Linux shell 编程

V 1.0

http://www.51testing.com

深圳博为峰信息技术有限公司

课程内容

- 第一部分：Linux Shell 简介
- 第二部分：Shell 程序设计基础
- 第三部分：Shell 程序设计流程控制
- 第四部分：Shell函数

第一部分

Linux Shell 简介

Linux Shell 简介

Shell 简介

命令解释语言　　程序设计语言

当一个用户登陆linux 系统后，系统就会为该用户创建一个shell进程。

Shell版本：

Bourne Shell：是贝尔实验室开发的，unix普遍使用的shell，在编程方面比较优秀，但在用户交互方面没有其他shell优秀。

BASH：是GNU的Bourne Again Shell，是GNU操作系统上默认的shell，在bourne shell基础上增强了很多特性，如命令补全，命令历史表等等

Korn Shell：是对Bourne Shell 的发展，在大部分内容上与Bourne Shell兼容，集成了C Shell和Bourne shell优点。

C Shell：是SUN公司Shell的BSD版本，语法与c语言相似，比bourne shell 更适合编程

……

Linux Shell 案例

Shell案例：

```
[root@sugarCRM ~]# vi myshell.sh
#!/bin/sh
echo "hello,world“

[root@sugarCRM ~]# chmod u+x myshell.sh

[root@sugarCRM ~]# ./myshell.sh
hello,world
```

第二部分

Shell 程序设计基础

Shell 程序设计基础

2.1 Shell 输入输出

2.2 Shell 后台执行命令

2.3 引号

2.4 Shell 变量，参数

2.1 Shell输入输出

2.11 echo

2.12 read

2.13 cat 和管道

2.14 tee

2.15 标准输入，输出和错误

结合使用标准输出和标准错误

合并标准输出和标准错误

2.11 Shell输入输出－echo

echo命令:用来显示文本行或变量取值，或者把字符串输入到文件中

格式： echo string

echo的常用功能：\c 不换行 \f 不进纸 \t 跳格 \n 换行

note：

对于linux系统，必须使用**-e**选项来使以上转义符生效

例：$ echo -e “hello\tboy”

$ hello boy

echo命令对特殊字符敏感，如果要输出特殊字符，需要用\屏蔽其特殊含义。

常用的特殊字符：双引号 “” 反引号`` 反斜线\

例：$ echo “ \“ \” ” //输出“”

$ “ ”

2.11 Exercise：echo

1、 编写shell脚本，借助echo命令分别输出系统中SHELL，PATH变量的取值。

2、 编写shell脚本，使用一个echo命令输出如下格式的内容(注意对齐格式)：

id	name	msg
01	mike	“hello”
02	john	“hi”

2.12 Shell输入输出－read

从键盘或者文件的某一行文本中读入信息，并将其赋给一个变量。

格式：　read var1 var2 …

例1：$ read name

Hello I am superman

$ echo $name

$ Hello I am superman　　//显示结果

如果输入的值个数多于变量个数，多余的值会赋给最后一个变量：

例2：$ read name surname

John Mike Kate

$ echo $surname

2.12 Exercise：read

1、 编写shell脚本，实现功能：接收用户输入的值，并显示在屏幕上。

2、 编写shell脚本，使用read命令读取user和password变量，

第一次：输入 mike 1234567

第二次：输入 mike

第三次：输入 mike john 123456

分别使用echo命令查看user，password的取值，比较有何不同？

2.13 Shell输入输出－cat ,管道

cat

- 可以用来显示文件，并且支持将多个文件串连接后输出

  note：该命令一次显示完整个文件，若想分页查看，需使用more

- 格式： cat [options] filename1 … filename2 …

  常用 options： -v 显示控制字符 -n 对所有输出行进行编号

  -b 与-n相似，但空白行不编号

  例：$ cat file1 file2 file3 同时显示三个文件

  $ cat –b file1 file2 file3

管道 |

- 可以通过管道把一个命令的输出传递给另外一个命令做为输入
- 格式： 命令1 | 命令2

  例：$ cat test.txt | grep ‘hello’

2.13 Exercise：cat 管道

1、 编写shell脚本，对文件file1，file2，file3内容合并，并对每行进行编号。

2、 编写shell脚本，对文件file1，file2，file3统计非空行共有多少？

2.14 Shell输入输出－tee

tee

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

如果想看到输出的同时，把输出也同时拷入一个文件，这个命令很合适

格式： tee -a file

-a 表示文件追加到末尾

file 表示保存输出信息的文件

tee命令一般和管道符 | 结合起来使用

例：$ who | tee who.info

该命令的信息返回在屏幕上，同时保存在文件who.info中

2.14 Exercise： —tee

1、 使用tee命令将当前系统中所有进程的信息保存到文件pid_info 中

2、 使用tee命令和who命令将当前系统中登陆的终端信息追加到文件 login_info 中

2.15 Shell输入输出－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，见下表

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个描述符，可以任意使用文件描述符3－9

- 标准输入　　对应文件描述符0，是命令的输入，默认是键盘
- 标准输出　　对应文件描述符1，是命令的输出，默认是终端
- 标准错误　　对应文件描述符2，是命令错误的输出，默认是终端

利用文件重定向功能对命令的标准输入，输出和错误进行修改：

2.15 Shell输入输出－文件重定向

常用文件重定向命令：

command >file：　标准输出重定向到一个文件,错误仍然输出屏幕

command >>file：　标准输出重定向到一个文件(追加)

command 1>file1：　标准输出重定向到一个文件

command 2>>file2：　标准错误重定向到一个文件(追加)

command 1>file 2>&1：标准输出和标准错误一起重定向到一个文件

command>>file 2>&1：　标准输出和标准错误一起重定向到一个文件(追加)

command < file1 >file2：以file1做为标准输入，file2做为标准输出

command <file：　以file做为文件标准输入

注：上面的command>file 2>&1 可以修改为command>file 2>>file

2.15 Shell输入输出－文件重定向

- 重定向标准输出：

  例：$ ls –l >>myfile.out

- 重定向标准输入：

  例：$ sort < name.txt > name.out

  从name.txt读入数据进行排序，然后将排序结果输出到文件name.out中

- 重定向标准错误

  例：$ ls ddd 2> /dev/null

  ddd不存在，标准错误信息会输送到系统垃圾箱，而不会输送到屏幕

  如果标准错误信息有用，可以将其存放到错误文件中

2.15 Exercise：一文件重定向

1、 使用文件重定向功能将MySQL-client-5.0.16-0.i386.rpm的安装信息记录到文件

mysql_install.log 文件中。

2、 使用文件重定向功能将MySQL-client-5.0.16-0.i386.rpm的包说明信息追加到

mysql_install.log 文件中。

3、 使用文件重定向功能将ls ddd的错误信息保存在错误日志error.log文件中，

说明：ddd 目录并不存在。

4、 使用文件重定向功能将 ls ddd 的错误信息既不输出到屏幕，也不输出到错误文件。

5、 编写脚本，实现功能：读取文件myfile中一行数据，并通过echo打印出来

2.2 Shell后台执行命令

2.21 设置crontab文件，并用它来提交作业

2.22 在后台提交作业 &

名词解释：

cron 系统调度进程，可通过它按照一定的时间间隔或固定的时间点运行作业

& 使用它在后台运行一个占用时间不长的进程

2.21 Shell后台执行命令－cron

- cron 是系统的调度进程，可在无人干预的情况下运行作业，通过crontab的命令允许用户提交，编辑或者删除相应的作业。
- 每个用户都有一个crontab文件来保存作业调度信息，通过该命令运行任意一个shell脚本或者命令
- 在大的系统中，系统管理员可以通过cron.deny和cron.allow这两个文件来禁止或允许用户拥有自己的crontab文件

----------crontab的域----------

第1列	分钟1～59
第2列	小时1～23(0表示子夜)
第3列	日1～31
第4列	月1～12
第5列	星期0～6(0表示星期天)
第6列	要运行的命令

2.21 Shell后台执行命令－cron

- **crontab**格式： **分<>时<>日<>月<>星期<>要运行的命令**

  <>表示空格

  note：如果要表示范围的话，如周一到周五，可以用1-5表示

  如果要列举某些值， 如周一，周五，可以用1,5表示

例1：30 21 * * * /apps/bin/cleanup.sh

表示：每天21点30分运行/app/bin目录下的脚本cleanup.sh

例2：0,30 18-23 * * * /apps/bin/dbcheck.sh

表示：每天的18:00到23:00之间每隔半小时运行脚本backup.sh脚本

2.21 Shell后台执行命令－cron

- crontab的命令选项

  格式：**crontab [-u user] -e -l -r**

  其中　-u　　用户名

  　　-e　　编辑crontab文件

  　　-l　　列出crontab文件中的内容

  　　-r　　删除crontab文件

  如果使用自己的名字登陆，就不用使用-u选项

- 创建一个新的crontab文件

  修改$HOME目录下的.bash_profile文件，加入环境变量

  EDITOR=vi; export EDITOR　　//注：修改后重新登陆

  1 创建一个文件，建议名为<user>cron,例wuxhcron,在文件中加入如下内容：

2.21 Shell后台执行命令－cron

1 * * * * /usr/local/apache2/bin/apachectl start

3 * * * * /usr/local/apache2/bin/apachectl stop

保存退出

2 提交刚刚创建的cron文件wuxhcron

$ crontab wuxhcron

$ ls /var/spool/cron/ 是否生成文件wuxh

- 列出crontab文件

  $ crontab –l

  $ crontab –l > $HOME/mycron 可以通过这种方法对crontab进行备份

- 编辑crontab文件

  $ crontab -e

  修改后保存退出，cron会对其进行必要的完整性检查

2.21 Shell后台执行命令－cron

- 删除crontab文件

  $ crontab –r

- crontab文件的恢复

  如果误删了crontab文件，假设在$HOME目录下还有备份，可以将这个备份文件拷贝到/var/spool/cron/<username>　username是用户名，如果由于权限问题无法拷贝，可以使用

  $ crontab <filename>

  note：filename是备份的crontab文件的名字

- crontab的重启

  $ crond stop

  $ crond start

2.21 Exercise： —cron

1、 分别说明以下任务的目的：

```
45  4  1,10,22  *  *  /apps/bin/backup.sh
10  1  *  *  6,0        /usr/local/alert/file_check.sh
59  23  28  *  *       /usr/local/alert/file_check.sh
```

2、 创建一个crontab任务，要求每小时第1分钟调用一个脚本cleanDir.sh

脚本cleanDir.sh 功能：对$HOME/tmp目录进行清除

2.22 Shell后台执行命令－&

- 当在前台运行某个作业时，终端被该作业占据，无法继续操作。
  我们可以借助&命令把作业放到后台执行

- 格式： **命令 &**

  注：1 需要用户交互的命令不要放在后台执行，否则机器一直等待输入

  2 后台程序在执行时，执行结果仍然会输出到屏幕，干扰我们的工作，
  建议将这样的信息重定向到某个文件

  即：command > out.file 2>&1 &

  将标准输出及标准错误都定向到一个out.file的文件中

  例：$ find /etc -name “hello” -print >find.dt 2>&1 &

2.22 Exercise： —&

1、 使用&符号使apache编译源码的进程到后台执行，并把编译的信息存放到文件make_log中

操作的过程中，思考：如何判断后台的这个任务已经完成？

2.31 引号—双引号

“”	双引号	`	反引号
‘’	单引号	\	反斜线

- 双引号

  可引用除字符$,`,\外的任意字符或者字符串,对$,`,\敏感

例1：$ echo “hello”

例2：$ echo “$$” //想输出字符$$ 结果看到的是数值3746

$ echo “\$$” //对特殊字符需要反斜线屏蔽其特殊含义

$$ //得到想要的结果

例3：$ echo “`V_V`” //想输出`V_V`字样 结果得到错误信息

$ echo “\`V_V\`” //得到`V_V`输出

2.32 引号-单引号

单引号

单引号和双引号的用法基本类似，不同的是单引号对特殊字符不敏感，可以将其做为普通字符输出出来

例：$ echo '$$'　　结果 $$ 不用借助\进行屏蔽

$ echo '`V_V`'　　结果`V_V`,和前页双引号比较

2.33 引号-反引号

该命令用于设置系统命令的输出到变量，shell将反引号中的内容做为命令执行。

例1：$ echo `hello`

$ sh: hello: command not found

例2：$ echo `date`

$ Thu Nov 1 08:48:17 CST 2007

对比 $ MYDATE=“date”

$ echo $MYDATE

$ date

反引号可以和双引号结合起来使用：

例3：$ echo “The date today is `date`”

$ The date today is Thu Nov 1 08:48:17 CST 2007

2.34 引号—反斜线

- 反斜线

  如果一个字符有特殊含义，为防止shell误解其含义，可用\屏蔽该字符

  具有特殊含义的字符

  & * + ^ $ ` “ | ?

  例1： $ echo “$$” //在屏幕上输出$$字符,结果显示3853

  $ echo “\$$” //用反斜线屏蔽，防止shell误解,结果显示$$

  例2： $ echo * //在屏幕上输出*字符，结果输出当前目录下内容

  $ echo * //用反斜线屏蔽，防止shell误解，输出*字符

2.3 Exercise：一引号

1、 使用 grep 命令查询 myfile 中是否有字符串“`hello`” 应该怎样查询？

2、 怎样使用echo输出一句话: **The pen is $2**

3、 怎样使用echo输出一句话 **The time is AA** ,注AA需要被当前系统时间代替。

4、 怎样使用expr 12 * 12 得到144的结果

5、 怎样在屏幕上输出字符串 The price is $19.99

2.4 Shell 变量

2.41 环境变量

2.42 本地变量

2.43 位置变量

2.44 特定变量

2.41 环境变量

环境变量适用于所有用户进程

在/etc/profile中进行定义

在用户进程使用前，必须用export命令导出；建议环境变量都大写，

- 设置环境变量：

  var_name=value; export var_name

  或者：var_name=value

  export var_name

- 查看环境变量取值：

  echo $var_name

- unset var_name　删除某个系统环境变量

  注：该命令只是从当前用户进程中删除，不会从文件/etc/profile删除

2.41 环境变量

嵌入shell变量

一般来讲，BASH有一些预留的环境变量名，这些变量名不能做其他用途，通常在/etc/profile中建立这些嵌入的环境变量，但这不绝对，取决于用户

shell的变量列表：

CDPATH; EXINIT; HOME; IFS; LOGNAME; MAIL;

MAILCHECK; PATH; PS1; PS2; SHELL; TERMINFO;

TERM; TZ

2.41 Exercise：一环境变量

1、 环境变量应该定义在哪个文件中？

2、 练习在/etc/profile文件中定义一个环境变量，
练习使用unset命令删除环境变量

2.42 本地变量

在用户当前的shell进程中使用

一般在 **$HOME/.bash_profile** 中定义。

也可以在命令行定义，但只在用户当前shell进程中有意义，如果在shell中启动另一个进程或退出，此值将无效。

用法：**var_name=value**

查看本地变量取值：

echo $var_name

or **echo ${var_name}** --建议使用

删除变量：**unset var_name**

- 结合变量值：**echo ${var_name1}${var_name2}…**
- 测试变量是否设置： **${var:=new_value}** 若未设置或未初始化，可用新值

2.42 本地变量

- 使用变量保存系统命令参数

例：$ SOURCE="/etc/passwd"

$ DEST="/home/wuxh/"

$ cp $SOURCE $DEST

- 设置只读变量

可设置某个变量为只读方式，只读变量不可更改，否则系统返回错误

用法：var_name=value

readonly var_name

例: $ myvar="100"

$ readonly myvar

$ myvar="200"

$ -bash: myvar: readonly variable

2.42 Exercise：一本地变量

1、本地变量定义在哪个文件中？

2、分别在命令行和$HOME/.bash_profile文件中定义本地变量，

练习查看该本地变量的取值

练习删除该本地变量。

3、练习在$HOME/.bash_profile文件中定义一个只读用户变量，如何验证其只读生效？

2.43 位置变量

属于只读变量

作用：向shell脚本传递参数，参数个数可以任意多，但只有前9个被访问到，
shift命令可以更改这个限制。

每个访问参数前加$，

第一个参数为0，表示预留保存实际脚本名字，无论脚本是否有参数，此值均可用,如：给脚本test传递信息：

Would you like to do it

———————————————————————————————

$0	$1	$2	$3	$4	$5	$6	$7	$8	$9
脚本名字	would	you	like	to	do	it			

例：**$ vi test**

2.43 位置变量

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

保存文件，执行 **$ test would you like to do it**

2.43 位置变量

note：上例中$0返回的信息中包含路径名，如果只想得到脚本名称，可以借助basename,将脚本中第一句修改为：

echo “The script name is : `basename $0` ”

保存文件，执行 test would you like to do it

note：basename 用``

- 向系统命令传递参数

可以在脚本中向系统命令传递参数

$ **vi findfile**

#!/bin/sh

find / -name $1

保存，执行

$ **./findfile passwd**

2.43 Exercise：一位置变量

1、 写一个简单的脚本文件catfile.sh，要求实现的功能：

用户随意输入3个文件名，这3个文件的内容能够被cat命令连接起来显示，并且所有行都被标号；

用户输入的文件名可能真实存在，也可能不存在，需要将标准输出和标准错误分别重定向到文件catfile.log和catfile.err

2.44 特定变量

特定变量

反映脚本运行过程中的控制信息

特定的shell变量列表：

$#	传递到脚本的参数个数
$*	以一个单字符串的形式显示所有向脚本传递的参数，与位置变量不同，此项参数可超过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
$?	显示最后命令的退出状态，0表示正确，其他任何值表示错误

2.44 特定变量

例：修改test脚本，最后添加粗体部分：

```
#!/bin/sh
echo “The script name is : $0 ”
echo “The first parameter is :$1”
…
echo “The ninth parameter is :$9”
echo “The number of arguments passed :$#”
echo “Show all arguments :$*”
echo “Show my process id :$$”
echo “Show me the arguments in quotes :$@”
echo “Did my script go with any errors :$?”
```

2.44 特定变量

最后的退出状态　$?

可以在任何脚本或者命令中返回此变量以获得返回信息，基于此信息，

可以在脚本中做更进一步的研究，返回0为成功，1为失败

例1：$ cp /etc/passwd /home/wuxh/myfile

$ echo $?

$ 0

例2：$ cp /etc/passwd /home/wuxh/mydir　<mydir不存在>

$ echo $?

$ 1

建议将返回值设置为一个有意义的名字，增加脚本的可读性

修改例2 $ cp_status=$?

$ echo $cp_status

第三部分

Shell程序设计流程控制

3 Shell 程序设计流程控制

3.1 test 语句－文件测试

- 文件测试

  测试文件状态：

  用法：test condition　或者 [condition]

  ----------文件状态列表----------

-d	目录	**-s**	文件长度大于0，非空
-f	正规文件	**-w**	文件可写
-L	符号文件	**-r**	文件可读
-x	文件可执行		

例：**$ ls -l hello**

$ [-w hello]

$ echo $?

3.1 test 语句－文件测试

- 文件测试

使用逻辑操作符：

测试文件状态是否ok，可以借助逻辑操作符对多个文件状态进行比较

-a 逻辑与，操作符两边均为真，结果为真，否则为假

-o 逻辑或，操作符两边一边为真，结果为真，否则为假

! 逻辑否，条件为假，结果为真

例1：**$ [-r myfile1 -a -w myfile2]**

$ echo $?

例2：**$ [-w myfile1 -o -x myfile2]**

$ echo $?

3.1 test 语句—字符串测试

- 字符串测试

字符串测试是错误捕获很重要的一部分，特别是用户输入或比较变量时尤为重要

格式：

test “string”

test string_operator “string”

test “string” string_operator “string”

[string_operator string]

[string string_operator string]

注：string_operator 的取值：

= 等于　　!= 不等于　　-z 空串　　-n 非空串

例：测试环境变量EDITOR是否为空

3.1 test 语句—字符串测试

```
$ [  -z   $EDITOR  ]
$  echo $?                    为空返回0，否则返回1
```

如果非空，取值是否为**vi**

```
$ [  $EDITOR = “vi” ]
$  echo $?
```

测试变量**string1**是否等于**string2**

```
$ string1＝”hello”
$ string2＝”Hello”
$ [  “$string1”  =   “$string2”  ]
$ echo $?
```

note：在进行字符串比较时，建议加引号

3.1 test 语句—数值测试

- 数值测试

格式：**“number” number_operator “number”**

或者：**[“number” number_operator “number”]**

number_operator 的取值范围：

-eq	数值相等	-gt	第一个数大于第二个数
-ne	数值不相等	-lt	第一个数小于第二个数
-le	第一个数小于等于第二个数		
-ge	第一个数大于等于第二个数		

例：**$ NUM1＝130**

$ [$NUM1 –eq 130]

$ echo $?

3.1 test 语句—数值测试

例：$ [990 -le 996 -a 123 -gt 33]

$ echo $?

3.1 Exercise：—test测试语句

1、 使用test命令判断一个对象是否是目录，并查看判断结果

2、 使用test命令判断一个文件是否是链接文件，并查看结果

3、 使用test命令判断一个文件是否非空并且可写，并查看结果

4、 使用test命令判断“hello”和“HELLO”字符串是否相等

5、 使用test命令判断“ ”是否为空串

6、 执行一个脚本文件时，需要用户输入3－6个位置参数，怎样使用test语句来判断用户输入的参数是在3－6个范围内

3.2 expr 语句—字符串测试和数值测试

一般用于整数值，也可以用于字符串；

格式：**expr argument operator argument**

- expr 也是个手工命令行的计数器

  $ expr 10 + 10　　注意空格

  $ expr 300 / 6 / 5

  $ expr 30 * 3　　注意：乘号必须用反斜线屏蔽其特定含义

- 增量计数

  expr在循环中用于增量计算，首选，循环初始化为0，然后循环加1，常用的做法：从expr接受输出赋给循环变量

例：**$ LOOP=0**

$ LOOP=`expr $LOOP + 1`

3.2 expr 语句—字符串测试和数值测试

- 数值测试

  可以用expr测试一个数，如果对非整数进行计算，则返回错误

  例：$ expr 1.1 + 1　　返回错误

  $ expr 1 + 1　　返回2

- 字符串测试

  注 expr 也有返回的状态，但与系统最后返回的值刚好相反，expr返回成功为1，其他值为失败。

  例：$ value=hello

  $ expr $value = “hello”　　//注意=前后都有空格

  $ 1　　//这是expr执行成功的值

  $ echo $?

  $ 0　　//这是系统返回的成功的值

3.2 Exercise：—expr测试语句

1、使用echo命令输出一句话:**300/5*6=**360

注意：结果部分360需要使用expr命令运算出来，而不是直接给出结果

2、如果一个脚本中定义了一个变量，这个变量可以按照用户执行脚本时指定的数值进行自增操作

这个变量的自增语句在脚本中该如何写？

3.3 if 条件语句

格式：**if　条件1**

then　命令1

elif　条件2

then　命令2

else　　命令3　　//注：蓝色部分为可选部分

fi

注意：使用if语句时，必须将then部分放在新行，否则会产生错误，如果要不分行，必须使用命令分割符，即：

```
if　条件1;　then
    命令1
fi
```

3.3 If 条件语句

例：$ vi myfile.sh

```
#!/bin/sh
If [ “`ls  -A $DIRECTORY`” =“” ] ;  then
  echo  “$DIRECTORY  is  indeed empty”
  else
  echo  “$DIRECTORY  is  not  empty”
fi
```

3.3 Exercise：－if 条件语句

1、 写一个脚本，要求用户可以随意输入一个帐号，打印出该帐号，如果帐号为空，打印“You did not enter any info”

2、 当前目录的user.txt文件中存放多个用户的信息，要求用户可以随意输入一个登陆帐号，脚本会到user.txt中查找是否存在该帐号，如果存在，打印“用户xxx可以登陆系统”，否则，打印“用户xxx是非法用户”，请在练习1的基础上修改.

3、 写一个脚本，实现创建目录的功能，目录的名称由用户给出，需要对如下情况进行判断处理：

1 用户没有给出参数

2 用户给出的目录名称是否在当前目录存在，如果存在，提示用户重新创建

3 目录创建成功或者失败，都给出说明信息

3.4 for 循环

- 格式：**for　变量名 in 列表**

  do

  命令1

  命令2

  done

- 说明：命令 可为任何有效的shell命令和语句

  变量名可以为任何单词

  in列表是可选的，如果没有列表，for循环会使用命令行的位置参数

  in列表可以包含替换，字符串，文件名

例：

3.4 For 循环

```
#!/bin/sh
for    loop1    in 1 2 4 5 6          //数字列表
do
   echo $loop1
done
for    loop2    in  he is a tall man          //字符串列表
do
   echo $loop2
done
for     loop3    in `ls`                  //替换列表
do
   echo $loop3
done
```

3.4 for 循环

对for 循环使用参数，当循环中省去in列表选项时，它将接受命令行特定变量做为参数即

for params in “$@” 或者 for params in “$*”

例1：文件名**for2.sh**

```
#!/bin/sh
for params in “$*”
do
echo “You supplied $params as a command line option”
done
echo $params

./for2.sh a b c d e f
```

3.4 for 循环

例2

```
#!/bin/sh
counter=0
for files in `ls`
do
counter =` expr $counter  +  1`
done
echo  “There are $counter  files in `pwd`  wu need to process”
```

3.5 while 和 until

格式：**while** 命令

do

命令1

命令2

……

done

note：**do**和**done**之间命令，只有前一个返回状态为0，后面命令才会被执行；否则则循环中止

格式：**until** 条件

do

命令1

…….

done

note：until执行一系列命令，直到条件为真时停止。

3.5 while 和 until

例1：

```
#!/bin/sh
while read  FILM
do
    echo “the info is: $FILM”
done  < data.file
```

使用while循环读取文件内容

3.5 while 和 until

例2：
```
#!/bin/sh
IS_ROOT=`who | grep root`
until [  “$IS_ROOT”  ]
do
    sleep  5
    IS_ROOT=`who | grep root`
done
echo “Watch it. Roots in ” | mail zhang
```

思考：为什么用sleep 5？

3.5 Exercise：－while循环语句

1、使用while循环，要求从文件names.txt中逐行读取数据，并把文件信息逐行显示出来,

names.txt 内容包含雇员名字，部门，及其id，如下所示：

Louise Conrad:Accounts:ACC8987

Peter James:Payroll:PR489

Fred Terms:Customer:CUS012

James Lenod:Accounts:ACC887

Frank Pavely:Payroll:PR489

3.6 case 条件选择

格式：**case** 变量 **in**

模式1)

命令1

……

;;

模式2)

命令2

……

;;

esac

case 取值后面必须为in，每个模式必须以右括号结束，取值可以为变量或者常数，找到匹配模式后，执行相关命令直到;;

3.6 case 条件选择

模式部分可以包含元字符，与命令行中文件扩展名中使用的匹配类型相符，

如 *　?　[..]

例：**vi abc.sh**

```
#!/bin/sh
    if [  $# != 1 ];  then
      echo “Usage:`basename $0` [start|stop|help]”   >&2
      exit 1
    fi
    OPT=$1
       case $OPT in
          start) echo “starting..`basename $0`”
          # code here to start a process
          ;;
```

3.6 case 条件选择

```
 stop) echo “stopping..`basename $0`”
# code here to stop a process
;;
help)
# code here to display a help page
;;
*) echo “Usage:`basename $0` [start|stop|help]”
;;
esac
```

3.7 break 和 continue

有时需要某些准则退出循环或者跳过循环步，就需要break和continue来实现

- **break** 允许跳出循环或者**case**语句，在嵌套循环里，可以指定跳出的循环个数，例在两层的嵌套循环内，**break** 可以跳出整个循环
- **continue** 类似于**break**，区别是**continue**只会跳过当前的循环步，而不会跳出整个循环

3.7 break 和 continue

例1：

```
#!/bin/sh
    while :
    do
    echo –n  “Enter any number [1..5]  :”
    read ANS
    case $ANS in
    1|2|3|4|5) echo “great! you entered a number between 1 and 5”
    ;;
    *) echo “wrong number..bye!”
    break
    ;;
    esac
    done
```

3.7 break 和 continue

例2：names2.txt 内容包含雇员名字，部门，及其id，如下所示：

----------内容如下----------

---LISTING OF PERSONNEL FILE----

--- TAKEN AS AT 06/1999----------

Louise Conrad:Accounts:ACC8987

Peter James:Payroll:PR489

Fred Terms:Customer:CUS012

James Lenod:Accounts:ACC887

Frank Pavely:Payroll:PR489

要求：读取names2.txt文件，将在职员工的名字，部门，部门id读取打印出来

说明：Peter James已经离职

3.7 break 和 continue

例3：

```
#!/bin/sh
    SAVEDIFS=$IFS
    IFS=:
    INPUT_FILE=names2.txt
    NAME_HOLD="Peter James"
    LINE_NO=0
    if [ -s $INPUT_FILE ]; then
      while read NAME DEPT ID
      do
      LINE_NO=`expr $LINE_NO +1`
      if [ "$LINE_NO " -le 2 ]; then
        continue
      fi
```

3.7 break 和 continue

```
            if  [   “$NAME” = “$NAME_HOLD”  ];  then
                continue
            else
                 echo “ Now processing …$NAME $DEPT $ID”
             fi
             done < $INPUT_FILE
    IFS=$SAVEDIFS
    else
    echo “`basename $0 ` : Sorry file not found or there is no data in the file
>&2”
    exit 1
    fi
```

第四部分

Shell 函数

Shell 函数

- Shell 允许将一组命令集或语句形成一个可用块，这些块称为shell函数，其组成部分：

  函数标题，函数体

  标题是函数名，应该唯一；函数体是命令集合

- 函数格式：函数名()

  {

  命令1

  …

  }

  或者 function 函数名()

  { ….

  }

Shell 函数

- 函数可以只放在同一个文件中做为一段代码，也可以放在只包含函数的单独文件中

1. 在脚本中定义并使用函数

注：函数必须在使用前定义，一般放于脚本开始部分，直至shell解释器首次发现它时，才可以使用

例脚本func1：

```
#!/bin/sh
hello() {
echo “Hello,today’s date is `date`”
 }
 echo “now, going to the function hello”
 hello
 echo “back from the function”
```

$ ./func1　　//执行脚本func1

Shell 函数

2. 向函数传递参数

向函数传递参数就象在一般脚本中使用特殊变量$1,$2..$9一样，函数取得所传参数后，将原始参数传回shell，可以在函数内定义本地变量保存所传的参数，一般这样的参数名称以_开头

例： 脚本对输入的名字进行检查，只能包含字母

$ vi func2　　./func2 start

```
#!/bin/sh
echo –n “what is your first name :”
read F_NAME
char_name()
{
_LETTERS_ONLY=$1
_LETTERS_ONLY=`echo $1|awk ‘{if($0~/[^a-z A-Z]/) print “1”}’`
```

Shell 函数

```
    if [ “$_LETTERS_ONLY” != “” ]
        then
            return 1              #find number
        else
            return 0             # not find number
     fi
     }
     char_name  $F_name;
     ret=$?
if  [ $ret  -eq  0  ];  then
       echo “ok”
    else
       echo “ERRORS”
    fi
```

Shell 函数

3. 从调用函数中返回

函数执行完毕或者基于某个测试语句返回时，可作两种处理：

1) 让函数正常执行到末尾，然后返回脚本中调用函数的控制部分

2) 使用return 返回脚本中函数调用的下一条语句，可以带返回值，0为无错误，1为有错误

格式： return 从函数中返回，用最后状态命令决定返回值

return 0 无错误返回

return 1 有错误返回

Shell 函数

4. 函数返回值测试

可以直接在脚本调用函数语句的后面使用最后状态命令来测试函数调用的返回值

例：hello　　#这里是hello函数被调用

```
if [  $? = 0   ]
    then
        echo “it is ok”
    else
        echo “something is wrong with hello function”
     fi
```

更好的办法是使用if语句测试返回0还是返回1，可以在if语句里面将函数调用用括号括起来，增加可读性，如 if　hello　;　then

Shell 函数

如果函数将从测试结果中反馈输出，可以使用替换命令保存结果，函数调用的替换格式为

variable_name=function_name

函数function_name输出被设置到变量variable_name中

5. 在shell中使用函数

常用的一些函数可以收集起来，放入函数文件，使用时，将文件载入shell中

文件头应该以#!/bin/sh开头，文件名可任意选取，但建议有说明性。

文件一旦载入shell，就可以在命令行或者脚本中调用函数，可以使用set产看所有定义的函数，输出列表包括已经载入shell的所有函数。

要想改动文件，首先用unset命令从shell中删除函数，注，这里不是真正的删除，修改完毕后，再将文件重新载入，有些shell会识别改动，不必使用unset，但建议改动时使用unset命令

Shell 函数

6. 创建函数文件

例：**function.main**

```
#!/bin/sh
findit() {
if [ $# -lt 1 ]; then
   echo “Usage :findit file”
   return 1
 fi
 find / -name $1 -print
        }
```

7. 定位文件(载入文件)

格式：**<点><空格><路径><文件名>**

$. ./function.main

Shell 函数

8. 检查载入的函数

   使用set命令查看已经载入shell中的函数

   $ set

9. 执行shell函数

   要执行函数，简单的键入函数名，如果需要参数，后跟参数即可

   $ findit hello

10. 修改shell函数

    如果需要对函数做改动，需要借助unset命令先从shell中删除函数，修改后，再重新载入

    $ unset findit

    修改后…重新载入

    $ . ./function.main

4 Exercise：一函数

练习：写一个脚本，和用户进行交互，要求用户输入一个目录名称，通过在脚本中调用函数的方法实现，要求函数能够对这个目录名称进行判断，如果不是目录，也不是文件，则返回1，否则返回0；

如果函数返回1，脚本会继续创建该目录，创建失败，返回错误信息。