第5章 shell输入与输出

在shell脚本中，可以用几种不同的方式读入数据：可以使用标准输入 —— 缺省为键盘，或者指定一个文件作为输入。对于输出也是一样：如果不指定某个文件作为输出，标准输出总是和终端屏幕相关联。如果所使用命令出现了什么错误，它也会缺省输出到屏幕上，如果不想把这些信息输出到屏幕上，也可以把这些信息指定到一个文件中。

大多数使用标准输入的命令都指定一个文件作为标准输入。如果能够从一个文件中读取数据，何必要费时费力地从键盘输入呢？

本章我们将讨论以下内容：

- 使用标准输入、标准输出及标准错误。
- 重定向标准输入和标准输出。

本章全面讨论了shell对数据和信息的标准输入、标准输出，对重定向也做了一定的介绍。

5.1 echo

使用echo命令可以显示文本行或变量，或者把字符串输入到文件。它的一般形式为：

```
echo string
```

echo命令有很多功能，其中最常用的是下面几个：

\c 不换行。
\f 进纸。
\t 跳格。
\n 换行。

如果希望提示符出现在输出的字符串之后，可以用：

```
$ echo  "What is your name :\c"
$ read name
```

上面的命令将会有如下的显示：

```
What is your name :□
```

其中“□”是光标。

如果想在输出字符之后，让光标移到下一行，可以用：

```
$ echo "The red pen ran out of ink"
```

还可以用echo命令输出转义符以及变量。在下面的例子中，你可以让终端铃响一声，显示出$HOME目录，并且可以让系统执行 tty命令(注意，该命令用键盘左上角的符号，法语中的抑音符引起来，不是单引号，)。

```
$ echo "\007your home directory is $HOME, you are connected on `tty`"

your home directory is /home/dave, you are connected on /dev/ttyp1
```

如果是LINUX系统，那么.....
必须使用- n选项来禁止echo命令输出后换行：

(续)

```
$ echo -n "What is your name :"
```

必须使用-e选项才能使转义符生效：

```
$ echo -e "\007your home directory is $HOME, you are connected on
`tty`"

your home directory is /home/dave, you are connected on /dev/ttyp1
```

如果希望在echo命令输出之后附加换行，可以使用\n选项：

```
$ pg echod
#!/bin/sh
echo  "this echo's 3 new lines\n\n\n"
echo "OK"
```

运行时会出现如下输出：

```
$ echod
this echo's 3 blank lines



OK
```

还可以在echo语句中使用跳格符，记住别忘了加反斜杠\：

```
$ echo "here is a tab\there are two tabs\t\tok"
here is a tab   here are two tabs                ok
```

如果是LINUX系统，那么...

别忘了使用- e选项才能使转义符生效：

```
$ echo -e "here is a tab\there are two tabs\t\tok"
here is a tab   here are two tabs                ok
```

如果想把一个字符串输出到文件中，使用重定向符号 >。在下面的例子中一个字符串被重定向到一个名为myfile的文件中：

```
$ echo "The log files have all been done"> myfile
```

或者可以追加到一个文件的末尾，这意味着不覆盖原有的内容：

```
$ echo "$LOGNAME carried them out at `date`">>myfile
```

现在让我们看一下myfile文件中的内容：

```
$ pg myfile
 The log files have all been done
 root carried them out at Sat May 22 18:25:06 GMT 1999
```

初涉shell的用户常常会遇到的一个问题就是如何把双引号包含到 echo命令的字符串中。引号是一个特殊字符，所以必须要使用反斜杠 \来使shell忽略它的特殊含义。假设你希望使用echo命令输出这样的字符串：“ /dev/rmt0 ”，那么我们只要在引号前面加上反斜杠\即可：

```
$ echo "\"/dev/rmt0"\"
"/dev/rmt0"
```

5.2 read

可以使用read语句从键盘或文件的某一行文本中读入信息，并将其赋给一个变量。如果只

指定了一个变量，那么 read将会把所有的输入赋给该变量，直至遇到第一个文件结束符或回车。

它的一般形式为：

```
read varible1 varible2 ...
```

在下面的例子中，只指定了一个变量，它将被赋予直至回车之前的所有内容：

```
$ read name
Hello I am superman
$ echo $name
Hello I am superman
```

在下面的例子中，我们给出了两个变量，它们分别被赋予名字和姓氏。 shell将用空格作为变量之间的分隔符：

```
$ read name surname
John Doe
$ echo $name $surname
John Doe
```

如果输入文本域过长， Shell 将所有的超长部分赋予最后一个变量。下面的例子，假定要读取变量名字和姓，但这次输入三个名字；结果如下；

```
$ read name surname
John Lemon Doe
$ echo $name
John
$ echo $surname
Lemon Doe
```

在上面的例子中，如果我们输入字符串 John Lemon Doe，那么第一个单词将被赋给第一个变量，而由于变量数少于单词数，字符串后面的部分将被全部赋给第二个变量。

在编写shell脚本的时候，如果担心用户会对此感到迷惑，可以采用每一个 read语句只给一个变量赋值的办法：

```
$ pg var_test
#!/bin/sh
# var_test
echo "First Name :\c"
read name
echo "Middle Name :\c"
read middle
echo "Last name :\c"
read surname
```

用户在运行上面这个脚本的时候，就能够知道哪些信息赋给了哪个变量。

```
$ var_test
First Name : John
Middle Name : Lemon
Surname : Doe
```

如果是LINUX系统，那么.....

别忘了使用“ -n ”选项。

```
$ pg var_test
#!/bin/sh
# var_test
echo "First Name :\c"
```

（续）

```
read name
echo "Middle Name :\c"
read middle
echo "Last name :\c"
read surname
```

5.3 cat

cat是一个简单而通用的命令，可以用它来显示文件内容，创建文件，还可以用它来显示控制字符。在使用cat命令时要注意，它不会在文件分页符处停下来；它会一下显示完整个文件。如果希望每次显示一页，可以使用 more命令或把cat命令的输出通过管道传递到另外一个具有分页功能的命令中，请看下面的例子：

```
$ cat myfile | more
```

或

```
$ cat myfile | pg
```

cat命令的一般形式为：

```
cat [options] filename1 ... filename2 ...
```

cat命令最有用的选项就是：

-v 显示控制字符

如果希望显示名为myfile的文件，可以用：

```
$ cat myfile
```

如果希望显示myfile1、myfile2、myfile3这三个文件，可以用：

```
$ cat myfile1 myfile2 myfile3
```

如果希望创建一个名为bigfile的文件，该文件包含上述三个文件的内容，可以把上面命令的输出重定向到新文件中：

```
$ cat myfile1 myfile2 myfile3 > bigfile
```

如果希望创建一个新文件，并向其中输入一些内容，只需使用 cat命令把标准输出重定向到该文件中，这时 cat命令的输入是标准输入 —— 键盘，你输入一些文字，输入完毕后按<CTRL-D>结束输入。这真是一个非常简单的文字编辑器！

```
$ cat > myfile
This is great
<CTRL-D>
$ pg myfile
This is great
```

还可以使用cat命令来显示控制字符。这里有一个对从 DOS机器上ftp过来的文件进行检察的例子，在这个例子中，所有的控制字符 <CTRL-M>都在行末显示了出来。

```
$ cat -v life.tct
ERROR ON REC AS12^M
ERROR ON REC AS31^M
```

有一点要提醒的是，如果在敲入了 cat以后就直接按回车，该命令会等你输入字符。如果你本来就是要输入一些字符，那么它除了会在你输入时在屏幕上显示以外，还会再回显这些

内容；最后按<CTRL-D>结束输入即可。

5.4 管道

可以通过管道把一个命令的输出传递给另一个命令作为输入。管道用竖杠 |表示。它的一般形式为：

命令1 |命令2

其中|是管道符号。

在下面的例子中，在当前目录中执行文件列表操作，如果没有管道的话，所有文件就会显示出来。当 shell看到管道符号以后，就会把所有列出的文件交给管道右边的命令，因此管道的含义正如它的名字所暗示的那样：把信息从一端传送到另外一端。在这个例子中，接下来grep命令在文件列表中搜索 quarter1.doc：

```
$ ls | grep quarter1.doc
quarter1.doc
```

让我们再来用一幅图形象地讲解刚才的例子（见图 5-1）：

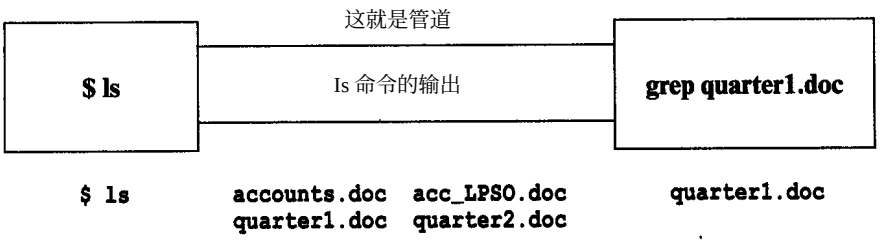

图5-1 管道

sed、awk和grep都很适合用管道，特别是在简单的一行命令中。在下面的例子中， who命令的输出通过管道传递给 awk命令，以便只显示用户名和所在的终端。

```
$ who | awk '{print $1"\t"$2}'
matthew    pts/0
louise     pts/1
```

如果你希望列出系统中所有的文件系统，可以使用管道把 df命令的输出传递给 awk命令，awk显示出其中的第一列。你还可以再次使用管道把 awk的结果传递给 grep命令，去掉最上面的题头filesystem。

```
$ df -k | awk '{print $1}'| grep -v "Filesystem"
/dev/hda5
/dev/hda8
/dev/hda6
/dev/hdb5
/dev/hdb1
/dev/hda7
/dev/hda1
```

当然，你没准还会希望只显示出其中的分区名，不显示 /dev/部分，这没问题；我们只要在后面简单地加上另一个管道符号和相应的 sed命令即可。

```
$ df -k | awk '{print $1}'| grep -v "Filesystem"|sed s'/\/dev\///g'
hda5
```

```
hda8
hda6
hdb5
hdb1
hda7
hda1
```

在这个例子中，我们先对一个文件进行排序，然后通过管道输送到打印机。

```
$ sort myfile | lp
```

5.5 tee

tee命令作用可以用字母 T来形象地表示。它把输出的一个副本输送到标准输出，另一个副本拷贝到相应的文件中。如果希望在看到输出的同时，也将其存入一个文件，那么这个命令再合适不过了。

它的一般形式为：

```
tee -a files
```

其中，-a表示追加到文件末尾。

当执行某些命令或脚本时，如果希望把输出保存下来， tee命令非常方便。

下面我们来看一个例子，我们使用 who命令，结果输出到屏幕上，同时保存在 who.out文件中：

```
$ who | tee who.out
louise    pts/1    May 20 12:58 (193.132.90.9)
matthew   pts/0    May 20 10:18 (193.132.90.1)

cat who.out
louise    pts/1    May 20 12:58 (193.132.90.9)
matthew   pts/0    May 20 10:18 (193.132.90.1)
```

可以用图5-2来表示刚才的例子。

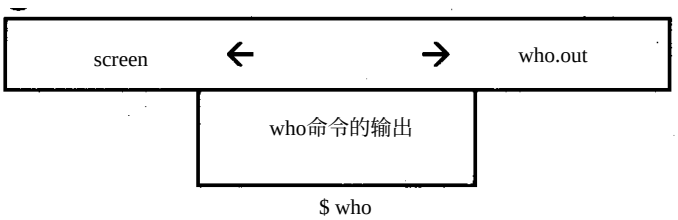

图5-2 tee

在下面的例子中，我们把一些文件备份到磁带上，同时将所备份的文件记录在 tape.log文件中。由于需要不断地对文件进行备份，为了保留上一次的日志，我们在 tee命令中使用了-a选项。

```
$ find  etc usr/local home -depth -print | cpio -ovC65536 -O \
/dev/rmt/0n | tee -a tape.log
```

在上面的例子中，第一行末尾的反斜杠 \告诉shell该命令尚未结束，应从下面一行继续读入该命令。

可以在执行脚本之前，使用一个 echo命令告诉用户谁在执行这个脚本，输出结果保存在

什么地方。

```
$ echo " myscript is now running, check out any errors...in
myscript.log" | tee -a myscript.log

$ myscript | tee -a myscript.log
```

如果不想把输出重定向到文件中，可以不这样做，而是把它定向到某个终端上。在下面的例子中，一个警告被发送到系统控制台上，表明一个磁盘清理进程即将运行。

```
$ echo "stand-by disk cleanup starting in 1 minute"| tee /dev/console
```

可以让不同的命令使用同一个日志文件，不过不要忘记使用 -a选项。

```
$ sort myfile | tee -a accounts.log

$ myscript | tee -a accounts.log
```

5.6 标准输入、输出和错误

当我们在shell中执行命令的时候，每个进程都和三个打开的文件相联系，并使用文件描述符来引用这些文件。由于文件描述符不容易记忆， shell同时也给出了相应的文件名。

下面就是这些文件描述符及它们通常所对应的文件名：

文　件	文件描述符
输入文件 —— 标准输入	0
输出文件 —— 标准输出	1
错误输出文件 —— 标准错误	2

系统中实际上有12个文件描述符，但是正如我们在上表中所看到的， 0、1、2是标准输入、输出和错误。可以任意使用文件描述符 3到9。

5.6.1 标准输入

标准输入是文件描述符0。它是命令的输入，缺省是键盘，也可以是文件或其他命令的输出。

5.6.2 标准输出

标准输出是文件描述符 1。它是命令的输出，缺省是屏幕，也可以是文件。

5.6.3 标准错误

标准错误是文件描述符 2。这是命令错误的输出，缺省是屏幕，同样也可以是文件。你可能会问，为什么会有一个专门针对错误的特殊文件？这是由于很多人喜欢把错误单独保存到一个文件中，特别是在处理大的数据文件时，可能会产生很多错误。

如果没有特别指定文件说明符，命令将使用缺省的文件说明符（你的屏幕，更确切地说是你的终端）。

5.7 文件重定向

在执行命令时，可以指定命令的标准输入、输出和错误，要实现这一点就需要使用文件

重定向。表5-1列出了最常用的重定向组合，并给出了相应的文件描述符。

在对标准错误进行重定向时，必须要使用文件描述符，但是对于标准输入和输出来说，这不是必需的。为了完整起见，我们在表 5-1中列出了两种方法。

表5-1 常用文件重定向命令

command > filename	把标准输出重定向到一个新文件中
command >> filename	把标准输出重定向到一个文件中 (追加)
command 1 > fielname	把标准输出重定向到一个文件中
command > filename 2>&1	把标准输出和标准错误一起重定向到一个文件中
command 2 > filename	把标准错误重定向到一个文件中
command 2 >> filename	把标准输出重定向到一个文件中 (追加)
command >> filename 2>&1	把标准输出和标准错误一起重定向到一个文件中 (追加)
command < filename >filename2	command命令以filename文件作为标准输入，以 filename2文件作为标准输出
command < filename	command命令以filename文件作为标准输入
command << delimiter	从标准输入中读入，直至遇到 delimiter分界符
command <&m	把文件描述符 m作为标准输入
command >&m	把标准输出重定向到文件描述符 m中
command <&-	关闭标准输入

5.7.1 重定向标准输出

让我们来看一个标准输出的例子。在下面的命令中，把 /etc/passwd文件中的用户 ID域按照用户命排列。该命令的输出重定向到 sort.out文件中。要提醒注意的是，在使用 sort命令的时候(或其他含有相似输入文件参数的命令)，重定向符号一定要离开 sort命令两个空格，否则该命令会把它当作输入文件。

```
$ cat passwd | awk -F: '{print $1}' | sort  1>sort.out
```

从表5-1中可以看出，我们也可以使用如下的表达方式，结果和上面一样：

```
$ cat passwd | awk -F: '{print $1}' | sort  >sort.out
```

可以把很多命令的输出追加到同一文件中。

```
$ ls -l | grep ^d >>files.out
$ ls account* >> files.out
```

在上面的例子中，所有的目录名和以 account开头的文件名都被写入到 file.out文件中。

如果希望把标准输出重定向到文件中，可以用 >filename。在下面的例子中， ls命令的所有输出都被重定向到ls.out文件中：

```
$ ls >ls.out
```

如果希望追加到已有的文件中 (在该文件不存在的情况下创建该文件)，那么可以使用 >>filename：

```
$ pwd >>path.out
$ find . -name "LPSO.doc" -print >>path.out
```

如果想创建一个长度为0的空文件，可以用'>filename'：

```
$ >myfile
```

5.7.2 重定向标准输入

可以指定命令的标准输入。在awk一章就会遇到这样的情况。下面给出一个这样的例子：

```
$ sort  < name.txt
```

在上面的命令中，sort命令的输入是采用重定向的方式给出的，不过也可以直接把相应的文件作为该命令的参数：

```
$ sort name.txt
```

在上面的例子中，还可以更进一步地通过重定向为 sort命令指定一个输出文件 name.out。这样屏幕上将不会出现任何信息(除了错误信息以外)：

```
$ sort  <name.txt >name.out
```

在发送邮件时，可以用重定向的方法发送一个文件中的内容。在下面的例子中，用户louise将收到一个邮件，其中含有文件contents.txt中的内容：

```
$ mail louise < contents.txt
```

重定向操作符command << delimiter是一种非常有用的命令，通常都被称为“此处”文挡。我们将在本书后面的章节深入讨论这一问题。现在只介绍它的功能。 shell将分界符delimiter之后直至下一个同样的分界符之前的所有内容都作为输入，遇到下一个分界符， shell就知道输入结束了。这一命令对于自动或远程的例程非常有用。可以任意定义分界符 delimiter，最常见的是EOF，而我最喜欢用MAYDAY，这完全取决于个人的喜好。还可以在 <<后面输入变量。下面给出一个例子，我们创建了一个名为 myfile的文件，并在其中使用了TERM和LOGNAME变量。

```
$ cat >> myfile <<MAYDAY
> Hello there I am using a $TERM terminal
> and my user name is $LOGNAME
> bye...
> MAYDAY

$ pg myfile
Hello there I am using a vt100 terminal
and my user name is dave
bye...
```

5.7.3 重定向标准错误

为了重定向标准错误，可以指定文件描述符 2。让我们先来看一个例子，因为举例子往往会让人更容易明白。在这个例子中， grep命令在文件missiles中搜索trident字符串：

```
$ grep "trident" missiles
grep: missiles: No such file or directory
```

grep命令没有找到该文件，缺省地向终端输出了一个错误信息。现在让我们把错误重定向到文件/dev/null中(实际就上是系统的垃圾箱)：

```
$ grep "trident" missiles 2>/dev/null
```

这样所有的错误输出都输送到了/dev/null，不再出现在屏幕上。

如果你在对更重要的文件进行操作，可能会希望保存相应的错误。下面就是一个这样的例子，这一次错误被保存到grep.err文件中：

```
$  grep "trident" missiles 2>grep.err
$ pg grep.err
grep: missiles: No such file or directory
```

还可以把错误追加到一个文件中。在使用一组命令完成同一个任务时，这种方法非常有用。在下面的例子中，两个 grep命令把错误都输出到同一个文件中；由于我们使用了 >>符号进行追加，后面一个命令的错误(如果有的话)不会覆盖前一个命令的错误。

```
$ grep "LPSO" * 2>>account.err
$ grep "SILO" * 2>>account.err
```

5.8 结合使用标准输出和标准错误

一个快速发现错误的方法就是，先将输出重定向到一个文件中，然后再把标准错误重定向到另外一个文件中。下面给出一个例子：

我有两个审计文件，其中一个的确存在，而且包含一些信息，而另一个由于某种原因已经不存在了(但我不知道)。我想把这两个文件合并到accounts.out文件中。

```
$ cat account_qtr.doc account_end.doc 1>accounts.out 2>accounts.err
```

现在如果出现了错误，相应的错误将会保存在 accounts.err文件中。

```
$ pg accounts.out
AVBD 34HJ   OUT
AVFJ   31KO  OUT
...

$ pg accounts.err
cat: account_end.doc: No such file or directory
```

我事先并不知道是否存在 account_end.doc文件，使用上面的方法能够快速发现其中的错误。

5.9 合并标准输出和标准错误

在合并标准输出和标准错误的时候，切记 shell是从左至右分析相应的命令的。下面给出一个例子：

```
$ cleanup >cleanup.out 2>&1
```

在上面的例子中，我们将 cleanup脚本的输出重定向到 cleanup.out文件中，而且其错误也被重定向到相同的文件中。

```
$ grep "standard"* > grep.out 2>&1
```

在上面的例子中， grep命令的标准输出和标准错误都被重定向到 grep.out文件中。你在使用前面提到的“ 此处 ”文挡时，有可能需要把所有的输出都保存到一个文件中，这样万一出现了错误，就能够被记录下来。通过使用 2>&1就可以做到这一点，下面给出一个例子：

```
$ cat>> filetest 2>&1 <<MAYDAY
> This is my home $HOME directory
> MAYDAY
$ pg filetest
This is my home /home/dave directory
```

上面的例子演示了如何把所有的输出捕捉到一个文件中。在使用 cat命令的时候，这可能

没什么用处，不过如果你使用“此处”文挡连接一个数据库管理系统 (例如使用 isql连接sybase)或使用ftp，这一点就变得非常重要了，因为这样就可以捕捉到所有的错误，以免这些错误在屏幕上一闪而过，特别是在你不在的时候。

5.10 exec

exec命令可以用来替代当前shell；换句话说，并没有启动子shell。使用这一命令时任何现有环境都将会被清除，并重新启动一个shell。它的一般形式为：

```
exec command
```

其中的command通常是一个shell脚本。

我所能够想像得出的描述exec命令最贴切的说法就是：它践踏了你当前的 shell。

当这个脚本结束时，相应的会话可能就结束了。 exec命令的一个常见用法就是在用户的.profile最后执行时，用它来执行一些用于增强安全性的脚本。如果用户的输入无效，该shell将被关闭，然后重新回到登录提示符。 exec还常常被用来通过文件描述符打开文件。

记住， exec在对文件描述符进行操作的时候（也只有在这时），它不会覆盖你当前的shell。

5.11 使用文件描述符

可以使用exec命令通过文件描述符打开和关闭文件。在下面的例子中，我选用了文件描述符4，实际上我可以在4到9之间任意选择一个数字。下面的脚本只是从 stock.txt文件中读了两行，然后把这两行回显出来。

该脚本的第一行把文件描述符 4指定为标准输入，然后打开 stock.txt文件。接下来两行的作用是读入了两行文本。接着，作为标准输入的文件描述符 4被关闭。最后，line1和line2两个变量所含有的内容被回显到屏幕上。

```
$ pg f_desc
#!/bin/sh
# f_desc
exec 4<&0 0<stock.txt
read line1
read line2
exec  0<&4
echo $line1
echo $line2
```

下面是这个小小的股票文件stock.txt的内容：

```
$ pg stock.txt
Crayons Assorted 34
Pencils Light    12
```

下面是该脚本的运行结果：

```
$ f_desc
Crayons Assorted 34
Pencils Light    12
```

上面是一个关于文件描述符应用的简单例子。它看起来没有什么用处。在以后讲解循环的时候，将会给出一个用文件描述符代替 cp命令拷贝文本文件的例子。

5.12 小结

本书通篇可见重定向的应用，因为它是 shell中的一个重要部分。通过重定向，可以指定命令的输入；如果有错误的话，可以用一个单独的文件把它们记录下来，这样就可以方便快捷地查找问题。

这里没有涉及的就是文件描述符的应用 (3～9)。要想应用这些文件描述符，就一定会涉及循环方法，在后面讲到循环方法的时候，我们会再次回过头来讲述有关文件描述符的问题。