

## 第 12 章 Shell 脚本编程



### 本章内容

- Shell 命令行的运行
- 编写、修改权限和执行 Shell 程序的步骤
- 在 Shell 程序中使用参数和变量
- 表达式比较、循环结构语句和条件结构语句
- 在 Shell 程序中使用函数和调用其他 Shell 程序

### 12-1 Shell 命令行书写规则



#### 学习目标

- ◆ Shell 命令行的书写规则

对 Shell 命令行基本功能的理解有助于编写更好的 Shell 程序,在执行 Shell 命令时多个命令可以在一个命令行上运行,但此时要使用分号 (;) 分隔命令,例如:

```
[root@localhost root]# ls a* -l;free;df
```

长 Shell 命令行可以使用反斜线字符 (\) 在命令行上扩充,例如:

```
[root@localhost root]# echo "this is \  
>long command"  
This is long command
```



注意:

“>”符号是自动产生的,而不是输入的。

### 12-2 编写/修改权限及执行 Shell 程序的步骤



#### 学习目标

- ◆ 编写 Shell 程序
- ◆ 执行 Shell 程序

Shell 程序有很多类似 C 语言和其他程序设计语言的特征,但是又没有程序语言那样复杂。Shell 程序是指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 操作系统一个接一个地解释和执行每条命令。**首先,来编写第一个 Shell 程序,从中学习 Shell 程序的编写、修改权限、执行过程。

## 12-2-1 编辑 Shell 程序

编辑一个内容如下的源程序，保存文件名为 date，可将其存放在目录/bin 下。

```
[root@localhost bin]#vi date
#!/bin/sh
echo "Mr.$USER,Today is:"
echo &date "+%B%d%A"
echo "Wish you a lucky day !"
```



注意：

#!/bin/sh 通知采用 Bash 解释。如果在 echo 语句中执行 Shell 命令 date，则需要在 date 命令前加符号“&”，其中%B%d%A 为输入格式控制符。

## 12-2-2 建立可执行程序

编辑完该文件之后不能立即执行该文件，需给文件设置可执行程序权限。使用如下命令。

```
[root@localhost bin]#chmod +x date
```

## 12-2-3 执行 Shell 程序

执行 Shell 程序有下面三种方法：

方法一：

```
[root@localhost bin]#./ date
Mr.root,Today is:
二月 06 星期二
Wish you a lucky day !
```

方法二：

另一种执行 date 的方法就是把它作为一个参数传递给 Shell 命令：

```
[root@localhost bin]# Bash date
Mr.root,Today is:
二月 06 星期二
Wish you a lucky day !
```

方法三：

为了在任何目录都可以编译和执行 Shell 所编写的程序，即把/bin 的这个目录添加到整个环境变量中。

具体操作如下：

```
[root@localhost root]#export PATH=/bin:$PATH
[root@localhost bin]# date
Mr.root,Today is:
二月 06 星期二
Wish you a lucky day !
```



实例 12-1: 编写一个 Shell 程序 mkf, 此程序的功能是: 显示 root 下的文件信息, 然后建立一个 kk 的文件夹, 在此文件夹下建立一个文件 aa, 修改此文件的权限为可执行。

分析: 此 Shell 程序中需要依次执行下列命令为:

进入 root 目录: `cd /root`

显示 root 目录下的文件信息: `ls -l`

新建文件夹 kk: `mkdir kk`

进入 root/kk 目录: `cd kk`

新建一个文件 aa: `vi aa` #编辑完成后需手工保存

修改 aa 文件的权限为可执行: `chmod +x aa`

回到 root 目录: `cd /root`

因此该 Shell 程序只是以上命令的顺序集合, 假定程序名为 mkf

```
[root@localhost root]#vi mkf
```

```
cd /root
```

```
ls -l
```

```
mkdir kk
```

```
cd kk
```

```
vi aa
```

```
chmod +x aa
```

```
cd /root
```

## 12-3 在 Shell 程序中使用的参数



### 学习目标

- ◆ 位置参数
- ◆ 内部参数

如同 `ls` 命令可以接受目录等作为它的参数一样, 在 Shell 编程时同样可以使用参数。Shell 程序中的参数分为位置参数和内部参数等。

### 12-3-1 位置参数

由系统提供的参数称为位置参数。位置参数的值可以用 `$N` 得到, `N` 是一个数字, 如果为 1, 即 `$1`。类似 C 语言中的数组, Linux 会把输入的命令字符串分段并给每段进行标号, 标号从 0 开始。第 0 号为程序名字, 从 1 开始就表示传递给程序的参数。如 `$0` 表示程序的名字, `$1` 表示传递给程序的第一个参数, 以此类推。

## 12-3-2 内部参数

上述过程中的\$0 是一个内部变量，它是必须的，而\$1 则可有可无，最常用的内部变量有\$0、\$#、\$?、\$\*，它们的含义如下。

- \$0:命令含命令所在的路径。
- \$#:传递给程序的总的参数数目。
- \$?:Shell 程序在 Shell 中退出的情况，正常退出返回 0，反之为非 0 值。
- \$\*:传递给程序的所有参数组成的字符串。



实例 12-2: 编写一个 Shell 程序，用于描述 Shell 程序中的位置参数为: \$0、\$#、\$?、\$\*，程序名为 test1，代码如下：

```
[root@localhost bin]#vi test1
#!/bin/sh
echo "Program name is $0" ;
echo "There are totally $# parameters passed to this program" ;
echo "The last is $?" ;
echo "The parameter are $*" ;
执行后的结果如下:
[root@localhost bin]# test1 this is a test program //传递 5 个参数
Program name is /bin/test1 //给出程序的完整路径和名字
There are totally 5 parameters passed to this program //参数的总数
The last is 0 //程序执行效果
The parameters are this is a test program //返回由参数组成的字符串
```



注意：命令不计算在参数内。



实例 12-3: 利用内部变量和位置参数编写一个名为 test2 的简单删除程序，如删除的文件名为 a，则在终端中输入的命令为: test a

分析：除命令外至少还有一个位置参数，即\$#不能为 0，删除不能为\$1，程序设计过程如下。

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test2
#!/bin/sh
if test $# -eq 0
then
echo "Please specify a file!"
else
gzip $1 //现对文件进行压缩
```

```
mv $1.gz $HOME/dustbin          //移动到回收站
echo "File $1 is deleted !"
fi
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test2
```

(3) 运行

```
[root@localhost bin]# test2 a (如果 a 文件在 bin 目录下存在)
File a is deleted!
```

## 12-4 在 Shell 程序中的使用变量



### 学习目标

- ◆ 变量的赋值
- ◆ 变量的访问
- ◆ 变量的输入

### 12-4-1 变量的赋值

在 Shell 编程中，所有的变量名都由字符串组成，并且不需要对变量进行声明。要赋值给一个变量，其格式如下：

变量名=值



注意：

等号 (=) 前后没有空格

例如：

x=6

a="How are you "

表示把 6 赋值给变量 x，字符串 "How are you " 赋值给变量 a。

### 12-4-2 访问变量值

如果要访问变量值，可以在变量前面加一个美元符号 "\$"，例如：

```
[root@localhost bin]#a="How are you "
[root@localhost bin]#echo "He juest said:$a"
A is:hello world
```

一个变量给另一个变量赋值可以写成：

变量 2=\$变量 1

例如：

x=\$i



i++可以写成:

i=\$((i+1))

### 12-4-3 键盘读入变量值

在 Shell 程序设计中, 变量的值可以作为字符串从键盘读入, 其格式为:

```
read 变量
```

例如:

```
[root@localhost bin]#read str
```

read 为读入命令, 它表示从键盘读入字符串到 str。



实例 12-4: 编写一个 Shell 程序 test3, 程序执行时从键盘读入一个目录名, 然后显示这个目录下所有文件的信息。

分析:

存放目录的变量为 DIRECTORY, 其读入语句为:

```
read DIRECTORY
```

显示文件的信息命令为: ls -a

```
[root@localhost bin]#vi test3
```

```
#!/bin/sh
```

```
echo "please input name of directory"
```

```
read DIRECTORY
```

```
cd $DIRECTORY
```

```
ls -l
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test3
```

(3) 执行

```
[root@localhost bin]#./test3
```



注意:

输入路径时需 "/"

/home



实例 12-5: 运行程序 test4, 从键盘读入 x、y 的值, 然后做加法运算, 最后输出结果。

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test4
```


```
#!/bin/sh
```

```
echo "please input x y"
```

```
read x,y
```

```
z=`expr $x+$y`
```

```
echo "The sum is $z"
(2) 设置权限
[root@localhost bin]#chmod +x test4
(3) 执行
[root@localhost bin]#./ test4
45 78
The sum is 123
```

 注意：

表达式 `total=`expr $total +$num`` 及 `num=`expr $num +1`` 中的符号 “`” 为键盘左上角的 “`” 键。

## 12-5 表达式的比较



### 学习目标

- ◆ 字符串操作符
- ◆ 逻辑运算符
- ◆ 用 `test` 比较的运算符
- ◆ 数字比较符
- ◆ 文件操作符

在 Shell 程序中，通常使用表达式比较来完成逻辑任务。表达式所代表的操作符有字符操作符、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 所独特的操作符。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，于是才提供了文件操作符。

### 12-5-1 字符串比较

作用：测试字符串是否相等、长度是否为零，字符串是否为 NULL。  
常用的字符串操作符如表 12-1 所示。

表 12-1 常用的字符串操作符

字符串操作符	含义及返回值
=	比较两个字符串是否相同，相同则为“真”
!=	比较两个字符串是否不相同，不同则为“真”
-n	比较两个字符串长度是否大于零，若大于零则为“真”
-z	比较两个字符串长度是否等于零，若等于零则为“真”



实例 12-6：从键盘输入两个字符串，判断这两个字符串是否相等，如相等输出。

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test5
#!/bin/Bash
read ar1
read ar2
[ "$ar1" = "$ar2" ]
echo $? #?保存前一个命令的返回码
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test5
```

(3) 执行

```
[root@localhost root]#./ test5
aaa
bbb
1
```



注意:

“ [ ”后面和“ ] ”前面及等号“ = ”的前后都应有一个空格；注意这里是程序的退出情况，如果 ar1 和 ar2 的字符串是不想等的非正常退出，输出结果为 1。



实例 12-7: 比较字符串长度是否大于零

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test6
#!/bin/Bash
read ar
[ -n "$ar" ]
echo $? //保存前一个命令的返回码
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test6
```

(3) 执行

```
[root@localhost bin]#./ test6
0
```



注意:

运行结果 1 表示 ar 的小于等于零，0 表示 ar 的长度大于零。

## 12-5-2 数字比较

在 Bash Shell 编程中的关系运算有别于其他编程语言，用表 12-2 中的运算符用 test 语句表示大小的比较。

表 12-2 用 test 比较的运算符



运算符号	含 义
-eq	相等
-ge	大于等于
-le	小于等于
-ne	不等于
-gt	大于
-lt	小于



实例 12-8：比较两个数字是否相等

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test7
#!/bin/Bash
read x,y
if test $x -eq $y
then
    echo "$x=$y"
else
    echo "$x!= $y"
fi
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test7
```

(3) 执行

```
[root@localhost bin]#./ test7
50 100
50!=100
[root@localhost bin]#./ test7
150 150
150= =150
```

## 12-5-3 逻辑操作

在 Shell 程序设计中的逻辑运算符如表 12-3 所示。

12-3 Shell 中的逻辑运算符

运算符号	含 义
!	反：与一个逻辑值相反的逻辑值
-a	与（and）：两个逻辑值为“是”返回值为“是”，反之为“否”
-o	或（or）：两个逻辑值有一个为“是”，返回值就是“是”



实例 12-9：分别给两个字符变量赋值，一个变量赋予一定的值，另一个变量为空，

求两者的与、或操作。

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test8
#!/bin/Bash
part1 =" 1111"
part2 =" "      #part2 为空
[ "$ part1"  -a  "$ part2" ]
echo $?         #保存前一个命令的返回码
[ "$ part1"  -o  "$ part2" ]
echo $?
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test8
```

(3) 执行

```
[root@localhost bin]#./ test8
1
0
```

## 12-5-4 文件操作

文件测试操作表达式通常是为了测试文件的信息，一般由脚本来决定文件是否应该备份、复制或删除。由于 test 关于文件的操作符有很多，在表 12-4 中只列举一些常用的操作符。

表 12-4 文件测试操作符

运算符号	含 义
-d	对象存在且为目录返回值为“是”
-f	对象存在且为文件返回值为“是”
-L	对象存在且为符号连接返回值为“是”
-r	对象存在且可读则返回值为“是”
-s	对象存在且长度非零则返回值为“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”



实例 12-10：判断 zb 目录是否存在于/root 下。

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test9
#!/bin/Bash
[ -d /root/zb ]
echo $?      #保存前一个命令的返回码
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test9
```

(3) 执行

```
[root@localhost bint]#./ test9
```

(4)在/root 添加 zb 目录

```
[root@localhost bin]#mkdir zb
```

(5)执行

```
[root@localhost bin]#. /test9
```

0



注意:

运行结果是返回参数“\$?”, 结果 1 表示判断的目录不存在, 0 表示判断的目录不存在。



实例 12-11: 编写一个 Shell 程序 test10, 输入一个字符串, 如果是目录, 则显示目录下的信息, 如为文件显示文件的内容。

(1) 用 vi 编辑程序

```
[root@localhost bin]#vi test10
```

```
#!/bin/Bash
```

```
echo "Please enter the directory name or file name"
```

```
read DORF
```

```
if [ -d $DORF ]
```

```
then
```

```
ls $DORF
```

```
elif [ -f $DORF ]
```

```
then
```

```
cat $DORF
```

```
else
```

```
echo "input error! "
```

```
fi
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test10
```

(3) 执行

```
[root@localhost bin]#. / test10
```

## 12-6 循环结构语句



### 学习目标

#### ◆ Shell 的循环语句

Shell 常见的循环语句有 for 循环、while 循环语句和 until 循环。

### 12-6-1 for 循环

语法:

```
for 变量 in 列表
do
    操作
done
```



注意:

变量要在循环内部用来指列表当中的对象。

列表是在 for 循环的内部要操作的对象, 可以是字符串也可以是文件, 如果是文件则为文件名。



实例 12-12: 在列表中的值: a, b, c, e, I, 2, 4, 6, 8 用循环的方式把字符与数字分成两行输出。

(1) 用 gedit 编辑脚本程序 test11

```
[root@localhost bin]#gedit test11
#!/bin/Bash
for i in a,b,c,e,I 2,4,6,8
do
echo $i
done
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test11
```

(3) 执行

```
[root@localhost bin]#. / test11
a,b,c,e,i
2,4,6,8
```



注意:

在循环列表中的空格可表示换行。



实例 12-13: 删除垃圾箱中的所有文件。

分析: 在本机中, 垃圾箱的位置是在 \$HOME/.Trash 中, 因而是在删除 \$HOME/.Trash 列表当中的所有文件, 程序脚本如下。

(1) 用 gedit 编辑脚本程序 test12

```
[root@localhost bin]#gedit test12
#!/bin/Bash
for i in $HOME/.Trash/*
do
    rm $ i
echo "$ i has been deleted!"
done
```

*\$HOME/.local/share/Trash/\**

(2) 设置权限

```
[root@localhost bin]#chmod +x test12
```

(3) 执行

```
[root@localhost bin]#./ test12
/root/.Trash/abc~ has been deleted!
/root/.Trash/abc1 has been deleted!
```



实例 12-14: 求从 1~100 的和。

(1) 用 gedit 编辑脚本程序 test13

```
[root@localhost bin]#gedit test13
#!/bin/Bash
total =0
for((j=1;j<=100;j++));
do
    total=`expr $total + $j`
done
echo "The result is $total"
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test13
```

(3) 执行

```
[root@localhost bin]#./ test13
The result is 5050
```



注意:

for 语句中的双括号不能省, 最后的分号可有可无, 表达式 `total=`expr $total + $j`` 的加号两边的空格不能省, 否则会成为字符串的连接。

## 12-6-2 while 循环

语法:

```
while 表达式
do
    操作
done
```

只要表达式为真, do 和 done 之间的操作就一定会进行。



实例 12-15: 用 while 循环求 1~100 的和。

(1) 用 gedit 编辑脚本程序 test14

```
[root@localhost bin]#gedit test13
total =0
```



```
num=0
while((num<=100));
do
    total=' expor $total +$ num'
done
echo "The result is $total"
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test14
```

(3) 执行

```
[root@localhost bin]#. / test14
The result is 5050
```

## 12-6-3 until 循环

语法:

```
until 表达式
do
    操作
done
```

重复 do 和 done 之间的操作直到表达式成立为止。



实例 12-16: 用 until 循环求 1~100 的和。

(1) 用 gedit 编辑脚本程序 test15

```
[root@localhost bin]#gedit test15
total =0
num=0
until [$sum -gt 100]
do
    total=' expor $total +$ num'
    num=' expr $num + 1'
done
echo "The result is $total"
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test15
```

(3) 执行

```
[root@localhost bin]#. / test15
The result is 5050
```

## 12-7 条件结构语句



### 学习目标

- ◆ Shell 的条件结构语句

Shell 程序中的条件语句主要有 if 语句与 case 语句。

### 12-7-1 if 语句

语法：

```
if 表达式 1 then
操作
elif 表达式 2 then
操作
elif 表达式 3 then
操作
.....
else
操作
fi
```

Linux 里的 if 的结束标志是将 if 反过来写成 fi；而 elif 其实是 else if 的缩写。其中，elif 理论上可以有无限多个。



实例 12-17：用 for 循环求 1~100 的和。

(1) 用 gedit 编辑脚本程序 test16

```
[root@localhost bin]#gedit test16
for((j=0;j<=10;j++))
do
    if(($j%2==1))
    then
        echo "$j"
    fi
done
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test16
```

(3) 执行

```
[root@localhost bin]#./ test16
13579
```

## 12-7-2 case 语句

语法:

```
case 表达式 in
值 1|值 2)
操作;;
值 3|值 4)
操作;;
值 5|值 6)
操作;;
*)
操作;;
esac
```

case 的作用就是当字符串与某个值相同是就执行那个值后面的操作。如果同一个操作对于多个值,则使用“|”将各个值分开。在 case 的每一个操作的最后面都有两个“;;”分号是必需的。



实例 12-18: Linux 是一个多用户操作系统,编写一程序根据不同的用户登录输出不同的反馈结果。

(1) 用 vi 编辑脚本程序 test17

```
[root@localhost bin]#gedit test17
#!/bin/sh
case $USER in
beechen)
echo "You are beichen!" ;;
liangnian)
echo "You are liangnian" ;           //注意这里只有一个分号
echo "Welcome !" ;;                //这里才是两个分号
root)
echo "You are root!" ;echo "Welcome !" ;;
//将两命令写在一行,用一个分号作为分隔符
*)
echo "Who are you?$USER?" ;;
easc
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test17
```

(3) 执行

```
[root@localhost bin]#./ test17
You are root
Welcome!
```

## 12-8 在 Shell 脚本中使用函数



### 学习目标

#### ◆ Shell 的函数

Shell 程序也支持函数。函数能完成一特定的功能，可以重复调用这个函数。

函数格式如下：

函数名 ( )

{

函数体

}

函数调用方式为

函数名 参数列表



实例 12-19：编写一函数 add 求两个数的和，这两个数用位置参数传入，最后输出结果。

(1) 编辑代码

```
[root@localhost bin]#gedit test18
#!/bin/sh
add()
{
a=$1
b=$2
z=' expr $a + $b'
echo "The sum is $z"
}
add $1 $2
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test18
```

(3) 执行

```
[root@localhost bin]#./ test18 10 20
The sum is 30
```



注意：

函数定义完成后必须同时写出函数的调用，然后对此文件进行权限设定，在执行此文件。

## 12-9 在 Shell 脚本中调用其他脚本



### 学习目标

#### ◆ Shell 脚本的调用

在 Shell 脚本的执行过程中，Shell 脚本支持调用另一个 Shell 脚本，调用的格式为：

程序名



实例 12-20：在 Shell 脚本 test19 中调用 test20。

(1) 调用 test20

```
#test19 脚本
#!/bin/sh
echo "The main name is $0"
./test20
echo "The first string is $1"
#test20 脚本
#!/bin/sh
echo "How are you $USER?"
```

(2) 设置权限

```
[root@localhost bin]#chmod +x test19
[root@localhost bin]#chmod +x test20
```

(3) 执行

```
[root@localhost bin]#./ test19 abc123
The main name is ./test19
How are you root?
the first string is abc123
```



注意：

- 1) 在 Linux 编辑中命令区分大小写字符。
- 2) 在 Shell 语句中加入必要的注释，以便以后查询和维护，注释以#开头。
- 3) 对 Shell 变量进行数字运算时，使用乘法符号“\*”时，要用转义字符“\”进行转义。
- 4) 由于 Shell 对命令中多余的空格不进行任何处理，因此程序员可以利用这一特性调整程序缩进，达到增强程序可读性效果。
- 5) 在对函数命名时最好能使用有含义且能容易理解的名字，即使函数名能够比较准确地表达函数所完成的任务。同时建议对于较大的程序要建立函数名和变量命名对照表。

## 12-10 本章小结

本章讲解了 Linux 下 Shell 脚本的定义和相关 Shell 脚本编写的基础，这些基础知识是学习 Shell 脚本编程的关键。接着讲解了 Shell 脚本的执行方式和 Shell 脚本的常见流程控制，



为 Shell 脚本的编程做了准备。

## 课后习题

### 1. 选择题

- (1) 下列说法中正确的是 ( )。
  - A. 安装软件包 fctix-3.4.tar.bz2, 要按顺序使用 ./configure;make;make install;tar 命令
  - B. 挂载 U 盘, mount /dev/sda /mnt/u -o iocharset=gb2312
  - C. 显示变量 PS1 的值用命令 echo PS1
  - D. 用命令 ./abc 与 sh abc 执行 Shell 脚本 abc, 所得的结果并不相同
- (2) 一个 Bash Shell 脚本的第一行是什么 ( )。
  - A. #!/bin/Bash
  - B. #/bin/Bash
  - C. #/bin/csh
  - D. /bin/Bash
- (3) 在 Shell 脚本中, 用来读取文件内各个域的内容并将其赋值给 Shell 变量的命令是 ( )。
  - A. fold
  - B. join
  - C. tr
  - D. read
- (4) 下列变量名中有效的 Shell 变量名是 ( )。
  - A. -2-time
  - B. \_2\$3
  - C. trust\_no\_1
  - D. 2004file
- (5) 下列对 Shell 变量 FRUIT 操作, 正确的是 ( )。
  - A. 为变量赋值: \$FRUIT=apple
  - B. 显示变量的值: fruit=apple
  - C. 显示变量的值: echo \$FRUIT
  - D. 判断变量是否有值: [ -f "\$FRUIT" ]
- (6) 在 Fedora 12 系统中, 下列关于 Shell 脚本程序说法不正确的是 ( )。
  - A. Shell 脚本程序以文本的形式存储
  - B. Shell 脚本程序在运行前需要进行编译
  - C. Shell 脚本程序由解释程序解释执行
  - D. Shell 脚本程序主要用于系统管理和文件操作, 它能够方便自如地处理大量重复性的系统工作
- (7) 在 Shell 编程中关于 \$2 的描述正确的是 ( )。
  - A. 程序后携带了两个位置参数
  - B. 宏替换
  - C. 程序后面携带的第二个位置参数
  - D. 携带位置参数的个数
  - E. 用 \$2 引用第二个位置参数
- (8) 在 Fedora 12 系统中, “run.sh” 是 Shell 执行脚本, 在执行 ./run.sh file1 file2 file3 的命令的过程中, 变量 \$1 的值为 ( )。
  - A. run.sh
  - B. file1
  - C. file2
  - D. file3

### 2. 填空题

- (1) 在 Shell 编程时, 使用方括号表示测试条件的规则是\_\_\_\_\_。
- (2) 编写的 Shell 程序运行前必须赋予该脚本文件\_\_\_\_\_权限。

### 3. 简答题

- (1) 用 Shell 编程, 判断一文件是不是字符设备文件, 如果是将其拷贝到 /dev 目录下。
- (2) 在根目录下有四个文件 m1.txt, m2.txt, m3.txt, m4.txt, 用 Shell 编程, 实现自动创建 m1, m2, m3, m4 四个目录, 并将 m1.txt, m2.txt, m3.txt, m4.txt 四个文件分别拷贝

到各自相应的目录下。

- (3) 某系统管理员需每天做一定的重复工作，请按照下列要求，编制一个解决方案：
- 在下午 4 :50 删除/abc 目录下的全部子目录和全部文件；
  - 从早 8:00~下午 6:00 每小时读取/xyz 目录下 x1 文件中每行第一个域的全部数据加入到/backup 目录下的 bak01. txt 文件内；
  - 每逢星期一下午 5:50 将/data 目录下的所有目录和文件归档并压缩为文件：backup. tar. gz；
  - 在下午 5:55 将 IDE 接口的 CD-ROM 卸载（假设：CD-ROM 的设备名为 hdc）；
  - 在早晨 8:00 前开机后启动。
- (4) 请用 Shell 编程来实现：当输入不同的选择时，执行不同的操作，如：输入 start 开始启动应用程序 myfiles，输入 stop 时，关闭 myfiles，输入 status 时，查看 myfiles 进程，否则执行\*) 显示“EXIT!”并退出程序。
- (5) 编写一个 Shell 程序，此程序的功能是：显示 root 下的文件信息，然后建立一个 abc 的文件夹，在此文件夹下建立一个文件 k. c，修改此文件的权限为可执行。
- (6) 编写一个 Shell 程序，挂载 U 盘，在 U 盘中根目录下所有. c 文件拷贝到当前目录，然后卸载 U 盘。
- (7) 编写一个 Shell 程序，程序执行时从键盘读入一个文件名，然后创建这个文件。
- (8) 编写一个 Shell 程序，键盘输入两个字符串，比较两个字符串是否相等。
- (9) 编写三个 Shell 程序，分别用 for、while、与 until 求从 2+4+...+100 的和。
- (10) 编写一个 Shell 程序，键盘输入两个数及+、-、\*、与/中的任一运算符，计算这两个数的运算结果。
- (11) 编写两个 Shell 程序 kk 及 aa，在 kk 中输入两个数，调用 aa 计算计算这两个数之间奇数的和。
- (12) 编写 Shell 程序，可以挂载 U 盘，也可挂载 Windows 硬盘的分区，并可对文件进行操作。
- (13) 编写 4 个函数分别进行算术运算+、-、\*、/，并编写一个菜单，实现运算命令。

## 课程实训

实训内容：编写一个 Shell 程序，呈现一个菜单，有 0-5 共 6 个命令选项，1 为挂载 U 盘，2 为卸载 U 盘，3 为显示 U 盘的信息，4 把硬盘中的文件拷贝到 U 盘，5 把 U 盘中的文件拷贝到硬盘中，选 0 为退出。

程序分析：把此程序分成题目中要求的 6 大功能模块，另外加一个菜单显示及选择的主模板。

(1) 编辑代码

```
[root@localhost bin]#vi test19
#!/bin/sh
#mountusb.sh
#退出程序函数
quit()
{
    clear
    echo "*****"
    echo "***          thank you to use,Good bye!          ***"
    exit 0
}
```

```

#加载 U 盘函数
mountusb()
{
    clear
    #在/mnt 下创建 usb 目录
    mkdir /mnt/usb
    #查看 U 盘设备名称
    /sbin/fdisk -l |grep /dev/sd
    echo -e "Please Enter the device name of usb as shown above:\c"
    read PARAMETER
    mount /dev/$PARAMETER /mnt/usb
}

#卸载 U 盘函数
umountusb ()
{
    clear
    ls -la /mnt/usb
}

#显示 U 盘信息函数
display()
{
    clear
    umount /mnt/usb
}

#拷贝硬盘文件到 U 盘函数
cpdisktousb()
{
    clear
    echo -e "Please Enter the filename to be Copide (under Current directory):\c"
    read FILE
    echo "Copying,please wait!..."
    cp $FILE /mnt/usb
}

#拷贝 U 盘函数到硬盘文件
cpusbtodisk()
{
    clear
    echo -e "Please Enter the filename to be Copide in USB:\c"
    read FILE
    echo "Copying ,Please wait!..."
    cp /mnt/usb/$FILE . #点 (.) 表示当前路径
}

clear
while true

```

```

do
echo "===== "
echo "***          LINUX USB MANAGE PROGRAM          ***"
echo "          1-MOUNT USB          "
echo "          2-UNMOUNT USB          "
echo "          3-DISPLAY USB INFORMATION          "
echo "          4-COPY FILE IN DISK TO USB          "
echo "          5-COPY FILE IN USB TO DISK          "
echo "          0-EXIT          "
echo "===== "
echo -e "Please Enter a Choice(0-5):\c"
read CHOICE
case $CHOICE in
    1) mountusb
    2) unmountusb
    3) display
    4) cpdisktousb
    5) cpusbtodisk
    0) quit
    *) echo "Invalid Choice!Corrent Choice is (0-5)"
    sleep 4
    clear;;
esac
done

```

(2) 修改权限

```
[root@localhost bin]#chmod +x test19
```

(3) 程序执行结果

```
[root@localhost bin]#./ test19
```

## 项目实践

这段时间陈飞在学习 Linux 下的 Shell 编程，感觉 Shell 编程和 C 语言很相似。王工程师今天来看陈飞，顺便问一下陈飞的学习情况。陈飞就和他说了自己对 Shell 编程的看法。王工程师听了后，笑着说，“一样不一样，你编个程序不久明白了吗。”“那编什么程序呢”。陈飞问道。“就俄罗斯方块吧”，王工程师说。“俄罗斯方块大家都会玩，而且你可以在网上找到用 C 语言编写的程序，你用 Shell 编程实现，和 C 语言版的对比一下，不就明白了它们之间的不同了吗”。王工程师走了，留下了陷入沉思的陈飞。他能完成吗？