID。表示该进程的所有者
%CPU	CPU 使用百分比
%MEM	内存使用百分比
VSZ	虚拟耗用内存大小
RSS	实际使用的内存大小。进程使用的物理内存 (RAM) 大小 (以 KB 为单位)
START	进程开启的时间。如果数值超过 24 个小时，那么将使用日期来显示

10.1.2 使用 top 命令动态查看进程信息

虽然 ps 命令可以显示有关机器运行情况的很多信息，但是它提供的只是在 ps 命令被执行时刻机器状态的一个快照。要查看机器运行情况的动态视图，我们可以使用 top 命令，如下所示。

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

top 程序将按照进程活动的顺序，以列表的形式持续更新显示系统进程的当前信息 (默认每 3 秒更新一次)。它主要用于查看系统“最高 (top)”进程的运行情况，其名字也来源于此。top 命令显示的内容包含两个部分，顶部显示的是

系统总体状态信息，下面显示的是一张按 CPU 活动时间排序的进程情况表。

top - 14:59:20 up 6:30, 2 users, load average: 0.07, 0.02, 0.00												
Tasks: 109 total, 1 running, 106 sleeping, 0 stopped, 2 zombie												
Cpu(s): 0.7%us, 1.0%sy, 0.0%ni, 98.3%id, 0.0%wa, 0.0%hi, 0.0%si												
Mem: 319496k total, 314860k used, 4636k free, 19392k buff												
Swap: 875500k total, 149128k used, 726372k free, 114676k cach												
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	
6244	me	39	19	31752	3124	2188	S	6.3	1.0	16:24.42	trackerd	
11071	me	20	0	2304	1092	840	R	1.3	0.3	0:00.14	top	
6180	me	20	0	2700	1100	772	S	0.7	0.3	0:03.66	dbus-dae	
6321	me	20	0	20944	7248	6560	S	0.7	2.3	2:51.38	multiloa	
4955	root	20	0	104m	9668	5776	S	0.3	3.0	2:19.39	Xorg	
1	root	20	0	2976	528	476	S	0.0	0.2	0:03.14	init	
2	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kthreadd	
3	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	migratio	
4	root	15	-5	0	0	0	S	0.0	0.0	0:00.72	ksoftirq	
5	root	RT	-5	0	0	0	S	0.0	0.0	0:00.04	watchdog	
6	root	15	-5	0	0	0	S	0.0	0.0	0:00.42	events/0	
7	root	15	-5	0	0	0	S	0.0	0.0	0:00.06	khelper	
41	root	15	-5	0	0	0	S	0.0	0.0	0:01.08	kblockd/	
67	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kseriod	
114	root	20	0	0	0	0	S	0.0	0.0	0:01.62	pdfflush	
116	root	15	-5	0	0	0	S	0.0	0.0	0:02.44	kswapd0	

系统总体状态信息包含很多有用的内容，表 10-3 将逐条解释这些字段，具体如下。

表 10-3 顶部信息中的字段

行	字段	含义
1	top	程序名
	14:59:20	一天中的当前时间
	up 6:30	正常运行时间（uptime）。从机器最后一次启动开始计算的时间总数。在这个例子中，系统已经运行了 6.5 小时。
	2 users	有两个用户已登录
	load average:	负载均值（load average）指的是等待运行的进程数；即共享 CPU 资源的处于可运行状态的进程数。显示的三个值分别对应不同的时间段第一个对应的是前 60 秒的均值，下一个对应的是前 5 分钟的均值，最后一个对应的是前 15 分钟的均值。该值小于 1.0 表示该机器并不忙
2	tasks:	统计进程数及各个进程的状态信息
	0.7%us	0.7% 的 CPU 时间被用户进程占用，这里指的是处于内核外的进程
	1.0%sy	1.0% 的 CPU 时间被系统进程（内核进程）占用
	0.0%ni	0.0% 的 CPU 时间被友好进程（nice）（低优先级进程）占用
	98.3%id	98.3% 的 CPU 时间是空闲的
	0.0%wa	0.0% 的 CPU 时间用来等待 I/O 操作
4	Mem:	显示物理 RAM（随机存取内存）的使用情况
5	Swap:	显示交换空间（虚拟内存）的使用情况

`top` 程序可以接受许多键盘指令，其中最常用的有两个：一个是 `h`，输入后将显示程序的帮助界面；另一个是 `q`，用来退出 `top` 命令。

主流的桌面环境都提供了用来显示类似 `top` 命令的输出信息的图形化应用程序（和 Windows 中任务管理器[Task Manager]的运行方式类似），但是 `top` 命令优于图形化版本，这是因为 `top` 命令运行得更快，而且消耗的系统资源要少得多。毕竟，系统监控程序不应该减缓正在被监控的系统的处理速度。

10.2 控制进程

既然我们已经知道了如何查看和监控进程，那么接下来让我们看看如何对进程进行控制。我们将使用一个称为 `xlogo` 的小程序作为实验对象。`xlogo` 程序是由 X 窗口系统（X Window System，使得显示器支持图形化界面的底层引擎）提供的一个示例程序，它只简单地显示一个包含 X 标识的可缩放窗口。首先，我们认识一下实验对象。

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

输入该命令后，包含该标识的一个小窗口将在屏幕的某个地方出现。有些系统中，`xlogo` 可能会输出一条告警信息，但是我们可以忽略它，因为它并不会造成什么影响。

注意	如果系统中不包含 <code>xlogo</code> 程序，那么试着使用 <code>gedit</code> 程序或者 <code>kwrite</code> 程序来替代。
----	--

我们可以通过改变窗口的大小来验证 `xlogo` 是否处于运行状态。如果该标识适应新的窗口大小被重新绘制了，则表明该程序正在运行。

注意，为什么这里 shell 提示符没有返回呢？这是因为 shell 正在等待该 `xlogo` 程序结束，就像以前使用的其他程序一样。如果关闭 `xlogo` 窗口，那么提示符将返回。

10.2.1 中断进程

让我们观察再次运行 `xlogo` 命令的时候会发生什么。首先，输入 `xlogo` 命令，并确保程序在正常运行。接下来，返回到终端窗口，按下 `Ctrl-C` 键。

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

在终端里按下 Ctrl-C 键将会中断 (interrupt) 一个程序，它意味着我们委婉地请求程序结束。按下 Ctrl-C 键后，xlogo 窗口将关闭，shell 提示符将返回。

许多（但不是所有）命令行程序都可以使用这种方法来实现中断。

10.2.2 使进程在后台运行

假设我们想要 shell 提示符返回，但又不终止 xlogo 程序，那么可以通过让该程序在后台 (background) 运行来实现。我们可以把终端想象为有一个前台 (foreground，表面上可见的内容，类似 shell 提示符) 和一个后台 (隐藏在表层下面的内容)。要想在启动程序时让该程序在后台运行，可以在命令后面加上和号字符 (&) 来实现。

```
[me@linuxbox ~]$ xlogo &
[1] 28236
[me@linuxbox ~]$
```

命令执行后，将出现 xlogo 窗口，而且 shell 提示符也将返回，但是同时也将打印一些有趣的数字信息。这条信息是 shell 的一个称为作业控制 (job control) 的特性表现。shell 通过这条信息来显示已经启动的作业编号为 1 ([1])，其对应的 PID 是 28236。如果执行 ps 命令，可以查看到当前运行的进程。

```
[me@linuxbox ~]$ ps
  PID TTY      TIME CMD
10603 pts/1    00:00:00 bash
28236 pts/1    00:00:00 xlogo
28239 pts/1    00:00:00 ps
```

shell 的作业控制特性也提供了一种方式来查看从该终端启动的所有作业。使用 jobs 命令可以得到如下列表信息。

```
[me@linuxbox ~]$ jobs
[1]+  Running                  xlogo &
```

输出结果显示存在一个编号为 1 的作业在运行，而且对应命令是 xlogo &.

10.2.3 使进程回到前台运行

后台运行的进程不会受到任何键盘输入的影响，包括试图用来中断它的 Ctrl-C 键。要想使得进程返回到前台来运行，可以使用 fg 命令来实现，参见下面的例子。

```
[me@linuxbox ~]$ jobs
[1]+  Running                  xlogo &
```

```
[me@linuxbox ~]$ fg %1
xlogo
```

我们可以通过在 `fg` 命令后面加上百分比符号和作业编号（称为 `jobspec` 选项）来实现这个功能。如果后台只有一个任务，那么可以不带 `jobspec` 选项。这个时候按下 `Ctrl-C` 键就可以终止 `xlogo` 命令。

10.2.4 停止（暂停）进程

如果我们只是想要暂停进程，而不是终止进程，那么通常需要我们将前台运行的进程移到后台去运行。我们为了暂停前台进程需要按下 `Ctrl-Z` 键。让我们试试如下操作，在命令提示符后输入 `xlogo`，按下 `Enter` 键后再按下 `Ctrl-Z` 键。

```
[me@linuxbox ~]$ xlogo
[1]+ Stopped                  xlogo
[me@linuxbox ~]$
```

在暂停 `xlogo` 命令后，我们可以通过试图改变 `xlogo` 窗口的大小来确认该程序是否真正被暂停了。可以发现，该进程看起来好像死了。这个时候，我们可以使用 `fg` 命令让进程在前台恢复运行，也可以使用 `bg` 命令让进程移到后台运行：

```
[me@linuxbox ~]$ bg %1
[1]+ xlogo &
[me@linuxbox ~]$
```

在使用 `fg` 命令的时候，如果只存在一个作业，那么可以不带 `jobspec` 选项。

如果用命令方式启动了一个图形化程序，但是忘记了在命令尾部加上“`&`”符号来让程序在后台运行，那么在这种情况下，把进程从前台移到后台去运行的方法将非常方便。

为什么会想要通过命令行的方式来启动一个图形化程序呢？原因有两个。首先，想要运行的程序可能并不在窗口管理器的菜单中（比如 `xlogo` 程序）。

其次，从命令行启动程序可以看到用图形化方式启动程序所看不到的错误信息。有时候从图形菜单中启动程序，程序会启动失败。但改用命令行方式启动的话，就可以得到错误提示信息，找到问题所在。另外，一些图形化程序也包含很多有意思的和有用的命令行选项。

10.3 信号

`kill` 命令通常用来“杀死”（终止）进程，它可以用来终止运行不正常的程

序或者反过来拒绝终止的程序。这里有一个例子，如下所示。

```
[me@linuxbox ~]$ xlogo &
[1] 28401
[me@linuxbox ~]$ kill 28401
[1]+  Terminated                  xlogo
```

我们首先在后台启动了 xlogo 程序。shell 将打印输出该后台进程的 jobspec 选项信息和 PID 信息。接着，我们使用了 kill 命令，并且指定想要终止进程的 PID。我们也可以使用 jobspec 选项（例如，%1）代替 PID 信息来指定该进程。

这些看起来都非常简单，但是事实上，它们包含着更多的内容。kill 命令准确地说并不是“杀死”进程，而是给进程发送信号（signal）。信号是操作系统和程序间通信的多种方式之一，在使用 Ctrl-C 键和 Ctrl-Z 键时已经见识过信号的作用。当终端接收到其中的一个输入时，它将发送信号到前台进程。在按下 Ctrl-C 键的情况下，它将发送一个称为 INT（中断，Interrupt）的信号；在按下 Ctrl-Z 的情况下，它将发送一个称为 TSTP（终端暂停，Terminal Stop）的信号。反过来，程序“侦听”信号，而且在接收到信号的时候按照它们的指示进行操作。程序可以侦听信号并且可以按照信号指示操作的这一特性，使得程序在接收到终止信号的时候可以保存当前正在进行的工作。

10.3.1 使用 kill 命令发送信号到进程

kill 命令最常用的语法格式如下。

```
kill [-signal] PID...
```

如果命令行中没有指定信号，那么默认发送 TERM（终止，Terminate）信号。kill 命令最常用来发送的信号如表 10-4 所示。

表 10-4 常用信号

信号编号	信号名	含义
1	HUP	挂起信号。这是美好的过去留下的痕迹，当时通过电话线和调制解调器来把终端和远端计算机连接在一起。该信号用来指示程序控制终端已被“挂起”。该信号的效果通过关闭终端会话的方式来表现。运行在终端上的前台程序收到该信号后将终止。该信号也被很多后台程序用来进行重新初始化。这就意味着，当一个后台进程接收到该信号时，它将重启并且重新读取它的配置文件。Apache Web 服务器就是后台进程使用 HUP 信号重新初始化的一个例子
2	INT	中断信号。执行效果和在终端按下 Ctrl-C 键的效果一样。通常用来终止一个程序

续表

信号编号	信号名	含义
9	KILL	杀死信号。该信号比较特殊。鉴于程序可以选择不同的方式来处理发送过来的信号，包括忽略所有的这些信号，KILL 信号将不会真正意义上地被发送到目标程序。而是内核宁愿立即终止了该进程。当进程以这种方式被终止时，它将没有机会对它自己进行“清理”或者对当前工作进行保存。考虑到这个原因，KILL 信号只能当作其他的终端信号都执行失败的情况下最后选择
15	TERM	终止信号。这是 kill 命令默认发送的信号类型。如果程序仍然有足够的“活力”(alive enough) 来接收信号，那么它将被终止
18	CONT	继续运行信号。恢复之前接受了 STOP 信号的进程
19	STOP	暂停信号。该信号将使进程暂停，而不是终止。和 KILL 信号类似，该信号不会被发送给目标进程，因此它不能被忽略

按照下面的方式使用 kill 命令。

```
[me@linuxbox ~]$ xlogo &
[1] 13546
[me@linuxbox ~]$ kill -1 13546
[1]+ Hangup                  xlogo
```

在这个例子中，我们首先在后台启动了 xlogo 程序，接着使用 kill 命令给它发送 HUP 信号。xlogo 程序将终止，shell 的输出信息表明这个后台进程已经接收了一个挂起信号。你也许需要多敲几次 Enter 键才能看到这条输出信息。注意，你可以通过信号编号或者信号名来指定信号，其中包含带有 SIG 前缀的信号名。

```
[me@linuxbox ~]$ xlogo &
[1] 13601
[me@linuxbox ~]$ kill -INT 13601
[1]+ Interrupt                  xlogo
[me@linuxbox ~]$ xlogo &
[1] 13608
[me@linuxbox ~]$ kill -SIGINT 13608
[1]+ Interrupt                  xlogo
```

尝试使用其他的信号重复执行上面的例子。记住，你也可以使用 jobspec 选项来代替 PID 信息。

和文件一样，进程也有所有者，只有进程的所有者（或者超级用户）才能使用 kill 命令来给它发送信号。

除了表 10-4 中列出的通常用于 kill 命令的信号之外，还存在其他一些经常被系统使用的信号。表 10-5 列出的是其他的一些常用信号。

表 10-5 其他常用信号

信号编号	信号名	含义
3	QUIT	退出信号
11	SEGV	段错误信号。如果程序非法使用了内存空间，即程序试图在没有写权限的空间执行写操作，那么系统将发送该信号
20	TSTP	终端暂停信号。在按下 Ctrl-Z 键时终端将发出该信号。与 STOP 信号不同的是，TSTP 信号由程序接收，但是程序可以选择忽略该信号
28	WINCH	窗口改变信号。当窗口改变大小时，系统将发送该信号。类似 top 和 less 的一些程序将会对该信号作出响应，重新绘制视图来适应新的窗口大小

如果想要查看更多的信号，使用如下命令将显示完整的信号列表。

```
[me@linuxbox ~]$ kill -l
```

10.3.2 使用 killall 命令发送信号给多个进程

通过使用 killall 命令，我们可以给指定程序或者指定用户名的多个进程发送信号。一般语法格式如下。

```
killall [-u user] [-signal] name...
```

要证明这一点，我们可以先启动两个 xlogo 程序实例，然后终止它们。

```
[me@linuxbox ~]$ xlogo &
[1] 18801
[me@linuxbox ~]$ xlogo &
[2] 18802
[me@linuxbox ~]$ killall xlogo
[1]- Terminated                  xlogo
[2]+ Terminated                  xlogo
```

记住，和 kill 命令一样，你必须具有超级用户权限，才能够使用 killall 命令给不属于自己的进程发送信号。

10.4 更多与进程相关的命令

由于进程监控是一项重要的系统管理任务，所以存在很多命令用来为它服务。表 10-6 列出了其中一些命令。

表 10-6 其他与进程相关的命令

命令	描述
pstree	以树状的模式输出进程列表，该模式显示了进程间的父/子关系
vmstat	输出系统资源使用情况的快照，包括内存，交换空间和磁盘 I/O。如果想要持续查看输出，可以在命令后面加上一个间隔时间（以秒为单位），命令将按照间隔时间来动态更新显示的内容（比如，vmstat 5）。按下 Ctrl-C 键可以终止输出
xload	用来绘制显示系统时间负载情况图形的一种图形化界面程序
tload	类似于 xload 程序，但是图形是在终端上绘制。按下 Ctrl-C 键终止输出

第二部分

配置与环境



第 11 章

环 境

前面讲到，在 shell 会话调用环境（environment）期间，shell 会存储大量的信息。程序使用存储在环境中的数据来确定我们的配置。尽管大多数系统程序使用配置文件（configuration file）来存储程序设置，但是也有一些程序会查找环境中存储的变量来调整自己的行为。知道这一点之后，用户就可以使用环境来自定义 shell。

本章会讲解下述命令。

- `printenv`: 打印部分或全部的环境信息。
- `set`: 设置 shell 选项。
- `export`: 将环境导出到随后要运行的程序中。
- `alias`: 为命令创建一个别名。

11.1 环境中存储的是什么

尽管 shell 在环境中存储了两种基本类型的数据，但是在 bash 中，这两种类

型基本上没有区别。这两种数据类型分别是环境变量（environment variable）和 shell 变量（shell variable）。shell 变量是由 bash 存放的少量数据，环境变量就是除此之外的所有其他变量。除变量之外，shell 还存储了一些编程数据（programmatic data），也就是别名和 shell 函数。本书第 5 章阐述了与别名有关的内容，而 shell 函数（主要与 shell 脚本有关）将会在本书的第四部分进行讲解。

11.1.1 检查环境

要了解环境中存储的内容，需要用到集成在 bash 中的 set 命令或 printenv 程序。不同的是，set 命令会同时显示 shell 变量和环境变量，而 printenv 只会显示环境变量。由于环境的内容可能会比较冗长，所以最好将这两个命令的输出以管道形式重定向到 less 命令中。

```
[me@linuxbox ~]$ printenv | less
```

得到的结果应当类似如下所示。

```
KDE_MULTIHEAD=false
SSH_AGENT_PID=6666
HOSTNAME=linuxbox
GPG_AGENT_INFO=/tmp/gpg-Pd0t7g/S.gpg-agent:6689:1
SHELL=/bin/bash
TERM=xterm
XDG_MENU_PREFIX=kde-
HISTSIZE=1000
XDG_SESSION_COOKIE=6d7b05c65846c3eaf3101b0046bd2b00-1208521990.996705-11770561
99
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/me/.gtkrc-2.0:/home/me/.kde/share/config/gtkrc
GTK_RC_FILES=/etc/gtk/gtkrc:/home/me/.gtkrc:/home/me/.kde/share/config/gtkrc
GS_LIB=/home/me/.fonts
WINDOWID=29360136
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
KDE_FULL_SESSION=true
USER=me
LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=40;33
;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe:
```

可以看到，输出结果是一系列的环境变量及其变量值。例如，让我们来看一个名为 USER 的变量，其值为 me。命令 printenv 也能够列出特定变量的值。

```
[me@linuxbox ~]$ printenv USER
me
```

在使用 set 命令时，如果不带选项或参数，那么只会显示 shell 变量、环境变量以及任何已定义的 shell 函数。

```
[me@linuxbox ~]$ set | less
```

与 printenv 命令不同的是，set 命令的输出结果是按照字母顺序排列的。

如需要查看单个变量的值，我们也可以使用 echo 命令，如下所示。

```
[me@linuxbox ~]$ echo $HOME  
/home/me
```

set 命令和 printenv 命令都不能显示的一个环境元素是别名。要查看别名，需使用不带任何参数的 alias 命令。

```
[me@linuxbox ~]$ alias  
alias l.='ls -d .* --color=tty'  
alias ll='ls -l --color=tty'  
alias ls='ls --color=tty'  
alias vi='vim'  
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --showtilde'
```

11.1.2 一些有趣的变量

环境中包含了相当多的变量，尽管你所使用的环境与这里的不相同，也会在你的环境中看到表 11-1 中所示的变量。

表 11-1 环境变量

变量	说明
DISPLAY	运行图形界面环境时界面的名称。通常为 <code>:0</code> ，表示由 X 服务器生成的第一个界面
EDITOR	用于文本编辑的程序名称
SHELL	本机 shell 名称
HOME	本机主目录的路径名
LANG	定义了本机语言的字符集和排序规则
OLD_PWD	先前的工作目录
PAGER	用于分页输出的程序名称。通常设置为 <code>/usr/bin/less</code>
PATH	以冒号分割的一个目录列表，当用户输入一个可执行程序的名称时，会查找该目录列表
PS1	提示符字符串 1。定义了本机 shell 系统提示符的内容。在后面我们会看到，可以灵活地自定义该变量
PWD	当前工作目录
TERM	终端类型的名称。类 UNIX 系统支持很多种终端协议；此变量设定了本机终端模拟器使用的协议
TZ	用于指定本机所处的时区。大多数类 UNIX 系统以协调世界时（UTC）来维护计算机的内部时钟，而显示的本地时间是根据本变量确定的时差计算出来的
USER	用户名

如果某些变量无法在该表中找到也不要紧，因为这些变量会因发行版本的不同而有差异。

11.2 环境是如何建立的

用户登录系统后，`bash` 程序就会启动并读取一系列称为启动文件的配置脚本，这些脚本定义了所有用户共享的默认环境。接下来，`bash` 会读取更多存储在主目录下的用于定义个人环境的启动文件。这些步骤执行的确切顺序是由启动的 shell 会话类型决定的。

11.2.1 login 和 non-login shell

shell 会话存在两种类型，分别为 `login shell` 会话和 `non-login shell` 会话。

`login shell` 会话会提示用户输入用户名和密码，如虚拟控制台会话。而我们在 GUI 中启动的终端会话就是一个典型的 `non-login shell` 会话。

`login shell` 会读取一个或多个启动文件，如表 11-2 所示。

表 11-2 `login shell` 的启动文件

文件	说明
<code>/etc/profile</code>	适用于所有用户的全局配置脚本
<code>~/.bash_profile</code>	用户的个人启动文件。可扩展或重写全局配置脚本中的设置
<code>~/.bash_login</code>	若 <code>~/.bash_profile</code> 缺失，则 <code>bash</code> 尝试读取此脚本
<code>~/.profile</code>	若 <code>~/.bash_profile</code> 与 <code>~/.bash_login</code> 均缺失，则 <code>bash</code> 尝试读取此文件。在基于 Debian 的 Linux 版本中（比如 Ubuntu），这是默认值

表 11-3 所示为 `non-login shell` 读取的启动文件。

表 11-3 `non-login shell` 的启动文件

文件	内容
<code>/etc/bash.bashrc</code>	适用于所有用户的全局配置脚本
<code>~/.bashrc</code>	用户的个人启动文件。可扩展或重写全局配置脚本中的设置

在读取以上启动文件之外，`non-login shell` 还会继承父类进程的环境，父类进程通常是一个 `login shell`。

用户可查看本机系统有哪些启动文件，需要注意的是这些文件大多数以“.”

开头（意味着这些文件是被隐藏的），所以用户在使用 ls 命令时，需要伴随使用 -a 选项。

在普通用户看来，`~/.bashrc` 可能是最重要的启动文件，因为系统几乎总是要读取它。`non-login shell` 会默认读取`~/.bashrc`，而大多数 `login shell` 的启动文件也能以读取`~/.bashrc` 文件的方式来编写。

11.2.2 启动文件中有什么

一个典型的`.bash_profile`（来自于 CentOS-4 系统）内容如下所示。

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
# User specific environment and startup programs

PATH=$PATH:$HOME/bin
export PATH
```

文件中以“#”开始的行是注释行，而 shell 是不会读取注释行的，注释是为提高用户可读性而存在的。一件有趣的事发生在第 4 行，如下所示。

```
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

这段代码被称作 if 复合命令，会在本书的第四部分进行讲解，现在可以将这段代码理解为如下所示的内容。

```
If the file "~/.bashrc" exists, then
read the "~/.bashrc" file.
```

可以看到这一段代码阐述了 `login shell` 读取`.bashrc` 文件的机制。以上启动文件中另一个很重要的元素是 PATH 变量。

在命令行输入一条命令后，你曾经疑惑过 shell 是怎样找到这些命令的吗？当用户输入命令 `ls`，shell 不会搜索整个系统来寻找`/bin/ls`（`ls` 命令的完整路径名），而是会搜索 PATH 变量中存储的目录列表。

PATH 变量通常是由启动文件`/etc/profile` 中的一段代码设定（并不总是如此，这取决于系统的发行版本）。

```
PATH=$PATH:$HOME/bin
```

这段代码将`$HOME/bin` 添加到了 PATH 值的尾部。这是一个参数扩展的

实例，本书第 7 章介绍过相关的内容。以下代码可以帮助用户理解参数扩展的机理。

```
[me@linuxbox ~]$ foo="This is some"
[me@linuxbox ~]$ echo $foo
This is some
[me@linuxbox ~]$ foo=$foo" text."
[me@linuxbox ~]$ echo $foo
This is some text.
```

使用参数扩展，用户可以将更多的内容添加到变量值的尾部。

在把字符串\$HOME/bin 添加到 PATH 值的尾部之后，当系统需要检索用户输入的命令时，\$HOME/bin 这个路径就会处于被搜索的路径列表中。这就意味着当我们想在主目录下创建名为 *bin* 的目录，并在此目录中存放自己的私有程序时，shell 已经为我们准备好了，我们要做的就是将创建的目录称之为 bin。

注意

很多 Linux 发行版本在默认情况下提供了该 PATH 设置。一些基于 Debian 的发行版本，如 Ubuntu，会在登录时检查~/bin 目录是否存在，若存在，就会自动将其添加到 PATH 变量中。

最后一行是如下代码：

```
export PATH
```

该 export 命令告诉 shell，将 shell 的子进程使用 PATH 变量的内容。

11.3 修改环境

现在用户已经知道了系统启动文件的位置和内容，就可以修改启动文件，来自定义我们的环境。

11.3.1 用户应当修改哪些文件

一般来说，在 PATH 中添加目录，或者定义额外的环境变量，需要将这些更改放入到.bash_profile 文件中（或者是其他的等效文件，这取决于系统的发行版本，比如 Ubuntu 系统使用的是.profile 文件），其他的改变则应录入.bashrc 文件中。除非是系统管理员需要修改用户公用的默认设置，普通用户只需对主目录下的文件作出修改即可。当然用户也可以修改其他目录下的文件，比如/etc 下的 profile 文件，而且很多情况会需要用户这样做，但是现在我们先保险一点操作。

11.3.2 文本编辑器

为了编辑（比如修改）shell 的启动文件，以及系统中的其他大多数配置文件，我们会用到一个称为文本编辑器的程序。文本编辑器类似于字处理器，它允许用户通过移动光标的方式来编辑屏幕中的文字。与字处理器不同的是，文本编辑器只支持纯文本，而且通常包含为编写程序而设计的特性。文本编辑器是软件开发人员编写代码的主要工具，系统管理员也可以使用文本编辑器来管理系统的配置文件。

Linux 系统可使用的文本编辑器有很多种，你的系统中可能装有不止一种的文本编辑器。为什么会有这么多种编辑器？主要是因为程序员热衷于编写文本编辑器，既然程序员在工作中会广泛地用到编辑器，他们希望文本编辑器能符合自己的工作方式。

文本编辑器可大概分为两类：图形界面的和基于文本的。GNOME（GNU 网络对象模型环境）和 KDE（K 桌面环境）都配备有一些流行的图形界面编辑器。GNOME 配备的编辑器叫做 gedit，在 GNOME 菜单中 gedit 通常被称为 Text Editor。KDE 则配备了三种编辑器，分别是 kedit、kwrite 和 kate（复杂程度递增）。

有很多种基于文本的编辑器，常见编辑器中较受用户欢迎的是 nano、vi 和 emacs。nano 是一种简单易用的编辑器，最初是为了替代 pico（由 PINE 电子邮件套件提供）而出现的。vi 是类 Unix 系统的传统文本编辑器（在大多数 Linux 系统中已被 vim——Vi Improved 的缩写——所取代），本书第 12 章的讲解主题就是 vi。而 emacs 编辑器最初由 Richard Stallman 编写，这是一个庞大的、万能的、可做任何事情的编辑环境。尽管 emacs 仍然可用，但是大多数 Linux 系统很少默认安装 emacs。

11.3.3 使用文本编辑器

所有的文本编辑器都可以通过在命令行输入编辑器名称和需编辑的文件名称的方式启动。如果输入的文件不存在，编辑器会认为用户想要创建一个新的文件。下面是一个使用 gedit 的范例。

```
[me@linuxbox ~]$ gedit some_file
```

如果 some_file 文件存在，这条命令将启动 gedit 编辑器，并载入 some_file。

因为所有的图形界面编辑器都非常易于理解，所以这里就不做赘述。接下

来，我们将通过.bashrc 文件的编辑过程来讲解 nano， nano 是第一个基于文本的文本编辑器。但在此之前，需要先采取一些安全措施。在修改一些重要的配置文件时，先对配置文件进行备份再进行编辑是一个很好的习惯。当用户把文件修改得一团糟的时候，备份就很有用处了。我们可以使用以下代码备份.bashrc：

```
[me@linuxbox ~]$ cp .bashrc .bashrc.bak
```

为备份文件取什么名字并不重要，只要备份文件的名称易于理解即可。扩展名.bak、.sav、.old 和.orig 是常用的标示备份文件的方法。需要说明的是，cp 命令会默默地覆盖现有的文件。

备份文件完成之后，就可以启动文件编辑器了。

```
[me@linuxbox ~]$ nano .bashrc
```

nano 启动后，屏幕显示内容如下所示。

```
GNU nano 2.0.3          File: .bashrc

# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions

[ Read 8 lines ]
^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos
^X Exit      ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell
```

注意

若系统没有安装 nano，也可以使用图形界面编辑器来进行操作。

屏幕显示内容分为三部分：顶端的标题（header）、中间的可编辑文本和底部的命令菜单。由于 nano 是替代电子邮件文本编辑器出现的，所以其编辑功能非常有限。

对于每一种文本编辑器，你都应该首先学习它的退出命令。就 nano 来说，可按 Ctrl-X 退出程序，在页面底部的命令菜单中有相关的介绍。“^X”代表了 Ctrl-X，这是控制字符的常见表示法，很多程序中都使用它。

我们需要了解的第二个命令就是如何保存我们的工作。就 nano 来说，按 Ctrl-O 完成保存。掌握这些知识之后，我们就可以进行文本编辑操作了。请使用向下箭头键或者向下翻页键使光标移动到文件的末尾，然后添加以下代码到.bashrc 文件中。

```
umask 0002
export HISTCONTROL=ignoredups
export HISTSIZE=1000
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

注意

用户系统的.bashrc 文件可能已经写入了这些代码的一部分，但是不用担心，重复的代码不会造成什么危害。

表 11-4 列出了以上代码的含义。

表 11-4 .bashrc 文件增加的代码

代码行	含义
Umask 0002	设置 umask 值以解决第 9 章中讨论过的共享目录的问题
Export HISTCONTROL=ignoredups	使 shell 的历史记录功能忽略与上一条录入的命令重复的命令
Export HISTSIZE=1000	使命令历史记录规模从默认的 500 行增加到 1000 行
alias l.='ls -d .* --color=auto'	创建新的命令：l.，功能是显示所有以 . 开头的目录条目
alias ll='ls -l --color=auto'	创建新的命令：ll，功能是以长格式来展示目录列表

可以看到，很多新增加的代码并不易于理解，所以就需要在.bashrc 文件中添加一些注释来帮助用户理解代码的含义。添加注释后的代码如下所示。

```
# Change umask to make directory sharing easier
umask 0002

# Ignore duplicates in command history and increase
# history size to 1000 lines
export HISTCONTROL=ignoredups
export HISTSIZE=1000

# Add some helpful aliases
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

这样一来就易懂多了。最后我们按 Ctrl-O 保存文档，按 Ctrl-X 退出 nano，这样对.bashrc 文件的修改就完成了。

11.3.4 激活我们的修改

因为只有在启动 shell 会话时才会读取.bashrc，所以对.bashrc 做出的修改只有在关闭 shell 终端会话并重启的时候才会生效。当然也可以使用以下命令强制

命令 bash 重新读取.bashrc 文件。

```
[me@linuxbox ~]$ source .bashrc
```

重新读取.bashrc 之后，文件中做出的修改就会生效。我们来试一下其中的一个新的别名。

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

为什么注释很重要

不管何时修改配置文件，随手添加一些注释来记录做出的改变是一个好的习惯。我们可以记住近期做出的修改，但是 6 个月之前的修改呢？所以，随手写上一些注释吧，这对用户本身有益。同时，在写注释的时候，附加一个修改记录是一个不错的主意。

在 shell 脚本和 bash 启动文件中，注释是以“#”开头的。其他的配置文件可能会使用其他的符号。大多数配置文件中都有注释，这些注释能起到很好的向导作用。

在配置文件中经常会看到一些代码被注释掉，以防止它们被相关程序读取。这是为了给读者示范可能的配置选项或者正确的配置方法。比如，Ubuntu 8.04 的.bashrc 文件包含以下内容。

```
# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'
```

最后三行被注释掉的代码定义的别名是有效的。如果将这三行代码去注释化，也就是去掉开头的“#”符号，那么这些别名将被激活。反之，如果在一行代码前添加“#”符号，就可以在保留原本信息的基础上使这行配置代码失效。

11.4 本章结尾语

本章节讲解了一项重要的基本技能——使用文本编辑器编辑配置文件。随着学习的继续，当我们查看某一命令的 man 文档时，记录下该命令支持的环境变量，可能会有有趣的发现。接下来的章节会讲到 shell 函数，shell 函数是一种强大的功能，用户可将其添加到 bash 启动文件中，以此来添加你的自定义命令。

第 12 章

VI 简介

有一个古老的笑话，讲的是一个观光客想去纽约著名的古典音乐厅，于是找到一个路人问路的故事。

观光客：“不好意思，请问怎样才能到达卡耐基音乐厅？”

路人：“你要练习，练习，再练习！”

就像一个人不可能一夕之间成为技艺高超的钢琴家，Linux 命令也不是花一个下午就能熟练掌握的，这需要很长时间的练习。本章节将介绍 UNIX 传统核心软件之一——文本编辑器 vi（发音是“vee eye”）。vi 用户界面的不友好是非常出名的，但是看一位 vi 专家在键盘前坐下并开始“演奏”，将是莫大的艺术享受。本章节并不能使读者成为 vi 专家，但是在学习之后，读者至少能够做到在 vi 中演奏“Chopsticks”。

12.1 为什么要学习 vi

现在这个时代存在着很多图形界面编辑器和易用的基于文本的编辑器，例

如 nano，那为什么还要学习 vi？这有三条充分的理由。

- vi 总是可用的。如果用户面前的系统没有图形界面，例如是远程服务器或者是本地系统的 X 配置不可用，那么 vi 就会成为救命的稻草。尽管 nano 已经得到了越来越广泛的应用，但是，迄今为止它还不是通用的。而 POSIX（一种 UNIX 系统的程序兼容标准）则要求系统必须配备有 vi。
- vi 是轻量级的软件，运行速度快。对很多任务来说，启动 vi 比在菜单中找到一个图形界面编辑器并等待几兆大小的编辑器载入要容易得多。另外，vi 的设计还非常利于打字。在接下来的讲解中读者可以了解到，vi 高手在编辑过程中甚至不需要把手指从键盘上移开。
- 用户不想被其他 Linux 和 UNIX 用户蔑视。

好吧，也许只有两条正当的理由。

12.2 VI 背景

1976 年，加州大学伯克利分校的学生，之后又成为 Sun 公司创始人之一的 Bill Joy 写出了 vi 的第一个版本。vi 出自单词“visual”，含义是能够在视频终端上用移动光标来进行编辑。在图形界面编辑器出现之前是行编辑器的天下，用户每次只能在一行文本上进行编辑。使用行编辑器的时候，用户需要告知编辑器是在哪一行进行什么样的操作，比如添加或者删除。而视频终端（而非基于打印机的终端，比如电报）的来临使得全屏幕编辑成为可能。由于 vi 融合了强大的行编辑器 ex，vi 用户也可以同时使用行编辑的命令。

大多数 Linux 发行版配备的并不是真正的 vi，而是 Bram Moolenaar 编写的 vi 加强版——vim（Vi Improved 的缩写）。vim 是传统 UNIX 系统中 vi 的实质性改良版。通常，vin 的硬连接（或别名）指向 Linux 系统的 vi 名称。接下来的讨论就是建立在用户使用名为 vi 的 vim 程序这样一个假设上的。

12.3 启动和退出 vi

输入以下命令启动 vi：

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

屏幕显示如下：

```

VIM - Vi Improved

version 7.1.138
by Bram Moolenaar et al.
Vim is open source and freely distributable

Sponsor Vim development!
type :help sponsor<Enter>    for information

type :q<Enter>              to exit
type :help<Enter> or <F1>    for on-line help
type :help version?<Enter>   for version info

Running in Vi compatible mode
type :set nocp<Enter>        for Vim defaults
type :help cp-default<Enter>  for info on this

```

像操作 nano 一样，现在应该先学习如何退出 vi。输入以下命令退出 vi（需要注意的是，冒号是命令的一部分）。

:q

此时 shell 会返回初始的操作窗口。如果因为一些原因，vi 不能够退出（通常是因为没有保存修改过的文件），可以通过在命令后添加感叹号的方式强制退出 vi。

:q!

注意 如果用户不能确定 vi 所处的状态，可以按 Esc 键两次返回初始状态。

12.4 编辑模式

再次启动 vi，并向其传递一个不存在的文件名，就可以通过 vi 创建新文件。

```
[me@linuxbox ~]$ rm -f foo.txt
[me@linuxbox ~]$ vi foo.txt
```

正常情况下，屏幕显示如下所示。

"foo.txt" [New File]

每行开头的波浪线代表此行没有任何内容，也就是说现在 `foo` 是空白的文件。
先不要输入任何内容！

讲解过如何退出 vi 之后，接下来需要了解的就是 vi 是一个模态编辑器。vi 启动后进入的是命令模式。在命令模式中，几乎键盘上的每一个按键都代表一条命令，所以在这时对 vi 进行普通输入的话，vi 基本上就要崩溃了，并且会把文件弄得一团糟。

12.4.1 进入插入模式

如果用户需要向文件中添加一些内容，那么首先要做就是按 I 键（或 i）进入插入模式。若此时 vim 是在增强模式下正常地运行，那么在屏幕底部会出现以下内容（若 vim 以兼容模式运行，则不会出现）：

— INSERT —

现在用户可以进行输入操作了，例如：

The quick brown fox jumped over the lazy dog

最后按 Esc 键退出插入模式并返回命令模式。

12.4.2 保存工作

要保存用户修改过的文件，在命令模式下输入一条 ex 命令，也就是按“：“键。这样之后，一个冒号会出现在屏幕的底部：

1

要将文件写入硬盘，在冒号之后输入 w，如下所示：

:w

文件写入硬盘驱动器之后，用户会在屏幕底部得到一条确认信息。

```
"foo.txt" [New] 1L, 46C written
```

注意

如果用户阅读 vim 的说明文档，会困惑地发现命令模式被称为普通模式，而使用 ex 命令则被称为命令模式。这方面需多加留意。

兼容模式

在前面示例的 vi 启动屏幕（取自 Ubuntu 8.04 版本）中，用户可以看到这样的内容：Running in Vi compatible mode。这意味着 vim 将以近似于 vi 的常规模式运行，而不是加强版的 vim 行为。为达到本章的目的，用户需要使用加强模式下的 vim。以下两种方式都可以达到此目的。

- 运行 vim 而不是 vi（如果此法可行，可以考虑在.bashrc 文件中添加别名 vi='vim'）。
- 使用以下命令在 vim 配置文件中添加一行内容。

```
echo "set nocomp" >> ~/.vimrc
```

Linux 发行版本不同，其 vim 包也就不同。一些版本在默认情况下只是安装了 vim 的最小版本，只支持有限的 vim 特性。在接下来的讲解中，可能会用到 vim 特性缺少的情况。如果是这样的话，你可安装一个完全版的 vim。

12.5 移动光标

在命令模式下，vi 提供了很多移动光标命令，其中有一些命令是与 less 命令共用的。表 12-1 列出了命令的一部分。

表 12-1 光标移动功能键

键	光标动作
L 或右方向键	右移一位
H 或左方向键	左移一位
J 或下方向键	下移一行
K 或上方向键	上移一行
数字 0	至本行开头

续表

键	光标动作
Shift-6(^)	至本行第一个非空字符
Shift-4(\$)	至本行的末尾
W	至下一单词或标点的开头
Shift-W(W)	至下一单词的开头，忽略标点
B	至上一单词或标点的开头
Shift-B(B)	至上一单词的开头，忽略标点
Ctrl-F 或 Page Down	下翻一页
Ctrl-B 或 Page UP	上翻一页
number-Shift-G	至第 number 行（如 1G 会将光标移到文件的第一行）
Shift-G(G)	至文件的最后一行

为什么使用 H、J、K 和 L 键来移动光标呢？这是因为在 vi 最初出现的阶段，并不是所有的视频终端都有方向键，这样的设计使得 vi 高手可以手不离键盘地移动光标。

像表 12-1 的 G 命令一样，许多 vi 的命令的前面都可以缀上数字。前缀数字可以控制命令执行的次数，比如 5j 可以使得光标下移 5 行。

12.6 基本编辑

插入、删除、剪切、复制等构成了基本的文本编辑操作，vi 也以其特殊的方式支持这些操作。同时 vi 还支持有限形式的撤销操作，在命令模式下按 U 键就可以撤销用户最后一步操作。这项功能在学习一些编辑命令的时候会很有帮助。

12.6.1 添加文本

有几种方式都可以进入 vi 的插入模式。现在假设已经使用 i 命令进入插入模式。

先回顾下 foo.txt 内容。

The quick brown fox jumped over the lazy dog.

因为光标不能跳出行末，所以单纯使用 i 命令并不能完成在文本末尾添加内容的任务。为此 vi 提供了在行末添加文本的 a 命令。当用户将光标移动到行的

末尾并使用 a 命令时，光标就会越过文本的末尾，同时 vi 进入插入模式。这样用户就可以在行末添加文本了。

```
The quick brown fox jumped over the lazy dog. It was cool.
```

输入结束后不要忘记按 Esc 键退出插入模式。

因为用户经常用到在行末添加文本的功能，所以 vi 提供了使光标移动到行末并进入插入模式的快捷方式——A 命令。现在我们就来试一下。

首先，使用 0 命令将光标移动到行的开头。接下来使用 A 命令将以下内容写入文件中。

```
The quick brown fox jumped over the lazy dog. It was cool.
```

```
Line 2
```

```
Line 3
```

```
Line 4
```

```
Line 5
```

按 Esc 键退出插入模式。

可以看到，A 命令使 vi 进入插入模式并自动将光标移动到行尾，非常好用。

12.6.2 插入一行

插入文本的另一种方式是在文本中重开一行，即在两行现存的文字中间插入空白行并进入插入模式。表 12-2 列出了插入一行的两种方式。

表 12-2 插入一行功能键

命令	开行
o	当前行的上方
O	当前行的下方

下面这个例子示范了这两种命令的作用。先将光标置于 Line 3，再输入 o，结果如下所示。

```
The quick brown fox jumped over the lazy dog. It was cool.
```

```
Line 2
```

```
Line 3
```

```
Line 4
```

```
Line 5
```

我们可以看到在第三行的下方 vi 插入了一行，并进入了插入模式。按 Esc 键退出插入模式，按 u 键取消上述操作。

继续输入命令 o，就会在第三行的上方插入了一行。

```
The quick brown fox jumped over the lazy dog. It was cool.  
Line 2  
  
Line 3  
Line 4  
Line 5
```

再次按 Esc 键退出插入模式并按 u 键取消操作。

12.6.3 删除文本

就像用户期望的一样，vi 提供了很多种删除文本的方式，每一种都需要进行一次至两次的按键操作。首先，X 键会删除光标处的字符。x 命令可加以数字前缀来明确删除的字符数目。D 键则使用得更加普遍。像 x 命令一样，d 命令也可加练数字前缀来明确删除的次数。另外，d 命令总是加以控制删除范围的光标移动命令作为后缀。表 12-3 给出了一些范例。

现在我们进行命令练习。我们将光标移至文件首行单词 It 的首字母，使用 x 命令直到完全删除本句。然后按 u 键直到所有的删除操作都被取消为止。

注意

实际上，vi 只能取消一次操作，vim 可取消多次操作。

表 12-3 文本删除命令

命令	删除内容
x	当前字符
3x	当前字符和之后 2 个字符
dd	当前行
5dd	当前行和之后 4 行
dW	当前字符到下一单词的起始
d\$	当前字符到当前行的末尾
d0	当前字符到当前行的起始
d^	当前字符到当前行下一个非空字符
dG	当前行到文件末尾
d20G	当前行到文件第 20 行

现在让我们练习使用 d 命令。我们再次将光标移动到单词 It，使用 dW 命令来删除整个单词。

```
The quick brown fox jumped over the lazy dog. was cool.  
Line 2  
Line 3  
Line 4  
Line 5
```

使用的 d\$ 删除光标至本行末尾的字符。

```
The quick brown fox jumped over the lazy dog.  
Line 2  
Line 3  
Line 4  
Line 5
```

使用 dG 删除当前行到文件末尾的内容。

```
-  
-  
-
```

使用 u 命令三次来取消以上操作。

12.6.4 剪切、复制和粘贴文本

命令 d 不只是删除文本，而是在“剪切”文本。用户每次使用 d 命令之后，都会复制删除的内容进缓存（类似剪贴板），然后用户就可以使用 p 命令将缓存中的内容粘贴到光标之后或使用 P 命令将内容粘贴到光标之前。

就像命令 d 剪切文本的形式一样，命令 y 会“复制”文本。表 12-4 列举了一些 y 命令与光标移动命令共同作用的范例。

表 12-4 复制命令

命令	复制内容
yy	当前行
5yy	当前行和之后 4 行
yW	当前字符到下一单词的起始
y\$	当前字符到当前行的末尾
y0	当前字符到当前行的起始
y^	当前字符到当前行下一个非空字符
yG	当前行到文件末尾
y20G	当前行到文件第 20 行

现在让我们来练习一下复制和粘贴。我们将光标移至文本的第一行，使用

yy 命令复制当前行。接下来，将光标移至最后一行 (G)，使用 p 命令将复制的内容粘贴到当前行的下方。

```
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
Line 3
Line 4
Line 5
The quick brown fox jumped over the lazy dog. It was cool.
```

命令 u 会取消我们的操作。将光标移至文件的最后一行，输入 p 命令将文本粘贴到当前行的上方。

```
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
Line 3
Line 4
The quick brown fox jumped over the lazy dog. It was cool.
Line 5
```

将表 12-4 中的其他命令都练习一下，以实际了解 p 命令和 P 命令的作用。练习结束后，将文件恢复到本来的样子。

12.6.5 合并行

vi 在行的概念上非常严格。通常来说，将光标移动到行的末端并删除行的末尾字符并不能将此行与下一行合并。因此，vi 专门提供了 J 命令（不要与移动光标的 j 命令混淆）来合并行。

若将光标置于第 3 行并输入 J 命令，将得到如下所示的结果。

```
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
Line 3 Line 4
Line 5
```

12.7 查找和替换

vi 提供了在一行或者整个文件中，根据搜索条件将光标移动至指定位置的功能。vi 还可以执行文本替换工作，用户可指定替换时是否需要用户确认。

12.7.1 行内搜索

命令 f 在行内进行搜索，并将光标移至搜索到的下一个指定字符。比如，命令 fa 就会将光标移动到本行下一处出现字符 a 的地方。在执行过一次行内搜索之后，

输入分号可以使 vi 重复上一次搜索。

12.7.2 搜索整个文件

同第 3 章中讲解过的 less 程序一样，命令 “/” 可以完成对单词或短语的搜索。当用户使用 “/” 命令后，一个 “/” 符号会出现在屏幕的底部。接下来，输入需要搜索的单词或短语，以 Enter 结束。光标就会移动到下一处包含被搜索字符串的地方。使用 n 命令可以重复此搜索。如下例所示。

```
The quick brown fox jumped over the lazy dog. It was cool.  
Line 2  
Line 3  
Line 4  
Line 5
```

将光标移至文件的第一行，并输入如下代码。

```
/Line
```

输入 Enter 以结束，光标将移动至第 2 行。接下来，输入 n，光标将继续移动至第 3 行。重复输入 n 直至光标移动到文档的最后，且找不到符合条件的字符串。尽管现在只讲解到 vi 的单词和词组的搜索模式，但是 vi 同样支持正则表达式（一种强大的表达复杂文本模式的方法）的应用。第 19 章将会讲解这方面的内容。

12.7.3 全局搜索和替换

vi 使用 ex 命令来执行几行之内或者整个文件中的搜索和替换操作。输入以下命令可将文件中的 Line 替换为 line。

```
:%s/Line/line/g
```

现在就来解析这条命令每一部分的功能（见表 12-5）。

表 12-5 全局搜索和替换语法范例

组成	含义
:	分号用于启动一条 ex 命令
%	确定了操作作用的范围。%简洁地代表了从文件的第 1 行到最后 1 行。本命令的范围还可以表示为 1,5（因为本文件只有 5 行），或者是 1,\$，意思是“从第 1 行到文件的最后一行”。如果不明确指出命令的作用范围，那么命令只会在当前行生效
s	指定了具体的操作——本次是替换操作（搜索和替换）

续表

组成	含义
/Line/line	搜索和替换的文本
g	代指 global (全局)，也就是说对搜索到的每一行的每一个实例进行替换。如果 g 缺失，那么只替换每一行第一个符合条件的实例

以下是执行过查找和替换命令之后的文档内容。

```
The quick brown fox jumped over the lazy dog. It was cool.
line 2
line 3
line 4
line 5
```

在命令末尾添加 c，则命令在每次替换之前都会请求用户确认。如下所示。

```
:%s/line/Line/gc
```

此命令将会将文件替换回原来的样子，但是每次替换前，vi 都会停下来询问用户是否确认执行替换。

```
replace with Line (y/n/a/q/l/^E/^Y)?
```

圆括号中的每一个字符都是一种可能的回答，表 12-6 具体阐述了每一个字符的含义。

表 12-6 替换确认功能键

功能键	行为
y	执行替换
n	跳过此次替换
a	执行此次替换和之后的所有替换
q 或者 ESC	停止替换
l	执行此次替换并退出替换。是 last 的缩写
Ctrl-E, Ctrl-Y	分别是向下滚动和向上滚动，能用于查看替换处的上下文

12.8 编辑多个文件

用户经常遇到需要同时编辑多个文件的情况。可能是需要对多个文件作出修改，或者是拷贝文件的部分内容到另一个文件。用户可以通过在命令行具体指定多个文件的方式使 vi 打开多个文件。

```
vi file1 file2 file3...
```

现在退出所处的 vi 会话，并创建一个用于编辑的新文件。输入:wq 来退出 vi 并保存做出的修改。接下来，使用 ls 命令的部分输出在主目录创建一个用于实验的新文件。

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

现在就用 vi 来同时编辑旧文件和新文件。

```
[me@linuxbox ~]$ vi foo.txt ls-output.txt
```

vi 启动后，屏幕显示内容如下所示。

```
The quick brown fox jumped over the lazy dog. It was cool.  
Line 2  
Line 3  
Line 4  
Line 5
```

12.8.1 切换文件

使用以下 ex 命令来从一个文件切换到下一个文件。

```
:n
```

切换回上一个文件。

```
:N
```

当用户从一个文件切换到另一个的时候，vi 要求用户必须先保存对当前文件做出的修改才能切换到其他文件。若要放弃对文件的修改并使 vi 强制切换到另一个文件，可在命令后添加感叹号 (!)。

除了以上描述的切换方法之外，vim（和一些版本的 vi）还提供了一些 ex 命令让用户可以更轻松地编辑多个文本。用户可使用:buffers 命令来查看正在编辑的文件列表。

```
:buffers
1 %a      "foo.txt"          line 1
2       "ls-output.txt"       line 0
Press ENTER or type command to continue
```

输入:buffer 加文件 (buffer) 编号可切换到另一个文件 (buffer)。如从文件 1 (foo.txt) 切换到文件 2 (ls-output.txt)，用户应当输入如下命令。

```
:buffer 2
```

现在屏幕展示的就是文件 2 的内容了。

12.8.2 载入更多的文件

我们也可以在现有的编辑会话中载入更多的文件。使用 ex 命令:e (edit 的缩写) 加文件名可以载入另一个文件。先退出现有的编辑会话并回到命令行模式。

重启 vi, 并只打开一个文件。

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

添加一个文件到编辑会话中, 输入下列代码。

```
:e ls-output.txt
```

屏幕将展示第二个文件的内容, 而第一个文件仍然处在编辑状态, 可使用:buffers 命令来证实。

```
:buffers
1 #    "foo.txt"                      line 1
2 %a   "ls-output.txt"                 line 0
Press ENTER or type command to continue
```

注意

使用:ez 载入的文件不会响应:n 或者:N 命令, 而需使用:buffer 加文件编号来切换文件。

12.8.3 文件之间的内容复制

用户在编辑多个文件的过程中, 有时会需要将一个文件中的一部分复制到另一个文件中。使用之前使用过的复制和粘贴命令即可完成此功能, 示范如下。首先, 在载入的两个文件中, 切换到文件 1 (foo.txt)。

```
:buffer 1
```

此时屏幕显示如下所示。

```
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
Line 3
Line 4
Line 5
```

接下来, 将光标移动到文件的第一行并输入 yy 命令来复制第一行。

输入如下命令以切换到文件 2。

```
:buffer 2
```

现在屏幕将会展示一份文件列表，如下所示（这里只展示了一小部分）。

```
total 343700
-rwxr-xr-x 1 root root      31316 2011-12-05 08:58 [
-rwxr-xr-x 1 root root      8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 root root     111276 2012-01-31 13:36 a2p
-rwxr-xr-x 1 root root     25368 2010-10-06 20:16 a52dec
-rwxr-xr-x 1 root root     11532 2011-05-04 17:43 aafire
-rwxr-xr-x 1 root root     7292 2011-05-04 17:43 aainfo
```

将光标移动到文件的第一行并使用 p 命令将从文件 1 复制的内容粘贴到本文件。

```
total 343700
The quick brown fox jumped over the lazy dog. It was cool.
-rwxr-xr-x 1 root root      31316 2011-12-05 08:58 [
-rwxr-xr-x 1 root root      8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 root root     111276 2012-01-31 13:36 a2p
-rwxr-xr-x 1 root root     25368 2010-10-06 20:16 a52dec
-rwxr-xr-x 1 root root     11532 2011-05-04 17:43 aafire
-rwxr-xr-x 1 root root     7292 2011-05-04 17:43 aainfo
```

12.8.4 插入整个文件

用户还可以将一个文件完全插入正在编辑的文件中。为了实际演示这项功能，先结束现有的 vi 会话并重启 vi 的同时只打开一个文件。

```
[me@linuxbox ~]$ vi ls-output.txt
```

屏幕将再次显示一份文件列表。

```
total 343700
-rwxr-xr-x 1 root root      31316 2011-12-05 08:58 [
-rwxr-xr-x 1 root root      8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 root root     111276 2012-01-31 13:36 a2p
-rwxr-xr-x 1 root root     25368 2010-10-06 20:16 a52dec
-rwxr-xr-x 1 root root     11532 2011-05-04 17:43 aafire
-rwxr-xr-x 1 root root     7292 2011-05-04 17:43 aainfo
```

将光标移动到文件的第三行并输入如下 ex 命令。

```
:r foo.txt
```

命令:r (read 的缩写) 将指定的文件内容插入到光标位置之前。现在的屏幕显示如下所示。

```
total 343700
-rwxr-xr-x 1 root root      31316 2011-12-05 08:58 [
-rwxr-xr-x 1 root root      8240 2011-12-09 13:39 411toppm
The quick brown fox jumped over the lazy dog. It was cool.
Line 2
```

```

Line 3
Line 4
Line 5
-rwxr-xr-x 1 root root      111276 2012-01-31 13:36 a2p
-rwxr-xr-x 1 root root      25368 2010-10-06 20:16 a52dec
-rwxr-xr-x 1 root root     11532 2011-05-04 17:43 aafire
-rwxr-xr-x 1 root root     7292 2011-05-04 17:43 aainfo

```

12.9 保存工作

就像其他功能一样，vi 提供了很多种方式来保存编辑过的文件。前面的章节已经介绍过用于此功能的 ex 命令:w，但是还有一些其他可用的方法。

在命令模式下，输入 ZZ 将保存当前文档并退出 vi。同样的，ex 命令:wq 组合了:w 和:q 这两个命令的功能，能够保存文件并退出 vi。

当命令:w 指定一个随意的文件名时，命令的功能就类似于“另存为”。例如，用户在编辑 foo.txt 的时候想要将其另存为 fool.txt，那么就可以输入如下内容。

```
:w fool.txt
```

注意

此命令在以新名称保存文件的同时，并不更改编辑中的原文件的名称。当用户继续编辑时，编辑的还是 foo.txt 而不是 fool.txt。

第 13 章

定制提示符

本章将会讲解一个看似微不足道的细节：shell 提示符。通过讲解，我们会发现 shell 和终端仿真器程序的内部工作机制。

和 Linux 中的很多程序一样，shell 提示符的可配置性很高。尽管大多数用户并不重视提示符，但是，一旦我们学会了怎样控制它，它就会成为一种相当有用的设备。

13.1 提示符的分解

系统的默认提示符看起来如下所示。

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

可以看到提示符中包含了用户名、主机名和当前的工作目录，但是为什么提示符是这个样子的呢？很简单，提示符就是这样定义的。提示符是由名为 PS1 (prompt string 1 的缩写，即提示符字符串 1) 的环境变量定义的。echo 命令可

以帮助用户看到 PS1 的值。

```
[me@linuxbox ~]$ echo $PS1
[ \u@\h \W]\$
```

注意

如果输出的结果同本书的范例不同，也不需要担心。每一个 Linux 发行版本对此提示符字符串的定义都会有所不同，有一些甚至定义得很奇怪。

可以看出，PS1 包含了一些提示符中出现的符号，比如方括号、@ 符号和美元符号，但是其余的部分则很令人困惑。聪明的读者会将这些符号与表 7-2 中所示的由反斜杠转义的特殊字符联系起来。

表 13-1 shell 提示符中使用的转义字符

转义字符	含义
\a	ASCII 铃声。在遇到该转义字符时，计算机发出哔哔声
\d	当前日期，以星期、月、日的形式表示，如“Mon May 26”
\h	本地机器的主机名，但是不带域名
\H	完整的主机名
\j	当前 shell 会话中进行的任务个数
\l	当前终端设备的名称
\n	换行符
\r	回车符
\s	shell 程序的名称
\t	当前时间（24 小时制），格式为小时：分钟：秒
\T	当前时间（12 小时制）
\@	当前时间（12 小时制），格式为 AM/PM
\A	当前（24 小时制），格式为小时：分钟
\u	当前用户的用户名
\v	shell 的版本号
\V	shell 的版本号和发行号
\w	当前工作目录名
\W	当前工作目录名称的最后一部分
\!	当前命令的历史编号
\#	当前 shell 会话中输入的命令数
\\$	在非管理员权限下输出“\$”。在管理员权限下输出“#”
\[标志一个或多个非打印字符序列的开始。用于嵌入非打印的控制字符，使其以一定方式操纵终端仿真器，比如移动光标或更改文本颜色
\]	标志着非显示字符序列的结束

13.2 尝试设计提示符

通过这个特殊字符列表，我们可以更改提示符来查看效果。我们首先备份现有的字符串，以便过后进行恢复。为此，将现有的字符串复制到我们创建的另外一个 shell 变量中。

```
[me@linuxbox ~]$ ps1_old="$PS1"
```

这样我们就创建了名为 ps1_old 的新变量，并将 PS1 的值赋给了 ps1_old。我们可以使用 echo 命令来验证 PS1 的值确实已经被复制了。

```
[me@linuxbox ~]$ echo $ps1_old
[\u0@h ~]$
```

在终端会话中，用户随时可以通过这个过程的逆操作来复原最初的提示符。

```
[me@linuxbox ~]$ PS1="$ps1_old"
```

现在一切准备就绪。接下来让我们看看如果提示符为空会发生什么。

```
[me@linuxbox ~]$ PS1=
```

若提示符为空，那么用户不会得到任何提示。根本就没有提示字符串嘛！尽管提示符就在那里，但是系统并不会显示。这样的提示看起来很令人困惑，所以现在将提示符设置为最简略的内容。

```
PS1="\$ "
```

这样就好多了，至少现在用户知道自己在做什么了。可以注意到双引号中末尾的空格。当显示提示符时，这个空格会把美元符号和光标分隔开。

在提示符中添加一个铃声。

```
$ PS1="\a\$ "
```

这样以来，每当系统显示提示符的时候，用户都会听到哔哔声。虽然这可能会使用户感到厌烦，但是在一些情况下可能会很有帮助，比如可以在一个耗时特别长的命令执行完毕时通知用户。

接下来，我们试着创建一个信息丰富的提示符，其中包括主机名和当天的时间信息。

```
$ PS1="\A \h \$ "
17:33 linuxbox $
```

如果我们需要记录某些任务的执行时间，在提示符中添加时间信息会比较有用。最后，我们定制一个类似于最初样式的提示符。

```
17:37 linuxbox $ PS1="<\u@\h \W>\$ "
<me@linuxbox ->$
```

用户可以尝试使用表 13-1 中其他的序列，看看能不能创造出一个奇妙的新提示符。

13.3 添加颜色

大多数终端都会响应某些非打印字符序列，来控制光标位置、字符属性（如颜色、粗体、文本闪烁等）等内容。本章稍后会讲解光标位置，现在先来讲解颜色。

混乱的终端

很久以前，当终端机与远程计算机还紧密联系在一起的时候，我们有很多不同品牌的终端机，并且每一种都以不同的方式工作。这些终端的键盘不同，对控制信息的诠释方式也不同。UNIX 和类 UNIX 系统都配备有两种非常复杂的子系统（分别叫做 `termcap` 和 `terminfo`）来处理终端控制领域的混乱局面。如果你查看一下终端仿真器最底层的属性设置，可能会找到一个关于终端仿真器类型的设置。

为了使所有的终端都使用同一种通用语言，美国国家标准委员会（ANSI）开发了一套标准的字符序列，来控制视频终端。使用过 DOS 的老用户一定会记得用来启用这些代码解释的 `ANSI.SYS` 文件。

字符颜色是由发送到终端仿真器的一个 ANSI 转义代码来控制的，该转义代码嵌入到了要显示的字符流中。控制代码不会“打印”到屏幕上，而是被终端解释为一条指令。在表 13-1 中可以看到，“[”和“]”这两个序列用来封装非打印字符串。一个 ANSI 转义代码以八进制 033（该代码由转义键[escape key]产生）开始，后面跟着一个可选的字符属性，之后是一条指令。例如，将文本颜色设置为正常（`attribute = 0`）、黑色的代码是`\033[0;30m`。

表 13-2 列出了可用的文本颜色。需要注意的是，这些颜色分为两组，区别在于是否应用了粗体（`bold`）属性（1），这个属性使得色彩分为深色和浅色。

表 13-2 设置文本颜色的转义序列

字符序列	文本颜色
\033[0;30m	黑色
\033[0;31m	红色
\033[0;32m	绿色
\033[0;33m	棕色
\033[0;34m	蓝色
\033[0;35m	紫色
\033[0;36m	青色
\033[0;37m	淡灰色
\033[1;30m	深灰色
\033[1;31m	淡红色
\033[1;32m	淡绿色
\033[1;33m	黄色
\033[1;34m	淡蓝色
\033[1;35m	淡紫色
\033[1;36m	淡青色
\033[1;37m	白色

现在让我们尝试创造红色的提示符（本书中表现为灰色）。我们将相应的转义代码插入提示符的开端。

```
<me@linuxbox ->$ PS1="\[\033[0;31m\]<\u0e\h \w>\$ "
<me@linuxbox ->$
```

事实证明操作可行，但是此时用户输入的所有文字也变成红色了。要修复这个问题，可以在提示符的末尾插入另一条转义码，以通知终端仿真器恢复到原来的颜色。

```
<me@linuxbox ->$ PS1="\[\033[0;31m\]<\u0e\h \w>\$\[\033[0m\]"
<me@linuxbox ->$
```

这样就好多了。

使用表 13-3 中的代码可以设置文本的背景颜色，背景颜色不支持粗体属性。

表 13-3 设置背景颜色的转义序列

字符序列	背景颜色
\033[0;40m	黑色
\033[0;41m	红色

续表

字符序列	背景颜色
\033[0;42m	绿色
\033[0;43m	棕色
\033[0;44m	蓝色
\033[0;45m	紫色
\033[0;46m	青色
\033[0;47m	淡灰色

通过为第一个转义代码做一些修改，就可以创建带有红色背景的提示符。

```
<me@linuxbox ->$ PS1="\[\033[0;41m\]<\u@\h \W>\$[\033[0m\] "
<me@linuxbox ->$
```

用户可以尝试使用其他颜色代码，看看分别可以创造出什么样的提示符。

注意

文本除了正常（0）和粗体（1）属性外，还可以设置为下划线（4）、闪烁（5）和斜体（7）。为了维持好的品味，许多终端仿真器拒绝使用闪烁属性。

13.4 移动光标

转义代码也可以用来定位光标。比如在提示符出现的时候，这些转义代码通常用在屏幕的不同位置（比如屏幕上方的一角）显示一个时钟或其他信息。表 13-4 所示为可以定位光标的转义代码。

表 13-4 光标移动转义序列

转义码	动作
\033[l;cH	将光标移动至 l 行 c 列
\033[nA	将光标向上移动 n 行
\033[nB	将光标向下移动 n 行
\033[nC	将光标向前移动 n 个字符
\033[nD	将光标向后移动 n 个字符
\033[2J	清空屏幕并将光标移动至左上角（第 0 行第 0 列）
\033[K	清空当前光标位置到行末的内容
\033[s	存储当前光标位置
\033[u	恢复之前存储的光标位置

通过使用这些代码，用户可以构建这样的一条提示符。每当提示符出现时，屏幕的上方会绘制出一个红色的横条，横条中有用黄色文本显示的时间。用于

提示符的编码就是一个看起来很可怕的字符串：

```
PS1="\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]<\u@\h \w>\$ "
```

表 13-5 分析了这个字符串中每一部分的作用。

表 13-5 复杂提示符的分解

字符序列	动作
\[开始一个非打印字符序列。其真正目的是为了让 bash 正确计算可见提示符的长度。如果没有该字符，命令行编辑功能无法正确定位光标
\033[s	存储光标位置。在屏幕的顶部横条绘制完成并显示时间后，读取并使光标返回此位置。需要注意的是，一些终端仿真器不支持该代码
\033[0;0H	将光标移动至左上角，即第 0 行第 0 列
\033[0;41m	将背景颜色设置为红色
\033[1;33m	将光标当前位置（左上角）到行末的内容清空。因为现在背景颜色已经是红色了，所以清空后的行就是红色，也就绘制出了红色的横条。需要注意的是，清空行的内容并不会改变光标的位置，光标仍处于屏幕左上角
\033[1;33m	将文本颜色设置为黄色
\t	显示当前时间。尽管这是一个可打印的元素，但是还是将其包含在提示符非打印部分中，这是因为 bash 在计算可见提示符的长度时，不应当将其计算在内
\033[0m	关闭颜色。对文本和背景均有效
\033[u	恢复之前存储的光标位置
\]	结束非打印的字符序列
<\u@\h \w>\\$	提示符字符串

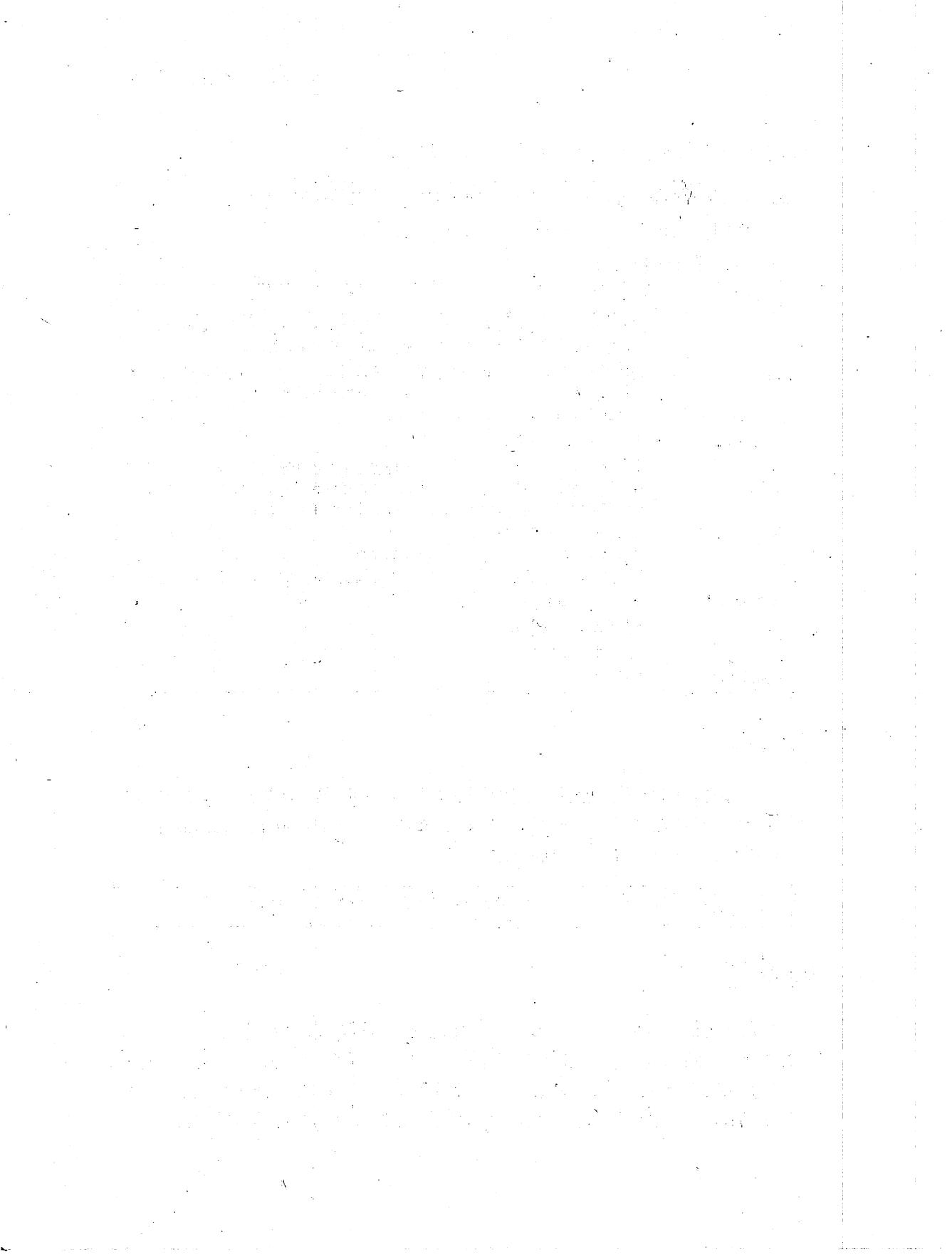
13.5 保存提示符

很显然，用户不会想要每次都输入这样长串代码，所以就需要将提示符存储在某个地方。将提示符添加到.bashrc 文件中是一个一劳永逸的解决办法，也就是将以下两行代码添加到文件中。

```
PS1="\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]<\u@\h \w>\$ "
export PS1
```

13.6 本章结尾语

无论你信是不信，还有很多事情也可以通过提示符来完成，这会涉及我们这里没有讲解到的 shell 函数和脚本，但这是一个好的开始。因为默认的提示符通常已经能让用户满意，所以并不是每个人都会想要对提示符做出修改。但是对于那些喜欢探索改进的用户来说，shell 提示符提供了很多制造琐碎乐趣的机会。



第三部分

常见任务和主要工具



第 14 章

软件包管理

如果用户经常访问 Linux 社区，那么针对众多 Linux 发行版本中哪一版是最好的这一问题，一定听到过诸多观点。通常，有关此问题的讨论都显得非常无聊，因为他们都将侧重点放在诸如哪个版本的桌面背景最漂亮（有些人居然因为 Ubuntu 的默认配色方案而不选择使用它）以及其他鸡毛蒜皮的小事上。

其实，决定 Linux 发行版本质量最重要的因素是软件包系统和支持该发行版本社区的活力。进一步接触 Linux，我们就会发现 Linux 软件的研究现状相当活跃。事物总是在不断变化，许多一流的 Linux 发行版本每 6 个月就有一个新版本问世，而且许多个人程序每天都在更新。要想同步这些日新月异的软件，我们就需要好的工具进行软件包管理。

软件包管理是一种在系统上安装、维护软件的方法。目前，很多人通过安装 Linux 经销商发布的软件包来满足他们所有的软件需求。这与早期的 Linux 形成了鲜明的对比。因为在 Linux 早期，想要安装软件必须先下载源代码，然后对其进行编译。这并不是说编译源代码不好，源代码公开恰是 Linux 吸引人

的一大亮点。编译源代码赋予用户自主检查、提升系统的能力，只是使用预先编译的软件包会更快、更容易些。

本章会介绍一些用于 Linux 软件包管理的命令行工具。虽然所有主流的 Linux 发行版本都提供了强大而复杂的维持系统运行的图形化界面操作程序，但学习命令行程序同样重要，因为它可以执行许多图形化程序很难甚至无法完成的任务。

14.1 软件包系统

不同的 Linux 发行版用的是不同的软件包系统，并且原则上，适用于一种发行版的软件包与其他版本是不兼容的。多数 Linux 发行版采用的不外乎两种软件包技术阵营，即 Debian 的.deb 技术和 Red Hat 的.rpm 技术。当然也有一些特例，比如 Gentoo、Slackware 和 Foresight 等，但多数版本采取的还是表 14-1 中所列的两个基本软件包系统。

表 14-1 主流软件包系统类

软件包系统	发行版本（只列举了部分）
Debian 类 (.deb 技术)	Debian、Ubuntu、Xandros、Linspire
Red Hat 类 (.rpm 技术)	Fedora、CentOS、Red Hat Enterprise Linux、openSUSE、Mandriva、PCLinuxOS

14.2 软件包系统工作方式

在非开源软件产业中，给系统安装一个新应用，通常需先购买“安装光盘”之类的安装介质，然后运行安装向导进行安装。

Linux 并不是这样。事实上，Linux 系统所有软件均可在网上找到，并且多数是以软件包文件的形式由发行商提供，其余则以可手动安装的源代码形式存在。本书第 23 章将会简述如何通过编译源代码安装软件。

14.2.1 软件包文件

包文件是组成软件包系统的基本软件单元，它是由组成软件包的文件压缩而成的文件集。一个包可能包含大量的程序以及支持这些程序的数据文件，包文件既包含了安装文件，又包含了有关包自身及其内容的文本说明之类的软件包

元数据。此外，许多软件包中还包含了安装软件包前后执行配置任务的安装脚本。

包文件通常由软件包维护者创建，该维护者通常（并不总是）是发行商的职员。包维护者从上游供应商（一般是程序的作者）获得软件源代码，然后进行编译，并创建包的元数据及其他必需的安装脚本。通常，包维护者会在初始源代码上做部分修改，从而提高该软件包与该 Linux 发行版本其他部分的兼容性。

14.2.2 库

虽然一些软件项目选择自己包装和分销，但如今多数软件包均由发行商或感兴趣的第三方创建。Linux 用户可以从其所使用的 Linux 版本的中心库中获得软件包。所谓的中心库，一般包含了成千上万个软件包，而且每一个都是专门为该发行版本建立和维护的。

在软件开发生命周期的不同阶段，一个发行版本可能会维护多个不同仓库。例如，通常会有一个测试库，该库里面存放的是刚创建的、用于调试者在软件包正式发布前查找漏洞的软件包。另外，一个发行版本通常还会有一个开发库，存放的是下一个公开发行的版本中所包含的开发中的软件包。

一个发行版本可能还会有相关的第三方库，这些库通常提供因法律原因，如专利或数字版权管理 (DRM) 等反规避问题而不能包括在发行版本中的软件，著名实例就是加密的 DVD 技术支持，该做法在美国不合法。第三方库主要用在软件专利和反规避法不适用的国家，这些库通常完全独立于它们所支持的 Linux 版本，用户必须充分了解后手动将其加入到软件包文件管理系统的配置文件中，才能使用它们。

14.2.3 依赖关系

几乎没有任何一个程序是独立的。与之相反，程序之间相互依赖彼此完成既定工作。一些共有的操作，比如输入/输出操作，就是由多个程序共享的例程执行。这些例程存储在共享库里面，共享库里面的文件为多个程序提供必要的服务。如果一个软件包需要共享库之类的共享资源，说明其具有依赖性。现代软件包管理系统都提供依赖性解决策略，从而确保用户安装了软件包的同时也安装了其所有的依赖关系。

14.2.4 高级和低级软件包工具

软件包管理系统通常包含两类工具——执行如安装、删除软件包文件等任

务的低级工具和进行元数据搜索及提供依赖性解决的高级工具。本章将要介绍 Debian 类型的系统（如 Ubuntu 等类似系统）所提供的软件包工具和最近的 Red-Hat 系列产品使用的工具。尽管所有 Red-Hat 系列版本都使用相同的低级工具（rpm），但使用的高级工具却不尽相同。下面我们将讨论高级软件包工具 yum 程序，它为高级 Fedora、Red Hat Enterprise Linux（红帽企业版 Linux）和 CentOS 等系统所用，而其他 Red Hat 系列的发行版本也提供功能与之相媲美的高级工具，具体见表 14-2。

表 14-2 软件包系统工具

发行版本	低级工具	高级工具
Debian 类	dpkg	apt-get、aptitude
Fedora、Red Hat Enterprise Linux、CentOS	rpm	yum

14.3 常见软件包管理任务

命令行软件包管理工具可以完成许多操作，下面我们介绍一些较常见的。有一点要说明，低级工具也支持软件包文件的创建，但不在本书的讨论范围。

在下面的讨论中，单词 package_name 指软件包的实际名称，而 package_file 则是指包含该软件包的文件名。

14.3.1 在库里面查找软件包

通过使用高级工具来搜索库元数据时，我们可以根据包文件名或其描述来查找该包。

表 14-3 包搜索命令

系统类型	命令
Debian 系统	apt-get update apt-cache search search_string
Red Hat 系统	yum search search_string

例如，在 Red Hat 系统的 yum 库中搜索 emacs 文本编辑器的代码如下。

```
yum search emacs
```

14.3.2 安装库中的软件包

高级工具允许从库中下载、安装软件包，同时安装所有的依赖包（见表

14-4)。

表 14-4 软件包安装命令

系统类型	命令行
Debian 系统	apt-get update apt-get install package_name
Red Hat 系统	yum install package_name

例如，在 Debian 系统上安装 apt 元数据库中的 emacs 文本编辑器的代码如下。

```
apt-get update; apt-get install emacs
```

14.3.3 安装软件包文件中的软件包

如果软件包文件并不是从库源中下载的，那么我们就可用低级工具直接安装（但并不安装依赖性关系），具体见表 14-5。

表 14-5 低级软件包安装命令

系统类型	命令
Debian 系统	dpkg --install package_file
Red Hat 系统	rpm -i package_file

例如，当 emacs-22.1-7.fc7-i386.rpm 软件包文件从非库资源网站下载时，可采用如下方式安装于 Red Hat 系统中。

```
rpm -i emacs-22.1-7.fc7-i386.rpm
```

注意

由于该方法采用低级 rpm 工具安装，所以并不会解决依赖性关系。一旦 rpm 在安装过程中发现缺少依赖包，rpm 就会跳出错误后退出。

14.3.4 删 除软件包

卸载软件包既可利用高级工具也可用低级工具，高级工具的相关命令见表 14-6。

表 14-6 软件包移除命令

系统类型	命令
Debian 系统	apt-get remove package_name
Red Hat 系统	yum erase package_name

例如，从 Debian 系统中卸载 emacs 软件包的代码如下。

```
apt-get remove emacs
```

14.3.5 更新库中的软件包

最常见的软件包管理任务是保持系统安装最新的软件包。高级工具仅需要一步便可完成此重要任务（见表 14-7）。

表 14-7 软件包更新命令

系统类型	命令
Debian 系统	apt-get update; apt-get upgrade
Red Hat 系统	yum update

例如，更新所有已安装于 Debian 系统中的可更新软件包的代码如下。

```
apt-get update; apt-get upgrade
```

14.3.6 更新软件包文件中的软件包

如果软件包的更新版本已从非库源中下载，那么我就可以用表 14-8 所列的命令进行安装更新从而取代原版本。

表 14-8 低级软件包更新命令

系统类型	命令
Debian 系统	dpkg --install package_file
Red Hat 系统	rpm -U package_file

例如，将 Red Hat 系统上已安装好的 emacs 程序更新为 emacs-22.1-7.fc7-i386.rpm 软件包文件中的版本的代码如下。

```
rpm -U emacs-22.1-7.fc7-i386.rpm
```

注意

与 rpm 命令不同，dpkg 命令在更新软件包时并没有指定的参数选项，只有在安装软件包时才有。

14.3.7 列出已安装的软件包列表

表 14-9 中所列出的命令用于显示系统上所有已安装的软件包列表。

表 14-9 软件包列表命令

系统类型	命令
Debian 系统	dpkg --list
Red Hat 系统	rpm -qa

14.3.8 判断软件包是否安装

表 14-10 中所列的为低级工具用于判断系统是否已安装某个软件的命令。

表 14-10 软件包状态命令

系统类型	命令
Debian 系统	dpkg --status package_name
Red Hat 系统	rpm -q package_name

例如，判断 emac 程序包在 Debian 系统中是否已安装的代码如下。

```
dpkg --status emacs
```

14.3.9 显示已安装软件包的相关信息

在已知已安装的软件包的名称的情况下，便可以用表 14-11 中的命令显示该软件包的描述信息。

表 14-11 软件包信息查看命令

系统类型	命令
Debian 系统	apt-cache show package_name
Red Hat 系统	yum info package_name

例如，查看 Debian 系统上 emac 软件包的描述信息的代码如下。

```
apt-cache show emacs
```

14.3.10 查看某具体文件由哪个软件包安装得到

表 14-12 中的命令判断某个特定的文件是由哪个软件包负责安装的。

表 14-12 查询文件所属命令

系统类型	命令
Debian 系统	dpkg --search file_name
Red Hat 系统	rpm -qf file_name

例如，查看 Red Hat 系统中哪个软件包安装了 /usr/bin/vim 目录下的文件的代码如下。

```
rpm -qf /usr/bin/vim
```

14.4 本章结尾语

在后面的章节，我们将会讨论许多应用非常广泛的程序。尽管其中多数程序由系统默认安装，但有时仍然需要我们安装一些额外的软件包。就本章中介绍的软件包管理方面的知识，我们足以安装和管理以后所需要的程序。

Linux 软件安装的神秘之处

学习使用 Linux 平台的人有时会觉得在该平台上安装软件很困难，而且不同的发行版本所用的多样化软件包策略对使用者来说也是一大障碍。确实，这是屏障，但也只是对于那些希望发布私有软件二进制版本的专有软件厂商而言。

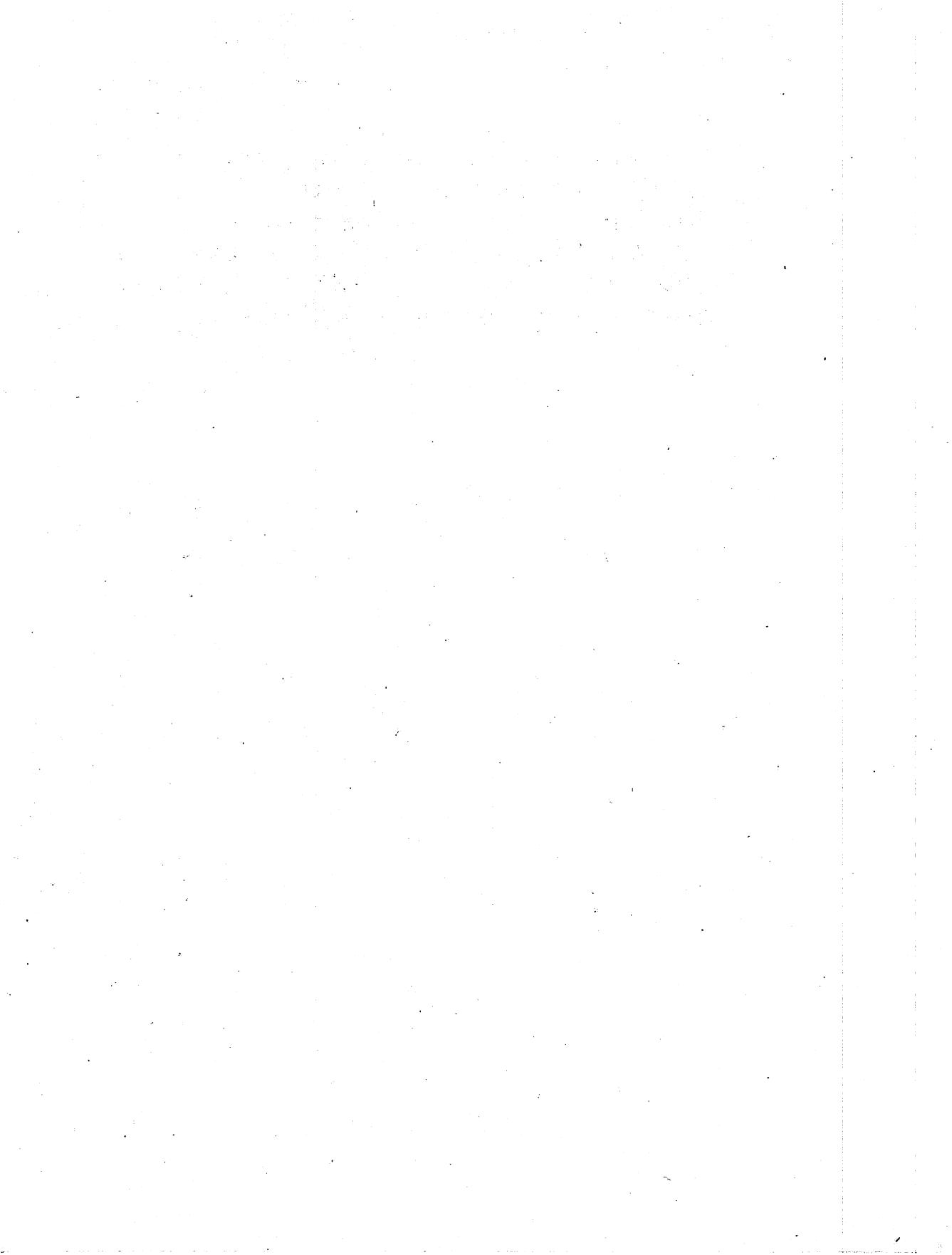
Linux 软件基于开放源代码的思想上。当程序的开发者发布了某产品的源代码后，很有可能会有一个与之相关的人打包该产品，并将其添加到该发行版的库中。这种方法可以确保该产品能够与发行版本保持很好的兼容性并且给使用者提供一个一站式的软件购买平台，而不需要搜索每个产品的网站主页。

除了那些已经加入到 Linux 内核的、不再是库中独立个体的设备驱动，其他设备驱动的处理方式都差不多。总体而言，Linux 世界中并没有“驱动光盘”这一类东西。Linux 系统很干脆，其内核要么支持该设备要么不支持。事实上，Linux 内核要比 Windows 内核支持的设备多很多。当然，如果你所需要的特定设备不被 Linux 支持，那即便比 Windows 多也无济于事。如果发生这样的情况，便需要寻找原因。设备不被支持一般是由以下三个原因造成的。

- 设备太新。由于许多硬件开发商并不积极支持 Linux 的开发，这就需要 Linux 社区的成员花时间编写该硬件设备的驱动代码。
- 设备太稀少。并不是所有的发行版本都包括了所有的设备驱动。每一个发行版本都建立了自己的内核，并且由于内核本身是可配置的（正是由于内核可配置，所以 Linux 才能在运行在从手表到大型主机的所有设备上），所以发行版本可能就会忽略了某个特定的设备。通过寻找以及下载该设备

的源代码，你完全可以自己编译、安装该驱动。这个过程虽不是那么困难，但却有点复杂。本书第 23 章将会讨论如何编译软件。

- 硬件供应商隐藏了某些东西。某些硬件既没有发布 Linux 驱动的源代码，也没有提供给他人编写驱动代码的技术文档，也就是说该硬件供应商打算保密此硬件编程接口。一般来说，使用者都不希望自己的计算机里有保密设备，所以建议移除这些硬件，并将其与其他无用文件一并放入垃圾箱。



第 15 章

存 储 介 质

前面的章节我们主要讨论了文件级别的数据处理，本章我们将会讨论设备级别的数据处理。对于诸如硬盘之类的物理存储器、网络存储器以及像 RAID（独立冗余磁盘阵列）和 LVM（逻辑卷管理）之类的虚拟存储器，Linux 都具有惊人的处理能力。

然而，本书并不是以介绍系统管理为主，所以在此我们并不打算深入探讨这个主题，我们只会简单介绍其基本概念以及用于管理存储设备的一些重要命令。

为了能够更好地练习本章中的例子，我们需要使用一个 USB 闪存、一张 CD-RW 光盘（用于可进行光盘刻录的系统）以及一张软盘（如果系统就是这么配置）。

本章将会介绍如下命令。

- `mount`: 挂载文件系统。
- `umount`: 卸载文件系统。

- **fdisk:** 硬盘分区命令。
- **fsck:** 检查修复文件系统。
- **fdformat:** 格式化软盘。
- **mkf:** 创建文件系统。
- **dd:** 向设备直接写入面向块数据。
- **genisoimage (mkisofs):** 创建一个 ISO 9600 映像文件。
- **wodim (cdrecord):** 向光存储介质写入数据。
- **md5sum:** 计算 MD5 校验码。

15.1 挂载、卸载存储设备

Linux 图形界面操作最近所取得的进展已使得图形界面操作用户能非常容易地管理存储设备。多数情况下，设备只要连接上系统就能运作。但是，过去（差不多在 2004 年），这必须通过手动操作。由于像服务器这类的非图形界面操作系统通常都有一些极致的存储需求和复杂的配置要求，所以在这类系统中管理存储设备很大程度上还是靠手动操作。

管理存储设备首先要做的就是将该设备添加到文件系统树中，从而允许操作系统可以操作该设备，这个过程称之为挂载。回忆一下第 2 章所讲的知识，类 UNIX 操作系统，与 Linux 相似，都只有一个文件系统树，设备则都连接到树的不同点上。这就与其他操作系统诸如 MS-DOS、Windows 等不同，它们对于每个设备都有独立的树（如 C:\、D:\等）。

/etc/fstab 文件内容列出了系统启动时挂载的设备（通常是硬盘分区）。下例所示的是某个 Fedora 7 系统上/etc/fstab 文件的内容。

LABEL=/12	/	ext3	defaults	1 1
LABEL=/home	/home	ext3	defaults	1 2
LABEL=/boot	/boot	ext3	defaults	1 2
tmpfs	/dev/shm	tmpfs	defaults	0 0
devpts	/dev/pts	devpts	gid=5,mode=620	0 0
sysfs	/sys	sysfs	defaults	0 0
proc	/proc	proc	defaults	0 0
LABEL=SWAP-sda3	swap	swap	defaults	0 0

此文件中列出的文件系统多数是虚拟的，不适用于当前的讨论。其前三项内容，才是我们所要关注的重点内容。

LABEL=/12	/	ext3	defaults	1 1
LABEL=/home	/home	ext3	defaults	1 2
LABEL=/boot	/boot	ext3	defaults	1 2

这三行内容其实指的是硬盘分区，文件中的每一行含 6 个字段，如表 15-1 所示。

表 15-1 /etc/fstab 文件 6 个参数含义

字段	内容	描述
1	设备	通常，该字段表示的是与物理设备相关的设备文件的真实名称，比如 dev/hda1 就代表第一个 IDE 通道上的主设备的第一块分区。但是如今的计算机都有很多可热拔插的设备（像 USB 驱动器），所以许多较新的 Linux 发行版用文本标签来关联设备。当设备与系统连接后，该标签（格式化后就会加到存储介质中）就会被操作系统识别。通过这样的方式，不管实际的物理设备被分配到哪个设备文件，它仍然能被正确识别
2	挂载节点	设备附加到文件系统树上的目录
3	文件系统类型	Linux 能够挂载许多文件系统类型，最常见的原始文件系统是 ext3，但也支持许多其他系统如 FAT16 (msdos)、FAT32 (vfat)、NTFS (ntfs)、CD-ROM (iso9660) 等
4	选项	文件系统挂载时可以使用许多选项参数，比如，可以设置文件系统以只读的方式挂载或是阻止任何程序修改它们（对于可移动设备是一个很有用的维护安全性的方法）。
5	频率	此数值被 dump 命令用来决定是否对该文件系统进行备份以及多久备份一次
6	优先级	此数值被 fsck 命令用来决定在启动时需要被扫描的文件系统的顺序

15.1.1 查看已挂载的文件系统列表

mount 命令用于文件系统挂载。不带任何参数输入该命令将会调出目前已经挂载的文件系统列表：

```
[me@linuxbox ~]$ mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda5 on /home type ext3 (rw)
/dev/sda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
/dev/sdd1 on /media/disk type vfat (rw,nosuid,nodev,noatime,
uhelper=hal,uid=500,utf8,shortname=lower)
twin4:/musicbox on /misc/musicbox type nfs4 (rw,addr=192.168.1.4)
```

列表的格式是：device on mount_point type filesystem_type (options)。例如，

上例中的第一行表示 dev/sda2 设备挂载在根目录下，可读写（后面的参数选项是 rw），属于 ext3 类型。然而，可以看到该列表的末尾有两个有趣的条目，倒数第二个条目表示 /media/disk 目录下挂载了读卡器中 2GB 的 SD 记忆卡，最后一个条目表示 /misc/musicbox 目录下挂载了一个网络驱动。

以 CD-ROM 为例，在我们插入 CD-ROM 前，首先查看一下系统信息：

```
[me@linuxbox ~]$ mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
```

该列表来自于一个使用 LVM 机制创建其根文件系统的 CentOS5 系统。与许多现代 Linux 发行版一样，此系统在 CD-ROM 插入后会自动进行挂载。光盘插入后，输入 mount 命令，便会显示如下系统信息：

```
[me@linuxbox ~]$ mount
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/hdc on /media/live-1.0.10-8 type iso9660 (ro,noexec,nosuid,nodev,uid=500)
```

与之前的信息列表相比，本列表只是在末尾处多了一个额外的条目，该条目表示 CD-ROM（本系统上的设备名是 /dev/hdc）已经挂载在了 /media/live-1.0.10-8 目录下并且是 iso9660 类型。此处只需要关注设备名，读者进行实验时，设备名很有可能就不一样。

警告

下面的例子中，要时刻注意自己使用的系统上显示的实际设备名，千万不要直接用本文中使用的名字。

同时，要注意音频 CD 与 CD-ROM 是不一样的。音频 CD 并不包含文件系统，所以通常意义上讲，音频 CD 不能被挂载。

获取 CD-ROM 的设备名之后，便可以卸载该设备，然后将其挂载在文件系统树的另外一个节点上。进行此操作，必须首先获得超级用户（使用适用于自己系统的命令切换为超级用户）权限，再使用 umount（注意拼写）命令卸载光盘。

```
[me@linuxbox ~]$ su -
Password:
[root@linuxbox ~]# umount /dev/hdc
```

接下来，我们为光盘创建一个新的挂载节点。挂载节点仅仅是文件系统上的某个目录，并没有什么特别之处，甚至都不需要是空目录，尽管如果在非空目录上挂载设备，该目录下原有内容将不可见直到此设备被卸载。作为演示，我们先创建一个新目录：

```
[root@linuxbox ~]# mkdir /mnt/cdrom
```

终于，CD 光盘挂载在了新的节点上，使用-t 选项指定文件系统类型：

```
[root@linuxbox ~]# mount -t iso9660 /dev/hdc /mnt/cdrom
```

之后，便可以通过新建的挂载节点访问 CD 光盘的内容：

```
[root@linuxbox ~]# cd /mnt/cdrom
[root@linuxbox cdrom]# ls
```

请注意，如果此时试图卸载 CD 光盘就会出现下面的问题：

```
[root@linuxbox cdrom]# umount /dev/hdc
umount: /mnt/cdrom: device is busy
```

为什么会出现这个问题？因为设备正在被某人或是某程序使用时是不能被卸载的。本例中的工作目录正好是 CD 光盘的挂载节点，所以导致了“设备繁忙”的错误警告。只要我们将工作目录改到挂载节点以外的地方就能轻松解决：

```
[root@linuxbox cdrom]# cd
[root@linuxbox ~]# umount /dev/hdc
```

如此该设备便卸载成功了。

为什么卸载如此重要

`free` 命令会输出关于存储器使用情况的一些数据，`buffer`（缓存）就包括在其中。计算机系统是以运行得尽可能快为原则设计的，阻碍计算机运行速度的一大因素就是低速设备。打印机则是一个典型的低速设备，从计算机的角度来看，即便是最快的打印机也已经是极其慢了。如果计算机必须停下来等待打印机完成一页的打印，那么计算机肯定会运行得相当慢。计算机出现早期，也就是还没有能够进行多任务处理的时期，这的确是个问题。每次打印电子表格或是文本文档时计算机就必须停止工作。计算机以打印机能够接受的最快速度向打印机传送数据，但是由于打印的速度很慢所以数据传送也很慢。打印缓冲区的问世，解决了数据传输慢的问题。打印缓冲区是存在于

计算机与打印机之间的 RAM 存储器设备，有了打印缓冲区，计算机就可以将准备发送给打印机的数据先发送给缓冲区，这个过程可以进行得很快因为 RAM 的存储速度很快，从而计算机能尽快返回去处理其他进程而不是停下来等待。与此同时，打印缓冲区再慢慢地以打印机能接受的速度从缓冲区存储中向打印机传送数据。

为了提升计算机的运行速度，缓冲区思想在计算机中得到了广泛运用。从/向较慢设备偶然性的读/写数据不再会妨碍系统的运行速度，操作系统会将那些已经从存储设备读取或是准备向存储设备写入的数据一直储存在内存中，直到这些数据确实需要与较慢设备发生交互。以 linux 系统为例，它似乎总是试图填满其内存，但这并不意味着 Linux 会用尽所有的存储器，只能说 Linux 正充分利用其可利用的一切内存进行尽可能多的缓存操作。

由于缓冲区的存在，向物理设备的写入操作被推迟到了未来某个时间，所以能快速向存储设备写入数据。与此同时，准备写入设备的数据不断向缓冲区堆积，而操作系统则不时地将这些数据写入物理设备。

卸载设备能确保缓存中的所有剩余数据全部写入设备，从而设备能被安全移除。如果设备事先没有卸载就被移除，那么缓存中就可能仍有剩余数据。有些情况下，这些未传输完的数据可能包含重要的目录更新信息，而这些信息会导致文件系统损坏，这时情况就糟透了。

15.1.2 确定设备名称

对于现在的系统，有时设备名称的确定比较困难。但是在过去，由于设备总是在一个地方固定不动，所以确定设备名并不是那么困难。类 UNIX 系统则偏爱这种方式，UNIX 开发初期，更改磁盘驱动就像用铲车将洗衣机大小的设备从机房间里面移除那样困难。近些年，典型台式机的硬件配置已经变得很灵活，而 Linux 也通过不断完善变得灵活了许多。

上例利用了现代 Linux 桌面系统的一种能力——自动挂载设备后确定设备名。但是如果操作的是一台服务器或者在不支持这样的自动挂载操作的情况下，我们那该怎么办？

要解决上述问题，让我们首先了解系统是如何命名设备的。查看/dev 目录（所有设备所在的目录）下的设备信息，我们会发现有海量的设备：

```
[me@linuxbox ~]$ ls /dev
```

`ls` 命令输出的表单内容揭示了设备命名的一些固定模式，表 15-2 列出了部分：

表 15-2 Linux 存储设备名称

模式	设备
<code>/dev/fd*</code>	软盘驱动器
<code>/dev/hd*</code>	较旧系统上的 IDE（或 PATA）硬盘。典型的主板有两个 IDE 连接点或通道，并且每个都有两个驱动器附着点。线缆上第一个驱动器叫做主设备，第二个叫做从设备。设备命名按照如下规则进行： <code>/dev/hda</code> 代表第一个通道上的主设备， <code>/dev/hdb</code> 代表第一个通道上的从设备； <code>/dev/hdc</code> 代表第二个通道上的主设备，以此类推，而末尾的数字代表设备的分区号。例如，当 <code>/dev/hda</code> 代表整个硬盘时， <code>/dev/hd1</code> 表示该硬盘驱动上的第一块分区
<code>/dev/lp*</code>	打印机设备
<code>/dev/sd*</code>	SCSI 硬盘，在最近的 Linux 系统上，内核把所有的类硬盘设备（包括 PATA/SATA 硬盘、闪存、USB 海量存储设备比如便携式音乐播放器或数码相机等）都当作 SCSI 硬盘。剩下的命名规则与上面所讲的 <code>/dev/hd*</code> 的命名规则类似
<code>/dev/sr*</code>	光驱（CD/DV 播放机和刻录机）

另外，我们经常能看到像`/dev/cdrom`、`/dev/dvd`、`/dev/floppy` 这样的符号链接，它们都是指向实际设备文件的，使用符号链接只是为了使用方便。

如果读者使用的系统不能自动挂载可移动设备，那么可以用下面介绍的方法来命名插入系统的可移动设备。首先，对`/var/log/messages` 文件进行实时查看（此操作可能需要超级用户权利）：

[me@linuxbox ~]\$ sudo tail -f /var/log/messages

文件的最后几行输出显示后停止该程序，接着插入可移动设备。以 16MB 的闪存为例，几乎在插入瞬间，内核就注意并且检测到了此设备：

```
Jul 23 10:07:53 linuxbox kernel: usb 3-2: new full speed USB device using uhci_h
cd and address 2
Jul 23 10:07:53 linuxbox kernel: usb 3-2: configuration #1 chosen from 1 choice
Jul 23 10:07:53 linuxbox kernel: scsi3 : SCSI emulation for USB Mass Storage dev
ices
Jul 23 10:07:58 linuxbox kernel: scsi scan: INQUIRY result too short (5), using
36
Jul 23 10:07:58 linuxbox kernel: scsi 3:0:0:0: Direct-Access Easy Disk 1.00 PQ:
0 ANSI: 2
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] 31263 512-byte hardware secto
rs (16 MB)
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Write Protect is off
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Assuming drive cache: write t
hrough
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] 31263 512-byte hardware secto
rs (16 MB)
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Write Protect is off
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Assuming drive cache: write t
hrough
Jul 23 10:07:59 linuxbox kernel: sdb: sdb1
```

```
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Attached SCSI removable disk
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: Attached scsi generic sg3 type 0
```

信息显示完成，按下 Ctrl-C 回到命令提示界面。查看输出信息，我们会看到很多地方重复提到[sdb]，这正好与 SCSI 类型的硬盘设备名相匹配。了解了这一点，我们就自然会额外注意下面的两行信息：

```
Jul 23 10:07:59 linuxbox kernel: sdb: sdb1
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Attached SCSI removable disk
```

以上信息告诉我们该设备名是/dev/sdb，/dev/sdb1 指的是此设备的第一个分区。正如大家所看到的，学习使用 Linux 是一项有趣的发现工作。

注意 `tail -f /var/log/messages` 命令行是用来进行实时系统监测的好方法。

获得设备名后，我们便可以挂载此闪存设备：

```
[me@linuxbox ~]$ sudo mkdir /mnt/flash
[me@linuxbox ~]$ sudo mount /dev/sdb1 /mnt/flash
[me@linuxbox ~]$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda2        15115452  5186944   9775164  35% /
/dev/sda5        59631908 31777376  24776480  57% /home
/dev/sda1        147764     17277   122858  13% /boot
tmpfs            776808       0   776808  0% /dev/shm
/dev/sdb1        15560       0   15560  0% /mnt/flash
```

只要设备一直与计算机保持连接并且系统没有重启，设备名就不会改变。

15.2 创建新的文件系统

将一个使用 FAT32 文件系统的闪存驱动器重新格式化为 Linux 本地文件系统，需要两个步骤：第一，（可选）在对现有的分区不满意的情况下创建一个新的分区布局；第二，在驱动器上创建一个新的空文件系统。

注意 下面是一个格式化闪存驱动器的例子。由于在使用过程中闪存中的所有内容都会被擦除，所以最好准备一个不含任何重要文件的驱动器。另外，要绝对确保指定的是自己系统上显示的实际设备名而不是直接用本文中使用的设备名，否则可能会格式化错误的驱动器！

15.2.1 用 fdisk 命令进行磁盘分区

`fdisk` 命令实现用户与磁盘设备（比如硬盘驱动器和闪存驱动器）进行较低

层次的直接交互。该工具可以用来编辑、删除以及创建设备分区。使用闪存前，我们必须首先将其卸载（如果需要的话）然后再启动 fdisk 程序：

```
[me@linuxbox ~]$ sudo umount /dev/sdb1
[me@linuxbox ~]$ sudo fdisk /dev/sdb
```

请注意使用 fdisk 命令指定设备时，设备名要是整个设备的而不是分区号。程序启动后，会出现下面的提示信息：

```
Command (m for help):
```

输入 m 后会出现下面的程序菜单：

```
Command action
a toggle a bootable flag
b edit bsd disklabel
c toggle the dos compatibility flag
d delete a partition
l list known partition types
m print this menu
n add a new partition
o create a new empty DOS partition table
p print the partition table
q quit without saving changes
s create a new empty Sun disklabel
t change a partition's system id
u change display/entry units
v verify the partition table
w write table to disk and exit
x extra functionality (experts only)
```

```
Command (m for help):
```

首先查看现有磁盘分区布局，可以通过输入字母 p 打印显示设备的分区表：

```
Command (m for help): p
```

```
Disk /dev/sdb: 16 MB, 16006656 bytes
1 heads, 31 sectors/track, 1008 cylinders
Units = cylinders of 31 * 512 = 15872 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		2	1008	15608+	b	W95 FAT32

可以看到此例中使用的是一个 16MB 大小的、仅有一个分区的闪存，并且只使用了可用的 1008 个磁柱（Cylinder）中的 1006 个。该分区被识别为是 Windows 95 FAT32 类型的分区，虽然有些程序会利用此身份标识来限制对磁盘可进行的操作种类，但大多数时候改不改变该标识无关紧要。然而，作为示范，我们还是需要将其改为 Linux 类型的分区。为此，首先我们得知道 Linux 分区使

用哪个身份识别 ID 码。从上表可以看出，字母 b 表示 Windows 95 FAT32 类型的分区。于是我们就需要查看有效分区类型对照表，参照先前的程序菜单，我们可以看到下面的菜单选项：

l list known partition types

按照提示输入 l，一张包含所有可能分区类型的对照表便显示出来。查表，可以看到现有的分区类型用 b 表示，而 Linux 分区类型则用 83 表示。

回到程序菜单，会看到用来改变分区 ID 的菜单选项：

t change a partition's system id

在提示命令框中输入 t 和新的 ID：

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 83
Changed system type of partition 1 to 83 (Linux)
```

完成了分区 ID 的修改，到目前为止，设备一直处于未开发状态（所有的变化都储存在了内存中而非物理设备上），所以下一步我们就该向设备写入修改后的分区表，然后退出。

在提示界面中输入 w 完成上述操作：

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
[me@linuxbox ~]$
```

如果不打算对设备做任何改动，可以输入 q，这样就可以不保存改动而退出程序了。在此过程中，我们可以毫无顾忌地忽略那些冠冕堂皇的警告信息。

15.2.2 用 mkfs 命令创建新的文件系统

分区编辑已经完成（虽然可能无足轻重），我们便可以在闪存上创建新的文件系统。mkfs（make filesystem 的缩写）命令可以用来创建各种类型的文件系统，例如我们要在设备上创建 ext3 文件系统，那就在想要格式化分区的设备名前面使用-t 参数选项指明创建的文件系统是 ext3 类型。

```
[me@linuxbox ~]$ sudo mkfs -t ext3 /dev/sdb1
mke2fs 1.40.2 (12-Jul-2012)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
3904 inodes, 15608 blocks
780 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=15990784
2 block groups
8192 blocks per group, 8192 fragments per group
1952 inodes per group
Superblock backups stored on blocks:
      8193

Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 34 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
[me@linuxbox ~]$
```

当 ext3 是指定的文件系统类型，该程序会输出许多信息。如果要将该设备重新格式化为原来的 FAT32 文件系统，则用 vfat 作为-t 的选项参数。

```
[me@linuxbox ~]$ sudo mkfs -t vfat /dev/sdb1
```

这种分区及格式化过程适用于任何有额外存储设备插入系统的时候。虽然此处仅仅用闪存作为例子讲解，但这样的过程同样适用于内部硬盘以及其他像 USB 磁盘驱动器之类的可移动存储设备。

15.3 测试、修复文件系统

前面在讨论/etc/fstab 文件的时候，我们会看到每一行的末尾有许多神秘的数字。系统每次启动时，挂载文件系统前都会惯例性地检查文件系统的完整性，此检查过程是由 fsck (filesystem check 的缩写) 程序完成的，fstab 文件每个条目末尾的数字正是对应设备的检查优先级。由上例中的 fstab 文件可知，根目录首先被检查，然后就是 home 目录和 boot 目录，末尾数字是 0 的设备表示不执行惯例性检查。

除了检查文件系统的完整性外，fsck 还能修复损坏的文件系统，修复程度取决于损坏程度。对于类 UNIX 文件系统，已修复的文件会存放在文件系统根目录下的 lost + found 目录中。

下面的命令可以用来检查闪存（闪存事先应该已卸载）

```
[me@linuxbox ~]$ sudo fsck /dev/sdb1
fsck 1.40.8 (13-Mar-2012)
e2fsck 1.40.8 (13-Mar-2012)
/dev/sdb1: clean, 11/3904 files, 1661/15608 blocks
```

依据我的经验，除非出现像磁盘驱动器不能工作这样的硬件问题，不然文件系统损坏是非常少见的。多数系统，如果在启动期间检测到文件系统损坏，系统会自动停止加载并且引导你运行 fsck 程序。

fsck 是什么

在 UNIX 文化中，fsck 通常用来取代一个与它有 3 个字母相同的流行词 (fuck)。这确实很合适，因为当你处于被迫运行 fsck 程序的情况时，你很有可能不自然地就会蹦出 fuck。

15.4 格式化软盘

对于那些仍然使用配备软盘驱动器的老式电脑的用户，同样可以管理这些软盘设备。如何准备一张空白软盘？首先，我们对软盘进行一个低级格式化操作，然后创建一个文件系统。dformat 程序后面输入指定的软盘设备名（通常是 /dev/fd0）即可完成格式化操作：

```
[me@linuxbox ~]$ sudo fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
```

接下来用 mkfs 命令为软盘创建一个 FAT 文件系统。

```
[me@linuxbox ~]$ sudo mkfs -t msdos /dev/fd0
```

请注意，这里我们使用 msdos 文件系统类型以期获得更小更早类型的文件分配表。软盘准备工作完成后，就可以像其他设备一样被挂载。

15.5 直接从/向设备转移数据

虽然，我们通常认为电脑上的数据都是以文件的形式存储的，但也有可能会认为数据以“原始”形式存储。以磁盘驱动器为例，它包含了许多被操作系

统当作目录或文件的数据“块”。如果可以简单地把磁盘驱动器当作一个大数据块集，那么我们就可以执行一些有用任务，诸如克隆设备等。

`dd` 程序可以完成这样的任务，该命令将数据块从一个地方复制到另一个地方。由于历史原因，该命令使用句法比较独特，其格式如下：

```
dd if=input_file of=output_file [bs=block_size [count=blocks]]
```

假定现在有两个容量一样的 U 盘，并且我们希望将第一个 U 盘里面的内容准确完全地复制到第二个 U 盘里面。如果这两个盘都已经连接到电脑上并且它们的设备名分别为`/dev/sdb` 和设备`/dev/sdc`，那么我们就可以用下面的命令行将第一个盘上的所有内容复制到第二个盘上：

```
dd if=/dev/sdb of=/dev/sdc
```

或者，如果只有第一个盘连接到电脑上，那么我们可以将其内容先复制到一个普通的文件里以备后续储存或复制：

```
dd if=/dev/sdb of=flash_drive.img
```

警告

`dd` 命令功能很强大。尽管该命令是由 `data definition` 两个单词缩写而来，但有时亦被称为“`destroy disk`”（摧毁磁盘），因为使用者经常把 `if` 或是 `or` 后面指定的设备名输错。一定要记住命令输入结束按 `Enter` 键前要反复检查你指定的输入输出是否与你的本意一致。

15.6 创建 CD-ROM 映像

向 CD-ROM（CD-R 或是 CD-RW）写入数据包括两个步骤：首先，创建一个 ISO 映像文件，也就是 CD-ROM 文件系统映像；其次，将此映像文件写入到 CD-ROM（只读光盘）介质中。

15.6.1 创建一个 CD-ROM 文件映像副本

如果我们想给现有的 CD-ROM 创建一个 ISO 映像，可以使用 `dd` 命令将 CD-ROM 中所有数据块复制到本地某个文件。假定我们有一张 Ubuntu 的 CD 光盘，并打算创建一个 ISO 文件以便后续制作更多副本。CD 光盘被插入且其设备名被确定后（假设是`/dev/cdrom`），我们便可以制作 ISO 文件：

```
dd if=/dev/cdrom of=ubuntu.iso
```

该方法同样适用于数据类 DVD，但不适用于音频 DVD，因为音频 DVD 并不使用文件系统实现存储。对于音频 CD，可以使用 cdrdao 命令。

用其他名字命名的程序……

如果你曾经看过创建和刻录 CD-ROM 和 DVD 之类的光学介质的在线教程，你肯定常听到 mkisofs 和 cdrecord 这两个程序。这两个程序包含在叫做 cdtools 的非常受欢迎的软件包里，Jörg Schilling 是该软件包的作者。2006 年的夏天，Schilling 先生对 cdtools 软件包中的部分内容提出了许可证变更，但是这些变更在许多 Linux 社区的人看来与 GNU 通用公共许可证不兼容。结果，cdtools 项目便开始分叉了，包括 cdrecord 和 mkisofs 程序分别被 wodim 和 genisoimage 取代。

15.6.2 从文件集中创建映像文件

genisoimage 程序通常用于创建包含一个目录内容的 ISO 映像文件。首先我们创建一个目录，该目录包含了所有我们希望加进映像文件里的文件，然后运行 genisoimage 程序创建映像文件。例如，假使事先我们已经创建了一个叫做 ~/cd-rom-files 的文件目录，并且该目录中包含了所有准备装入 CD-ROM 中的文件，那么接下来，我们就可以用 genisoimage 命令创建 ISO 映像文件，示例如下：

```
genisoimage -o cd-rom.iso -R -J ~/cd-rom-files
```

-R 选项添加了允许 Rock Ridge 延伸的元数据，此延伸允许在 Linux 中使用较长文件名的文件以及 POSIX 风格的文件。同样，-J 选项允许 Joliet 延伸，此延伸允许在 Windows 中使用较长文件名的文件。

15.7 向 CD-ROM 写入映像文件

映像文件创建好后，下一步便是将其刻录进光学介质中。下面我们所讨论的大部分命令都适用于 CD-ROM 和 DVD 介质。

15.7.1 直接挂载 ISO 映像文件

当 ISO 映像文件仍在硬盘上时，我们可以把它当作已存在于光学介质中，

并且有一个窍门可以实现该映像文件的挂载。那就是通过增加-o loop 选项来挂载（与-t iso9660 文件系统类型参数一起使用），如此便可以把映像文件当作设备一样挂载在文件系统树上了。

```
mkdir /mnt/iso_image
mount -t iso9660 -o loop image.iso /mnt/iso_image
```

在上面的例子中，我们创建了一个叫做/mnt/iso_image 挂载节点并将 image.iso 映像文件挂载在该节点上。映像文件挂载成功后，就可以把它当作真实的 CD-ROM 或 DVD。记住，当不需要该映像文件的时候要将其卸载。

15.7.2 擦除可读写 CD-ROM

可擦写 CD-ROM 在重用之前需要被擦除或清空，我们可以通过 wodim 命令指定光盘刻录机操作对象的设备名以及所要执行的擦除类型来完成。wodim 程序提供多种擦除类型，最基本的就是 fast 类型。

```
wodim dev=/dev/cdrw blank=fast
```

15.7.3 写入映像文件

同样我们使用 wodim 命令写入映像文件，通过指定写入的光介质刻录设备的名字以及映像文件的名字来完成：

```
wodim dev=/dev/cdrw image.iso
```

除了设备名和映像文件名，wodim 命令还支持很多的选项。两个比较常见的就是-v 和-dao 选项，-v 选项强调输出显示详细信息，而-dao 选项则决定光盘刻录采用一次刻录模式，这种模式一般用于光盘商业化生产。wodim 命令的默认模式是一次轨道模式，一般用于录制音乐曲目。

15.8 附加认证

通常，确认所下载的 ISO 映像文件是否完整大有必要。多数情况下，ISO 映像文件的发行商会提供一个校验和文件，校验和是通过一种奇异的数学计算得到而以数字形式表示的计算值，它代表了目标文件的内容。即便文件内容只有某个位发生了变化，校验和的结果也会有很大不同。通常我们使用 md5sum 程序生成校验和，示例如下，输出结果是一个独特的十六进制数：

```
md5sum image.iso  
34e354760f9bb7fbf85c96f6a3f94ece image.iso
```

映像文件下载结束后，你可以用 `md5sum` 命令得出其校验和，并与供应商提供的 `md5sum` 校验和数值进行比较验证。

`md5sum` 程序除了用来检查下载文件的完整性外，还可以用来验证新刻录的光学介质。检验光学介质，首先生成映像文件的校验和，然后再生成光学介质的校验和，比较这两个校验和，就可以判别映像文件是否完整地写入光学介质。该算法的关键在于如何只计算光学介质中包含该映像文件部分的校验和，常用的方法如下，计算映像文件中包含的 2K 字节块的数量（光学介质总是以 2K 字节的块为单位进行擦写），然后从光学介质中读取同样多数量的块，计算这些块的校验和。其实，有些类型的光学介质是不需要计算块数量的，像一次刻录模式刻录的 CD-R 就可以直接用下面的命令行检验：

```
md5sum /dev/cdrom  
34e354760f9bb7fbf85c96f6a3f94ece /dev/cdrom
```

但是也有许多类型的介质像 DVD 这类的，需要精确计算块的数量，示例如下。该命令行检查了 `dvd-image.iso` 映像文件的完整性以及 DVD 播放器内/`/dev/dvd` 光盘的完整性。你能弄明白该命令行的工作原理吗？

```
md5sum dvd-image.iso; dd if=/dev/dvd bs=2048 count=$(( $(stat -c "%s" dvd-image.iso) / 2048 )) | md5sum
```

第 16 章

网 络

在网络连接方面，Linux 可以说是万能的。Linux 工具可以建立各种网络系统及应用，包括防火墙、路由器、域名服务器、NAS（网络附加存储）盒等。

由于网络连接涉及的领域很广，所以用于控制、配置网络的命令自然很多。本章只着重讲一些经常用到的命令，涉及网络监测以及文件传输等方面。此外，我们还会探讨一下用于远程登录的 ssh 命令，所涉及的命令如下所示。

- **ping**: 向网络主机发送 ICMP ECHO_REQUEST 数据包。
- **traceroute**: 显示数据包到网络主机的路由路径。
- **netstat**: 显示网络连接、路由表、网络接口数据、伪连接以及多点传送成员等信息。
- **ftp**: 文件传输命令。
- **lftp**: 改善后的文件传输命令。
- **wget**: 非交互式网络下载器。

- **ssh:** OpenSSH (SSH 协议的免费开源实现) 版的 SSH 客户端 (远程系统登录命令)。
- **scp:** secure copy 的缩写, 是远程复制文件命令。
- **sftp:** secure file transfer program 的缩写, 安全文件传输程序。

接下来给大家补充一些网络方面的背景知识。在当今这个互联网时代, 每一个计算机用户都需要对网络这个概念有一个基本了解。便于更好地理解本章内容, 首先熟悉下面的术语。

- **IP (Internet protocol) address:** 互联网协议地址。
- **host and domain name:** 主机名和域名。
- **URI (uniform resource identifier):** 统一资源标识符。

注意

下面涉及到的一些命令可能需要安装额外的软件包 (取决于使用的 Linux 版本), 当然这些软件包可以从所使用的 Linux 版本的库源中下载。另外, 有些命令可能要求超级用户的权利才能执行。

16.1 检查、监测网络

即使你不是系统管理员, 经常检查网络的性能和运行情况也是有必要的。

16.1.1 ping——向网络主机发送特殊数据包

最基本的网络连接命令就是 ping 命令。ping 命令会向指定的网络主机发送特殊网络数据包 ICMP ECHO_REQUEST。多数网络设备收到该数据包后会做出回应, 通过此法即可验证网络连接是否正常。

注意

有时从安全角度出发, 通常会配置部分网络通信设备 (包括 Linux 主机) 以忽略这些数据包, 因为这样可以降低主机遭受潜在攻击者攻击的可能性。当然, 防火墙经常被设置为阻碍 ICMP 通信。

例如, 想要验证是否可以登录网站 <http://www.linuxcommand.org/> (我最喜欢登录的网站之一), 可以按如下方式使用 ping 命令。

```
[me@linuxbox ~]$ ping linuxcommand.org
```

一旦程序启动, ping 命令便以既定的时间间隔(默认是 1s) 传送数据包直到该命令被中断。

```
[me@linuxbox ~]$ ping linuxcommand.org
PING linuxcommand.org (66.35.250.210) 56(84) bytes of data.
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=1 ttl=43 time=10
7 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=2 ttl=43 time=10
8 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=3 ttl=43 time=10
6 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=4 ttl=43 time=10
6 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=5 ttl=43 time=10
5 ms
64 bytes from vhost.sourceforge.net (66.35.250.210): icmp_seq=6 ttl=43 time=10
7 ms

-- linuxcommand.org ping statistics --
6 packets transmitted, 6 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 105.647/107.052/108.118/0.824 ms
```

按 Ctrl-C 键终止 ping 程序(此时正好是本例中第 6 个数据包传输结束后), ping 程序会将反映运行情况的数据显示出来。数据包丢失 0% 表示网络运行正常, ping 连接成功则表明网络各组成成员(接口卡、电缆、路由和网关)总体处于良好的工作状态。

16.1.2 traceroute——跟踪网络数据包的传输路径

traceroute 程序(有些系统则使用功能相仿的 tracepath 程序代替)会显示文件通过网络从本地系统传输到指定主机过程中所有停靠点的列表。下例列出了数据在连接到网站 <http://www.slashdot.org/> 时所经过的站点。

```
[me@linuxbox ~]$ traceroute slashdot.org
```

输出如下内容。

```
traceroute to slashdot.org (216.34.181.45), 30 hops max, 40 byte packets
1  ipcop.localdomain (192.168.1.1)  1.066 ms  1.366 ms  1.720 ms
2  * * *
3  ge-4-13-ur01.rockville.md.bad.comcast.net (68.87.130.9)  14.622 ms  14.885
ms  15.169 ms
4  po-30-ur02.rockville.md.bad.comcast.net (68.87.129.154)  17.634 ms  17.626
ms  17.899 ms
5  po-60-ur03.rockville.md.bad.comcast.net (68.87.129.158)  15.992 ms  15.983
ms  16.256 ms
6  po-30-ar01.howardcounty.md.bad.comcast.net (68.87.136.5)  22.835 ms  14.23
3 ms  14.405 ms
```

```

    7 po-10-ar02.whitemarsh.md.bad.comcast.net (68.87.129.34) 16.154 ms 13.600
ms 18.867 ms
    8 te-0-3-0-1-cr01.philadelphia.pa.ibone.comcast.net (68.86.90.77) 21.951 ms
21.073 ms 21.557 ms
    9 pos-0-8-0-0-cr01.newyork.ny.ibone.comcast.net (68.86.85.10) 22.917 ms 21
.884 ms 22.126 ms
   10 204.70.144.1 (204.70.144.1) 43.110 ms 21.248 ms 21.264 ms
   11 cr1-pos-0-7-3-1.newyork.savvis.net (204.70.195.93) 21.857 ms cr2-pos-0-0-
3-1.newyork.savvis.net (204.70.204.238) 19.556 ms cr1-pos-0-7-3-1.newyork.sav
vis.net (204.70.195.93) 19.634 ms
   12 cr2-pos-0-7-3-0.chicago.savvis.net (204.70.192.109) 41.586 ms 42.843 ms
cr2-tengig-0-0-2-0.chicago.savvis.net (204.70.196.242) 43.115 ms
   13 hr2-tengigabitethernet-12-1.elkgrovech3.savvis.net (204.70.195.122) 44.21
5 ms 41.833 ms 45.658 ms
   14 csr1-ve241.elkgrovech3.savvis.net (216.64.194.42) 46.840 ms 43.372 ms 4
7.041 ms
   15 64.27.160.194 (64.27.160.194) 56.137 ms 55.887 ms 52.810 ms
   16 slashdot.org (216.34.181.45) 42.727 ms 42.016 ms 41.437 ms

```

由该列表可知，从测试系统到 <http://www.slashdot.org>/网站的连接需要经过 16 个路由器。对于那些提供身份信息的路由器，此列表则列出了它们的主机名、IP 地址以及运行状态信息，这些信息包含了文件从本地系统到路由器 3 次往返时间。而对于那些因为路由器配置、网络堵塞或是防火墙等原因不提供身份信息的路由器，则直接用星号行表示，如上面输出信息中的 2 号停靠站。

16.1.3 netstat——检查网络设置及相关统计数据

netstat 程序可以用于查看不同的网络设置及数据。通过使用其丰富的参数选项，我们可以查看网络启动过程的许多特性。示例如下，使用-*ie* 选项，我们可以检查系统中的网络接口信息。

```

[me@linuxbox ~]$ netstat -ie
eth0      Link encap:Ethernet HWaddr 00:1d:09:9b:99:67
          inet addr:192.168.1.2 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::21d:9bff:fe9b:9967/64 Scope:Link
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:238488 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:403217 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:100
                  RX bytes:153098921 (146.0 MB) TX bytes:261035246 (248.9 MB)
                  Memory:fdfe0000-fdfc0000

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
                  UP LOOPBACK RUNNING MTU:16436 Metric:1
                  RX packets:2208 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:2208 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:111490 (108.8 KB) TX bytes:111490 (108.8 KB)

```

以上的输出信息显示，测试系统有两个网络端口：第一个称为 eth0，是以太网端口；第二个称为 10，是系统用来自己访问自己的回环虚拟接口。

对网络进行日常诊断，关键是看能否在每个接口信息第四行的开头找到 UP 这个词以及能否在第二行的 inet addr 字段找到有效的 IP 地址。第四行的 UP 代表着该网络接口已启用，而对于使用动态主机配置协议的系统(DHCP)，inet addr 字段里面的有效 IP 地址则说明了 DHCP 正在工作。

使用 -r 选项将显示内核的网络路由表，此表显示了网络之间传送数据包时网络的配置情况。

```
[me@linuxbox ~]$ netstat -r
Kernel IP routing table
Destination     Gateway      Genmask      Flags   MSS Window irtt Iface
192.168.1.0    *           255.255.255.0 U        0 0          0 eth0  default
192.168.1.1    0.0.0.0    UG         0 0          0 eth0
```

此例显示的是运行在防火墙/路由器后面的局域网（LAN）上一客户端的典型路由表。该表的第一行表示接收方的 IP 地址为 192.168.1.0，IP 以 0 结尾表示接收方是网络而非个人主机，也就是说接收方可以是局域网（LAN）上的任何主机。下面的 Gateway 参数字段，表示的是建立当前主机与目标网络之间联系的网关（或路由）的名称或 IP 地址，此参数值是星号表示无需网关。

最后一行包含默认的接收方，这意味着所有通信都以该网络为目的地。上例中，网关被定义为 IP 地址是 192.168.1.1 的路由器，该路由器对双方通信做出最佳路径判断。

netstat 程序也有很多参数选项，而以上只列举了其中两个，要了解其整个参数列表可以查看 netstat 的 man 手册页。

16.2 通过网络传输文件

只有掌握了如何通过网络转移文件，才会明白网络的作用之大。有许多命令可以用于传送网络数据，现在我们就介绍其中的两个，后续章节将会介绍更多。

16.2.1 ftp——采用 FTP（文件传输协议）传输文件

ftp 是 Linux 比较经典的命令之一，由 File Transfer Protocol 协议缩写而来。ftp 用作下载文件工具在因特网上使用很广泛，大多数 Web 浏览器都支持 ftp 命令，读者肯定也经常遇到以 ftp:// 协议开头的 URI。

ftp 程序比 Web 浏览器出现得早，它用来与 FTP 服务器进行通信，所谓 FTP 服务器就是那些包含供网络上传、下载文件的机器。

FTP（原来的表示形式）并不安全，因为它以明文的方式传送账户名以及密码。这意味着这些信息并没有加密，任何一个接触网络的人都能看到它们。鉴于此，几乎所有使用 FTP 协议进行的网络文件传输都是由匿名 FTP 服务器处理的。匿名服务器允许任何人使用 anonymous 登录名以及无意义的密码登录。

下面是一个典型的 ftp 会话，其功能是从匿名 FTP 服务器 fileserver 上的 /pub/cd_images/Ubuntu-8.04 目录中下载 Ubuntu ISO 映像文件。

```
[me@linuxbox ~]$ ftp fileserver
Connected to fileserver.localdomain.
220 (vsFTPD 2.0.1)
Name (fileserver:me): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd pub/cd_images/Ubuntu-8.04
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-rw-r-- 1 500      500    733079552 Apr 25 03:53 ubuntu-8.04-desktop-
i386.iso
226 Directory send OK.
ftp> lcd Desktop
Local directory now /home/me/Desktop
ftp> get ubuntu-8.04-desktop-i386.iso
local: ubuntu-8.04-desktop-i386.iso remote: ubuntu-8.04-desktop-i386.iso
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for ubuntu-8.04-desktop-i386.iso
(733079552 bytes).
226 File send OK.
733079552 bytes received in 68.56 secs (10441.5 kB/s)
ftp> bye
```

表 16-1 列出了这段代码中所输入的每个命令行的含义与解释。

表 16-1 ftp 命令实例

命令	代表的含义
ftp fileserver	启动 ftp 程序，建立与 FTP 服务器 fileserver 的连接
anonymous	登录名，登录提示框出现之后就是密码输入提示框。一些服务器可以接受空白密码，其他的则要求密码以邮件地址的形式。在那样的情况下，就尝试 user@example.com 这样的格式
cd pub/cd_images/Ubuntu-8.04	打开远程系统上含有所需文件的目录。注意，对于多数匿名服务器，供公开下载的文件一般都存放在 pub 目录下