第6章 Shell编程

本章内容

6.1 熟悉Shell程序的创建

6.2 Shell变量

6.3 变量表达式

6.4 Shell条件判断语句

6.5 Shell循环控制语句

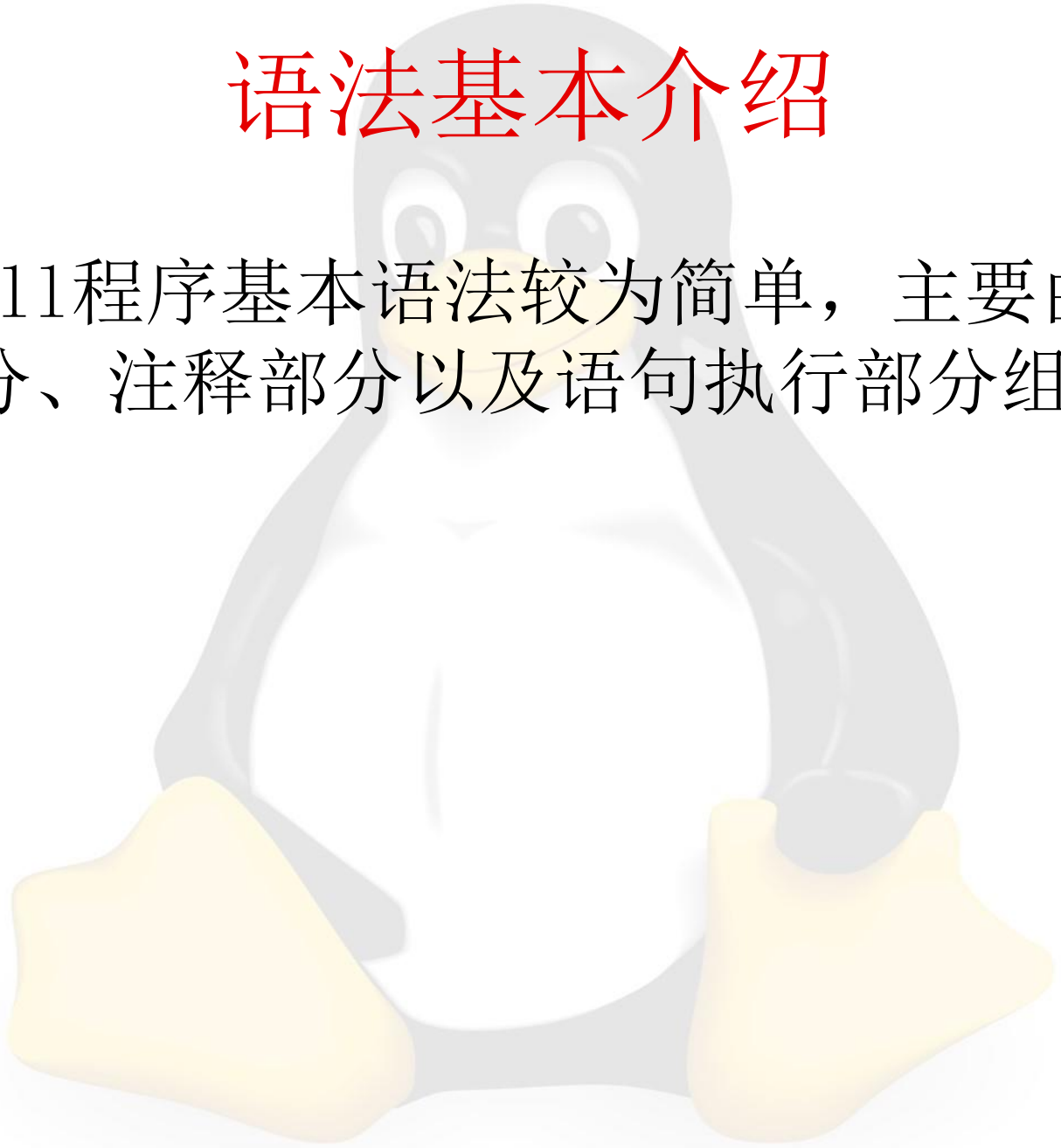
- 通常情况下，从命令行每输入一次命令就能够得到系统响应，如果需要一个接着一个地输入命令才得到结果的时候，这样的做法效率很低。使用Shell程序或者Shell脚本可以很好地解决这个问题。

6.1 熟悉Shell程序的创建

- 作为命令语言互动式地解释和执行用户输入的命令是Shell的功能之一，Shell还可以用来进行程序设计，它提供了定义变量和参数的手段以及丰富的过程控制结构。使用Shell编程类似于使用DOS中的批处理文件，称为Shell脚本，又叫做Shell程序或Shell命令文件。

语法基本介绍

- Shell程序基本语法较为简单，主要由开头部分、注释部分以及语句执行部分组成。



1. 开头

- Shell程序必须以下面的行开始（必须放在文件的第一行）。

  #!/bin/bash

- 符号“#!”用来告诉系统它后面的参数是用来执行该文件的程序，在这个例子中使用/bin/bash来执行程序。当编辑好脚本时，如果要执行该脚本，还必须使其可执行。
- 要使脚本可执行，需赋予该文件可执行的权限，使用如下命令文件才能运行。

  chmod u+x [文件名]

2. 注释

- 在进行Shell编程时，以“#”开头的句子表示注释，直到这一行的结束，建议在程序中使用注释。如果使用注释，那么即使相当长的时间内没有使用该脚本，也能在很短的时间内明白该脚本的作用及工作原理。

3. 执行命令

- 在Shell程序中可以输入多行命令以得到命令的结果信息，这样就提高系统管理的工作效率。

Shell程序的创建过程

- Shell程序就是放在一个文件中的一系列Linux命令和实用程序，在执行的时候，通过Linux系统一个接着一个地解释和执行每个命令，这和Windows系统下的批处理程序非常相似。

1. 创建文件

- 在/root目录下使用vi编辑器创建文件date，该文件内容如下所示，共有3个命令。

```
#!/bin/bash
#filename:date
echo “Mr.$USER,Today is:”
echo `date`//此处为反引号
echo Whish you a lucky day !
```

2．设置可执行权限

- 创建完date文件之后它还不能执行，需要给它设置可执行权限，使用如下命令给文件设置权限。

[root@rhel ~]# chmod u+x /root/date

[root@rhel ~]# ls -l /root/date

-rwxr--r--. 1 root root 88 6月　3 05:37 /root/date

//可以看到当前date文件具有可执行权限

3．执行Shell程序

- 输入整个文件的完整路径执行Shell程序，使用如下命令执行。

```
[root@rhel ~]# /root/date
Mr.root,Today is:
2012年 06月 03日 星期日 05:37:34 CST
Whish you a lucky day !
//可以看到Shell程序的输出信息
```

4．使用bash命令执行程序

- 在执行Shell程序时需要将date文件设置为可执行权限。如果不设置文件的可执行权限，那么需要使用bash命令告诉系统它是一个可执行的脚本，使用如下命令执行Shell程序。

```
[root@rhel ~]#  bash /root/date
Mr.root,Today is:
2012年 06月 03日 星期日 05:37:34 CST
Whish you a lucky day !
```

6.2 Shell变量

- 像高级程序设计语言一样，Shell也提供说明和使用变量的功能。对Shell来讲，所有变量的取值都是一个字符，Shell程序采用“$var”的形式来引用名为var的变量的值。

Shell定义的环境变量

- Shell在开始执行时就已经定义了一些与系统的工作环境有关的变量，用户还可以重新定义这些变量。

常用的Shell环境变量

Shell环境变量	描述
HOME	用于保存用户主目录的完全路径名
PATH	用于保存用冒号分隔的目录路径名，Shell将按PATH变量中给出的顺序搜索这些目录，找到的第一个与命令名称一致的可执行文件将被执行
TERM	终端的类型
UID	当前用户的UID，由数字构成
PWD	当前工作目录的绝对路径名，该变量的取值随cd命令的使用而变化
PS1	主提示符，在root用户下，默认的主提示符是“#”，在普通用户下，默认的主提示符是“$”
PS2	在Shell接收用户输入命令的过程中，如果用户在输入行的末尾输入“\”然后按回车键，或者当用户按回车键时Shell判断出用户输入的命令没有结束时，就显示这个辅助提示符，提示用户继续输入命令的其余部分，默认的辅助提示符是“>”

【例6.1】 查看当前用户Shell定义的环境变量的值。

```
[root@rhel ~]# echo $HOME
/root
[root@rhel ~]# echo $PWD
/root
[root@rhel ~]# echo $PS1
[\u@\h \W]\$
[root@rhel ~]# echo $PS2
>
[root@rhel ~]# echo $PATH
/usr/lib/ccache:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@rhel ~]# echo $TERM
vt100
[root@rhel ~]# echo $UID
0
```

用户定义的变量

- 用户可以按照下面的语法规则定义自己的变量。
  变量名=变量值
- 在定义变量时，变量名前不应该加符号“$”，在引用变量的内容时则应在变量名前加符号“$”。
- 在给变量赋值时，等号两边一定不能留空格，若变量中本身就包含了空格，则整个字符串都要用双引号括起来。在编写Shell程序时，为了使变量名和命令名相区别，建议所有的变量名都用大写字母来表示。

【例6.2】用户定义变量的使用。

[root@rhel ~]# $AS=120

bash: =120: 未找到命令... //在定义变量名时，变量名前加符号“$”就报错

[root@rhel ~]# AS=120

[root@rhel ~]# echo $AS //在引用变量名时，在变量名前加符号“$”

120

[root@rhel ~]# AA="hello word" //变量值中包含了空格，需将整个字符串用双引号括起来

[root@rhel ~]# echo $AA

hello word

- 有时需要在说明一个变量并对它设置为一个特定值后就不再改变它的值时，可以用下面的命令来保证一个变量的只读性。

  readonly 变量名

- 在任何时候创建的变量都只是当前Shell的局部变量，所以不能被Shell运行的其他命令或Shell程序所利用，而export命令可以将一个局部变量提供给Shell命令使用，其格式是：

  export 变量名
- 也可以在给变量赋值的同时使用export命令：

  export 变量名=变量值
- 使用export说明的变量在Shell以后运行的所有命令或程序中都可以访问到。

位置参数

- 位置参数是一种在调用Shell程序的命令行中按照各自的位置决定的变量，是在程序名之后输入的参数。位置参数之间用空格分隔，Shell取第一个位置参数替换程序文件中的$1，第二个替换$2，依次类推。$0是一个特殊的变量，它的内容是当前这个Shell程序的文件名，所以$0不是一个位置参数，在显示当前所有的位置参数时是不包括$0的。

预定义变量

- 预定义变量和环境变量相类似，也是在Shell一开始时就定义了的变量。所不同的是，用户只能根据Shell的定义来使用这些变量，所有预定义变量都是由符号“$”和另一个符号组成的。

常用的Shell预定义变量

预定义变量	描述
$#	位置参数的数量
$*	所有位置参数的内容
$?	命令执行后返回的状态，0表示没有错误，非0表示有错误
$$	当前进程的进程号
$!	后台运行的最后一个进程号
$0	当前执行的进程名

参数置换的变量

- Shell提供了参数置换功能以便用户可以根据不同的条件来给变量赋不同的值。参数置换的变量有四种，这些变量通常与某一个位置参数相联系，根据指定的位置参数是否已经设置决定变量的取值。

1. 变量=${参数-word}

- 如果设置了参数，则用参数的值置换变量的值，否则用word置换，即这种变量的值等于某一个参数的值。如果该参数没有设置，则变量就等于word值。

2. 变量=${参数=word}

- 如果设置了参数，则用参数的值置换变量的值，否则把变量设置成word，然后再用word替换参数的值。位置参数不能用于这种方式，因为在shell程序中不能为位置参数赋值。

3. 变量=${参数?word}

- 如果设置了参数，则用参数的值置换变量的值，否则就显示word并从shell中退出，如果省略了word，则显示标准信息。这种变量要求一定等于某一个参数的值。如果该参数没有设置，就显示一个信息，然后退出，这种方式常用于出错指示。

4. 变量=${参数+word}

- 如果设置了参数，则用word置换变量，否则不进行置换。

```
[root@rhel ~]# var="hello"
[root@rhel ~]# echo $var ${title-"somebody"}!
hello somebody!
[root@rhel ~]# echo $var ${title+"somebody"}!
hello !
[root@rhel ~]# echo $var ${title?"title is null or empty"}!
bash: title: title is null or empty
[root@rhel ~]# echo $var ${title="tom and jerry"}!
hello tom and jerry!
[root@rhel ~]# echo $var ${title+"somebbody"}!
hello somebbody!
```

6.3 变量表达式

- test是Shell程序中的一个表达式，通过和Shell提供的if等条件语句相结合可以方便地测试字符串、文件状态和数字。其语法如下所示。

  test [表达式]

- 表达式所代表的操作符有字符串操作符、数字操作符、逻辑操作符以及文件操作符。其中文件操作符是一种Shell特有的操作符，因为Shell里的变量都是字符串，为了达到对文件进行操作的目的，于是才提供了这样的一种操作符。

字符串比较

- 字符窜比较是用来测试字符串是否相同、长度是否为0、字符串是否为null。
- 常用的字符串比较符号：

字符串比较符号	描述
=	比较两个字符串是否相同，相同则为“是”
!=	比较两个字符串是否相同，不同则为“是”
-n	比较字符串的长度是否大于0，如果大于0则为“是”
-z	比较字符串的长度是否等于0，如果等于0则为“是”

【例6.6】 字符串比较的使用。

```
[root@rhel ~]# str1=abcd
[root@rhel ~]# test $str1 = abcd
[root@rhel ~]# echo $?
0
//结果显示0表示比较字符串str1确实等于abcd
```

数字比较

- 数字比较是用来测试数字的大小。
- 常用的数字比较符号：

数字比较符号	描述
-eq	相等
-ge	大于等于
-le	小于等于
-ne	不等于
-gt	大于
-lt	小于

【例6.7】 数字相等比较。

```
[root@rhel ~]# int1=1234
[root@rhel ~]# int2=01234
[root@rhel ~]# test $int1 -eq $int2
[root@rhel ~]# echo $?
0
//结果显示0表示字符int1和int2比较，二者值一样大
```

【例6.8】 数字大于比较。

```
 [root@rhel ~]# int1=4
[root@rhel ~]# test $int1 -gt 2
[root@rhel ~]# echo $?
0
//结果显示0表示字符int1的值确实大于2
```

逻辑测试

- 逻辑测试是用来测试文件是否存在。
- 常用的逻辑测试符号：

逻辑测试符号	描述
!	与一个逻辑值相反的逻辑值
-a	两个逻辑值为“是”返回值才为“是”，反之为“否”
-o	两个逻辑值有一个为“是”，返回值就为“是”

【例6.9】 逻辑测试。

```
[root@rhel ~]# test -r empty -a -s empty
[root@rhel ~]# echo $?
1
//结果显示1表示文件empty存在且可读以及长度为0
```

文件操作测试

- 文件操作测试表达式通常是为了测试文件的文件操作逻辑。
- 常用的文件操作测试符号：

文件操作测试符号	描述
-d	对象存在且为目录则返回值为“是”
-f	对象存在且为文件则返回值为“是”
-L	对象存在且为符号链接则返回值为“是”
-r	对象存在且可读，则返回值为“是”
-s	对象存在且长度非0则返回值为“是”
-w	对象存在且可写，则返回值为“是”
-x	对象存在且可执行，则返回值为“是”
!	测试条件的否定

【例6.10】 文件操作测试。

```
[root@rhel ~]# cat /dev/null>empty
[root@rhel ~]# cat empty
[root@rhel ~]# test -r empty
[root@rhel ~]# echo $?
0
//结果显示0表示文件empty存在且只读
[root@rhel ~]# test -s empty
[root@rhel ~]# test ! -s empty
[root@rhel ~]# echo $?
0
//结果显示0表示文件empty存在且文件长度为0
```

6.4 Shell条件判断语句

- Shell提供了用来控制程序和执行流程的命令，包括条件分支和循环结构，用户可以用这些命令创建非常复杂的程序。
- 在Shell程序中使用条件判断语句可以使用if条件语句和case条件语句，两者的区别在于使用case语句的选项比较多而已。

if条件语句

- Shell程序中的条件分支是通过if条件语句来实现的，其语法格式有if-then-fi语句和if-then-else-fi语句两种。

1.if-then-fi语句

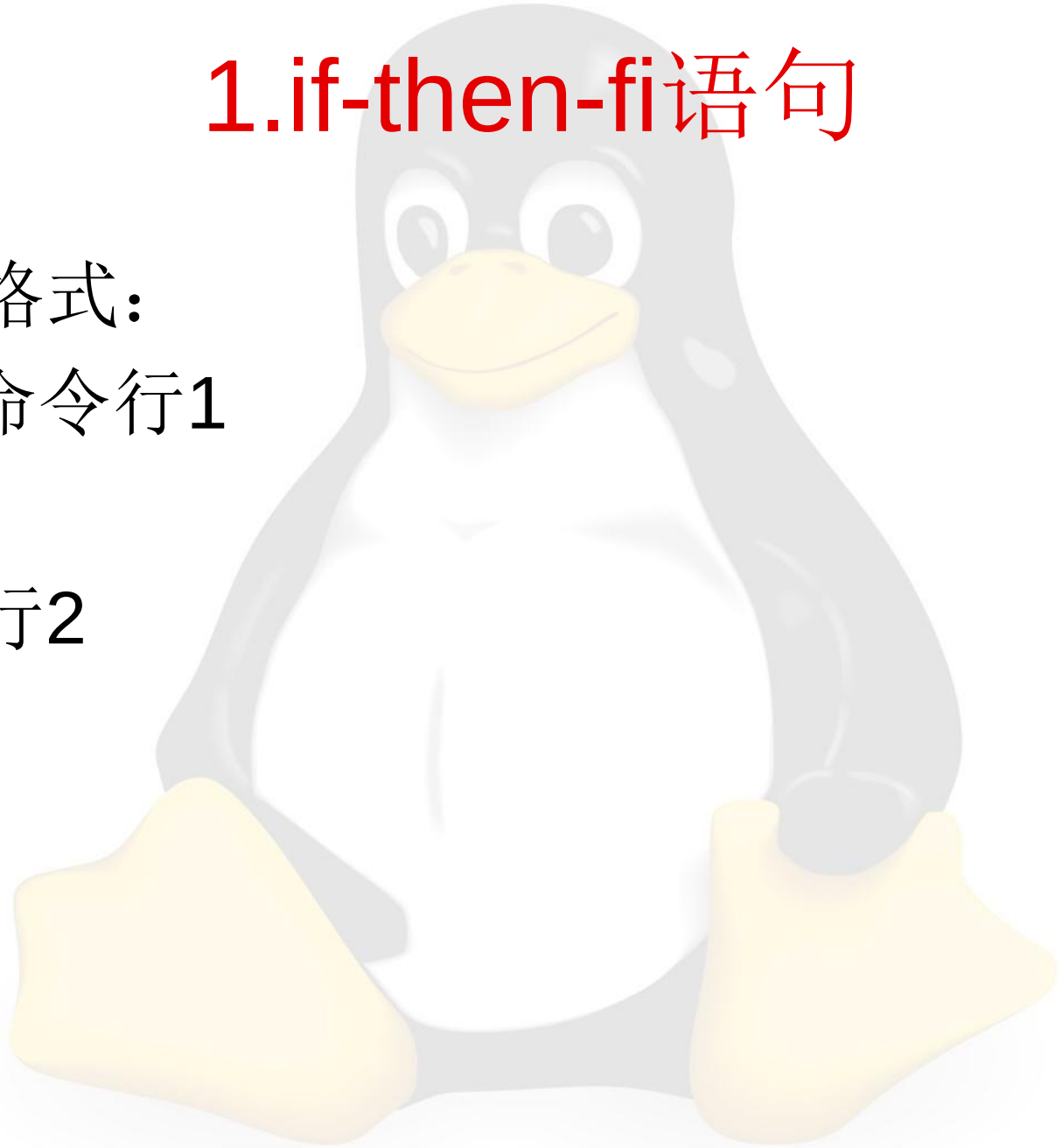
语法格式：

if　命令行1

then

命令行2

fi



【例6.11】 使用if-then 语句创建简单的Shell程序。

使用vi编辑器创建Shell程序，文件名为bbbb，文件内容如下所示。

```
#!/bin/bash
#filename:bbbb
echo -n "Do you want to continue: Y or N"
read ANSWER
if [ $ANSWER=N -o $ANSWER=n ]
then
exit
fi
```

运行Shell程序bbbb，输出内容如下所示。

```
[root@rhel ~]# bash bbbb
Do you want to continue: Y or N
```

2.if-then-else-fi语句

语法格式：

if

命令行1

then

命令行2

else

命令行3

fi

【例6.12】 使用if-then-else语句创建一个根据输入的分数判断分数是否及格的Shell程序。

使用vi编辑器创建Shell程序，文件名为ak，文件内容如下所示。

```
#! /bin/bash
#filename:ak
echo -n "please input a score:"
read SCORE
echo "You input Score is $SCORE"
if [ $SCORE -ge 60 ];
then
echo -n "Congratulation!You Pass the examination。"
else
echo -n "Sorry !You Fail the examination!"
fi
echo -n "press any key to continue!"
read $GOOUT
```

运行Shell程序ak，输出内容如下所示。

```
[root@rhel ~]# bash /root/ak
please input a score:80                    //输入数值80
You input Score is 80
Congratulation!You Pass the examination。press any key to continue!
[root@rhel ~]# bash /root/ak
please input a score:30                    //输入数值30
You input Score is 30
Sorry !You Fail the examination!press any key to continue!
```

case条件语句

- if条件语句用于在两个选项中选定一项，而case条件选择为用户提供了根据字符串或变量的值从多个选项中选择一项的方法。

语法格式：

```
case string in
exp-1)
    若干个命令行1
;  ;
exp-2)
    若干个命令行2
;  ;
.....
*)
    其它命令行
esac
```

- Shell通过计算字符串string的值，将其结果依次与运算式exp-1和exp-2等进行比较，直到找到一个匹配的运算式为止。如果找到了匹配项，则执行它下面的命令直到遇到一对分号（；；）为止。
- 在case运算式中也可以使用Shell的通配符（“*”，“？”，“[ ]”）。通常用“*”作为case命令的最后运算式以便在前面找不到任何相应的匹配项时执行“其他命令行”的命令。

【例6.13】 使用case语句创建一个菜单选择的Shell脚本。

使用vi编辑器创建Shell程序，文件名为za，文件内容如下所示。

```
#!/bin/bash
#filename:za
#Display a menu
echo _
echo "1 Restore"
echo "2 Backup"
echo "3 Unload"
echo
#Read and excute the user's selection
echo -n "Enter Choice:"
read CHOICE
case "$CHOICE" in
1) echo "Restore";;
2) echo "Backup";;
3) echo "Unload";;
*) echo "Sorry $CHOICE is not a valid choice
exit 1
esac
```

运行Shell程序za，输出内容如下所示。

```
[root@rhel ~]# bash /root/za

_
1 Restore
2 Backup
3 Unload
Enter Choice:
```

6.5 Shell循环控制语句

- 在Shell程序中使用循环控制语句可以使用for语句、while语句以及until语句。下面分别对其进行介绍。

for循环语句

- for循环语句对一个变量的可能的值都执行一个命令序列。赋给变量的几个数值既可以在程序中以数值列表的形式提供，也可以在程序以外以位置参数的形式提供。

for循环语句的语法格式如下所示。

```
for   变量名     [in数值列表]
do
    若干个命令行
done
```

- 变量名可以是用户选择的任何字符串，如果变量名是var，则在in之后给出的数值将顺序替换循环命令列表中的“$var”。如果省略了in，则变量var的取值将是位置参数。对变量的每一个可能的赋值都将执行do和done之间的命令列表。

【例6.14】 使用for语句创建简单的Shell程序。

使用vi编辑器创建Shell程序，文件名为mm，文件内容如下所示。

```
#!/bin/bash
#filename:mm
for ab in 1 2 3 4
do
echo $ab
done
```

运行Shell程序mm，输出内容如下所示。

```
[root@rhel ~]#bash /root/mm
1
2
3
4
```

【例6.15】 使用for语句创建求命令行上所有整数之和的Shell程序。

使用vi编辑器创建Shell程序，文件名为qqq，文件内容如下所示。

```
#!/bin/bash
#filename:qqq
sum=0
for INT in $*
do
sum='expr $sum + $INT'
done
echo $sum
```

运行Shell程序qqq，输出内容如下所示。

```
[root@rhel ～]# bash /root/qqq 1 2 3 4 5
15
```

while循环语句

- While语句是用命令的返回状态值来控制循环的。

语法格式：

```
while
    若干个命令行1
do
    若干个命令行2
done
```

- 只要while的“若干个命令行1”中最后一个命令的返回状态为真，while循环就继续执行“do...done”之间的“若干个命令行2”。

【例6.16】 使用while语句创建一个计算1到5的平方的Shell程序。
使用vi编辑器创建Shell程序，文件名为zx，文件内容如下所示。

```
#!/bin/bash
#filename:zx
int=1
while [ $int -le 5 ]
do
sq='expr $int \* $int'
echo $sq
int='expr $int + 1'
done
echo "Job completed"
```

运行Shell程序zx，输出内容如下所示。

```
[root@rhel ～]# bash /root/zx
1
4
9
16
25
Job completed
```

【例6.17】 使用while语句创建一个根据输入的数值求累加和（1+2+3+4+……+n）的Shell程序。

使用vi编辑器创建Shell程序，文件名为sum，文件内容如下所示。

```
#!/bin/bash
#filename:sum
echo -n "Please Input Number:"
read NUM
number=0
sum=0
while [ $number -le $NUM ]
do
echo number
echo "$number"
number='expr $number + 1'
echo sum
echo "$sum"
sum='expr $sum + $number'
done
echo
```

运行Shell程序sum，输出内容如下所示。

```
[root@rhel ～]# bash /root/sum
Please Input Number:4       //在这里输入了数字4
number
0
sum
0
number
1
sum
1
number
2
sum
3
number
3
sum
6
number
4
sum
10
```

until循环语句

- until循环语句是另外一种循环结构，它和while语句相类似。

语句格式：

```
until
    若干个命令行1
do
    若干个命令行2
done
```

- until循环和while循环的区别在于：while循环在条件为真时继续执行循环，而until则是在条件为假时继续执行循环。
- Shell还提供了true和false两条命令用于创建无限循环结构，它们的返回状态分别是总为0或总为非0。

【例6.18】使用until语句创建一个输入exit退出的Shell程序。

使用vi编辑器创建Shell程序，文件名为hk，文件内容如下所示。

```
#!/bin/bash
#filename:hk
echo "This example is for test until....do "
echo "If you input [exit] then quit the system "
echo -n "please input:"
read EXIT
until [ $EXIT = "exit" ]
do
read EXIT
done
echo "OK!"
```

运行Shell程序hk，输出内容如下所示。

[root@rhel ~]# bash /root/hk
This example is for test until....do
If you input [exit] then quit the system
please input:**exit**　　//输入exit退出
OK!