第18章　编写shell命令脚本程序

如果你需要下列问题的一个快速解决方案	请查阅节号
编写一个命令脚本程序	18.2.1
给变量赋值	18.2.2
使用某个变量的值	18.2.3
提示和接受输入数据	18.2.4
忽略元字符	18.2.5
编写条件if语句	18.2.6
接受命令行输入	18.2.7
添加注释语句	18.2.8
添加一个帮助组件	18.2.9
添加一个for循环语句	18.2.10
添加一个while循环语句	18.2.11
添加一个菜单	18.2.12
二次检查用户输入数据	18.2.13
比较文件、字符串以及规则表达式	18.2.14
结束一个命令脚本程序	18.2.15
测试一个命令脚本程序	18.2.16
调试一个命令脚本程序	18.2.17

18.1　概述

shell命令脚本程序是一种极其有用的系统管理工具。它可以用来自动完成通常在命令行上执行的重复或者复杂的工作。对那些并不很了解 Linux操作系统的用户来说，命令脚本程序还可以用来自动完成一些基本的任务，或者用来完成一些只有对 Linux操作系统有着深厚功底的人才能采用其他方法完成的任务。

18.1.1　shell概述

shell是用户与Linux操作系统内核之间的接口。它是用户工作在其中的环境，而用户可以选择使用哪一种 shell来进行工作，每一种 shell都各有特色，因而吸引了有着不同个人需要的人，帮助他们完成工作。每一种 shell对用户输入的命令进行处理时，还要使用它自己的语法、环境变量等等：

- bash shell是Linux操作系统的缺省shell，也是根帐户最经常使用的shell。
- C shell之所以得到这么个名字是因为它的语法很像编程用的 C语言；因此它在使用 C语言的程序员中很流行。
- Linux操作系统中的Korn shell实际上并不是标准的Korn shell，前者只是后者的一个名为pdksh（Public Domain Korn Shell）的公共域版本。它包括了一些 C shell中的特色，又加上了一些新的特色，这就使它成为大量编写 shell命令脚本程序的人们的首选。它在Red Hat和Caldera发行版本的CD-ROM光盘中都有相应的软件包。

相关解决方案	请查阅节号
挂装到文件系统上	9.2.2
安装一个RPM软件包	15.2.1

18.1.2 良好的命令脚本程序编程习惯

对于命令脚本程序的编写工作来说，并没有什么放之四海而皆准的指导原则，下面列出的是一些应该具备的良好编程习惯：

- 基于文件和路径管理两方面的考虑，把全部的 shell命令脚本程序统一保存到某个固定位置的做法是非常明智的。不论是为超级用户还是为普通用户编写命令脚本程序，这一条都是适用的。许多人会在他们的用户目录中建立一个名为 bin（~/bin）的目录。
- 但上面这个做法需要注意的是：如果系统管理员为一部分用户或者全部用户编写了命令脚本程序，就要考虑应该把这些命令脚本程序存放到什么位置，其原则也就是为某个程序选择存放位置所必须考虑的因素。
- 不要过分相信自己的记忆力，认为能够记住所编写的命令脚本程序的功能与实现方法的每一个细节。如果需要在几个月或几年后对命令脚本程序进行修改或升级的话，最好还是尽可能在程序中加上好的注释。
- 使用缩进格式排列命令脚本程序里的条件语句，这样就可以很清楚地看出哪个命令属于哪个子句。在18.2.6节中给出了一个这类格式的例子。

18.1.3 编程示范

下面我们将通过一个简单的示范程序带领你走过编写命令脚本程序的全过程。在这些步骤的介绍中，还包括了在计划和编写 shell命令脚本程序时容易忽略的因素。不必担心，编写命令脚本程序实现起来并不难。

1. 命令脚本程序的功能

这个命令脚本程序的基本功能是对一个 Web主机上的某个Web文件进行大量、相同的修改。因为这个例子中的管理员没有足够的时间来编写一个复杂的命令脚本程序，所以这个例子本身也就只能够完成全部修改工作中最初始的部分。这个命令脚本程序可以在今后逐步加以改进，使它能够完成更大和更复杂的任务 —— 比如对一组被选中的目录中的每一个文件进行修改等等。

2. 准备编写命令脚本程序

可能很多人都会认为立刻开始编写命令脚本程序是最有效率的。但是在现实情况中，先进行一些计划准备工作将会使程序的编写工作进展得更顺利。要不然，就会因为又发现还需要考虑某些其他方面而不得不回过头去重复修改以前的程序。下面是编写这个命令脚本程序需要的几个步骤：

1) 首先，决定把命令脚本程序解释器运行在哪一种 shell环境下。因为这个例子是准备编写一个系统监管性质的命令脚本程序，缺省的 shell又是bash，所以我们就沿用bash shell好了。

2) 在开始编程之前，首先需要确定在命令行提示符完成这个工作通常都会用到哪些命令。在这个例子中，第一个目标是搜索与替换一个 Web页上面的某个 URL地址；最终的目标是建立一个能够完成更多功能的程序集合。我们选择使用 sed命令来完成这个搜索与替换操作。

3) 请仔细阅读sed命令的使用手册页和其他有关文档。事实上，就是不使用命令脚本程序的方法，sed命令也可以完成这个任务；编写命令脚本程序的目的就是使够更简单轻松地执行同样或者类似任务，而不必再去学习如何掌握sed命令复杂的语法。

4) 引起大家注意的第一个问题就是必须使用一个分隔符来把搜索与替换的两个短语分隔开。缺省的分隔符是正斜杠（/）。但是，搜索与替换的两个短语都是URL地址，因此它们自己本身就将带有斜杠字符。这个问题我们有两种方法可以解决：

- 使用转义字符（escape character）——即反斜杠（\）——来转换正斜杠字符。转义字符序列会通知解释器程序，该序列中的字符应该按照它显示的样子而非一个特殊字符（比如分隔符）来对待。举例来说，要想查找URL地址“www.green.org/lime”，需要输入下面的搜索短语：

```
www.green.org\ /lime\ /
```

在本章的例子中，我们将选用上面的方法，因为这是程序接受正规表达式所使用的标准格式。

- 使用完全不同的一个分隔符。并不是只有正斜杠符号才能被用做分隔符。举例来说，因为在搜索与替换短语中不会使用到URL地址中的“http://”部分，所以冒号（:）也可以用来作为分隔符。

窍门 “元字符(meta -character)”是那些给出关于其他字符信息的字符。它们是“特殊的字符”；用到它们的时候一般都是为了特殊的目的，不会是按照它们本身的样子被使用的。举例来说，正斜杠（/）在sed命令中通常就被当做是一个元字符，因为它是用来分开命令中不同组成部分的。

5)现在，写出sed命令的语句。我们打算搜索并替换的是URL地址“www.peach.org”；打算把它替换为“http://watermelon.com/pink”。因为第一个部分——即“hppt://”——不需要改变，整个语句应该是下面的样子：

```
s / www\ .peach\ .org\ //watermelon\ .com\ /pink\ //
```

语句中各部分的意义如下所示：

- s/ ——告诉sed命令这个语句执行的是替换操作。
- www\ .peach\ .org\ // ——就是我们查找的文本。之所以会有这么多的转换字符（\），是因为句号（.）和URL地址中的正斜杠（/）符号都必须被转换。请注意在语句的最后还有一个正斜杠符号，它表示sed命令的搜索短语到此结束。
- watermelon\ .com\ /pink\ / ——是我们想用来替换原文的文字内容。
- 语句中最后的一个斜杠表示sed命令到此结束。

3. 建立命令脚本程序文件

要想建立这个命令脚本程序，首先需要我们打开并编辑一个文本文件：

1) 输入“mkdir ~/bin”命令，在用户目录中建立一个bin目录。

2) 输入“cd ~/bin”命令，把路径切换到bin目录。

3) 输入“PATH = $PATH:~/bin”命令，把bin目录添加到path路径中去。

4) 使用喜欢的文本编辑器程序打开这个文件。如果使用的是vi编辑器的话，可以输入“vi webchange”。

4. 编写命令脚本程序

到这里我们就已经把命令和语法都确定下来了，现在开始编写命令脚本程序的内容：

1)每个shell命令脚本程序的第一行都是一个解释器声明语句。对这个命令脚本程序来说，它的解释器是bash shell。所以webchange文件的第一行内容就是：

```
# ! /bin/bash
```

从此往下，在命令脚本程序中添加的任何内容都将提交给 bash shell的解释器。

2) 因为建立这个命令脚本程序的最初目的只是完成一个任务，但最终的目的是把它的用途扩展到更普遍的情况，所以千万不要忘记在学习 sed命令时掌握的知识。请使用井字号（ #）打头，在命令脚本程序中加上注释语句。

注意 如果在命令脚本程序中某一行的开头用了“ #! ”字符，shell命令会认为这是在定义一个用来运行命令脚本程序的程序。

这个命令脚本程序的一个注释例子如下所示：

```
# Sed statement built to search for the first URL and
        # replace it with the
# second . Notice the escape
        # characters ensuring that the periods and front
# slashes are seen as actual characters and not meta-characters .
```

3) 接下来，是sed命令的语句本身：

```
sed s / www\ .peach\.org\ // watermelon \ .com \ / pink \ // test.html
```

我们先不要冒险去真的修改某个 Web文件，把这个 Web文件拷贝到一个临时测试文件中（本例子中这个文件就是test.html），这样就可以在使用这个命令脚本程序处理重要的数据之前，确认它不会造成什么无法预见的问题。

4) 保存文件并退出。

5) 使用类似于“ chmod +x webchange ”这样的命令修改文件的存取权限，使它成为一个可执行文件。

5. 测试命令脚本程序

现在来测试命令脚本程序，确认它可以正常执行。如果：

- 已经在path语句中加上了这个命令脚本程序的目录名。
- 已经在~/bin目录中拷贝或者建立了一个测试文件。
- 已经使用类似于“ chmod +x webchange ”这样的命令把这个命令脚本程序设置为一个可执行文件。

那么，就可以输入这个文件的文件名“ webchange ”，对它进行测试。

18.2 快速解决方案

18.2.1 编写一个命令脚本程序

编写一个shell命令脚本程序需要经过以下几个基本步骤：

1) 从编辑包含用户的命令脚本程序的文本文件开始。

2) 添加shell定义语句，这样Linux操作系统就可以知道用户使用的是哪一种 shell语言。

3) 编写shell命令脚本程序

4) 保存命令脚本程序并退出。

5) 使用“ chmod +x webchange ”命令改变这个命令脚本程序的存取权限和所有权限，使它可以被适当的用户帐户执行使用。

6) 运用这个命令脚本程序测试它。

7) 如有必要可进行程序纠错，然后返回第 6步。

8) 如有必要可把完整的命令脚本程序移动到它该去的位置。

9) 如有必要可在path 语句中加上它的路径。

18.2.2 给变量赋值

你可以直接给变量分配一个值，也可以让变量等于某个命令的输出结果。具体如何给变量赋值主要取决于准备分配给变量的值来自什么地方。

1. 直接赋值

请按照下面的格式直接给变量赋一个值：

```
variable = value
```

请记住，如果想在某个数据值使用巨字符，就必须转换或者忽略它们（请阅读 18.2.5节中的内容）。另外，如果使用了包含空格字符的字符串，就必须把它们放在双引号（ ""）中，如下所示：

```
name1 = " Smith , Adam "
```

2. 通过命令给变量赋值

如果想让变量等于某个命令的输出结果，需要把命令放在单引号（'）中，格式如下所示：

```
variable = ' command '
```

举例来说，使用：

```
now = 'date'
```

把变量now的值设置为date命令的输出结果，也就是当前的日期和时间。

18.2.3 使用某个变量的值

如果想使用某个变量的值，按照下面的格式输入这个变量：

```
$variable
```

18.2.4 提示和接受输入数据

在命令脚本程序中你可以提示用户输入不要的信息。先使用 echo命令（请阅读18.2.17节中的内容）显示提示信息。然后按照下面的格式使用 read命令：

```
read variable_name
```

read命令告诉命令脚本程序暂停执行，等待用户的输入；用户按下回车键结束输入后命令脚本程序继续向下执行。

18.2.5 忽略元字符

下面是一个元字符清单。如果你想把它们用作普通字符(比如说在 echo语句中)就需要用一个反斜杠对它们进行转换：

/ . > * ? $

其他使元字符普通化的方法还有：

- 用单引号（''）把元字符或者包含了元字符的字符串括起来。如下所示：

```
echo ' --> Cut Here and send $8 with Application <-- '
```

- 把元字符或者包含了元字符的字符串用双引号（ " "）括起来。但是这个方法不能够使美元符号（$）和单引号（' '）普通化。如果想使用双括号得到与前面这个例子中同样的输出效果，必须输入如下所示的内容：

```
echo " --> Cut Here and send \$8 with Application <-- "
```

18.2.6 编写条件if语句

当在命令脚本程序中加上一个if语句的时候，它可以有表18-1所示的组成部分。

表18-1 if语句的组成部分

组成部分	用途	必要吗
if	这类语句的开始部分	是
fi	这类语句的结束部分	是
then	如果if语句为真的话，需要执行的任务	是
else	如果if语句为假的话，需要执行的任务	否
elif	意思是“ else if ”。如果第一个if语句为假，开始假如接下来的这个if语句	否

只要遵守正确的语句封闭规则（下面就要讨论到），就可以根据自己的需要多层次嵌套这些命令。

注意 嵌套语句的意思是在语句中使用语句。

1. if-then语句

这些组成部分按照一定的模式可以有许多种组合。其中最简单的就只包括一个条件和一个结果，如下所示：

```
if conditions
        then result
fi
```

注意 if语句必须以fi结束。

2. if-then-else语句

通常，当条件不能满足的时候，包括其他形式的结果也是不错的。加上了 else子句的这种语句格式如下所示：

```
if conditions
        then result
        else alternate_result
fi
```

3. if-then-else-elif语句

有时候条件过于复杂，一个if-then-else结构不能表达清楚。这样就需要在同一个语句中使用多个if结构。在这种情况下，可以加上一个elif（else if）子句，如下所示：

```
if conditions
        then result
        else alternate_result
        elif secondary_conditions
                then secondary_result
        fi
fi
```

注意 elif语句就像if语句一样，也必须以fi结束。

18.2.7 接受命令行输入

如果想让一个命令脚本程序带有执行参数，可以在从命令行上调用这个命令脚本程序时传递过去。在大多数shell命令脚本语言中，这些参数分别被分配为$1到$9，而这个命令脚本程序本身则被分配为$0。两个参数之间使用空格分开；如果参数内部有空格，需要使用双引号（" "）把整个参数都括起来。下面给出两个例子来说明这一点：

• 第一个例子是：

```
myscript a b cdef g
```

在这个例子中，命令脚本程序的名称即$0的值是myscript。$1的值是a，$2的值是b，$3的值是cdef，而$4的值是g。

• 第二个例子是：

```
myscript "a b" cdef g
```

在这个例子中，命令脚本程序的名称即$0的值还是myscript。$1的值是a b，$2的值是cdef，而$3的值是g。

18.2.8 添加注释语句

注释语句在命令脚本程序文件中是以井字号（#）打头的。注释部分从这个符号后面开始一直到用户在这一行上按下回车键为止。请看下面的两个例子：

1) 一条注释语句可以从一行开始，延伸到好几行。

```
# Comments are very useful in scripts, since you might learn
# a lot while writing them but forget it by the next time
# you look at the code .
```

2) 一条注释语句可以跟在程序代码的后面，写满一行为止。

```
echo "Hello world ."   # Prints Hello world .
```

18.2.9 添加一个帮助组件

如果一个命令脚本程序在命令行上可能包括比较复杂的数据，那么给这个命令脚本程序加上一个帮助组件是很有用的。只要你觉得可能会在运行它之前忘记什么东西，就应该给这个命令脚本程序加上帮助性的内容。

请按照下面的方法添加一个帮助组件：

1)选择一种或者多种访问帮助组件的方法。其中常见的有：

• 输入命令脚本程序的名称，不带任何参数。

• 输入命令脚本程序的名称，再加上一个单词“ help ”。
• 输入命令脚本程序的名称，再加上一个参数“ -help ”。
我们这里选用第一个方法，因为它是Linux操作系统中shell命令脚本和程序编程的常见做法。
2) 拟订好帮助语句的文字内容。
3) 在这个命令脚本程序可是的部分建立一个条件 if语句。其测试条件检查$1参数（请阅读18.2.7节中关于$1参数的介绍）是否为空。这个语句的简单形式如下所示：

```
if $1 = ""
        then echo " Here is how to use this script ."
exit
fi
```

4) 你可以把命令脚本程序放在这个 if语句的else子句部分，但是必须在 exit语句之前。或者从fi后面写命令脚本程序的正式内容。

18.2.10 添加一个for循环语句

如果某个代码段需要重复执行固定次数（比如对命令行上的每个参数各执行一次、对目录中的每一个文件各执行一次、或者对星期的每一天各执行一次等等情况），for循环语句就是一个很不错的选择。for语句的基本结构如下所示：

```
for item_to_count do
        Commands  to execute within loop
done
```

注意 在for语句中还可以再使用其他循环语句或者条件语句。

18.2.11 添加一个while循环语句

如果需要让某个代码段保持执行，直到某个条件满足为止， while循环语句就是不错的选择。while语句的基本结构如下所示：

```
while item_to_watch do
        Commands to execute within loop
done
```

18.2.12 添加一个菜单

如果想在命令脚本程序中加上一个菜单，使用户可以对某些个选项进行选择，请使用 case命令。case常用的结构如下所示：

```
echo "Choose one of the following options:"
echo series of options
...
read variable
case $variable in
     potential_value) Action;;
     ...
esac
```

举例来说，现在有一些对 Linux操作系统的使用方法不很熟悉的用户。那么你可能会为他

们编写一个包含了如下内容的命令脚本程序：

```
echo "Choose one of the following options to work with files:"
echo "[E]dit or create a file"
echo "[D]elete a file"
echo "[M]ove a file"
echo "[R]ename a file"
read action
case $action in
     E|e) echo "Name of file to edit? Please provide full path."
          read filename
          pico $filename;;
     D|d) echo "Name of file to delete? Please provide full path."
          read filename
          rm $filename;;
     M|m) echo "Name of file to move? Please provide full path."
          read filename
          echo "Where do you want to move the file to?"
          read moveto
          mv $filename $moveto;;
     R|r) echo "Name of file to rename? Please provide full path."
          read filename
          echo "What do you want to rename the file to?"
          read newname
          mv $filename $newname;;
esac
```

注意 请注意，上面这个命令脚本程序没有对用户输入的菜单选择做有效性检查。

18.2.13 二次检查用户输入数据

虽然用户没有办法完全保证一个命令脚本程序不会出现任何意外问题，但是可以预先分析在哪些地方容易发生问题，并提前准备好处理措施。许多 shell提供的工具之一就是test命令。在与条件语句联合使用的时候，它也是很有用的。

这个命令（测试情况为真， true）最基本的形式如下所示：

```
test option_flag item_to_test
```

如果测试成功，返回值就是0（真， true）。反之就是1（假， false）。用户可以任意对这两个返回值进行测试。如果打算进行情况为假（ false）的测试，请使用下面的格式：

```
test ! option_flag item_to_test
```

1. 测试文件

最能够帮助避免出错的测试项目是对用户选中的文件进行测试，以保证命令脚本程序可以对它们进行处理。表18-2中列出了可以使用的文件测试参数。

表18-2 test命令的文件测试参数

测试参数	测试情况	0	1
-a	文件存在吗？	是	否
-b	它是一个块设备文件吗？	是	否
-c	它是一个字符设备文件吗？	是	否

(续)

测试参数	测试情况	0	1
-d	它是一个目录吗？	是	否
-f	它是一个普通文件吗？	是	否
-g	文件的setgid位已经被置位了吗？	是	否
-k	文件的sticky位已经被置位了吗？	是	否
-p	它是一个管道或者FIFO文件吗？	是	否
-r	文件是可读的吗？	是	否
-s	文件中有内容吗？	是	否
-u	文件的setuid位已经被置位了吗？	是	否
-w	文件是可写的吗？	是	否
-x	文件是可执行的吗？或者目录是可以搜索的吗？	是	否
-G	它与shell的拥有者是属于同一个分组ID的吗？	是	否
-L	它是一个符号链接吗？	是	否
-O	它是由shell的用户ID所拥有的吗？	是	否
-S	它是一个套接字吗？	是	否

举例来说，如果准备让用户输入一个文件名，然后需要对它进行测试，保证这个文件

- 存在
- 属于该用户
- 不是一个目录

那么，大概就会写下如下所示的代码段：

```
echo "Enter the name (with full path) of the file you wish to
delete:"
read filename
# Does the item exist?
if test -a $filename
   then
   # The item exists. Does it belong to the user?
   if test -O $filename
      then
      # The item exists. Is it a directory? If it is, the test
gives False.
      if test ! -d $filename
         then
         # All tests passed. Delete the file.
         rm $filename
         else
         echo "The item is a directory, not a file."
         exit 1
      fi
      else
      echo "The item is not yours to delete."
      exit 1
   fi
   else
   echo "The item does not exist."
   exit 1
fi
```

2. 测试字符串

你还可以测试文本字符串以便确认用户按要求输入了数据。对字符串进行测试的参数列在表18-3中。

举例来说，我们看看下面的代码段：

```
echo "Enter last name:"
read last
# If user entered no data, go into a loop that keeps requesting
# data until the user types something before they press Enter.
while test -z $last do
        echo "Enter last name:"
        read last
done
```

表18-3 test命令的字符串测试参数

测试参数	测试情况	0	1
-n	是非空字符串？	是	否
-z	是空字符串？	是	否

18.2.14 比较文件、字符串以及正则表达式

有时候，为了排序或者其他的目的，需要把两个事物进行比较。这个问题又把我们带回到test命令（详细资料请阅读18.2.13节中的内容）。这个命令可以比较两个事物，也可以对单独一个数据项进行测试。

使用test命令进行比较操作的基本格式如下所示：

```
test item1 option_flag item2
```

1. 比较文件

虽然test命令并不会真正比较两个文件的具体内容，但是它可以用来在文件之间进行一些基本的检查。请查看表18-4中给出的文件比较参数。

下面给出一个使用了其中一个比较操作的test命令示例：

```
test workdata -nt work_data
```

2. 比较字符串

你可以使用test命令比较两个字符串。请查看表18-5中给出的字符串比较参数。

举例来说，下面的比较结果为真，也就是结果为0值：

```
test cart = ca?t
```

窍门 比较模板是包括了通配符的一组字符。通配符代表了字符串中某些未知的部分。在模板中可以使用的通配符有以下几个：

- 一个星号（*）——代表一个或者多个未知字符。模板c*t可以代表cat、cart、court等数据项。
- 一个问号（?）——代表一个未知字符。模板c?t可以代表cat。模板c??t可以代表cart。
- 方括号（[ ]）——表示一个字符范围。模板c[a-c]t可以代表cat，但是不能够代表cart或者cot。

3. 比较数学表达式

你可以使用test命令比较等式和数字。请查看表 18-6中给出的算术比较参数。

举例来说，下面的比较结果为假，也就是结果为 1值：

```
test 1+4 -eq 5-2
```

表18-4 test命令的文件比较参数

比较参数	比较情况	0	1
-nt	哪一个文件比较新？	文件1比较新	文件2比较新
-ot	哪一个文件比较旧？	文件1比较旧	文件2比较旧
-ef	它们是同一个文件吗？	是，它们是同一个	否，它们不是同一个

表18-5 test命令的字符串比较参数

比较参数	比较情况	0	1
=	字符串与模板匹配吗？	是	否
!=	字符串与模板不相同吗？	是	否
<	按照字母表顺序，前一个字符串比后一个字符串靠前吗？	是	否
>	按照字母表顺序，后一个字符串比前一个字符串靠前吗？	是	否

表18-6 test命令的算术比较参数

比较参数	比较情况	0	1
-eq	表达式相等吗？	是	否
-ne	表达式相等吗？	否	是
-lt	哪一个表达式比较小？	第一个	第二个
-gt	哪一个表达式比较大？	第一个	第二个
-le	第一个表达式小于或者等于第二个表达式吗？	是	否
-ge	第一个表达式大于或者等于第二个表达式吗？	是	否

18.2.15 结束一个命令脚本程序

在一个命令脚本程序文件的末尾并没有什么特殊的记号或者代码可用。当 shell执行到命令脚本程序的末尾或者出现错误的情况时，它就会自动退出，并且把最后一个命令的执行结果作为程序出口状态值。如果想了解基本程序出口状态值更详细的资料，请阅读 18.2.16节中的内容。

下面给出一个例子：

```
if test $value9 -eq 0
   then
        exit 1              # Division by 0 error would result.
   else
        value16=`expr $value5 / $value9`
        echo "$value5 divided by $value9 equals $value16."
        exit
fi
```

在上面的例子中，语句的then部分有一个明确定义的程序出口状态值，却没有定义明确的else出口状态值。但是由于整个语句中没有出现错误，它的程序出口状态值将是 0。

18.2.16 测试一个命令脚本程序

在测试一个命令脚本程序的时候，请注意下面几个问题：

- 除非一个命令脚本程序经过了完全的测试，否则不要对有效文件（也就是那些重要的和系统上正在使用的文件）进行修改或者保存所做的修改。可以把修改保存到临时文件或者使用一个测试文件来执行命令脚本程序。
- 逐步测试命令脚本程序是很明智的做法。在命令脚本程序中增加一个新的功能，然后立即测试命令脚本程序。然后增加另外一个新功能，再进行测试。与一次性编写一个长长的命令脚本程序并把它作为一个大的整体进行测试相比，这个方法可以比较容易找出问题出现的原因。

18.2.17 调试一个命令脚本程序

本小节给出了一些在进行命令脚本程序调试的时候需要遵守的基本做法。这些做法可以把一个也许混乱不清的调试过程变成比较平静的搜索和缩小问题潜在原因的过程。

在bash shell中，有一个很方便的命令行调试工具，就是“ set -o xtrace ”或者“ set -x ”命令。如果在调试一个命令脚本程序之前先输入了这个命令，那么 shell会在命令脚本程序执行的时候把所有的命令和参数输出显示到屏幕上。如果想关闭这个功能，可以输入“ set +o xtrace ”或者“ set +x ”命令。

1. 回显 —— 向屏幕输出文字

当命令脚本程序执行的时候，通过 echo命令加上显示其相关信息的语句可以让你跟踪掌握变量值发生的变化。举例来说，如果你怀疑处理变量 count的时候可能出现了问题，就可以在命令脚本程序第一步完成的时候加上如下所示的语句：

```
echo " The value of the variable count after step one is $count . "
```

因为使用了美元符号（ $），所以命令脚本程序会把 $count正确地处理为这个变量的值。如果变量count的值是14，那么在遇到这个echo命令的时候，命令脚本程序将会显示：

```
The value of the variable count after step one is 14 .
```

2. “ Unexpected EOF ”出错信息

如果在试图运行一个命令脚本程序的时候看到了“ Unexpected End Of File (EOF) ”出错信息（意思是错误地遇到文件尾标记），那么极有可能是因为你没有正确地关闭所有的 if和elif语句。

3. 给出一个退出状态

shell会记录一个命令脚本程序是否正确地退出了执行，也就是可以报告它退出的时候是没有出现错误、出现隐含错误、还是出现了明显的错误。如果想让命令脚本程序在它退出的时候显示其程序出口状态值，需要加上下面的语句：

```
echo $?
```

如果命令脚本程序有很多输出的话，可以使用下面效果更为明显的语句：

```
echo " The exit status is $? "
```

如果这个程序出口状态值是 0的话，这个命令脚本程序就是正常结束退出的。如果这个值是1，就说明出现了错误。

4. 使用注释标记进行程序调试

如果在命令脚本程序中不太容易找出其中的 bug，有一个很有用的方法可供使用参考，那就是使用注释标记把某些代码段转换为注释语句，使它们不再被执行。然后，你就可以检查出在没有了那些代码段之后，原来的程序错误是否还继续出现。

这个方法的一个不足之处是：当你把某些代码段转换为注释语句之后，程序代码可能会前进到另外一个方向去。请一定要仔细检查，看看在你准备忽略的代码段中是不是有对命令脚本程序中的其他代码产生关键影响的操作。