第九章　登录环境

登录系统后，在进入命令提示符前，系统要做两个工作。键入用户名和密码后，系统检查是否为有效用户，为此需查询/etc/passwd文件；如果登录名正确并且密码有效，开始下一步过程，即登录环境。

账户文件/etc/passwd是一个文本文件，可以任意修改其中的文本域，但要小心。此文本有7个域，并用冒号作分隔符。

[1][2][3][4][5][6][7]

kvp:JFqMmk9.uRioA:405:413:K.V.Pally:/home/sysdev/kvp:/bin/sh

dhw:hi/G4U1CUd9aI,B/0J:407:401:D.Whitely:/home/dept47/dhw:/bin/sh

aec:ILgHtxJ9kXtSc,B/GI:408:401:A.E.Cloody:/b_user/dept47/aec:/bin/sh

gdw:iLFu9BB8RNjpc,B/MK:409:401:G.D.Wilcom:/b_user/dept47/gdw:/bin/sh

第1域是登录名，第2域是加密的密码，第5域是用户全名。第6域是用户家目录，第7域是用户使用的shell。这里/bin/sh意即缺省为常规Bourne Shell。passwd文件可能还有其他格式。

登录成功后，系统执行两个环境设置文件，第一个是/etc/profile，第二个是.profile，位于用户家目录下。系统还会处理其他的初始化文件。这里只涉及profile文件。

9.1 /etc/profile

用户登录时，自动读取/etc目录下.profile文件，此文件包含：

- 全局或局部环境变量。
- PATH信息。
- 终端设置。
- 安全命令。
- 日期信息或放弃操作信息。

下面就来详细解释上述各项内容。设置全局环境变量便于用户及其进程和应用访问它。

PATH定位可执行文件、库文件及一般文本文件的目录位置，便于用户快速访问。

终端设置使系统获知用户终端的一般特性。

安全命令包括文件创建模式或敏感区域的双登录提示。

日期信息是一个文本文件，保存用户登录时即将发生事件的记录或放弃登录的信息文件。

9.2 用户的$HOME.profile

/etc/profile文件执行时，用户将被放入到自己的$HOME目录中，回过头来观察passwd文件，用户的$HOME目录在倒数第2列。可以将之看作用户根目录，因为正是在这里存储了所有的私有信息。

回到.profile，一般来说创建帐户时，一个profile文件的基本框架即随之创建。不要忘了在 .profile文件中可以通过设置相关条目以不同的值或使用 set 命令来覆盖/etc/profile文件中的设置。如果愿意，可以定制用户自己的.profile文件。先来看看标准的.profile文件。

```
$ pg .profile
#.profile
set -a
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH
export PATH
#
```

现在加入两个环境变量，如EDITOR，以使cron或其他应用获知正在使用的编辑器；将TERM变量设置为vt100。

也可以创建bin目录，将之加入路径（PATH），目录结构中加一个bin目录是一个好习惯。

在这里可以保存所有脚本，将之加入PATH后，就不必写入脚本的文件路径名全称，只键入脚本名即可。

如果想知道已登录系统的用户数，可使用命令who和wc。

```
$ echo "`who|wc -l` users are on today"

19 users are on today.
```

将上述设置加入.profile文件。如果要使 .profile或/etc/.profile文件改动生效，可以退出系统然后再登入。

以下为改动过的.profile文件

```
$ pg .profile
#.profile
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH:$HOME:bin
#
EDITOR=vi
TERM vt100
ADMIN=/usr/adm
PS1="`hostname`>"
PS2="`echo "\0251"`:"
export EDITOR TERM ADMIN PATH PS1
echo "`who|wc -l` users are on to-day"
```

9.3 创建 .logout 文件

使用Bourne shell与其他shell不同，其缺点是不包含.logout文件。此文件保存有执行exit命令时，在进程终止前执行的命令。

但是通过使用trap命令（trap和信号将在后面讨论），Bourne shell也可以创建自己的.logout文件。方法如下：编辑.profile文件，在最后一行加入下列命令，然后保存并退出。

```
trap "$HOME/.logout" 0
```

再键入一个.logout文件，敲入下列执行命令。
如果愿意，可以在此脚本中加入任何命令。

```
$ pg .logout
rm -f $HOME/*.log
rm -f $HOME/*.tmp
echo "Bye...bye $LOGNAME"
```

用户退出时，调用.logout文件。过程如下：用户退出一个shell时，传送了一个信号0，意即从现在shell中退出，在控制返回shell继续退出命令前， .profile文件中trap行将捕获此信号并执行 .logout。

第十章　环境和shell变量

为使shell编程更有效，系统提供了一些shell变量。shell变量可以保存诸如路径名、文件名或者一个数字这样的变量名。shell将其中任何设置都看做文本字符串。

有两种变量，本地和环境。严格地说可以有4种，但其余两种是只读的，可以认为是特殊变量，它用于向shell脚本传递参数。

本章内容有：

- shell变量。
- 环境变量。
- 变量替换。
- 导出变量。
- 特定变量。
- 向脚本传递信息。
- 在系统命令行下使用位置参数。

10.1 什么是shell变量

变量可以定制用户本身的工作环境。使用变量可以保存有用信息，使系统获知用户相关设置。变量也用于保存暂时信息。例如：一变量为EDITOR，系统中有许多编辑工具，但哪一个适用于系统呢？将此编辑器名称赋给变量EDITOR，这样，在使用cron或其他需要编辑器的应用时，会缺省使用该编辑器。

10.2 本地变量

本地变量在用户现在的shell生命期的脚本中使用。例如，本地变量var取值为123，这个值只在用户当前shell生命期有意义。如果在shell中启动另一个进程或退出，此值将无效。这个方法的优点就是用户不能对其他的shell或进程设置此变量有效。

10.2.1 显示变量

使用echo命令可以显示变量值，需在变量名前加$，例如：

```
$ GREAT_PICTURE="die hard"
$ echo ${GREAT_PICTURE}
die hard

$ DOLLAR=99
$ echo ${DOLLAR}
99

$ LAST_FILE=ZLPSO.txt
$ echo ${LAST_FILE}
ZLPSO.txt
```

可以结合使用变量，下面将错误信息和环境变量LOGNAME设置到变量ERROR_MSG。

```
$ ERROR_MSG=" Sorry this file does not exist user $LOGNAME"
$ echo ${ERROR_MSG}
Sorry this file does not exist user dave
```

上例中，shell首先显示文本，然后查找变量$LOGNAME，最后扩展变量以显示变量值。

10.2.2 清除变量值

使用unset命令清除变量值。

unset variable-name

```
$ PC=enterprise
$ echo ${PC}
enterprise
$ unset PC
$ echo ${PC}
$
```

10.2.3 显示所有本地shell变量

使用set命令显示所有本地定义的shell变量。

```
$ set
...
PWD=/root
SHELL=/bin/sh
SHLVL=1
TERM=vt100
UID=7
USER=dave
dollar=99
great_picture=die hard
last_file=ZLPSO.txt
```

set输出可能很长。

查看输出时可以看出shell已经设置了一些用户变量以使工作环境更加容易使用。

10.2.4 合并输出变量值

将变量并排可以使其结合在一起：

```
echo ${variable_name}${variable_name}...
```

```
$ FIRST="Bruce "
$ SURNAME=Willis
$ echo ${FIRST}${SURNAME}
Bruce Willis
```

10.2.5 测试变量是否已经设置

有时要测试是否已设置或初始化变量。如果未设置或初始化，就可以使用另一值。此命令格式为：

${ variable : -value }

如果设置了变量值，则使用它，如果未设置，则用新值显示。例如：

```
$ COLOUR=blue
$ echo "The sky is ${COLOUR:-grey} today"
The sky is blue today
```

```
$ COLOUR=blue
$ unset COLOUR
$ echo "The sky is ${COLOUR:-grey} today"
The sky is grey today
```

上面的例子并没有将实际值传给变量，需使用下述命令完成此功能：

${variable:=value}

10.2.6 使用变量来保存系统命令参数

可以用变量保存系统命令参数的替换信息。下面的例子使用变量保存拷贝命令中的文件名信息。变量source保存passwd文件的路径，dest保存目标文件的信息。

```
$ SOURCE="/etc/passwd"
$ DEST="/tmp/passwd.bak"
$ cp ${SOURCE} ${DEST}
```

10.2.7 设置只读变量

设置变量后，不想再改变其值，可以将之设置为只读方式。如果有人(包括用户本人)想要改变它，则返回错误信息。格式如下：

variable-name = value

readonly variable-name

下面的例子中，设置变量为系统磁带设备之一的设备路径，将之设为只读，任何改变其值的操作将返回错误信息。

```
$ TAPE_DEV="/dev/rmt/0n"
$ echo ${TAPE_DEV}
/dev/rmt/0n
$ readonly TAPE_DEV
$ TAPE_DEV="/dev/rmt/1n"
sh: TAPE_DEV: read-only variable
```

要查看所有只读变量，使用命令 readonly即可

```
$ readonly
declare -r FILM="Crimson Tide"
declare -ri PPID="1"
declare -r TAPE_DEV="/dev/rmt/0n"
declare -ri UID="0"
```

10.3 环境变量

环境变量用于所有用户进程（经常称为子进程）。登录进程称为父进程。shell中执行的用户进程均称为子进程。不像本地变量（只用于现在的shell）环境变量可用于所有子进程，这包括编辑器、脚本和应用。

环境变量可以在命令行中设置，但用户注销时这些值将丢失，因此最好在.profile文件中定义。系统管理员可能在/etc/profile文件中已经设置了一些环境变量。将之放入.profile文件意味着每次登录时这些值都将被初始化。

传统上，所有环境变量均为大写。环境变量应用于用户进程前，必须用export命令导出。环境变量与本地变量设置方式相同。

10.3.1 设置环境变量

VARIABLE_NAME = value； export VARIABLE_NAME

两个命令之间是一个分号，也可以这样写：

VARIABLE_NAME = value

export VARIABLE_NAME

10.3.2 显示环境变量

显示环境变量与显示本地变量一样，例：

```
$ CONSOLE=tty1; export CONSOLE
$ echo $CONSOLE
tty1

$ MYAPPS=/usr/local/application; export MYAPPS
$ echo $MYAPPS
/usr/local/application
```

使用env命令可以查看所有的环境变量

```
$ env
HISTSIZE=1000
HOSTNAME=localhost.localdomain
LOGNAME=dave
MAIL=/var/spool/mail/root
TERM=vt100
HOSTTYPE=i386
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin:
CONSOLE=tty1
HOME=/home/dave
ASD=sdf
SHELL=/bin/sh
PS1=$
USER=dave
...
```

10.3.3 清除环境变量

使用unset命令清除环境变量：

```
$ unset MYAPPS
$ echo $MYAPPS

$
```

10.3.4 嵌入shell变量

Brourne shell有一些预留的环境变量名，这些变量名不能用作其他用途。通常在/etc/profile中建立这些嵌入的环境变量，但也不完全是，这取决于用户自己。以下是嵌入shell变量列表。

1. HOME

   HOME目录，通常定位于passwd文件的倒数第2列，用于保存用户自身文件。

2. IFS：IFS用作shell指定的缺省域分隔符。

3. LOGNAME

   此变量保存登录名，应该为缺省设置，

4. PATH

   PATH变量保存进行命令或脚本查找的目录顺序。

5. SHELL：SHELL变量保存缺省shell。

6. PS1

shell基本提示符，缺省时超级用户为#，其他为$。可以使用任何符号作提示符，以下为两个例子：

```
$ PS1="star trek:"; export PS1
star trek:
$ PS1="->" ; export PS1
->
```

7. PS2

PS2为附属提示符，缺省为符号>。PS2用于执行多行命令或超过一行的一个命令。

```
$ PS2="@:"; export PS2
$ for loop in *
@:do
@:echo $loop
...
```

10.3.5 set命令

在$HOME . profile文件中设置环境变量时，还有另一种方法导出这些变量。使用set命令- a选项，即set -a指明所有变量直接被导出。不要在/etc/profile中使用这种方法，最好只在自己的$HOME .profile文件中使用。

```
$ pg .profile
#.profile
set -a
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH:$HOME:bin
#
EDITOR=vi
TERM vt220
ADMIN=/usr/adm
PS1="`hostname`>"
```

10.3.6 将变量导出到子进程

shell新用户碰到的问题之一是定义的变量如何导出到子进程。前面已经讨论过环境变量的工作方式，现在用脚本实现它，并在脚本中调用另一脚本（这实际上创建了一个子进程）。

以下是两个脚本列表father和child。

father脚本设置变量film，取值为A Few Good Men，并将变量信息返回屏幕，然后调用脚本child，这段脚本显示第一个脚本里的变量film，然后改变其值为Die Hard，再将其显示在屏幕上，最后控制返回father脚本，再次显示这个变量。

```
$ pg father
#!/bin/sh
# father script.
echo "this is the father"
FILM="A Few Good Men"
echo "I like the film :$FILM"
# call the child script
child
echo "back to father"
echo "and the film is :$FILM"

$ pg child
#!/bin/sh
# child
echo "called from father..i am the child"
echo "film name is :$FILM"
FILM="Die Hard"
echo "changing film to :$FILM"
```

脚本显示结果:

```
$ father
this is the father
I like the film :A Few Good Men
called from father..i am the child
film name is :
changing film to :Die Hard
back to father
and the film is :A Few Good Men
```

因为在**father**中并未导出变量**film**，因此**child**脚本不能将**film**变量的值显示。

在father脚本中加入export命令后，child脚本中film变量的值可显示。

```
pg father
#!/bin/sh
# father script.
echo "this is the father"
FILM="A Few Good Men"
echo "I like the film :$FILM"
# call the child script
# but export variable first
export FILM
child
echo "back to father"
echo "and the film is :$FILM"
```

```
$ father2
this is the father
I like the film :A Few Good Men
called from father..i am the child
film name is  :A Few Good Men
changing film to :Die Hard
back to father
and the film is :A Few Good Men
```

因为在脚本中加入了**export**命令，因此可以在任意多的脚本中使用变量**film**，它们均继承了**film**的所有权。不可以将变量从子进程导出到父进程，可尝试通过重定向来做到这一点。

10.4 位置变量参数

本章开始提到有4种变量，本地、环境，还有两种变量被认为是特殊变量，因为它们是只读的。这两种变量即为位置变量和特定变量参数。先来看一看位置变量。

如果要向一个shell脚本传递信息，可以使用位置参数完成此功能。参数相关数目传入脚本，此数目可以任意多，但只有前9个可以被访问，使用shift命令可以改变这个限制。参数从第一个开始，在第9个结束；每个访问参数前要加$符号。第一个参数为0，系统预留保存实际脚本名字。无论脚本是否有参数，此值均存在。

如果向脚本传送Did You See The Full Moon信息，下面讲解了如何访问每一个参数。

$0	$1	$2	$3	$4	$5	$6	$7	$8	$9
脚本名字	Did	You	See	The	Full	Moon			

10.4.1 在脚本中使用位置参数

在下面脚本中使用上面的参数。

```
$ pg param
#!/bin/sh
# param
echo "This is the script name          : $0"
echo "This is the first parameter      : $1"
echo "This is the second parameter     : $2"
echo "This is the third parameter      : $3"
echo "This is the fourth parameter     : $4"
echo "This is the fifth parameter      : $5"
echo "This is the sixth parameter      : $6"
echo "This is the seventh parameter    : $7"
echo "This is the eighth parameter     : $8"
echo "This is the ninth parameter      : $9"
```

```
$ param Did You See The Full Moon
This is the script name          : ./param
This is the first parameter      : Did
This is the second parameter     : You
This is the third parameter      : See
This is the fourth parameter     : The
This is the fifth parameter      : Full
This is the sixth parameter      : Moon
This is the seventh parameter    :
This is the eighth parameter     :
This is the ninth parameter      :
```

这里只传递6个参数，7、8、9参数为空，

10.4.2 向系统命令传递参数

可以在脚本中向系统命令传递参数。下例中，使用参数$1指定要查找的文件名。

```
$  pg findfile
#!/bin/sh
# findfile
find / -name $1 -print

$ findfile passwd
/etc/passwd
/etc/uucp/passwd
/usr/bin/passwd
```

下例以**$1**向**grep**传递一个用户**id**号，**grep**使用此**id**号在**passwd**中查找用户全名。

```
$ pg who_is
#!/bin/sh
# who_is
grep $1 passwd | awk -F: {print $4}'

$ who_is seany
Seany Post
```

10.4.3 特定变量参数

脚本运行时通过特定变量来反映一些相关控制信息。SHELL共有7个特定变量，见下表

$# 传递到脚本的参数个数

$* 以一个单字符串显示所有向脚本传递的参数。与位置变量不同，此选项参数可超过9个

$$ 脚本运行的当前进程ID号

$! 后台运行的最后一个进程的进程ID号

$@ 与$#相同，但是使用时加引号，并在引号中返回每个参数

$- 显示shell使用的当前选项，与set命令功能相同

$? 显示最后命令的退出状态。0表示没有错误，其他任何值表示有错误。

在脚本param中加入各种特定变量并重新运行。

```
$ pg param
#!/bin/sh
# allparams
echo "This is the script name                : $0"
echo "This is the first parameter            : $1"
echo "This is the second parameter           : $2"
echo "This is the third parameter            : $3"
echo "This is the fourth parameter           : $4"
echo "This is the fifth parameter            : $5"
echo "This is the sixth parameter            : $6"
echo "This is the seventh parameter          : $7"
echo "This is the eighth parameter           : $8"
echo "This is the ninth parameter            : $9"
echo "The number of arguments passed         :$#"
echo "Show all arguments                     : $*"
echo "Show me my process ID                  : $$"
echo "Show me the arguments in quotes        :" "$@"
echo "Did my script go with any errors       :$?"
```

```
$ param Merry Christmas Mr Lawrence
This is the script name                 :./param
This is the first parameter             : Merry
This is the second parameter            : Christmas
This is the third parameter             : Mr_Lawrence
This is the fourth parameter            :
This is the fifth parameter             :
This is the sixth parameter             :
This is the seventh parameter           :
This is the eighth parameter            :
This is the ninth parameter             :
The number of arguments passed          :3
Show all arguments                      : Merry Christmas Mr_Lawrence
Show me my process ID                   : 630
Show me the arguments in quotes         : "Merry" "Christmas" "Mr_Lawrence "
Did my script go with any errors        :0
```

特定变量的输出使用户获知更多的脚本相关信息。可以检查传递了多少参数，进程相应的ID号等，以免我们想杀掉此进程。

10.4.4 最后的退出状态

使用**$?**。可以在任何命令或脚本执行后获得退出时的状态信息，依此可以做更进一步的研究。返回**0**意味着成功，**1**为出现错误。

下例拷贝文件到**/tmp**，并使用**$?**检查结果。

```
$ cp ok.txt /tmp
$ echo $?
0
```

现在尝试将一个文件拷入一个不存在的目录：

```
$ cp ok.txt /usr/local/apps/dsf
cp: cannot create regular file '/usr/local/apps/dsf': No such file or
directory


$ echo $?
1
```

使用**$?**检验返回状态，可知脚本有错误，但同时发现cp：cannot...，因此检验最后退出状态已没有必要。在脚本中可以用系统命令处理输出格式，要求命令输出不显示在屏幕上。为此可以将输出重定向到/dev/null中。这时可以用最后退出状态命令知道脚本正确与否。

```
$ cp ok.txt /usr/local/apps/dsf >/dev/null 2>&1
$ echo $?
1
```

通过将包含错误信息的输出重定向到null中，不能获知最后命令返回状态，但是通过使用$?，（其返回值为1）可知拷贝失败。

检验脚本退出状态时，最好将返回值设置为一个有意义的名字，这样可以增加脚本的可读性。

```
$ cp ok.txt /usr/local/apps/dsf >/dev/null 2>&1
$ cp_status=$?
$ echo $cp_status
1
```

10.5 小结

变量可以使shell编程更容易。它能够保存输入值并提高效率。shell变量几乎可以包含任何值。特定变量增强了脚本的功能并提供了传递到脚本的参数的更多信息。

第十一章　引号的使用

上一章介绍了变量和替换操作，在脚本中执行变量替换时最容易犯的一个错误就是引用错误。在命令行中引用是很重要的。

本章内容有：

- 引用的必要性。
- 双引、单引和反引号。
- 使用反斜线实现屏蔽。

11.1 引用必要性

SHELL命令会使用一个经选择的元字符集。所谓元字符是拥有特殊解释的字符。如管道(|)除了被用于与SHELL通信，还被广泛地用作普通文本。因此，无论将它们用作特殊字符，还是普通文本，都需要用某种方法(即使用引号，包括各式引用或反斜线)告知SHELL解释程序。一些用户在对文本字符串进行输出时觉得使用引用很麻烦。有时不注意，只引用了一半，这时问题出现了。最好在文本字符串输出时使用双引号。

```
$ echo Hit the star button to exit *

Hit the star button to exit DIR_COLORS HOSTNAME Muttrc X11 adjtime
aliases alias
...
```

文本返回了，但由于未使用双引号，*被shell误解，shell认为用户要做目录列表。用双引号后结果如下：

```
$ echo "Hit the star button to exit *"
Hit the star button to exit *
```

SHELL引用类型

“ ”双引号　｀反引号　‘ ’单引号　\反斜线

11.2 双引号

使用双引号可引用除字符$、`、\外的任意字符或字符串。这些特殊字符分别为美元符号，反引号和反斜线，对**shell**来说，它们有特殊意义。如果使用双引号将字符串赋给变量并反馈它，实际上与直接反馈变量并无差别。

```
$ STRING="MAY DAY, MAY DAY, GOING DOWN"
$ echo "$STRING"
MAY DAY, MAY DAY, GOING DOWN

$ echo $STRING
MAY DAY, MAY DAY, GOING DOWN
```

如果要查询包含空格的字符串，经常会用到双引号。以下使用grep抽取名字“Davey Wire”，因为没有加双引号， grep将“Davey”认作字符串，而把“Wire”当作文件名。

```
$ grep   Davey  Wire   /etc/passwd
grep: wire: No  such  file  or  directory
```

要解决这个问题，可将字符串加双引号。这样shell会忽略空格，当使用字符时，应总是使用双引号，无论它是单个字符串或是多个单词。

```
$ grep "Davey Wire" /etc/passwd
davyboy:9sdJUK2s:106:Davey Wire:/home/ap
```

在一个反馈命令里可以使用双引号将变量引起来。下面的例子中，shell反馈文本行，遇到符号$，知道这是一个变量，然后用变量值boy替换变量$BOY。

```
$ BOY="boy"
$ echo " The $BOY did well"
 The boy did well

$ echo " The "$BOY" did well"
 The boy did well
```

11.3 单引号

由单引号括起来的字符都作为普通字符出现。

```
$ echo ′ The time is `date`,the file is $HOME/abc′
  The time is `date`,the file is $HOME/abc
```

但双引号能屏蔽单引号的特殊含义。

```
$ GIRL='girl'
$ echo "The '$GIRL' did well"
The 'girl' did well
```

11.4 反引号

反引号用于设置系统命令的输出到变量。shell将反引号中的内容作为一个系统命令执行。反引号可以与双引号结合使用。

```
$ echo `hello`
sh: hello: command not found
```

现在用date命令再试一次。

```
$ echo `date`
Sun May 16 16:40:19 GMT 1999
```

这次命令有效，shell正确执行。

```
$ echo "The date today is `date`"
The date today is Sun May 16 16:56:53 GMT 1999
```

打印当前系统上用户数目:

```
$ echo "There are `who | wc -l` users on the system"
There are  13   users on the system
```

11.5 反斜线

如果下一个字符有特殊含义，反斜线可以屏蔽其特殊含义。下述字符包含有特殊意义：

& * + ^ $ ` " | ?

假定**echo**命令加*，意即顺序打印当前整个目录列表，而不是显示一个星号*。

```
$ echo *

conf.linuxconf conf.modules cron.daily cron.hourly cron.monthly
cron.weekly crontab csh.cshrc default dosemu.conf dosemu.users exports
fdprm fstab gettydefs gpm-root.c
onf group group- host.conf hosts hosts.allow hosts.deny httpd inetd
...
```

为屏蔽星号特定含义，可使用反斜线。

```
$ echo \*
*
```

上述语句同样可用于**$$**命令， **shell**解释其为现在进程**ID**号，使用反斜线屏蔽此意，仅打印**$$**。

```
$ echo $$
284
$ echo \$$
$$
```

使用命令expr时，用*表示乘法会出现错误，在*前加上反斜线才会正确。

```
$ expr 12 * 12
expr: syntax error


$ expr 12 \* 12
144
```

在echo命令中输出元字符时，须用反斜线\屏蔽其特殊含义。下例中要显示价格$19.99。其中$屏蔽与否将产生不同的结果。

```
$ echo "That video looks a good price for $19.99"
That video looks a good price for 9.99
```

使用反斜线屏蔽$，可得到正确的结果：

```
$ echo "That video looks a good price for \$19.99"
That video looks a good price for $19.99
```

11.6 小结

引用时遵循的两条规则：

1) 反馈字符串用双引号；但不要引用反馈(echo)本身。

2) 如果使用引用得到的结果不理想，再试另一种，毕竟只有三种引用方式，可以充分尝试。