第6学时　使　用　shell

本学时教程介绍shell。虽然最近10年来个人计算的潮流从命令行转向了鼠标界面，但是shell在Linux操作系统中的生命力依然很旺盛，并且为众多的 Linux操作系统程序员所使用。

本学时教程不打算探讨 Linux操作系统或者 UNIXshell的历史，也不打算加入到哪一种计算界面更容易或者更好使用的讨论中。本学时教程也不打算深入探讨　shell的编程 —— 因为本学时教程(甚至本书)都没有足够的篇幅这么做。本学时教程只讨论那些可以在本书所附的　CD-ROM光盘中找到的各种不同的shell并介绍这些shell的使用方法，让你在Linux操作系统中的工作更有乐趣，并集中讨论使用 shell以及命令行的一些基础知识。

6.1　什么是shell

如果现在还使用X窗口系统，那么shell就是使用到的最重要的程序之一。 shell提供了一个到Linux操作系统的界面以方便运行程序。事实上，　shell也只不过是另外一个 Linux操作系统程序而已。

虽然没有shell也可以运行Linux操作系统(请阅读第 21学时“ 系统监管基础 ”)，但是即使使用的是图形化接口的程序，仍可能会发现需要使用 shell的命令行。

shell是一个命令解释器，它可以用来启动、挂起、停止甚至编写程序。 shell是Linux操作系统的一个整体组成部分，也是 Linux操作系统和UNIX设计的一部分。如果把 Linux操作系统的内核想象成一个球体的中心，那么 shell就是包围内核的外层。从 shell或其他程序向Linux操作系统传递命令的时候，内核就会做出相应的反应。

有许多类型的 shell，但是至少有 5种(不包括图形化的 shell，如Midnight Commander等，请阅读第5学时教程“ 操作和搜索命令 ”)在 CD-ROM光盘上。可以查看 /etc/passwd文件的内容或者根据用户名查询文件来决定在登录进入到 Linux操作系统的时候想使用哪种 shell。请看下面的例子：

```
# fgrep bball etc/passwd
bball : x : 100 : 100 : William H. Ball , , , , : /home/bball : /bin/bash
```

shell被列在passwd文件数据项的末尾部分(在上面使用 fgrep命令的例子中就是/bin/bash)。

6.2　系统中都有哪几种shell

为了帮助你开始学习，本小节列出了 OpenLinux操作系统上可以使用的全部的 shell以及它们各自的特色。这些 shell中的每一个都可以运行其他的程序，而你也许正想探讨研究它们工作的原理。本小节还强调了它们彼此之间的区别，并搞清楚哪些是重要的配置文件。

所有这些shell都通过一个内建的 cd子目录切换命令支持切换子目录的操作。有趣的是 ash和tcsh这两种shell因为没有内建的pwd显示工作子目录命令而必须依靠存放在 /bin目录中的pwd命令。

第一次安装 OpenLinux操作系统的时候，应该已经选择了在OpenLinux操作系统任务操作中使用某个特定的 shell。如果你想改变这个缺省的shell，可以使用chsh命令，后面加上你想使用的 shell的完整的路径名。例如，如果你想把缺省的 shell改为bash，请按照下面的样子使用 chsh命令和它的-s参数：

```
# chsh -s /bin/bash
```

这些shell如表6-1的清单中所示，它们各有各的特性。请阅读它们各自的使用手册页来了解它们各自的详细情况。一些值得注意和研究的特色开列如下：

- •.这种shell的内建命令都有哪些？
- •.怎样进行任务控制(也就是后台处理，将在本学时教程后面的“ 在后台运行程序 ”一节中讨论)？
- •.这种shell是否支持命令行编辑？
- •.这种shell是否支持命令行历史记录？
- •.什么是它的重要的开机启动文件或者配置文件？
- •.各个shell的重要环境变量有哪些？
- •.可以使用什么样的命令行提示符？
- •.它支持什么样的编程框架？

OpenLinux操作系统中还包括其他几种 shell：tclsh，一个简单的shell程序和Tcl解释器；wish，一种用于X11的窗口shell程序；scotty，一个Tcl解释器和shell程序；rsh，一个用来通过网络运行命令的远端 shell。如果想了解它们的详细资料，请查阅 tclsh、wish、scotty和rsh的使用手册页。

表6-1 OpenLinux操作系统中包括的shell

名 称	说 明
ash	袖珍的sh兼容的shell
bash	Bourne Again Shell(与ksh和sh兼容)
csh	对tcsh的一个符号链接
ksh	pdksh，公共域Korn(与ksh兼容)shell
sh	对bash的一个符号链接
tcsh	与csh兼容的shell
zsh	Z-shell,一个与csh、ksh、和sh兼容的shell

6.2.1 ash的特色

由Kenneth Almquist编写的ash shell是Linux操作系统上尺寸最小的shell之一。这个shell有24个不同的内建命令和10个不同的命令行参数。 ash shell支持大多数常见的shell命令，如cd命令以及大多数普通的命令行操作符(请阅读本学时教程后面的“ 了解 shell命令行 ”一节)。

它是一个很流行的shell，通常在以下几种情况下使用：以单用户状态(在 LILO启动提示符

下输入linux single)启动到OpenLinux操作系统、用于安全恢复状态或者用于 Linux操作系统的软盘版本。它自身的尺寸很小，通常只有 bash shell的十分之一大小，这使得ash成为小文件系统理想的选择。

6.2.2 Linux操作系统缺省的shell —— bash的特色

bash就是由Brian Fox和Chet Ramey编写的Bourne Again Shell，它是OpenLinux操作系统上最流行的shell之一。它有48个内建的命令和十多个命令行参数。 Bash shell运行的时候，就像sh shell一样，而在子目录 /bin中就有一个叫做 sh的符号链接，这个符号链接指向 bash shell。

bash在运行的时候不仅仅像 sh shell，它还有一些csh和ksh shell的特点。因为bash被相当广泛地使用着，所以把它用做本学时教程中的示范。后面将介绍如何使用 bash shell来对命令行提示符进行定制设置。

bash shell有许多特色。可以使用方向键查阅以前输入的命令(历史记录功能)、可以对某个命令行进行编辑、而且在忘记了某个程序的名字的时候，甚至可以请求这个 shell使用命令行补充功能对你进行帮助。只要敲入一个命令的一部分然后再按下 Tab键就可以了。例如，先按下l键，再按下Tab键，可以看到下面的内容：

```
# l<TAB>
laser       less        listres     locale      look        lsac
last        lesskey     lkbib       localedef   lookbib     lsattr
lastb       let         lmorph      locate      lpq         lsc
latex       lex         ln          lockfile    lpr         lsl
lbxproxy    lha         lndir       logger      lprm        lynx
ld          lightning   loadkeys    login       lptest      lz
ld86        lisa        loadunimap  logname     ls
ldd         lispmtopgm  local       logout      lsa
```

bash shell会列出所有已知的以字母 l开头的命令(或者当前子目录中所有可执行文件)作为响应。假如记不住复杂的命令拼写方法，这个功能就会非常有帮助。

bash shell还有内建的帮助功能，能够列出所有的内建命令和关于每个命令的帮助信息，如下所示：

```
#  help
GNU bash, version 1.14.7(1)
Shell commands that are defined internally. Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
A star (*) next to a name means that the command is disabled.

 %[DIGITS ¦ WORD] [&]               . filename
 :                                  [ arg... ]
 alias [ name[=value] ... ]         bg [job_spec]
 bind [-lvd] [-m keymap] [-f filena break [n]
 builtin [shell-builtin [arg ...]]  case WORD in [PATTERN [¦ PATTERN].
 cd [dir]                           command [-pVv] [command [arg ...]]
 continue [n]                       declare [-[frxi]] name[=value] ...
...
 while COMMANDS; do COMMANDS; done  { COMMANDS }
```

如果想得到关于某个命令的帮助信息，在 help命令之后敲入这个命令的名称就可以了。举例来说，如果想得到help命令的帮助信息，敲入下面的内容：

```
# help help
help : help [ pattern . . . ]
```

Display helpful information about builtin commands . If PATTERN is specified , gives detailed help on all commands matching PATTERN , otherwise a list of the builtins is printed .

新术语 bash shell有几个非常重要的文件叫资源文件、运行配置文件和shell启动文件。第一次登录进入OpenLinux操作系统的时候, 这些文件被用来定义或者共享预定义的设置值和定义值, 比如所使用的终端类型、缺省的文本编辑器程序和打印机以及可执行文件存放的位置等等。初始化设置文件/etc/profile用来设置全局(对所有用户都起作用的)参数，比如环境变量(后面将要讨论)或者在第一次登录进入的时候给你发送一条消息 (比如一条欢迎标语)。也可以使用用户子目录中的.bashrc登录文件按照个人喜好控制 bash shell启动运行的方式或者对不同的击键(如退格键)的响应；在用户子目录中还有一个 .profile文件，它被用来通知 shell在登录进入OpenLinux操作系统之后应该使用哪一个资源文件。

如果希望了解更多关于bash shell的资料，可以使用info命令阅读它的使用手册页、 info信息页文本和保存在子目录/usr/doc/bash中的文档。

6.2.3 公共域Korn Shell —— pdksh

pdksh，也就是公共域Korn shell，最初是由Eric Gisin编写的，它有42个内建的命令和20个命令行参数。这个shell存放在子目录/bin下，但在子目录/usr/bin中还有一个符号链接。

pdksh shell在Linux操作系统中叫做ksh。它和bash shell一样，如果在用户子目录中找不到一个名为.profile的文件的话，就会去读取初始化设置文件 /etc/profile。不太幸运的是，这个shell并不支持与bash shell使用的相同的命令行提示符。但这个 shell确实支持任务控制(本学时教程后面将讨论)，所以还可以从命令行上挂起、后台执行、唤醒或终止程序。

这个shell和商业化UNIX版本中的商业化Korn shell版本几乎是完全兼容的。关于这个 shell的有关文档存放在ksh的使用手册页和子目录pdksh中(在子目录/usr/doc/pdksh中)。

6.2.4 与csh兼容的shell —— tcsh的特色

由William Joy(以及其他47名参加人员)编写的tcsh shell有52个内建的命令和18个命令行参数。这个shell仿真csh shell，但是增加了许多功能，包括一个带拼写检查功能的命令行编辑器程序。

这个shell不仅能够与bash shell提示符兼容，而且还可以提供比 bash shell更多的提示符定制参数。如果在用户子目录中找不到 .tcshrc或.cshrc文件的话， tcsh就会使用子目录/etc/directory中的csh.cshrc文件。与bash shell类似，可以查看曾经输入过的命令并编辑命令行。

使用builtins命令(tcsh没有像bash shell那样的帮助信息功能)可以得到一份 tcsh中允许使用的命令的清单，如下所示：

```
# builtins
:          @          alias      alloc      bg         bindkey    break
breaksw    builtins   case       cd         chdir      complete   continue
default    dirs       echo       echotc     else       end        endif
endsw      eval       exec       exit       fg         filetest   foreach
glob       goto       hashstat   history    hup        if         jobs
kill       limit      log        login      logout     ls-F       nice
```

```
nohup     notify     onintr     popd       printenv   pushd      rehash
repeat    sched      set        setenv     settc      setty      shift
source    stop       suspend    switch     telltc     time       umask
unalias   uncomplete unhash     unlimit    unset      unsetenv   wait
where     which      while
```

如果想了解关于这个 shell的详细资料，请阅读 tcsh的使用手册页。或者查阅子目录/usr/doc/tcsh下的子目录tcsh中它的常见问题答疑FAQs文件和其他文本文件。

6.2.5 zsh

由Paul Falstad最初开始编写的zsh shell是Linux操作系统所使用的最大的 shell之一，它有84个内建的命令和超过50个命令行参数，并能仿真sh和ksh shell的命令。

与bash和tcsh shell类似，zsh shell也可以让你查看曾经输入过的命令并对命令进行补充、编辑或者拼写检查。它还可以让你使用任务控制来管理运行中的程序。这个 shell有高级的命令行参数用来搜索和匹配文件格式。

这个shell的系统级初始化文件存放在子目录 /etc中，包括：

/etc/zlogin
/etc/zlogout
/etc/zshenv
/etc/zshrc

如果zsh shell在用户子目录中找不到以句号(.)开头的与上面列出的文件相对应的文件如.zlogin等的话，就会分析这些保存在 /etc子目录中的文件。与其他的 shell —— 如bash或者tcsh相比，这个shell有着更多的命令行参数。你会发现这个 shell有类似于所有其他 shell的特色，在被当作符号链接调用的时候可以仿真 sh或ksh shell(尽管OpenLinux操作系统已经把sh链接到了bash shell)。

关于这个shell有许多的资料：它有多达 10页的使用手册页，另外还有一个装满帮助文件、示范以及其他最新的有用信息的 /usr/doc/zsh子目录。

6.3 了解shell的命令行

通过命令行使用 shell启动一个程序的时候， shell会解释你的命令，而这个命令则把它自己的输出回显到屏幕上。而使用了 shell的时候，就可以把这个程序的输出送到别的地方，比如某个文件中。事实上， OpenLinux操作系统中几乎所有的东西都可以被当作是文件；这个规则对诸如终端及打印机等也不例外，甚至有许多程序都可以接受来自其他文件的输入。这个shell还可以把某个程序的输出当作是另外一个程序的输入，甚至让某个文件从另外一个文件中读入数据作为自己的输入。举例来说，可以把 ls命令的标准输出重定向到一个文件：

```
# touch /tmp/trash/file1 /tmp/trash/file2 /tmp/trash/file3
➥ /tmp/trash/file4
# ls -w 1 /tmp/trash/* > trashfiles.txt
```

新术语 上面的第一个命令行在子目录tmp/trash中建立了四个文件。第二个命令行使用 ls命令的输出生成了一个文本文件，这个文本文件中包含了子目录 /tmp/trash中的文件名。大于号(>)叫做“ **标准输出重定向操作符** ”，用来把一个命令的输出重定向到另外的地方。你还可以使用小于号(<)，也就是“ **标准输入重定向操作符** ”向其他程序输入信息。

本学时教程后面使用的所有例子都将使用 bash shell。如果想试试这些命令行中的一部分，请在命令行上输入 bash并按下回车键，这样就可以开始使用bash shell了。完成操作的时候，输入 exit并按下回车键，就可以退回到原来的shell中去了。

作为一个小例子，可以使用cpio命令把刚才建立的存放着文件名清单的那个文件作为输入来生成一个档案文件。如果想要做到这一点，只需要通过标准输入把那个文件名文件重定向馈入到cpio命令就可以了，如下所示：

```
# cpio -o < trashfiles.txt > trash.cpio
1 block
```

从上面的命令行中可以看到，cpio命令从标准输入 —— 也就是trashfiles.txt文件中读入了一个文件名的清单，然后把自己的输出经由标准输出送到一个我们起名为 trash.cpio的文件中去，这样生成了一个档案文件。

cpio命令的作用与tar命令和dump命令类似，都是用来对OpenLinux操作系统的文件进行归档或者备份操作的。请阅读第 23学时教程“备份和恢复系统”去学习更多关于对文件进行备份的知识。

一般来说，大多数能够在 shell的命令行上运行的程序都具备从标准输入读和向标准输出写的能力。除了标准输入和标准输出之外，还有一个标准出错信息输出(它几乎总是显示到屏幕上)。还使用上面的例子，如果送入到 cpio命令的文件清单中包含有一个错误， cpio命令将发出警告并在屏幕上显示出一个出错信息，如下所示：

```
# rm -fr /tmp/trash/file3
# cpio -o < trashfiles.txt > trash.cpio
cpio : /tmp/trash/file3 : No such file or directory
1 block
```

在上面的例子中，一个已经存在的文件被删除了，但是 cpio命令的输入文件trashfile.txt中依然是没有修改过的文件名清单。试图使用这个文件名清单作为一个有效的输入去生成一个档案文件的时候， cpio命令会发出警告并在屏幕上显示一个出错信息(但仍然会生成归档文件)。

这个shell给每个输入文件和输出文件都分配了一个文件号。对标准输入来说，它的文件号是0；对标准输出来说，它的文件号是 1；而对标准错误信息输出(也就是程序把出错信息经常送去的地方)来说它的文件号是 2。掌握了这些知识，就可以无声无息地运行 cpio命令，并把任何出错信息都送到某个文件去。采用把标准输出重定向操作符和标准错误信息输出文件号使用在一起的方法，就可以做到这一点，如下所示：

```
# cpio -o < trashfiles.txt > trash.cpio 2 > cpio.errors
# cat cpio.errors
cpio : /tmp/trash/file3 : No such file or directory
1 block
```

正如你所看到的，cpio命令平常都会把它的出错信息送往标准错误信息输出(显示器)，但是这一次却送到一个名称为cpio.errors的文件中去了。

在一般的情况下，每当把命令的输出重定向到一个文件的时候，这个指定的输出文件或

者被创建；或者如果它已经存在的话，就将被覆盖，而其中原来保存的内容也就无可挽回地丢失了。

在使用文件重定向的时候要非常小心。如果把命令输出重定向到一个现有的文件，就会丢失原来的文件，而这可能并不是你想进行的操作。

如果想在进行重定向操作的时候保留某个文件原来的内容，可以使用重定向附加操作符(>>)把某个程序的输出结果附加到一个文件上去，如下所示：

```
# cpio -o < trachfiles.txt > trash.cpio 2 >> cpio.errors
```

上面的命令行执行后保留了 cpio.errors文件中原来的内容，并把新的出错信息附加到这个文件的末尾。每当运行 cpio命令的时候，使用了这个方法就可以保存一份出错信息的记录。事实上，当自己登录到Linux操作系统上的时候，这个功能就已经被激活了。可以阅读一下子目录/var/log(要以根操作员身份进行操作)中 messages文件的内容。

如果还能想起我们在第4学时教程“阅读与浏览命令”中使用 cat命令进行简单的文本编辑的那个例子，就应该记得我们是在 cat命令中使用了标准输出重定向操作符从终端上读取输入而创建了那个文本文件的，如下所示：

```
# cat > file.txt
this is a line
this is another line
EOF
# cat file.txt
this is a line
this is another line
```

新术语 上面例子中的“文件结束符”，即EOF符号，是用来代表一个文件结尾的标志，它是按住Ctrl键再按下D键输入到文件中去的。 使用重定向操作符 <<可以给这个简单的编辑器程序再增添一个新的功能，这个重定向操作符告诉 shell程序紧跟在<<符后面符号就是文件结束符EOF，如下所示：

```
# cat > file.txt <<
> This is a line of text .
> This is another line of text .
.
# cat file.txt
This is a line of text .
This is another line of text .
```

请注意在上面的例子中，当我们在某一行文本上单独敲入一个句号后文件立刻就被关闭了，这比使用Ctrl组合键关闭文件的方法要来的更方便。可以使用任何字符或者字符组合来表示EOF文件结束符。

现在我们把egrep命令(请阅读第5学时教程“操作与搜索命令”)和 <<重定向操作符结合在一起使用，通过这样的方法来建立一个简单的自生成数据库文件。首先，使用一个喜欢的文本编辑器程序(比如包括在电子邮件程序 pine中的pico)建立一个名为db的文件。在db文件中输入一堆通讯地址，再把 grep命令放在这个文件的开头，把表示输入结束的字符串放在这个文件的末尾，然后保存这个文件。建立这个文件的过程如下所示：

```
#pico db
```

pico编辑器程序可能很快就会成为你最喜欢的 OpenLinux操作系统文本编辑器程序，因为它易于使用、快速以及能够安全地保存文本文件。请阅读第14学时教程“ 文字处理程序 ”学习这个了不起的小编辑程序。

然后，输入egrep命令，再输入一小段通讯地址表，最后在 db文件的结尾输入与 <<重定向操作符匹配的字符串，如下所示：

```
egrep -i $1 <<zzzz
Debby, 275 Collins Rd., Vestal NY 13850
Cathy, 1001 N. Vermont St., Arlington, VA 22003
Scotty, 2064 N. 16th St., Arlington, VA 22001
Bill, 4000 N. Pennsylvania Ave., Washington, DC 10000
Fred, Slip 417, N. Woodward Ave., Boca Raton, FL 46002
zzzz
```

用Ctrl+X组合键保存这个文件。最后，使用 chmod命令把这个文件转换为可执行程序，如下所示：

```
# chmod +x db
```

这样就建立了一个简单的通讯地址数据库。如果想试试这个数据库，请输入 db，后面再跟上一个姓名、城市、州或者邮政编码，如下所示：

```
# db deb
Debby, 275 Collins Ave., Vestal NY 13850
```

这个数据库的用法是：使用 egrep命令读取文件直到遇见代表输入结束的字符串 zzzz。命令行参数-i使得egrep命令不区分字符的大写或小写情况。字符串 $1是一个shell变量(我们将在下一小节讨论)，它表示的是传递到 egrep命令中去的命令行参数。

在shell中的输入和输出重定向技术可以有许多种不同的用法。我们以后还会继续遇到使用输入和输出重定向的情况。下面我们将介绍 shell变量和一些对 shell进行定制设置方面的小技巧。

6.3.1 对shell进行定制设置

新术语 使用shell的时候，是在一个包含了“ 环境变量 ”的环境中运行 shell的。环境变量是那些在用户子目录和子目录 /etc中的各种资源文本文件中预先定义使用的数值。对 bash shell来说，缺省的全局性系统环境变量是在文件 /etc/profile中定义的，而一些定制的设置可以在用户子目录中的文件.bashrc中找到。

有许多不同的环境变量。使用 printenv命令或者 set命令可以看到一个当前正在使用中的环境变量的清单，如下所示：

```
# printenv
...
PATH=/usr/local/bin:/bin:/opt/kde/bin:/usr/bin:.:/usr/X11R6/bin:/home/
➥bball/bin
HOME=/home/bball
SHELL=/bin/bash
...
```

本学时教程没有列出为 OpenLinux操作系统的shell所定义的所有环境变量的清单，但是应该知道最重要环境变量之一就是 $PATH变量。这个变量告诉 shell在哪里才能找到可执行程序。

如果没有环境变量，就不得不输入完整的路径 —— 也就是某个命令的完整的目录结构来运行这个程序。举例来说，假如想使用 ifconfig命令来检查网络连接的状态，首先可能会在命令行上敲入这个命令的名字来试试：

```
# ifconfig
bash：ifconfig：command not found
# whereis ifconfig
ifconfig：/sbin/ifconfig
```

正如所看到的，这并不表示 ifconfig命令不存在或者是在系统上没有安装这个命令；只是shell程序不知道到哪里去寻找这个程序罢了。如果想运行 ifconfig命令，可以在这个命令行中输入完整的路径名，但是如果需要经常使用这个程序的话，就可以把它的子目录加到 shell的$PATH环境变量中的已知路径清单中去。

我们在命令行上使用 bash shell的export命令(这个命令会保存新的环境变量定义并使得它对shell起作用)把子目录/sbin添加加到$PATH变量中去，这样就可以做到这一点，如下所示：

```
# ifconfig
bash: ifconfig: command not found
# PATH=$PATH:/sbin ; export PATH
# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Bcast:127.255.255.255  Mask:255.0.0.0
          UP BROADCAST LOOPBACK RUNNING  MTU:3584  Metric:1
          RX packets:436 errors:0 dropped:0 overruns:0
          TX packets:436 errors:0 dropped:0 overruns:0
```

正如所看到的，现在shell程序就知道到哪里去寻找ifconfig 命令了。但是这个作用只是暂时的，只有在保持在登录状态或者运行特定终端程序的时候才能够维持其作用，而退出系统之后再重新进入的时候还会出现找不到命令程序位置的情况。所以如果想让这个设置在每次登录进入系统的时候都能够起作用，就必须采用在用户子目录中的 .bash-profile文件中加上这个路径的办法；或者如果你就是根操作员并且希望所有的用户都能够享受到这种设置的好处的时候，采用在子目录/etc的profile文件中加上这个路径的办法。

在.bash-profile文件中找到下面这一行：

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

使用一个文本编辑器程序，在其中加上子目录 /sbin：

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

如果你是根操作员，就要在子目录 /etc下的profile文件中下面的这一行上加上子目录 /sbin：

```
PATH = " /bin : /sbin : /usr/bin "
```

进行完这些修改之后，使用bash shell程序的source资源命令读入这个新的$PATH环境变量，如下所示：

```
# source /etc/profile
```

在这个profile文件中还会看到类似于下面内容的一行：

```
export PS1 = " [ \u@\h \w ] \\$ "
```

这一行看起来非常古怪，它是用来对 $PS1环境变量，也就是提示符格式字符串进行定义用的，它定义了显示在控制台命令行或者使用了某种 shell的时候出现在终端窗口中的命令提示符字符串。可以改变这个格式字符串来定义几乎任何形式的命令行提示字符串。上面例子中的定义将显示为下面这个样子：

```
[ username@host base_working_directory ]$
```

bash shell有15种不同的提示符格式字符(tcsh有18种， ksh有35种)，可以把它们和其他普

通字符串一起使用，对命令行提示符进行定制设置。比如说，可以在命令行提示符中加上当前日期和时间。如果使用了bash shell的export命令，就可以在命令行试用不同的提示符形式：

```
# PS1 = ' Date : \d Time : \t -> ' ; export PS1
Date: Thu Dec 10 Time: 22 : 19 : 01 ->
```

在上面的例子中，每当输入一个命令或者按下回车键的时候其中的时间部分都会更新。如果想让命令行中出现shell的名称和当前子目录，可以使用\s和\w提示符格式字符：

```
# PS1 = ' \s: \w> ' ; export PS1
bash: /usr/bin>
```

还可以使用不同的转移字符键序列和终端键序列来控制命令行提示符字符串的属性。可以给命令行提示符字符串增加下划线、加粗、闪烁和其他的属性：

```
# PS1 = ' [ \033 [ 4m\u\033 [ 0m@\033 [ 4m\h\033 [ 0m ] : ' ; export PS1
[ bball@localhost ] :
```

上面的例子对X11的终端程序xterm(可以使用printenv命令查看$TERMCAP环境变量找到它)使用了适当的转移字符键序列，从而对它的命令行提示符字符串做了改变，它给命令行提示符中的用户名和主机名加上了下划线。可以在子目录 /etc中的termcap文件和termcap使用手册页中找到这些定义。如果想了解更多的示例和关于某种特定的 shell的命令行提示符格式字符串的资料，请阅读shell的使用手册页。

新术语 改变命令行提示符只是对shell中的工作方式进行定制设置的一种方法。还可以定义程序命令的快捷键，也就是它的“假名(alias)”，这样也可以对你喜欢的命令定制设置它的工作方式。至少可以在子目录/etc/directory中的bashrc文件中找到一两个为你的系统定义的假名(alias)。

对系统全局范围的假名进行的定义是由根操作员输入的。可以把自己的定义放在用户子目录中的.bashrc文件中，但是如果本身就是根操作员，你至少应该为所有的用户把下面三个假名加入进去：

```
alias rm = ' rm -i '
alias cp = ' cp -i '
alias mv = ' mv -i '
```

上面的三个假名使得对文件进行删除、拷贝和改名(移动)操作的时候能够交互式地进行，这样在对文件进行删除、拷贝和改名(移动)操作的时候就多少能提供点安全保障。如果没有 -i参数，用户在删除或覆盖文件之前可能根本就不会有什么考虑。

也可以使用定义假名来构成新的命令或者提供常见命令的变形以避免输入过长的命令行参数。比如，ls命令有许多不同的参数，但可以定义几种变形使工作简单化一些。我们把一些你可能想试试的假名定义开列如下：

```
# list the current directory using color filenames
alias lsc='ls --color'
# long format listing
alias lsl='ls -l'
# show all files except . and ..
alias lsa='ls -AF'
# show all file except . and .. in color
alias lsac='ls -AF --color'
```

在把这些改变输入到用户子目录中的 .bashrc文件中去之后，可以再次使用 bash shell的source命令来使用这些假名：

```
# source.bashrc
```

现在输入lsc、lsl、lsa或者lsac的时候就不用再加上命令行参数了。但是一定要保证没有重复定义一个现有的命令。如果的确有一些非常有用的假名，请注意要保证它们在系统上的每个shell中都有它们的定义。

在系统上提供了某种shell并不意味着用户就能够使用这种 shell来登录进入系统。作为根操作员，可以通过编辑子目录 /etc下的shells文件来维护一个在系统上可以使用的 shell的清单。在没有修改过的时候，这个文件中为 OpenLinux操作系统列出了下面几种shell：

```
/bin/bash
/bin/sh (a symbolic link to the bash shell)
/bin/ash
/bin/tcsh
/bin/csh (a symbolic link to the tcsh shell)
/bin/ksh
/bin/zsh
```

如果不想让用户使用某种 shell，只需简单地把它从表中删除就行了！如果想使用 chsh命令改变shells文件以便在下一次登录进入系统的时候能够使用某个特别的 shell的话，这个shell就必须是列在文件 /etc/shells的清单里面的。否则 chsh命令会报告出错并退出执行，而且也不会做任何改动，如下所示：

```
# chsh -s /bin/zsh
Changing shell for bball.
Password :
chsh : " bin/zsh " is not listed in /etc/shells .
chsh : use -l option to see list
```

请注意，上面的做法并不会限制用户在登录进入系统之后使用某种 shell。明确设定哪些人可以使用哪些shell的唯一有效的管理办法是改变某个 shell的所有权或文件权限(具体做法请阅读第22学时教程“ 管理文件和文件系统 ”)。

6.3.2 在后台运行程序

新术语 大多数的shell还都提供了一种以“ 后台 ”进程的方式启动并执行程序的方法。在后台启动某个程序的意思是这个命令继续在内存中执行，而在这同时， shell命令行的控制权已经返回到了控制台。这是完成工作的一个便利的方法，特别当你是工作在独立的终端上、工作在X11环境却又屏幕空间有限或者是拥有大量内存等情况的时候更是如此。对巨型文件进行排序或者对子目录和其他类型的文件系统进行搜索等等操作都是可以放到后台去执行的好例子。

虽然当Linux操作系统不工作在 X环境(通过Alt和功能键进行操作)的时候提供了虚拟控制台，而且许多的 X11窗口管理器程序也提供了好几个不同的桌面，但是还是有可能会在使用Linux操作系统的同时在后台多次运行一些程序。

如果想从shell的命令行让程序在后台运行，需要在启动这个程序的命令行的末尾使用 &操作符。比如说，如果想在 X11下启动另一个终端程序，并且希望这个程序在后台运行以便空出当前的终端来进行其他输入，需要使用：

```
# rxvt &
```

这个命令启动了 rxvt终端程序，然后命令行提示符又重新出现了。新的 rxvt终端程序被分配了一个进程号，可以使用查看进程状态的 ps命令看到这个进程号，如下所示：

```
# ps
...
 291   1 R  0 : 03 rxvt
...
```

在上面的例子中，为了简略起见我们并没有列出全部正在运行的进程。可以使用 shell的kill命令和某个程序的进程号来中止这个程序的运行：

```
# kill 291
[1] + Terminated          rxvt
```

新术语 使用kill命令是控制后台程序的一个比较粗鲁的方法。还有一些更精细地使用其他shell命令的方法。根据当前正在使用的 shell的运行情况，可以把运行中的程序放入后台、挂起这个程序、在后台继续运行这个程序、终止这个程序或者把这个程序再带回到前台。这就是所谓的**任务控制**。

如果正在运行的是bash shell，在键盘上按住Ctrl键再按下Z键就可以把一个正在运行的程序放入后台并把它的操作挂起来：

```
# pine
... program is running... (ctrl-z)...

Pine suspended. Give the "fg" command to come back.

[1]+  Stopped (signal)        pine
# fg
.... program returns
```

为了可以挂起pine邮件程序，必须使用 pine程序的配置菜单激活挂起功能。在运行 pine程序的时候，按下 S键进入设置菜单，再按下 C键进入配置菜单。然后，在配置菜单的选项中找到“ enable-suspend ”选项并把高亮度光标块移动到这个选项上，再按下 X键激活这个选项。接着，按下E键退出配置画面，再按下 Y键保存所做的修改。现在你就可以挂起 pine邮件程序了。

把一个正在运行的程序送入后台并挂起其运行后，可以使用 fg命令把这个正在运行的程序再调回到前台显示，或者使用 bg命令继续让程序运行。假定想启动一个程序，比方说是一个新闻阅读器程序(请阅读第 11学时教程“ 配置因特网电子邮件 ”和第 12学时教程“ 配置因特网新闻 ”)，然后在这个程序进行耗时漫长的操作(比如更新某个新闻组的内部名单)的时候挂起并继续其运行，而同一时间还可以在前台运行其他的程序，这么做有多方便。

使用bash shell，你可以启动、挂起并运行好几个程序，然后再通过这些程序的任务号有选择地把某个后台程序调回到前台显示，如下所示：

```
# pine
... program is running (ctrl-z)...

Pine suspended. Give the "fg" command to come back.
[1]+  Stopped (signal)        pine
# sc
.... program is running (ctrl-z)...

[2]+  Stopped                 sc
# fg %1
... pine program returns...
```

在上面的例子中，pine邮件阅读器程序在启动后被挂起送到后台，然后我们启动了 sc电子表程序。因为pine程序是第一个被挂起的任务，所以 bash shell把任务号1分配给了这个邮件程序。接着sc电子表程序也被挂起并被分配给任务号 2，然后我们使用 fg %1命令通过指定的任务号返回到了邮件程序中。

如果你运行并在后台挂起了许多任务，你可能无法通过某个程序的任务号想起这个程序或者是想起哪几个程序被挂起来了。在这种情况下，可以使用 bash shell的jobs命令得到一个被挂起的程序的清单：

```
# jobs
[1]   Stopped (signal)          pine
[2]-  Stopped                   sc
[3]+  Stopped                   emacs-nox
# fg %sc
... sc program is running ...
```

上面的例子显示出有三个任务： pine邮件程序， sc电子表程序和emacs编辑器程序在当前的shell中被挂起来了。请注意我们并没有通过引用 sc电子表程序的任务号的方法来重新启动它，我们使用的是fg %命令加上被挂起来的任务的程序名方法把 sc电子表程序调回到前台的。

当然也可以使用同样的方法来终止程序的运行。我们既可以使用先用 ps命令查找某个程序的进程号再使用kill命令的方法；也可以直接使用 kill命令和%操作符的方法来完成同样的操作：

```
# kill %1
[1] - Stopped (signal)  pine
# kill %emacs-nox
[3] + Stopped       emacs-nox
```

上面看到了把kill命令与任务号或者程序名一起使用来中止程序的两种方法。这比需要使用ps命令的情况要简单得多了，特别是在后台中运行着大量程序或其他进程的情况下更是如此。

OpenLinux操作系统中包括了一个不属于任何 shell的killall命令，可以使用这个命令和进程名称来终止一个或者几个进程。我们还是以前面使用kill命令终止rxvt终端仿真程序的例子来说明。可以输入下面的内容代替输入kill和跟在它后面的一个进程号：

killall rxvt

按下回车键之后，这个终端仿真程序的进程就被终止了，它的窗口也消失了。killall命令还有一个 -i交互参数，可以使用这个参数来有选择地终止某个程序，另外也可以指定需要终止的进程号。

使用shell的任务控制功能是运行多个程序、提高工作效率的强有力的方法，特别是只使用一个显示器或者控制台程序的时候更是如此。包括在 OpenLinux操作系统的CD-ROM光盘上的所有 shell都可以通过这样或者那样的形式进行任务控制。在下一小节里我们将向你介绍另外在shell的命令行上只使用一条命令就运行多个程序的一个好方法。

6.3.3 怎样使用管道

新术语 已经看到过如何把一个程序的输出重定向到一个文件以及如何再把这个文件的内容

重定向到另外一个程序中。但是现在可以立刻使用我们称之为“管道”(pipe)的垂直竖字符(|)完成上述的工作而不必再使用什么临时文件。使用管道字符把不同的命令在命令行上串在一起，这个方法我们称之为管道线，它是增强单个命令功能的有效手段，代表了 Linux操作系统和其他版本的UNIX操作系统的独一无二的过人之处。

在开始学习如何使用Linux操作系统的时候肯定会用到管道命令。这不仅是因为管道可以节省时间，还因为可以通过各种管道命令不同的组合使计算任务与工作方法和运行使用的程序互相适应。在刚开始的时候，使用的管道命令可能会很简单，但随着信心的加强及理解的加深，你将有能力构造出相当复杂的管道线。

新术语 管道在Linux操作系统中工作得很好，因为许多命令同时也是“过滤器”程序，它们接受某种类型的输入数据、以某种方法对之进行处理和修改然后把处理结果作为输出数据送到标准输出或者另外的某个程序中去。管道能够使用在几乎所有的计算任务中，它们可以用来快速地查找信息、生成报表、传输数据或者查看结果。下面先来看看四个简单的例子：

```
# ls | 1pr
# printenv | fgrep EDITOR
# nroff -man mymanpage.1 | less
# cat document.txt | wc | mail -s " Document.txt Report " bball
```

上面的第一个命令行使用管道把当前子目录的列表清单送到行式打印机命令打印出了一份报表。第二个例子检索正在使用的 shell的环境变量清单并显示出缺省文本编辑器的设置值。第三个例子把某个使用手册页排版之后的文档输出显示到显示器上供浏览并检查错误。最后一个例子把一个文本文件通过管道传送到单词统计程序 wc中，然后再通过电子邮件把关于这个文件中的字符数、单词和文本行数的报告传送到用户 bball处去。

将会使用管道命令来面对并解决通常靠单个程序无法解决的日常问题。而这也就是使用OpenLinux操作系统的shell程序的秘密和威力之一。举个例子，比如说在系统上存放了大量的文件可却想不起来在哪个文件里有个什么词儿的时候，可以很快地找到它们 —— 不需要运行字处理软件来打开每一个文件，只需输入下面的命令：

```
# find /home -name *.doc | xargs fgrep administration | less
```

上面的命令行使用 find命令在子目录/home中找出所有以.doc结尾的文件，再把文件名通过管道传递到 xargs命令。然后 xargs命令再运行 fgrep命令到每个文件中去查找单词“administration”，最后经过less页命令把结果显示出来。使用管道，不仅可以查找信息，还可以处理数据和生成新文件：

```
# find *.doc | xargs cat | tr ' ' '\n' | sort | uniq | tee dict | less
```

上面的命令行建立了一个叫做 dict的文件，这个文件中是一个经过整理和排序的字处理文件中使用过的所有单词的一个清单列表。把这种文件叫什么？当然是字典了！虽然其中的单词不见得都是拼写正确的。

上面的命令的工作过程是：先把每个找到的文件经管道传递到 tr命令，它把每个字符空格转换成回车符，这样就把输入字符流转换成每行只有一个单词的形式；然后这个字符流中的单词经过排序；再使用 unique命令把那些重复相似的单词行排除为只剩下一个。应该注意还可以使用tee命令把这个字符流的输出保存到一个文件中去。

zsh shell中包含了一些对输入和输出重定向的改进，这样你在 zsh shell的命令行中使用管道的时候可能就不必使用 tee命令了。详细资料请查阅zsh shell的文件。

tee命令用来在某个指定的环节保存一个管道的运行结果。当希望在建立管道的同时测试其结果，或者希望在某个复杂管道的某个地方保存中间结果的时候，这个命令是很方便的。请看下面的例子：

```
# xwd -out wd.xwd
# xwdtopnm < wd.xwd ¦ ppmtogif ¦ tee wd.gif ¦ giftopnm ¦ tee wd.pnm ¦
➥ pnmtotiff > wd.tif
```

上面的例子中，我们先使用X11的xwd命令建立了一幅窗口转储画面，xwd命令是用来捕捉并保存某个X窗口或桌面内容的。然后使用了xwdtopnm命令把这幅图像转换为便携位图格式的图形。再把xwdtopnm命令的输出馈入ppmtogif命令。此处使用了tee命令以GIF格式将其输出保存到一个wd.gif文件。同时使用giftopnm命令把这个文件再次转换为一个便携位图格式图像并再次使用tee命令把这个便携位图图形格式图像文件保存到文件wd.pnm中。最后，pnmtotiff命令把这个便携位图图形格式文件以TIF格式保存起来。这样只使用一条命令就进行了四次图形转换！

在Linux操作系统中使用管道是完成任务的简单方法。随着深入学习 Linux操作系统，很快就会开发出自己喜欢的一套命令行集合。一旦开发出喜欢的命令行，就可以建立自己的shell命令，这就是我们下一小节要学习内容。

6.4 建立shell的命令脚本程序

新术语 如果希望为Linux操作系统编写命令，不必非得是一个程序员。熟悉了各种命令并且发现自己反反复复地敲入着相同的命令行的时候，就可以把它们保存为一个文本文件，然后把它们转换为shell的“命令脚本程序”。简单地说，一个shell的命令脚本程序就是经常使用的一个或者几个命令行。请看下面的例子：

```
# rxvt -geometry 80×11+803+375 -bg white -fg black -e pico &
# rxvt -geometry 80×24+806+2 -bg white -fg black -e pine &
```

上面的两个命令行在一个860×600点分辨率的显示器上的第二个桌面上的两个 X11的rxvt终端窗口中分别启动了pico编辑器程序和pine邮件程序。这些当然不是每次需要运行这些程序的时候你愿意敲入的命令行。虽然在移到其他桌面之后可以手动启动这些终端窗口，但是调整窗口尺寸和启动程序总是会用去一些时间的。可以先使用文本编辑器程序把它们保存到一个文件中，再使用 chmod命令把这个文本文件转换成可执行文件，再通过下面的操作就可以把这些命令行转换为一个可执行文件了：

```
# chmod +x d2
```

现在，当想运行这些程序的时候，需要做的全部事情只是敲入下面的两个字符就行了，这可太简便了：

```
# d2
```

还可以通过使用shell变量$1和$2的方法使这个新命令更具灵活性。$1和$2分别代表了某个shell命令的第一个和第二个命令行参数。编辑建立的文件并把其中的程序名改为这两个变量：

```
rxvt -geometry 80×11+803+375 -bg white -fg black -e $2 &
```

rxvt -geometry 80×24+806+2 -bg white -fg black -e $1 &

请注意变量的顺序并不重要。现在当执行命令的时候，可以在命令行给出程序名,如下所示：

```
# d2 pine pico
```

虽然其结果和前面是一样的,但是从现在起将能够在终端窗口中运行几乎任何想运行的程序。

以上关于使用shell的学习最终归结到一个简单的 shell命令脚本程序上,可以使用这个脚本程序来安全地删除文件。程序 6-1中的rmv脚本程序虽没有什么特别之处，但是它却显示了shell的命令脚本程序的威力。

程序6-1 rmv命令的安全删除shell脚本程序

```
#!/bin/bash
# rmv - a safe delete program
# uses a trash directory under your home directory
#
# when run, always create a directory called .trash
mkdir $HOME/.trash 2>/dev/null
cmdlnopts=false
delete=false
empty=false
list=false

# get any command-line options
while getopts "dehl" cmdlnopts; do
  case "$cmdlnopts" in
    d ) /bin/echo "deleting: \c" $2 $3 $4 $5 ; delete=true ;;
    e ) /bin/echo "emptying the trash..." ; empty=true ;;
    h ) /bin/echo "safe file delete v1.0"
        /bin/echo "rmv -d[elete] -e[mpty] -h[elp] -l[ist] file1-4" ;;
    l ) /bin/echo "your .trash directory contains:" ; list=true ;;
  esac
done
# d - delete any files found on the command line
if [ $delete = true ]
then
  mv $2 $3 $4 $5 $HOME/.trash
  /bin/echo " rmv finished."
fi

# e - empty the trash?
if [ $empty = true ]
then
  /bin/echo "empty the trash? \c"
  read answer
  case "$answer" in
    y) rm -fr $HOME/.trash/* ;;
    n) /bin/echo "trashcan delete aborted." ;;
  esac
fi

# l - show any files in the .trash directory
if [ $list = true ]
then
  ls -l $HOME/.trash
fi
```

这个命令脚本程序的第一行调入 bash shell来运行这个程序。在使用喜欢的文本编辑器程序输入这个脚本程序后，要记得使用 chmod命令把这个脚本程序转换为可执行的：

```
# chmod +x rmv
```

这个shell脚本程序把不想要的文件移入到用户子目录中的一个名为 .trash的子目录中。当

确实想删除这些文件的时候，可以检查这些文件并清空这个“垃圾桶”。因为我们想把这个shell命令脚本程序设计成一个常规的命令，所以我们还在其中包括了一个短小的内建帮助命令。

如果想试试这个rmv脚本程序，一定要正确输入它的内容，或者把“rm -fr $HOME/.trash/* ;;”改为“rm -i $HOME/.trash/*;;”，这样就比无条件进行删除要安全多了。如果把这一行敲错了，那就可能会删除整个的用户子目录！

这个命令脚本程序的工作过程是：先在用户子目录(可以使用环境变量 $HOME查到)中建立一个.trash子目录，如果这个子目录已经存在， mkdir命令会产生一个出错信息，但我们把标准错误信息输出到/dev/null设备去了。这个/dev/null设备是个把所有出错信息都送去的好地方，因为所有对它的输入都被丢弃了。这个设备也用在不关心某个程序的输出或者希望从显示器上隐藏输出结果的时候。接着定义了脚本程序的四个内部变量： cmdlnopts、delete、empty和list。

这个命令脚本使用了bash shell的getopts命令来检查命令行中是否有什么参数。如果 case语句找到了匹配字符，两个分号之前的脚本命令就将被执行。比如，如果想得到使用这个脚本程序的提示信息，就可以使用-h参数，如下所示：

```
# rmv -h
safe file delete v1.0
rmv -d[elete] -e[mpty] -h[elp] -l[ist] file1-4
```

因为脚本程序在命令行检测到了字母 h，它就会显示上面所示的帮助信息，并使用 echo命令把这个脚本程序的用法显示出来。如果你想要删除文件，就必须使用 -d参数：

```
# rmv -d berries.jpg bowtie.jpg face.gif
deleting : berries.jpg bowtie.gif face.gif  rmv finished.
```

这样，脚本程序就把想要删除的三个文件移入到用户子目录中的 .trash子目录中去了。因为检测到了-d参数，这个脚本程序显示一个提示信息并在屏幕上回显想要删除的文件名，然后再把delete变量设置为真。又因为delete变量被设置为真，则if . . . then语句中的mv命令就被执行了。

可以使用-l参数看看“垃圾桶”中的文件，验证一下文件确实已经被移走了。

```
# rmv -l
your .trash directory contains：total 36
- r w - r - - r - -          1 bball    users      11967 Nov 6 14:18 berries.jpg
- r w - r - - r - -          1 bball    users      14010 Nov 6 14:18 bowtie.gif
- r w - r - - r - -          1 bball    users       8681 Nov 6 14:18 face.gif
```

和上面一样，因为在命令行上检测到了字母 l，脚本程序设置list变量为真，并执行ls命令显示.trash子目录中的文件。如果肯定自己确实想删除这些文件，可以接着使用 -e参数：

```
# rmv -e
emptying the trash...
empty the trash? n
trashcan delete aborted.
# rmv -e
emptying the trash...
empty the trash? y
# rmv -l
your .trash directory contains:
total 0
```

rmv脚本程序会问你是否真的想删除这些文件。如果输入一个字母 n，删除操作就不会继续执行；如果敲入一个字母 y，子目录.trash中的文件就将被删除。因为在命令行检测到了字母e，empty变量就被设置为真，跟在if . . . then后面的语句被执行。命令脚本在显示器上显示一条提示，并使用 read命令等待输入一个应答；然后对应答进行测试，如果是 yes，在子目录.trash中的文件就被删除了。

使用shell，可以很快地建立起完成主要任务的简单程序。请随意修改上面的程序，增加它的功能或者改进命令行处理文件名的方式。改进之一可能会是增加这个脚本程序处理通配符或者整个子目录的能力。另外一个改进可能会是在进行文件删除操作的过程中使用交互询问的方法。如果希望了解更多关于可以用在 shell命令脚本程序中的shell命令和操作符的资料，请阅读bash的使用手册页。

6.5 课时小结

本学时教程介绍了在OpenLinux操作系统中使用shell的基本知识。不要惧怕命令行。掌握和使用错综复杂又各有特点的各种 shell需要进行大量的实践。千万要保证所进行的练习是安全的：使用假名把那些有可能造成危害后果的命令修改为交互地执行、使用通配符删除文件的时候要多想两遍特别是绝对不要总是以 OpenLinux操作系统根操作员的身份进行操作。对于OpenLinux操作系统来说，学习得越多，在使用 shell遇到问题的时候和建立自己具有个人风格的解决方案的时候你就会越有信心。

6.6 专家答疑

问：帮帮忙！我正在使用 shell的时候，一个程序不见了，而且突然之间屏幕上满是乱七八糟的字符！

答：使用重启动命令试试。希望 shell还能够明白你从键盘输入的东西。输入 reset再按下回车键就可以重新启动终端或者控制台。

问：怎样才能了解更多关于bash的资料？

答：除了bash的使用手册页以外，在子目录 /usr/doc/bash中至少还可以找出三个额外的bash资料文件。也可以去喜欢的书店找找关于这种 shell的参考书。

问：用在OpenLinux操作系统中的最好的shell是哪一种？

答：这可是个难于回答的问题，因为每个 shell都有它不同的特点和不足。但是一般来说，bash shell看起来是最流行的了，而且它还是其他的 Linux操作系统发行版本缺省使用的 shell。大多数OpenLinux操作系统的用户还是比较喜欢这个 shell的，而且许多为Linux操作系统开发的shell脚本程序都可以工作在bash中。

6.7 练习题

1. 请找出新用户使用的缺省的 .bashrc设置。修改这个文件把rm、cp、和mv命令的安全化假名包括进去。

2. 如果想添加供自己个人使用的假名，那么应该修改哪个文件呢？

3. 请修改rmv脚本程序命令，对那个“垃圾桶”子目录的目录结构位置或者它的名称做一些改动。

4. 如果在命令行上使用了 -p参数但是没有给出任何的文件名，在 rmv脚本程序中的mv命令会显示出错信息，报告出现了错误。应该怎样改进？