函数的基本知识

1、函数的定义和调用

函数是shell脚本中自定义的一系列执行命令，一般来说函数应该设置有返回值（正确返回0，错误返回非0，对于错误返回，可以定义返回其他非0正值来细化错误。）使用函数最大的好处是可避免出现大量重复代码，同时增强了脚本的可读性。

函数定义方法：

```
function   FUNCTION_NAME() {          #关键字function可以省略
        comment1  #函数体中可以有多个语句，不允许有空语句
        comment2   #语句可以是任意的shell命令，也可以调用其他的函数
}
```

如果在函数中使用exit命令，可以退出整个脚本，通常情况，函数结束之后会返回调用函数的部分继续执行。

可以使用break语句来中断函数的执行。

declare –f 可以显示定义的函数清单

declare –F 可以只显示定义的函数名

unset –f 可以从Shell内存中删除函数

export –f 将函数输出给Shell

另外，函数的定义可以放到 .bash_profile 文件中，也可以放到使用函数的脚本中，还可以直接放到命令行中，还可以使用内部的unset命令删除函数。一旦用户注销，Shell将不再保持这些函数。

例如：

```
[root@sunday-test shell-script]# cat function01.sh
#/bin/bash
function sayHello() {
        echo "hello , you are calling the function"
}
echo "first time call the function "
sayHello                        #调用函数
echo "second time call the function"
sayHello                        #再次调用函数
```

例子2：

```
[root@sunday-test shell-script]# cat function02.sh
#!/bin/bash
```

```
FILE=/etc/passwd
function countLine(){
        local i=0
        while read line
        do
          let ++i
        done < $FILE
        echo "$FILE have $i lines"
}

echo "call function countLine"
countLine
```

运行结果：

```
[root@sunday-test shell-script]# bash function02.sh
call function countLine
/etc/passwd have 53 lines
```

函数的返回值：

函数的返回值又叫函数的退出状态，实际上是一种通信方式。使用return关键字，函数中的关键字“return”可以放到函数体的任意位置，通常用于返回某些值，Shell在执行到return之后，就停止往下执行，返回到主程序的调用行，return的返回值只能是0~256之间的一个整数，返回值将保存到变量“$?”中。

例如1：

```
[root@sunday-test shell-script]# cat function03.sh
#!/bin/bash
function abc() {
    RESULT=`expr $1 \% 2`  #表示取余数
    if [ $RESULT -eq 0 ] ; then
        return 0
    else
        return 1
    fi
}
echo "Please enter a number who can devide by 2"
```

```
read -p "this number is: " N
abc $N
case $? in
    0)
        echo "yes ,it is"
        ;;
    1)
        echo "no ,it isn’ t"
        ;;
esac
```

例如2：

```
[root@sunday-test shell-script]# cat function04.sh
#!/bin/bash
function checkNum(){
        echo -n "Please input a number: "
        read NUM
        if [ $NUM -ge 0 -a $NUM -lt 10 ];then
                return 0
        elif [ $NUM -ge 10 -a $NUM -lt 20 ]; then
                return 1
        elif [ $NUM -ge 20 -a $NUM -lt 30 ]; then
                return 2
        else
                return 3
        fi
}
echo "call function checkNum"
checkNum
RTV=$?
if [ $RTV -eq 0 ]; then
   echo "the number is between [0,10]"
elif [ $RTV -eq 1 ]; then
```

```
    echo  "the number is between [10,20]"
  elif [ $RTV -eq 2 ]; then
    echo  "the number is between [20,30]"
  else
     echo  "unknown input"
  fi
```

函数参数的传递

函数可以通过位置变量传递参数。例如

函数名 参数1 参数2 参数3 参数4

当函数执行时，$1 对应 参数1，其他依次类推。

例如：

```
[root@sunday-test shell-script]# cat function05.sh
#!/bin/bash
function show() {
   echo "hello , you are calling the function  $1"
}
echo "first time call the function"
show first
echo "second time call the function"
show second
```

执行结果：

```
[root@sunday-test shell-script]# bash function05.sh
first time call the function
hello , you are calling the function  first
second time call the function
hello , you are calling the function  second
```

移动位置参数

在shell中使用shift命令移动位置参数，shift命令可让位置参数左移一位，实例如下：

```
[root@sunday-test shell-script]# cat function06.sh
#!/bin/bash
until [ $# -eq 0 ]
```

```
do
    #打印当前的第一个参数$1和参数的总个数$#
   echo "Now \$1 is: $1, total parameter is: $#"
   shift   #移动位置参数
done
```

运行结果：

```
[root@sunday-test shell-script]# bash function06.sh a b c d e f
Now $1 is: a, total parameter is: 6
Now $1 is: b, total parameter is: 5
Now $1 is: c, total parameter is: 4
Now $1 is: d, total parameter is: 3
Now $1 is: e, total parameter is: 2
Now $1 is: f, total parameter is: 1
```

以下例子是利用shift来计算脚本中所有参数的和：

```
[root@sunday-test shell-script]# cat function07.sh
#!/bin/bash
function Total() {
        local SUM=0
        until [ $# -eq 0 ]
        do
          let "SUM=SUM+$1"
          shift
        done
        echo $SUM
}
Total $@
```

运行结果：

```
[root@sunday-test shell-script]# bash function07.sh 20 30 56 60
166
```

函数库

对某些很常用的功能，必须考虑将其独立出来，集中存放在一些独立的文件中，这些文件就称为“函数库”。好处是后期开发过程中可以直接使用，在实践中为了和

一般函数区分开，建议库函数使用下划线开头。

例如自己开发一个函数库lib01.sh,功能是判断文件是否存在?

```
[root@sunday-test shell-script]# cat lib01.sh
#!/bin/bash
_checkFileExists() {
    if [ -f $1 ];then
       echo "File:$1 exists"
    else
       echo "File:$1 not exist"
    fi
}
```

其他脚本希望使用_checkFileExists函数时，可以通过直接加载lib01.sh函数库的方式实现。有两种方式加载：

使用“点”命令

```
[root@sunday-test shell-script]# . /PATH/TO/LIB
```

使用source命令

```
[root@sunday-test shell-script]# source  /PATH/TO/LIB
```

写一个脚本来加载直接调用_checkFileExists函数：

```
[root@sunday-test shell-script]# cat calllib01.sh
#!/bin/bash
source ./lib01.sh
_checkFileExists $1
```

运行结果演示：

```
[root@sunday-test shell-script]# bash calllib01.sh /etc/passwd
File:/etc/passwd exists
[root@sunday-test shell-script]# bash calllib01.sh /etc/passwddd
File:/etc/passwddd not exist
```

函数嵌套：

函数可以进行嵌套，实例： 一般不建议使用

```
[root@sunday-test shell-script]# cat function08.sh
#!/bin/bash
function first() {
```

```
    function second() {
        function third() {
            echo "-----------this is third"
        }
        echo "this is the second"
        third
    }
    echo "this is the first"
    second
}
echo "start....."
first
```