

## 第6章 命令行

在相当长的时间里，命令行参数使得 UNIX 系统高效而又灵活。偶尔到 UNIX 世界一游的旁观者经常会惊讶于一些精心组合的命令输入后所产生的效果。但是这种能力也使得 UNIX 对普通用户来说不容易理解和掌握。因为这个原因，对许多 UNIX 工具软件来说，图形化的用户界面（GUI）已经成为举足轻重的标准装备。

但是那些经验比较丰富的用户发现 GUI 工具很难提供全部的可选参数。而要想提供完备的参数，一般又会使操作界面变得与其对应的命令行程序同样复杂。GUI 的设计思路就是要简化操作，因此有经验的用户基本都会返回到命令行灵活的能力上去。

窍门 在开始争论哪一种界面更好之前，请别忘记两种类型的界面都是为了同一个目的。每一种都有它各自的缺点与优点，而最终结果是，选择掌握两种方法的人将走在前面。

在我们开始学习 Linux 操作系统的命令行操作界面之前，请理解本章并不是一个终极的教科书。我们并不打算没有任何深度地讨论全部工具，我们将深入介绍那些相信对日常工作最为关键的工具软件。

注意 就本章来说，我们假设你是以根用户身份登录进入系统并具有标准的 X-Windows 环境。如果你使用的是 GNOME（这是最新版本的 Red Hat Linux 的缺省设置），可以启动一个“X 终端”。单击屏幕左下角的 GNOME 脚印图标，选择“Utilities”（工具），然后选择“Color Xterm”（彩色 Xterm 终端）。在本章中输入的全部命令都应该输入到选择“Color Xterm”（彩色 Xterm 终端）后出现的窗口里。

### 6.1 BASH 简介

在第 5 章，我们知道了用户的口令数据项中有一个参数是登录 shell，它是用户登录上机到某个工作站之后第一个运行的程序。shell 可以被比做 Windows 中的“Program Manager”（程序管理器），其区别是 shell 的选用不再是唯一的。

shell 只不过是一个程序，提供了一个到系统的操作界面。具体到 BASH shell（Bourne Again），它是一个只提供了命令行操作的界面，包括许多内建命令，具有启动其他程序和控制从它启动的程序的能力（作业控制）。我们可以大致把它想像成 command.com 或者 .cmd 那样的程序。

存在着各种各样的 shell，大多数都有类似的功能和不同的实现方式。单纯从比较的角度看，你可以把各种 shell 想像成不同的互连网浏览器：虽然有好几种不同的浏览器，但是它们的功能是相同的——显示来自互连网的内容。在这样一种情况下，虽然每个人都宣称他们的 shell 比其他人的都好，但那只不过是个人爱好问题而已。

本小节我们将仔细研究一些 BASH 的内建命令。关于 BASH 的完整介绍本身就能够很容易地写成一本书，因此我们将重点放在对系统管理员的日常工作影响最大的那些命令上。但是，

我郑重建议你一定要好好学习 BASH 的操作。关于这个标题的书可真不缺。

### 6.1.1 作业控制

工作在 BASH 环境中的时候, 可以从同一个提示符启动多个程序。每个程序是一个“作业”。当一个作业启动的时候, 它就占据了整个“终端”(这让我们想起了过去那些日子: 那时 VT-100 和 Wyse-50 等型号的哑终端被用作主计算机的操作界面)。在如今的计算机主机上, “终端”这个词的意思或者是计算机上当开机时你看到的全文本界面, 或者是 BASH 运行于其上的由 X-Windows 相同创建的窗口。如果一个作业有对终端的控制权, 就可以发出控制码, 这样全文本界面(比如 Pine 电子邮件阅读程序)就可以更加吸引人。一旦程序运行完毕, 就又把控制权交还给 BASH, 重新显示一个提示符给用户。

但是, 也并非所有的程序都需要这类终端控制权。包括通过 X-Windows 与用户发生联系的程序在内的一些程序可以按照指令交出终端控制权, 让 BASH 提供用户提示符, 哪怕调用的程序还在运行都没有问题。在下面的例子中, Netscape 公司的 Navigator 浏览器就接到了这样一个指令, 就是那个“&”符号:

```
[ root@ford /root ] # netscape &
```

当按下回车键之后, BASH 立刻就显示出一个提示符。这叫做把作业“放入后台运行”。

如果某个程序已经在运行并且拥有终端的控制权, 你可以在终端窗口内按下 CTRL+Z 组合键让这个程序交出控制权。这将立刻暂停运行中的作业, 把控制权交给 BASH, 这时可以再输入新的命令。

任何时候都可以输入下面的命令查看 BASH 跟踪多少个作业:

```
[ root@ford /root ] # jobs
```

这个命令列出的运行中的程序可以是两种状态之一: 正在运行或者暂停。如果某个作业被暂停了, 你可以在后台启动它继续运行, 这样允许你保留着终端的控制权。或者也可以把一个暂停的程序调回前台运行, 把终端的控制权交还给程序。

在后台运行一个作业, 请输入如下所示的命令:

```
[ root@ford ] # bg number
```

其中的 number 是打算放到后台的作业号。如果想在前台运行一个作业, 请输入:

```
[ root@ford ] # fg number
```

其中的 number 是你打算调回到前台的作业号。

注意 不能够把一个需要通过终端窗口进行交互式操作的作业放到后台。

### 6.1.2 环境变量

UNIX 系统中的环境变量在概念上与 Windows NT 中的在很大程度上是一样的; 唯一的区别是它们设置、查看和删除的方法不一样。

#### 1. 查看环境变量

使用 printenv 命令可以查看用户全部的环境变量, 如下所示:

```
[ root@ford /root ] # printenv
```

如果想查看某个特定的环境变量, 把这个变量作为 printenv 命令的参数。举例来说, 下面

的命令查看环境变量 OSTYPE :

```
[ root@ford /root ] # printenv OSTYPE
```

## 2. 设置环境变量

使用下面的命令格式设置一个环境变量 :

```
[ root@ford /root ] # variable = value
```

其中的variable是变量的名称, value是打算赋给该变量的设置值。举例来说, 下面的命令把值BAR设置给环境变量FOO :

```
[ root@ford /root ] # FOO = BAR
```

赋值之后, 立刻使用export命令使之生效。export命令的格式如下所示 :

```
[ root@ford /root ] # export variable
```

其中的variable是变量的名称。还使用设置FOO的例子, 我们应该输入 :

```
[ root@ford /root ] # export FOO
```

窍门 可以把设置环境变量和使用export命令两个步骤合并在一起, 如下所示 :

```
[ root@ford /root ] # export FOO = BAR
```

如果打算设置的环境变量的值中包含有空格, 需要使用引号把变量括起来。还是使用上面的例子, 如果要把FOO设置为“Welcome to the BAR of FOO”, 我们需要输入下面的命令 :

```
[ root@ford /root ] # export FOO = " Welcome to the BAR of FOO . "
```

## 3. 取消已设置的环境变量

使用unset命令可以删除一个环境变量 :

```
[ root@ford /root ] # unset variable
```

其中的variable是打算删除的变量的名称。举例来说, 下面是删除环境变量FOO的命令 :

```
[ root@ford ] # unset FOO
```

### 6.1.3 管道

管道的机理是这样的: 通过它可以把一个程序的输出发送到另外一个程序作为输入。各自独立的程序可以一环连一环地组成功能极为强大的工具。

我们使用grep程序向大家介绍一个使用了管道的简单例子。给定一个输入流, grep命令可以从中找出与以命令参数形式给定的字符串匹配的内容行, 并把这些行显示出来。举例来说, 如果我们想要查找全部包含着“OSTYPE”字符串的环境变量, 可以输入下面的命令 :

```
[ root@ford /root ] # printenv | grep "OSTYPE"
```

命令中的垂直字符(|)代表着printenv和grep命令之间的管道。

Windows下的shell命令方式也利用了管道功能。两者最基本的区别是: 一个Linux管道中的所有命令都是同时执行的; 而Windows则是按顺序运行每一个程序, 使用临时文件保存中间结果。

### 6.1.4 重定向

通过“重定向”我们可以获得一个程序的输出, 并且立刻把它自动送入一个文件。这个

过程是由 shell 而非程序本身来完成的，这样执行重定向任务的时候就有了一个标准的办法（使用重定向与需要记忆每个程序的使用方法相比要简单得多）。

重定向有三种方式：输出到文件、附加到文件末尾和发送文件作为输入。

如果要把程序的输出收集到一个文件中，在命令的末尾加上一个大于号（>）和接受重定向输出的文件名。举例来说，下面是把对子目录进行列文件清单操作的结果输出重定向收集到一个名为 /tmp/directory-listing 中的命令：

```
[ root@ford /root ] # ls > /tmp/directory-listing
```

如果是重定向到一个现有的文件，并且打算把数据附加到文件的末尾，需要连续使用两个大于号（>>），再跟上文件名。继续使用前面文件列表的例子，我们可以把字符串“Directory Listing”加到 /tmp/directory-listing 文件的末尾，请输入下面的命令：

```
[ root@ford /root ] # echo " Directory Listing " >> /tmp/directory-listing
```

重定向的第三种方式，是使用一个文件作为输入，需要使用小于号（<），再跟上一个文件名。举例来说，下面是把 /etc/passwd 文件送入 grep 程序中的命令：

```
[ root@ford /root ] # grep 'root ' < /etc/passwd
```

### 6.1.5 BASH 的命令行快捷键

转移到一个命令后界面——特别是从类似于 command.com 这样的命令工具上转移过去——的困难之一是如何使用好一个有许多命令行快捷键的 shell。如果你不小心，这些细微之处会让你不知所措。本小节将讨论这些快捷键最常用的方法和它们的行为。

#### 1. 文件的扩展名

在诸如 BASH 之类的 UNIX 型 shell 下，命令行上的通配符是在把它们传递到应用程序“之前”展开的。这与基于 DOS 的工具软件缺省的做法形成鲜明的对照，基于 DOS 的工具软件通常需要自己进行通配符扩展。UNIX 使用的方法还意味着你在使用通配符的时候必须小心谨慎。

BASH shell 中使用的通配符本身与 command.com 中使用的完全一样：星号（\*）可以匹配任意的文件名字符，而问号（?）匹配单个的文件名字符。不管什么原因，如果你需要使用通配符作为另外一个参数的组成部分，就必须在其前面使用反斜杠字符（\）进行字符转换。这将使得 shell 把星号和问号当作正常字符而不是通配符使用。

#### 2. 环境变量作为参数

在 BASH 下，你可以使用环境变量作为命令行上的参数（虽然 command.com 也可以这么做，但是并不常见，因而这个方便法门常常被忘记）。比如说，使用参数 \$FOO 将引起环境变量 FOO 的值而不是字符串“\$FOO”被传递过去。

#### 3. 多个命令的联合使用

在 BASH shell 下，多个命令可以在同一个命令行上执行，各个命令以分号（;）隔开。举例来说，如果想在同一个命令行上执行下面的两个命令：

```
[ root@ford /root ] # ls -l
[ root@ford /root ] # cat /etc/passwd
```

可以输入下面的内容：

```
[ root@ford /root ] # ls -l ; cat /etc/passwd
```

#### 4. 反单引号

下面这个想法怎么样：你可以使用一个程序的输出结果作为另外一个程序的参数。听起来不着边际？好吧，应该慢慢习惯它——这是所有UNIX操作系统的shell都具备的最为有用和独特的功能。

反单引号（```）允许你把命令嵌入到其他命令的参数中去。你将会经常在本书中看到这样的技巧：查出代表某个文件的编号，再把这个编号作为参数传递到 `kill` 命令中去。这个方法的典型应用是用来终止 DNS 服务器 `named` 的运行。当 `named` 启动的时候，它把自己的进程 ID 号写入到文件 `/var/run/named.pid` 中去。这样，终止 `named` 进程的基本方法就是先使用 `cat` 命令从文件 `/var/run/named.pid` 中查出这个号码，再把这个数值使用到 `kill` 命令里。如下所示：

```
[ root@ford /root ] # cat /var/run/named.pid
253
[ root@ford /root ] # kill 253
```

使用这种方法终止 `named` 进程的一个问题是它不能自动地进行——需要有人先知道 `/var/run/named.pid` 文件中的那个值才能把号码告诉 `kill` 命令。另外一个问题倒不算什么真正的问题是，需要两个步骤才能终止 DNS 服务器。

使用反单引号，我们就可以把这两个步骤合二为一，并且按照能够自动执行的方式操作。使用反单引号的命令看起来应该如下所示：

```
[ root@ford /root ] # kill `cat /var/run/named.pid`
```

当 BASH 遇到这条命令的时候，它会先运行 `cat /var/run/named.pid` 并保存其结果。接着再运行 `kill` 命令，并把保存的结果传递给它。从我们的观点看，这个步骤就解决了问题。

注意 本章到目前为止，我们介绍的功能特色都是属于 BASH shell 内部的。本章接下来的部分将讨论几个可以从 BASH 外使用的常见命令。

## 6.2 文档工具

Linux 操作系统带有两个对阅读文档非常有用的工具程序：`man` 和 `info`。如今，这两种文档系统之间有许多转换工具，因为许多应用程序都在把它们的文档转换为 `info` 格式。这种格式是优于 `man` 的，因为它允许文档以类似于互连网应用的形式超连接在一起，但是又不必真的编写成 HTML 格式。

从另外一个角度看，`man` 格式已经存在几十年了。有数以千计的软件以 `man` 使用手册页作为它们唯一的文档。进一步说，还有许多应用程序继续使用 `man` 格式，因为许多其他的 UNIX 族操作系统（比如 Sun Solaris）使用着 `man` 格式。

`man` 和 `info` 文档系统都将会延续很长的时间。我强烈推荐熟练掌握这两种格式。

### 6.2.1 man 命令

在本书很早我们就已经提到 `man`（`manual` 的缩写，使用手册），使用手册页是在线查询的文档，包含着工具程序和对应配置文件的使用方法。`man` 命令的格式如下所示：

```
[ root@ford /root ] # man program_name
```

其中的 `program-name` 就是你想查询的程序。比如说：

```
[ root@ford /root ] # man ls
```

当查询关于UNIX或者与UNIX有联系的信息来源（新闻组等等）时，你可能会遇到命令索引的后面跟着应该带括号的数字——比如说ls (1)。这个数字代表着需要查阅使用手册页的第几小节（参见表 6-1）。每个小节讨论不同的主题，这是为了适应下面的事实：有些工具程序（比如printf）既是C语言中的命令，同时又是命令行命令。

表6-1 使用手册页的章节排列顺序

使用手册页小节	主 题	使用手册页小节	主 题
1	用户工具	5	配置文件
2	系统调用	6	游戏
3	C函数库调用	7	软件包
4	设备驱动程序信息	8	系统工具

如果想查阅使用手册页中某个特定的小节，只需要把小节序号作为 man命令的第一个参数、命令名称作为第二个参数就可以了。举例来说，如果想查阅关于 printf命令的C语言编程资料，可以输入下面的命令：

```
[ root@ford /root ] # man 3 printf
```

如果想看看命令行资料，可以输入下面的命令：

```
[ root@ford /root ] # man 1 printf
```

在缺省的情况下，序号最小的小节先显示出来。

但是这个资料组织方法有时候不太好用。你可能会发现对这个文档宝库使用图形化的操作界面更有帮助，GNOME项目的一个组成部分叫做 gnome-help-browser。可以在窗口屏幕底部的工具栏上找到这个操作界面的大图标、或者在屏幕左下角的菜单中找到有关的选项。如果你没有找到这个图标，随时可以使用下面的方法从一个终端窗口中启动它：

```
[ root@ford /root ] # gnome-help-browser
```

窍门 man命令有一个方便的参数 -k，用在作为参数的命令前面。这个参数让 man 命令从全部的使用手册页中查找总结性的有关资料，并且把匹配了你指定命令的那些使用手册页、以及它们的小节序号列出来，如下所示：

```
[ root@ford /root ] # man -k printf
```

## 6.2.2 texinfo系统

文档的另外一种常见的格式是 texinfo格式。作为GNU组织建立的标准，texinfo的文档系统与 World Wide Web国际互连网上超链接的格式很类似。由于文档可以超链接在一起，texinfo通常都比较容易阅读、使用和检索。

如果想阅读关于某个特定工具程序或者应用程序的 texinfo文档，需要使用info命令，并把工具程序的名称指定为参数。举例来说，要想阅读 emacs的资料，需要输入下面的命令：

```
[ root@ford /root ] # info emacs
```

一般来说，在使用info命令之前最好先看看是否存在着使用手册页（目前使用 man格式保存的信息资料要比 texinfo格式多得多）。另一方面，有些使用手册页中明确指出其对应的



texinfo文件更具权威性，应该首先阅读。

## 6.3 文件列表、所有权和访问权限

Linux中的文件管理与Windows NT中的文件管理是不同的，与Windows 95 / 98中的文件管理就更不一样了。在本小节里，我们来讨论Linux中几个基本的文件管理工具程序。我们的话题从一些通用命令开始，逐步深入到它们的内部背景中。

### 6.3.1 列出文件清单命令ls

ls命令列出一个子目录中的全部文件。它有26个命令行参数，下面列出来的是它最常用的几个。这些参数可以任意地组合使用。完整的命令行参数清单请阅读其texinfo文档。

ls命令的参数	说 明
-l	长列表。除了文件名之外，还列出文件的大小、日期/时间、访问权限、所有者以及用户分组信息
-a	全部文件。列出该子目录中所有的文件，包括隐藏文件
-l	单列列表
-R	递归地列出所有的文件和下级子目录

使用长列表方式列出某个子目录中的全部文件，使用下面的命令：

```
[ root@ford /root ] # ls -la
```

列出子目录中以字母A打头的全部非隐藏文件，使用下面的命令：

```
[ root@ford /root ] # ls A*
```

### 6.3.2 文件和子目录类型

在Linux（以及大多数UNIX）操作系统下，几乎任何东西都被归结为一个文件。这样做的目的最初是为了简化程序员的工作，代替直接与设备驱动程序进行通信的工作，人们采用特殊的文件（这些文件从应用程序方面看来与普通的文件一样）作为一个过渡桥梁。几种文件类型就可以满足全部这些文件的应用情况。

#### 1. 普通文件

普通文件就是——普通的文件。它们保存着数据和可执行程序，而操作系统本身对它们中的内容不做任何规定。

#### 2. 子目录

子目录文件是普通文件的一种特殊形态。子目录文件中保存的是其他文件的存放位置。一般说来，子目录文件中的内容对你的日常操作不会有重要的影响；除非你必须不通过现有的应用程序打开并读取这些文件的内容——这就像是试图不通过command.com而直接读取DOS的文件分配表（File Allocation Table）一样；或者类似于使用findfirst / findnext系统调用。

#### 3. 硬链接（hard link）

Linux文件系统中的一个文件都有它自己的i-结点（i-node）。每个i-结点都保存了一个文件的属性和它在硬盘上的位置。如果你需要使用两个不同的文件名代表同一个文件的话，就可以建立一个硬链接。硬链接具有与原始文件同样的i-结点，因此其外部表现和内部行为都

和原始文件一模一样。每建立一个硬链接，就给“链接计数器”增加一个值；每删除一个硬链接，链接计数器就减去一个值。在链接计数器的值减到零之前，这个文件不会从硬盘上真正被删除。

请注意：不同硬盘分区上的两个文件之间不能够建立硬链接。这是因为硬链接是通过结点指向原始文件的，而文件的 i-结点在不同的文件系统中可能会不同。

#### 4. 符号链接 (symbolic link)

与通过 i-结点指向某个文件的硬链接不一样，符号链接是通过文件名指向另外一个文件的。这就允许符号链接（经常简称为 symlinks）指向位于其他分区、甚至是其他网络硬盘上的某个文件。

#### 5. 块设备 (block device)

各种设备驱动程序都是通过文件系统进行访问的，块设备类型的文件被用做磁盘之类设备的接口。一个块设备文件有三个辨认特征：它有一个主号码、一个从号码，另外当使用 `ls -l` 命令列出它的时候，其访问权限的头一个字母是 b。如下所示：

```
[root@ford /root]# ls -l /dev/hda
brw-rw---- 1 root    disk      3,   0 May  5 1998 /dev/hda
```

请注意上面的列表中文件访问权限的头一个字母是 b；3 是主号码、0 是从号码。

一个块设备文件的主号码指明它所代表的设备驱动程序。当对这个文件进行访问的时候，从号码就被作为参数传递到设备驱动程序中去，通知它正在访问的是哪一个设备。举例来说，如果有两个串行口，它们将共享同一个设备驱动程序，也就是使用相同的主号码，但是每个串行口都有一个唯一的从号码。

#### 6. 字符设备 (character device)

与块设备类似，字符设备也是允许通过文件系统访问硬件设备的特殊文件。块设备和字符设备最明显的区别就是：块设备与真实硬件设备之间的数据交换是以数据块为单位的；而字符设备每次只能交换一个字符的数据（硬盘是一个块设备；调制解调器是一个字符设备）。字符设备的访问权限的头一个字母是 c，并且文件具有一个主号码和一个从号码，如下所示：

```
[root@ford /root]# ls -l /dev/ttyS0
crw----- 1 root    tty       4,  64 May  5 1998 /dev/ttyS0
```

#### 7. 命名管道 (named pipe)

命名管道也是一种特殊的文件类型，它用来在进程之间进行通信。使用 `mknod` 命令（这个命令在本章的后面介绍）就可以建立一个命名管道文件，一个进程可以打开它进行读操作，而另外一个进程则可以打开它进行写操作，这样就可以让这两个进程彼此进行通信。如果某个程序拒绝命名管道读取的输入，另外一个程序又必须对这个程序输出数据，而磁盘上又没有足够临时文件使用的空间，在这种情况下，命名管道正好管用。

一个命名管道文件的访问权限的头一个字母是 p。如下所示：

```
[root@ford /root]# ls -l mypipe
prw-r--r-- 1 root    root      0 Jun 16 10:47 mypipe
```



### 6.3.3 改变文件的所有权命令 chown

chown命令可以把一个文件的所有权修改为别人的。只有根用户能够进行这样的操作（普通用户不能“转移”文件的所有权或者从其他用户那里“偷得”所有权）。这个命令的格式如下所示：

```
[ root@ford /root ] # chown [-R] username filename
```

其中的username是打算分配给所有权的那个用户的登录名，filename是对之进行操作的文件的名字。filename也可以是子目录名。

当指定的filename是一个子目录的名字的时候，就需要使用 -R参数。这个参数告诉命令要递归地逐层进入这个子目录的树状结构，把新的所有权分配给这个子目录本身和其中全部的文件及下级子目录。

### 6.3.4 改变用户分组命令 chgrp

chgrp命令可以改变一个文件的用户分组设置情况。它与 chown命令很像，下面是它的格式：

```
[ root@ford /root ] # chown [-R] groupname filename
```

其中的groupname是打算分配给文件 filename所有权的那个用户分组的名称，filename也可以是子目录名。

当指定的filename是一个子目录的名字的时候，就需要使用 -R参数。类似于chown命令的情况，这个参数告诉命令要递归地逐层进入这个子目录的树状结构，把新的所有权分配给这个子目录本身和其中全部的文件及下级子目录。

### 6.3.5 改变文件属性命令 chmod

访问权限分为四个部分：

- 第一个部分就是访问权限的头一个字母。“普通”文件不具有任何特殊的值，头一个字母就使用一个短划线（-）字符。如果该文件具有特殊的属性，它就用一个字母来表示。我们在这里最感兴趣的两种特殊属性是子目录（d）和符号链接（l）。
- 访问权限的第二、三、四部分是三个字符一组的数据段。第一个部分表示的是文件所有者的访问权限；第二个部分表示的是文件所在分组的访问权限；最后一个部分表示的是全系统的访问权限。在UNIX操作系统的术语里，“全系统”意味着系统中的全部用户，不管他们的分组设置情况如何。

下面列出了用来代表访问权限的字母以及它们对应的数值。当你组合属性的时候，把它们的数值逐个相加。chmod命令用来设置访问权限的数值。

字 母	访问权限	数 值
r	读	4
w	写	2
x	执行	1

虽然chmod命令确实有更“容易”理解的访问权限格式，但是了解掌握数字方法仍然是十分重要的，因为这个原理被用来进行编程。再说，并不是每个人都使用字母命名方法。正常

的假设是：如果你知道文件的访问权限，也就应该知道这些数字的含义。

下面的内容是这三种访问权限最常见的组合形式。其他的组合，比如 `-wx`，确实存在，但是实践中极少用到。

字母组合	数 值	访问权限
---	0	无访问权限
r--	4	只读
rW-	6	读和写
rWX	7	读、写、执行
r-X	5	读和执行
--X	1	只能执行

对每一个文件来说，三个这样的三字符组组合在一起。第一个字符组表示的是文件所有者的访问权限；第二个字符组表示的是文件所在分组的访问权限；最后一个字符组表示的是系统中全部用户的访问权限。表 6-2 给出了一些常见的文件访问权限设置值。

表6-2 文件访问权限组合

访问权限	对应数值	说 明
-rw-----	600	所有者拥有读和写权限，大多数文件都有这个设置
-rw-r--r--	644	所有者拥有读和写权限；用户分组和全系统拥有只读权限。 要确定你真的想让别人读取这个文件
-rw-rw-rw-	666	人人都拥有读和写权限。不推荐这种做法，因为此组合允许任何人从系统中的任何地点访问该文件
-rwx-----	700	所有者拥有读、写和执行权限。对所有者打算运行的程序来说（从 C 或者 C++ 程序编译而来的结果文件），这是最佳的访问权限组合
-rwxr-xr-x	755	所有者拥有读、写和执行权限。其他人拥有读和执行权限
-rwxrwxrwx	777	任何人都拥有读、写和执行权限。和设置值 666 一样，这个组合必须避免使用
-rwx--x--x	711	所有者拥有读、写和执行权限。其他人只拥有执行权限。 适用于打算让其他人执行但不想让他们拷贝的程序
drwx-----	700	这是一个使用 <code>mkdir</code> 命令建立的子目录。只有所有者能够在这个子目录里进行读写操作。请注意：所有的子目录必须设置执行位 <code>x</code>
drwxr-xr-x	755	这个子目录只能够由所有者进行改动，但是其他人可以查看它的内容
drwx--x--x	711	让子目录对全系统可读，但是限制使用 <code>ls</code> 命令的访问。只有那些知道其名字的人才能对子目录中的文件进行读操作

6.4 文件管理和操作

本小节介绍用来对文件和子目录进行管理的命令行基本工具程序。大多数内容对曾经使用过命令行界面的人来说应该都是比较熟悉的。相同的功能，新的命令。

6.4.1 拷贝文件命令 cp

`cp` 命令用来拷贝文件，它有数量相当多的参数。详细资料请查阅它的使用手册页。在缺

省的情况下，这个命令工作的时候不做任何显示；只有在出现一个错误情况的时候才显示状态信息。下面是cp命令最常见的参数：

cp命令的参数	说 明
-f	强制性拷贝；不要求确认
-i	交互式拷贝；在每一个文件被拷贝之前，要求用户确认

```
把index.html拷贝为index-orig.html：
[ root@ford /root ] # cp index.html index-orig.html

交互式地把全部以.html结尾的文件拷贝到/tmp子目录中去：
[ root@ford /root ] # cp -i *.html /tmp
```

6.4.2 移动文件命令mv

mv命令用来把文件从一个位置移动到另外一个位置。文件也可以从一个分区移动到另外一个分区。这会引起拷贝操作，这样文件移动命令的用时可能会长一点。下面是 mv命令最常见的参数：

mv命令的参数	说 明
-f	强制性移动
-i	交互式移动

```
把一个文件从/usr/src/myprog/bin/*移动到拷贝为/usr/bin：
[ root@ford /root ] # mv /usr/src/myprog/bin/* /usr/bin

没有明确进行重命名存在的命令，因此我们可以使用 mv命令。如果想把/tmp/blah文件重新命名为/tmp/bleck，可以使用下面的命令：
[ root@ford /root ] # mv /tmp/bleck /tmp/blah
```

6.4.3 链接文件：ln命令

ln命令用来建立硬链接和软链接（请查阅本章前面的 3.2小节“文件和子目录类型”中的内容）。ln命令的一般格式如下所示：

```
[ root@ford /root ] # ln original_file new_file

虽然ln命令有许多参数，其中的大多数基本上都很少使用。最常用的参数是 -s，建立一个符号链接而不是一个硬链接。如果想建立一个符号链接，让 /usr/bin/myadduser指向 /usr/local/bin/myadduser，请使用下面的命令：
[ root@ford /root ] # ln -s /usr/local/bin/myadduser /usr/bin/myadduser
```

6.4.4 查找文件命令find

find命令可以根据各种检索条件查找文件。和我们已经讨论过的工具程序相类似， find命令也有许多参数，详细资料请查阅它的使用手册页。下面是 find命令的一般格式：

```
[ root@ford /root ] # find start_dir [options]
```

其中的start-dir是开始进行搜索的子目录。表 6-3列出了find命令最常用的参数。

表6-3 find命令的常用参数

find命令的参数	说 明
-mount	不搜索与搜索起点不同的文件系统
-atime <i>n</i>	至少在 <i>n</i> *24小时以内没有访问过的文件
-ctime <i>n</i>	至少在 <i>n</i> *24小时以内没有修改过的文件
-inum <i>n</i>	拥有 <i>i</i> -结点值为 <i>n</i> 的文件
-amin <i>n</i>	<i>n</i> 分钟之前访问过的文件
-cmin <i>n</i>	<i>n</i> 分钟之前修改过的文件
-empty	文件为空
-mmin <i>n</i>	<i>n</i> 分钟之前修改过的文件
-mtime <i>n</i>	<i>n</i> 小时之前修改过的文件
-nouser	文件的UID值在/etc/passwd文件中没有对应的真正用户
-nogroup	文件的GID值在/etc/group文件中没有对应的真正用户分组
-perm <i>mode</i>	文件的访问权限被准确设置为 <i>mode</i>
-size <i>n</i> [ <i>bck</i> ]	文件的长度至少为 <i>n</i> 块/字符/千字节。每块等于512字节
-print	列印找到的文件名
-exec <i>cmd</i> \;	对每一个找到的文件，运行 <i>cmd</i> 命令。重要事项：每一个 <i>cmd</i> 的后面必须跟上“\;”符号，要不然BASH会不知道如何继续操作
-name <i>name</i>	文件的名字必须是 <i>name</i> 。这里可以使用规则表达式

如果想查找/tmp子目录中至少7天没有被访问过的文件，请使用下面的命令：

```
[ root@ford /root ] # find /tmp -atime 7 -print
```

如果想找出/usr/src子目录中名字为core的文件并删除它们，请使用下面的命令：

```
[ root@ford /root ] # find /usr/src -name core -exec rm { } \;
```

如果想找出/home中以.jpg结尾并且长度超过100K的文件，请使用下面的命令：

```
[ root@ford /root ] # find /home -name "*.jpg" -size 100k
```

6.4.5 转换并拷贝文件命令dd

dd命令读出一个文件的内容再把它送到另外一个文件中去。它与 cp命令的不同之处在于，dd命令可以即时进行文件格式转换，还可以从磁带或者软驱甚至其他设备上接受数据。当 dd命令访问每个设备的时候，它不对文件系统进行任何假设，只是把设备上的数据原样读下来。因此，dd命令可以用来生成磁盘的映像，即使磁盘使用的是另类格式也没问题。

表6-4给出了dd命令最常用的参数。

如果想生成一个软盘的映像（特别适用于另类的文件格式），请使用下面的命令：

```
[ root@ford /root ] # dd if = /dev/fd0 of = /tmp/floppy_image
```

表6-4 dd命令的常用参数

dd命令的参数	说 明
<i>if</i> = <i>infile</i>	把输入文件指定为 <i>infile</i>
<i>of</i> = <i>outfile</i>	把输出文件指定为 <i>outfile</i>
<i>count</i> = <i>blocks</i>	设定dd命令在退出之前至少应该对 <i>blocks</i> 个数据块进行操作
<i>ibs</i> = <i>size</i>	把输入设备的数据块长度设置为 <i>size</i>
<i>obs</i> = <i>size</i>	把输出设备的数据块长度设置为 <i>size</i>

(续)

dd命令的参数	说 明
seek = blocks	在输出时跳过 blocks个数据块
skip = blocks	在输入时跳过 blocks个数据块
swab	把大endian的输入转换为小endian，反之亦可

6.4.6 文件压缩命令gzip

注意 gzip命令与PKZip和WinZip使用的文件格式不同；但是 WinZip可以解压缩gzip文件。

在UNIX最早的发行版本里，用来进行文件压缩的工具程序叫做 compress。但不幸的是这个算法被某些想利用它发大财的人申请了专利。为了避免付费，大多数站点寻找到另外一个非专利算法的压缩工具程序 gzip。更进一步说：gzip可以很稳定地获得比compress更好的压缩效率。

窍门 从文件的扩展名通常可以看出一个使用 gzip压缩过的文件来。这些文件一般都以.gz后缀结尾（使用compress压缩的文件以.Z结尾）。

经常与gzip一起使用的可选参数如下所示；详细清单请查阅它的使用手册页。

参 数	说 明
-c	把压缩后的文件写到 stdout标准输出（这样可以使输出经过管道送入其他程序）
-d	解压缩
-r	递归找出全部需要压缩的文件
-9	最大压缩效果
-l	最快压缩时用

请注意，gzip是“当场”对文件进行压缩的，也就是说在压缩操作结束之后，原始文件将被删除，剩下的就只有压缩好的文件了。

压缩一个文件再解压缩它，请使用下面的命令：

```
[ root@ford /root ] # gzip myfile
[ root@ford /root ] # gzip -d myfile.gz
```

使用最大压缩效果方法对全部以.html结尾的文件进行压缩，请使用下面的命令：

```
[ root@ford /root ] # gzip -9 *.html
```

建立特殊文件命令mknod

我们在前面已经介绍过，Linux操作系统通过文件来访问它的各种设备。系统中用做某个设备接口的文件必须是块或者字符类型的，还必须有主号码和从号码。为了使用必要的数值建立这一类型的文件，需要使用mknod命令。mknod命令还可以用来建立命名管道文件。

这个命令的格式是：

```
[ root@ford /root ] # mknod name type [major] [minor]
```

其中的name是文件的名称；type是对应块设备的字母b、对应字符设备的字母c、或者对应命名管道的字母p。如果建立一个块设备或者字符设备（当安装一个需要它们的设备驱动程序时），还需要再给出主号码和从号码。随那个驱动程序而来的文档中应该有主号码和从号码

的设置值。

如果想建立一个名为/tmp/mypipe的命名管道，请使用下面的命令：

```
[ root@ford /root ] # mknod /tmp/mypipe p
```

#### 6.4.7 建立子目录命令mkdir

Linux中的mkdir命令与其他UNIX操作系统中同名的命令是完全一样的，与MS-DOS中的一样。它唯一的参数是-p，使用这个参数可以在没有上级子目录的情况下完成操作。举例来说，如果想建立一个/tmp/bigdir/subdir/mydir子目录，但是当前只存在一个/tmp子目录，使用了-p参数之后就会把bigdir、subdir和mydir都自动地建立起来。

如果想建立一个名为mydir的子目录，请使用下面的命令：

```
[ root@ford /root ] # mkdir mydir
```

注意 mkdir不能像DOS中那样简化为md。

#### 6.4.8 删除子目录命令rmdir

rmdir命令对那些熟悉DOS命令的人们毫不陌生；它用来删除某个现有的子目录。它唯一的命令行参数是-p，使用这个参数可以把上级子目录一起删除掉。举例来说，如果在一个名为/tmp/bigdir/subdir/mydir的子目录中想把bigdir到mydir之间的子目录全部删除掉，可以使用下面的命令：

```
[ root@ford /root ] # rmdir -p bigdir/subdir/mydir
```

如果想删除一个名为mydir的子目录，请使用下面的命令：

```
[ root@ford /root ] # rmdir mydir
```

注意 rmdir不能像DOS中那样简化为rd。

#### 6.4.9 显示当前工作子目录命令pwd

下面这样的情况肯定会发生：你坐到一台已经登录上机的工作站前，但是不知道自己在目录树的哪一个位置。如果想获得这个信息，需要使用pwd命令。它没有参数，而它唯一的作用就是显示当前工作目录。DOS下与此对应的命令是在命令行上单独输入cd命令；但是BASH的cd命令将把你带回到你的登录子目录中去。

如果想知道当前工作子目录，请使用下面的命令：

```
[ root@ford /root ] # pwd
/usr/local/src
```

#### 6.4.10 磁带文件归档命令tar

如果你对PKZIP程序很熟悉，就会知道压缩工具可以减少文件的长度，还可以把压缩过的多个文件归档到压缩档案文件中。Linux中的这个过程需要分别使用两个工具：gzip和tar。

tar程序把多个文件合并成一个大文件。它独立于压缩工具程序，因此可以让你选择使用哪一种压缩工具或者是否需要使用压缩。另外，tar还可以与dd命令以几乎同样的方法对设备



进行读写操作，这使得tar成为备份磁带设备的良好工具。

注意 虽然在tar程序的名称中有单词tape（磁带），在建立档案文件的时候并不必须对一个磁带机进行读写操作。事实上，在日常的（备份方面）工作中，你很少会在一个磁带机上使用tar命令。

下面是tar命令的结构、它最常用的参数和几个用法示例：

```
[ root@ford /root ] # tar [ commands and options ] filename
```

tar命令的参数	说 明
-c	建立一个新的档案文件
-t	查看档案文件的内容
-x	释放档案文件的内容
-f	定义档案文件所在文件（或者设备）的名字
-v	操作过程中显示流程信息
-z	假设该文件已经（或者将要）使用 gzip进行压缩

如果想建立一个包含 /usr/src/apache子目录中全部文件的名为 apache.tar的档案文件，请使用下面的命令：

```
[ root@ford /src ] # tar -cf apache.tar /usr/src/apache
```

如果想建立一个包含 /usr/src/apache子目录中全部文件的名为 apache.tar的档案文件，并且在操作过程中显示流程信息，请使用下面的命令：

```
[ root@ford /src ] # tar -cvf apache.tar /usr/src/apache
```

如果想建立一个包含 /usr/src/apache子目录中全部文件的经过 gzip压缩的名为 apache.tar.gz的档案文件，并且在操作过程中显示流程信息，请使用下面的命令：

```
[ root@ford /src ] # tar -cvzf apache.tar.gz /usr/src/apache
```

如果想释放一个名为 apache.tar.gz的经过 gzip压缩的tar档案文件，并且在操作过程中显示流程信息，请使用下面的命令：

```
[ root@ford /root ] # tar -xvzf apache.tar.gz
```

6.4.11 合并文件命令cat

cat程序扮演了一个非常简单的角色：显示文件的内容。虽然利用它也可以实现许多有创意的东西，但是这个命令几乎总是以最简单的面目出现，被用来显示文本文件中的内容——就像DOS中的type命令一样。因为在命令行上还可以同时使用多个文件名，它也可以用来把许多文件合并成一个独立连续的大文件。它与 tar命令的不同之处是 cat命令的结果文件中不包含控制信息，不能给出不同的被合并文件之间的界线。

如果想显示/etc/passwd文件，请使用下面的命令：

```
[ root@ford /root ] # cat /etc/passwd
```

如果想显示/etc/passwd文件和/etc/group文件，请使用下面的命令：

```
[ root@ford /root ] # cat /etc/passwd /etc/group
```

如果想把/etc/passwd文件和/etc/group文件合并到文件 /tmp/complete中去，请使用下面的

命令：

```
[ root@ford /root ] # cat /etc/passwd /etc/group > /tmp/complete
```

如果要把/etc/passwd文件附加到现有的名为/tmp/orb的文件末尾，请使用下面的命令：

```
[ root@ford /root ] # cat /etc/passwd >> /tmp/orb
```

#### 6.4.12 分屏显示文件命令 more

more命令执行的情况与DOS中的同名程序差不多是一样的。它使用一个输入文件，以每次一屏的方式显示其内容。输入文件可以来自它的标准输入 stdin，也可以来自命令行参数。

这个命令的其他命令行参数虽然很少被用到，但是可以在它的使用手册页中查到。

如果想每次一屏地显示/etc/passwd文件，请使用下面的命令：

```
[ root@ford /root ] # more /etc/passwd
```

如果想每次一屏地查看由ls命令显示的子目录清单，请使用下面的命令：

```
[ root@ford /root ] # ls | more
```

#### 6.4.13 磁盘操作工具命令 du

系统管理员经常需要确定是哪个工作站上的哪个人消耗了磁盘空间，特别是当它的性能越来越低的时候！du命令能够一个接一个地检查每一个子目录的磁盘工作性能。下面是可供选用的参数：

du命令的参数	说 明
-c	在运行完毕后，对各数据项产生一个总计
-h	以易于理解的格式显示空间使用情况
-k	以千字节而不是块为单位显示空间使用情况（请注意：在Linux操作系统中，每个数据块的长度等于1K，但是并非所有的UNIX操作系统都这么规定）
-s	总结，每个变量只给出一个输出值

如果想以易于理解的格式列出/home中每一个子目录占用的空间，请使用下面的命令：

```
[ root@ford /root ] # du -sh /home/*
```

#### 6.4.14 查找文件保存在哪个子目录里命令 which

which命令在用户的全部路径中对在它命令行上给出的文件进行查找。如果找到了，命令输出中就包括了文件真实的路径名。这个命令用来查找文件所在子目录的完整路径。

如果想查找ls命令保存在哪个子目录中，请使用下面的命令：

```
[ root@ford /root ] # which ls
```

#### 6.4.15 查找命令的保存位置命令 whereis

whereis命令搜索用户的路径，给出程序的名称和它所在的子目录、源文件（如果有的话）以及该命令的使用手册页（如果有的话）。

如果想找出grep命令的程序、源程序和它的使用手册页的存放位置，请使用下面的命令：

```
[ root@ford /root ] # whereis grep
```

6.4.16 释放磁盘空间命令df

df程序给出每一个分区自由空间的总量。为了得到这些信息，欲对之操作的硬盘驱动器或者分区必须已经挂装在系统上。 NFS网络文件系统的信息也可以采用这个方法收集到。下面列出了一些df命令的参数，那些不常用的请查阅df的使用手册页。

df命令的参数	说 明
-h	以易于理解的数字而不是自由块的格式给出自由空间大小
-l	只对本地挂装的文件系统进行操作。不显示任何网络挂装文件系统的信息

如果想给出所有本地挂装的硬盘驱动器的自由空间，请使用下面的命令：

```
[ root@ford /root ] # df -l
```

如果想以易于理解的数字给出当前工作子目录所在硬盘驱动器上的自由空间，请使用下面的命令：

```
[ root@ford /root ] # df -h .
```

如果想以易于理解的数字给出/tmp子目录所在文件系统的自由空间，请使用下面的命令：

```
[ root@ford /root ] # df -h /tmp
```

6.4.17 同步磁盘命令sync

与大多数其他现代的操作系統一样，Linux使用了磁盘缓存技术以提高性能。这样做的缺点是：你打算写到磁盘上的东西并不都被及时地写到磁盘上。

如果想把缓存中的内容写到磁盘上，则需要使用 sync命令。如果sync检测到已经安排了把缓存中的内容写到磁盘上，内核就会立刻清空缓存。这个命令没有命令行参数。

如果想确保缓存中的内容立刻被写到磁盘上，请使用下面的命令：

```
[ root@ford /root ] # sync ; sync
```

6.5 进程管理

对Linux操作系统（以及UNIX操作系统）来说，每个运行中的程序至少由一个进程组成。从操作系统的立场出发，每个进程与其他进程都是彼此独立的。除非某个进程发出与其他进程共享资源的特殊请求，一般情况下它是被局限在分配给它的内存和 CPU位置上的。跨出其分配内存的进程（它们可能会引起另外一个运行程序的崩溃并使系统不稳定）将立刻被终止。管理进程的这个方法对 UNIX系统的稳定性起了非常大的作用：一个用户的应用程序不会干扰到其他用户的程序或者操作系统本身。

本小节介绍用来列出和管理进程的工具程序。对一个系统管理员的日常工作来说，它们是非常重要的元素。

6.5.1 列出进程清单命令ps

ps命令列出系统中全部的进程，包括他们的状态、大小、名称、所有者、 CPU时间、已运行时间等方面的信息。它有许多命令行参数，表 6-5中列出了其中最常用的一些。

### 表6-5 ps命令的命令行参数

ps命令的参数	说 明
-a	列出带有控制终端的全部进程，不仅仅是当前用户的进程
-r	只列出正在运行中的进程（请参考本小节后面对进程状态的说明）
-x	列出没有控制终端的那些进程
-u	列出进程的所有者
-f	给出进程之间的父/子关系
-l	按长格式显示清单
-w	显示进程的命令行参数（最多半行）
-ww	显示进程的全部命令行参数，不管其长度是多少

ps命令最常用的参数组合是 -auxww。这些参数将列出全部的进程（不管它们是否属于哪个控制终端）、每个进程的所有者、以及进程全部的命令行参数。我们来看看一个 ps-auxww的输出结果。

[illegible]

```
sshah    16675  0.0  0.8  1960 1112 pts/10   S    Jun14   0:00 -tcsh
root     18243  0.0  0.9  2144 1216 tty1      S    Jun14   0:00 login -- sshah
sshah    18244  0.0  0.8  1940 1080 tty1      S    Jun14   0:00 -tcsh
```

输出结果的第一行给出了清单的内容标题，它们是：

- USER 谁拥有这个进程。
- PID 进程的标识号码。
- %CPU 进程占用CPU的百分比。对一个多处理器系统来说，这一行数字相加的结果可能会大于100%。
- %MEM 进程占用内存的百分比。
- VSZ 进程占用虚拟内存的总量。
- RSS 进程占用真实（驻留）内存的总量。
- TTY 进程的控制终端。在这一列中出现的问号（？）意味着那个进程不再与某个控制终端相关连。
- STAT 进程的状态。

S 进程休眠中。所有准备运行的进程（即那些被安排为多任务的进程，但是CPU当前正在处理其他事情）都是休眠状态的。

R CPU正在处理的进程。

D 不可中断休眠状态（通常与输入输出有关）。

T 正在被纠错程序跟踪或者已经被终止的进程。

Z “昏迷”的进程。它的意思是：(1)父进程没有使用wait系统调用通知它的子进程的终止；(2)父进程被非正常终止；但是在该父进程完全终止之前，init进程（参考第9章）无法管理其子进程本身。一个“昏迷”的进程通常都意味着软件编写得不好。

另外，每个进程的STAT数据项还可以有如下所示的说明符：W = 内存中没有驻留页面（它已经全部交换出内存）；< = 高优先权进程；N = 低优先权进程；L = 内存页面被锁定在那里（通常就表示需要实时操作功能）。

- START 进程开始的时间。
- TIME 进程已经使用的CPU时间。
- COMMAND 进程名称和它的命令行参数。

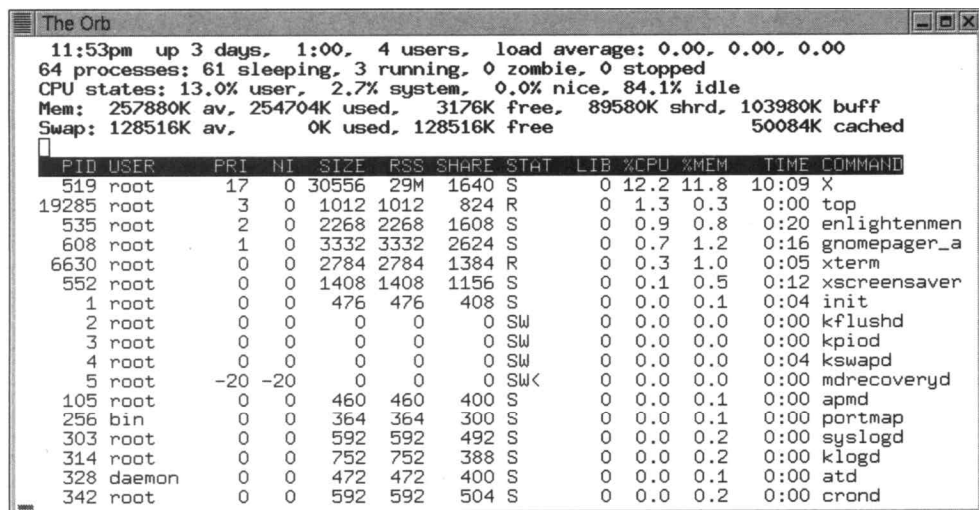
## 6.5.2 交互列出进程清单命令top

top命令是一个交互式操作的ps命令版本。它不是给出一个静态的输出，top命令每隔2-3秒钟（用户可调）就会刷新进程清单的显示画面。从这个进程清单中，用户可以重新安排优先权，或者终止它们。图6-1就是一个top命令的画面。

top命令的严重不足是它会占用CPU。在一个拥挤的系统上，这个程序会使系统管理方面的问题复杂化。用户运行top命令来查看系统上发生了什么问题，却发现另外几个用户也正使用着这个程序，这使系统变得更慢了。

缺省情况下，每个人都可以使用top命令。根据所处的操作环境，你可能会发现限制top命令只能由根用户使用会是谨慎的措施。如果想这么做，需要使用下面的命令来修改这个程序的访问权限：

```
[ root@ford /root ] # chmod 0700 /usr/bin/top
```



```
11:53pm up 3 days, 1:00, 4 users, load average: 0.00, 0.00, 0.00
64 processes: 61 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 13.0% user, 2.7% system, 0.0% nice, 84.1% idle
Mem: 257880K av, 254704K used, 3176K free, 89580K shrd, 103980K buff
Swap: 128516K av, 0K used, 128516K free 50084K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	LIB	%CPU	%MEM	TIME	COMMAND
519	root	17	0	30556	29M	1640	S	0	12.2	11.8	10:09	X
19285	root	3	0	1012	1012	824	R	0	1.3	0.3	0:00	top
535	root	2	0	2268	2268	1608	S	0	0.9	0.8	0:20	enlightenmen
608	root	1	0	3332	3332	2624	S	0	0.7	1.2	0:16	gnomepager_a
6630	root	0	0	2784	2784	1384	R	0	0.3	1.0	0:05	xterm
552	root	0	0	1408	1408	1156	S	0	0.1	0.5	0:12	xscreensaver
1	root	0	0	476	476	408	S	0	0.0	0.1	0:04	init
2	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kflushd
3	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kpiod
4	root	0	0	0	0	0	SW	0	0.0	0.0	0:04	kswapd
5	root	-20	-20	0	0	0	SW<	0	0.0	0.0	0:00	mdrecoveryd
105	root	0	0	460	460	400	S	0	0.0	0.1	0:00	apmd
256	bin	0	0	364	364	300	S	0	0.0	0.1	0:00	portmap
303	root	0	0	592	592	492	S	0	0.0	0.2	0:00	syslogd
314	root	0	0	752	752	388	S	0	0.0	0.2	0:00	klogd
328	daemon	0	0	472	472	400	S	0	0.0	0.1	0:00	atd
342	root	0	0	592	592	504	S	0	0.0	0.2	0:00	crond

图6-1 执行中的top程序

### 6.5.3 向某个进程发送消息命令 kill

这个程序的名字有一些误导：其实它并不真的“杀死”进程。它的作用是向正在运行的进程发送消息。缺省情况下，操作系统会为每个进程提供一套标准的“信号处理句柄”（signal handlers）来处理接收到的信号。从一个系统管理员的立场来看，最重要的是处理编号为9和15的信号句柄，它们分别是终止进程和中断进程运行。调用 kill 的时候，最少需要一个参数：从 ps 命令获得的进程标识号码（PID）。如果只有一个 PID 参数，kill 就发出信号 15，“中断进程运行”。有些程序截获这个信号并执行一系列的操作，让它们自己能够顺利地关闭。另外一些只是停止在运行的过程中。不管哪一种情况，kill 都不是一个确保进程停止的方法。

#### 1. 信号

kill 命令的可选参数是 -n，其中的 n 是信号的编号。作为一名系统管理员，我们最感兴趣的是信号 9（中断程序运行）和 1（挂起）。

kill 的信号 9 是一个不礼貌的终止进程运行的方法。操作系统收到这个信号后，不是发一个请求去停止进程的运行，而是简单地终止这个进程的运行。只有当进程正处于系统调用（比如打开一个文件的操作过程中）的时候，这个方法才不能立刻奏效；但是当那个进程从系统调用中一返回，也就停止运行了。

挂起信号 1 多少令人想起使用 VT100 终端的 UNIX 早期时代。如果在一个任务的执行过程中用户的终端连接掉了线，那么这个终端上全部运行着的进程都会收到一个挂起信号（一般叫做 SIGHUP 或者 HUP 信号）。这就给那些进程一个顺利关闭的机会；而那些后台进程一般都会忽略这个信号。发展到今天，HUP 信号用来通知某些服务器应用程序重新读它们的配置文件（本章的后面有这样的实例）。大多数应用程序一般会简单地忽略这个信号。

#### 2. kill 的安全性

能够中断进程运行的能力明显是很强大的，这就更加突出了安全预警的重要性。如果非根用户试图向一个不属于他的进程发送信号，则会看到返回的错误提示。根用户是这种限制的例外；根用户可以对系统中的全部进程发送信号。这当然就意味着在使用 kill 命令的时候，



根用户必须非常谨慎。

3. 示例

如果想中断第2059号进程，请使用下面的命令：

```
[ root@ford /root ] # kill 2059
```

如果想“比较有把握”地中断第593号进程，请使用下面的命令：

```
[ root@ford /root ] # kill -9 593
```

如果想向init程序（它的进程号永远是1）发送一个HUP信号，请使用下面的命令：

```
[ root@ford /root ] # kill -1 1
```

6.6 其他工具

下面将要介绍的工具程序不属于我们在本章前面介绍的任何类别。但是它们对日常的系统管理工作都具有很重要的作用。

6.6.1 显示系统名称命令 `uname`

`uname`程序给出一些系统细节信息，在某些场合是很有帮助的。可能你花了好大功夫远程登录到十多个不同的计算机上，但是搞不清自己在什么位置了。这个工具程序对命令脚本程序的编写人员也很有帮助，因为它可以让他们根据系统信息切换命令脚本程序的路径。

下面是`uname`命令的命令行参数：

uname命令的参数	说 明
-m	给出机器的硬件类型（比如“i686”表示Pentium Pro或者更好的体系结构）
-n	给出机器的主机名
-r	给出操作系统的发行名称
-s	给出操作系统的名称
-v	给出操作系统的版本
-a	列出以上全部的信息资料

如果想查看操作系统的名称和发行版本，请使用下面的命令：

```
[ root@ford /root ] # uname -s -r
```

注意 `-s`参数看起来没有什么用处（我们都知道这是Linux操作系统），但是这个参数在大多数UNIX类的操作系统中确实是很有用的。在SGI工作站上，`uname -s`命令返回的是IRIX；在Sun工作站上，返回的是SunOS。在多操作环境中工作的人们编写的命令脚本程序通常都会根据操作系统的不同而采取不同的操作；而带`-s`参数的`uname`命令就是确定此信息的稳妥办法。

6.6.2 查看用户命令 `who`

在允许用户登录进入到其他用户的计算机或者特殊目的的服务器主机上的系统时，系统管理员需要了解都有哪些人正在上机。可以使用`who`命令产生这样一个报告：

```
[ root@ford /root ] # who
```

`who`命令的报告看起来如下所示：

```
sshah    tty1      Jun 14 18 : 22
root     pts/9     Jun 14 18 : 29 ( :0 )
root     pts/11    Jun 14 21 : 12 ( :0 )
root     pts/12    Jun 14 23 : 38 ( :0 )
```

### 6.6.3 改变用户身份命令 su

当你以某个用户的身份登录进入到系统中之后，如果需要模拟另外一个身份（比如说根用户身份），可以不必经历退出再重新登录这样的麻烦。使用 `su` 命令就可以切换身份了。这个命令只有两个命令行参数，而且都是可选的。

不带任何参数直接执行 `su` 命令将自动尝试将你改变为根用户。系统会提示你输入根用户的口令，如果回答得正确，就会进入到根用户的 `shell` 中。如果你就是根用户，想切换到其他的用户身份，当使用这个命令的时候不必输入任何新的口令。

举例来说，如果你是以自己的身份登录进入系统的，现在想切换为根用户，需要输入下面的命令：

```
[ sshah@ford ~ ] $ su
```

如果你是以根用户的身份登录进入系统的，现在想切换为其他用户——比如说用户 `sshah`，需要输入下面的命令：

```
[ root@ford /root ] # su sshah
```

可选用的波浪号（`~`）参数告诉 `su` 命令切换身份后还需要运行新用户身份的登录脚本程序。举例来说，如果你是以根用户的身份登录进入系统的，现在想切换为用户 `sshah`，并且希望具备这个用户全部的登录和 `shell` 配置情况的话，需要输入下面的命令：

```
[ root@ford /root ] # su - sshah
```

### 6.6.4 编辑器程序

编辑器程序可以说是常用工具程序中种类最多的了，同时它们也可以说是最有用的。如果没有它们，对文本文件的任何修改都会变得难上加难。不管你使用的是哪一种 Linux 发行版本，其中都会带有好几种编辑器程序。在你开始对付其他问题之前，应该花些功夫去熟悉它们。

注意 不同的发行版本不一定都包括这里介绍的所有编辑器程序。

#### 1. vi 编辑器

`vi` 编辑器程序从 70 年代开始就已经出现在基于 UNIX 操作系统的系统中了，它具有独特的编辑操作界面。也许它可以算得上是最后一个使用分开的命令状态和数据输入状态的编辑器程序了，而其结果是：许多新手发现它不那么便于使用。但是在你拒绝它之前，最好还是先花点时间熟悉熟悉它。在某些复杂的情形中，你手边可能不会有一个图形化的好编辑器，而 `vi` 在各种 UNIX 操作系统的用法都是一致的。

Linux 发行版本中的 `vi` 编辑器的版本是 VIM（“VI iMproved”，改进型 `vi` 编辑器的简写）。它有许多使 `vi` 广受欢迎的东西，还有许多使 `vi` 在当今典型的操作环境中大显身手的功能（如果运行 X-Windows 的话，则还会有一个图形化的界面）。

如果想启动 `vi` 编辑器程序，只需简单地输入：

```
[ root@ford /root ] # vi
```

学习使用vi编辑器程序最简单的方法是启动它，再输入“:help”。

## 2. emacs编辑器

有人曾经争论过emacs本身就是一个操作系统。它规模庞大、功能丰富、可以扩展、可以编程、如此种种令人惊叹。如果你原来习惯于使用 GUI，那肯定会认为emacs是一个开始工作的好环境。表面上看，它的工作界面就像 Windows中的Notepad（记事本）；而其内涵则是一个完整的GNU开发环境接口、一个电子邮件阅读器、一个新闻阅读器、一个网络浏览器，甚至是一位心理分析专家（当然不那么专业）。

如果想启动emacs，需要输入下面的命令：

```
[ root@ford /root ] # emacs
```

emacs启动之后，如果你想访问哪位心理分析专家，先按下 Esc+X组合键，再输入doctor就可以了。如果想查看使用emacs的帮助信息，请按下Ctrl+H组合键。

## 3. joe编辑器

我们在这里讨论的编辑器程序当中，joe最像是一个简单的文本编辑器。它工作的时候就象是Windows中的Notepad（记事本）程序，并且在屏幕上提供了操作帮助。那些还记得最早时期WordStar命令集的人们一定会高兴地发现那些 Ctrl+K系列组合键命令还可以用在joe编辑器中。

如果想启动joe，需要输入下面的命令：

```
[ root@ford /root ] # joe
```

## 4. pico编辑器

pico是另外一个以简单性著称的编辑器程序。它一般用在 pine电子邮件系统中，但是也可以用作一个独立的编辑器程序。类似于 joe编辑器，pico工作的时候也很像是 Windows中的Notepad（记事本）程序，但是它使用了自己的一套组合键命令集。而且全部组合键总是显示在屏幕的底部。

如果想启动pico，需要输入下面的命令：

```
[ root@ford /root ] # pico
```

## 6.7 小结

在本章中，我们讨论了Linux操作系统中BASH shell的命令行操作界面。随着进一步的学习，你将会发现有越来越多与这一章内容有联系的东西，因此请一定要努力掌握好命令行的使用方法。刚开始的时候可能会有些不习惯，特别是如果你已经习惯于使用一个 GUI来完成本章中介绍的大多数基本的任务。但是一定要坚持下去，最终你可能会发现自己使用命令行进行操作要比通过GUI还要快！

很明显，本章不可能讨论到包括在你缺省 Linux安装中的全部命令行工具程序。我强烈推荐花些时间阅读一些给出关于这个主题详细文档的参考书籍。另外，还有大量关于 shell编程方面的文字，涉及各种不同的水平和不同的视角。选择适合你的东西，即使你不负责系统管理方面的工作，对shell进行编程也是一个值得好好学习的技巧。