Shell高级篇

函数的作用与在于，当你需要重复的使用一段代码，如果是一小段代码重复还好，若是大段重复代码就太麻烦了。函数就可以解决这个问题。

函数是定义一个代码块，且命名，然后可以在脚本里任何位置调用。

创建函数

```
创建函数语法
function name {
  commands
}
```

案例

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

function func1 {
  echo "This is an example of a function"
}
count=1
while [ $count -le 5 ]
do
  func1
  count=$[ $count + 1 ]
done
echo "This is the end of the loop"
func1
echo "Now this is the end of the script"
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
This is an example of a function
This is an example of a function
This is an example of a function
This is an example of a function
This is an example of a function
This is the end of the loop
This is an example of a function
Now this is the end of the script
```


每次应用函数名`func1`，shell就会执行func1里定义好的命令。

##函数简写

```
[root@chaogelinux shell]# cat test.sh
```

```
#!/bin/bash
# 超哥带你学shell

demoFunc(){
  echo "This is a shell function."
}

echo "start.."
demoFunc
echo "end.."
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
start..
This is a shell function.
end..
```

##函数常见的坑

要注意代码的上下路径，要先定义函数，再执行函数。


```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

count=1
echo "This line comes befor the function definition."

function func1 {
  echo "This is an example of a function"
}

while [ $count -le 5 ]
do
  func1
  count=$[ $count + 1 ]
done
echo "This is the end of the loop."
func2
echo "Now this is the end of the script"

function func2 {
  echo "This is an example of a function"
}




[root@chaogelinux shell]#
```

```
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
This line comes befor the function definition.
This is an example of a function
This is an example of a function
This is an example of a function
This is an example of a function
This is an example of a function
This is the end of the loop.
test.sh:行17: func2: 未找到命令
Now this is the end of the script
```

这里就有代码报错了，就因为func2函数还未定义就调用了，所以报错。

坑2

注意函数名唯一，重复定义则会覆盖函数体的代码。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

function func1 {
  echo "This is the first definition of the function name"
}
func1



function func1 {
  echo "This is the repeat definition of the function name"
}

func1
echo "This is the end of the script"


[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
This is the first definition of the function name
This is the repeat definition of the function name
This is the end of the script
[root@chaogelinux shell]#
```


func2还是能够正常工作，但是就覆盖了func1的内容，这就很容易产生错误了。

返回值

bash脚本在退出的时候会有状态码，前面已经和大家说过了。

函数结束后，也会有退出码，来看

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

func1() {
  echo "trying to display a non-existent file"
  ls -l chaoge666
}

echo "testing the function:"
func1
echo "The exit status is :$?"
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
testing the function:
trying to display a non-existent file
ls: 无法访问chaoge666: 没有那个文件或目录
The exit status is :2

[root@chaogelinux shell]# echo $?
0
```

这个脚本特点是，函数执行报错了，退出状态码是2，脚本应该会报错，但是最后一行命令，确实正确执行的，那么整个脚本的执行退出码，就是0，那这就是有隐藏问题的了。

##函数使用return

###使用$?

函数可以用return命令来制定特定的退出状态码。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

funcWithReturn(){
  read -p "Enter a value: " value
  read -p "Enter another value:" value2
  return $(($value*$value2))
}
funcWithReturn
echo "The final answer is $?"
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Enter a value: 8
Enter another value:7
```

```
The final answer is 56
```

函数返回值是在调用函数之后，通过`&?`来获得。

但是这种方式会有坑，记住

- 函数结束后会立即取返回值
- 退出状态码在0~255

如果在提取函数返回值之前执行了其他的命令，函数的返回值会丢失。

例如

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

funcWithReturn(){
  read -p "Enter a value: " value
  read -p "Enter another value:" value2
  return $(($value*$value2))
}
funcWithReturn
ls /tt
echo "The final answer is $?"
```

因此，记住，`$?`取得的是脚本最后一条命令的状态码。

还有就是超出0~255的范围。

使用返回值变量

可以吧函数的执行结果，保存到变量里。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell


function db1 {
  read -p "Enter a value: " value
  echo $[ $value * 2 ]
}
result=$(db1)
echo "The new value is $result"
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Enter a value: 5
The new value is 10
```

函数中处理变量

作用域指的就是变量可见的区域，函数里定义的变量和普通变量作用域不同，函数里的变量属于是局部的。

函数可以使用两种变量

- 全局变量
- 局部变量

全局变量

全局变量是在整个shell脚本里都有效的变量。可以在函数哪读取全局变量的值。

默认情况下，脚本里定义的任何变量都是全局变量，只要不是写在函数体内。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

db1() {
  value=$[ $value * 2 ]
}
read -p "Enter a value: " value
db1
echo "The new value is : $value"

[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Enter a value: 9
The new value is : 18
```


这里即使value是在函数外赋值，在db1函数执行后，也会计算value值，重新赋值。

局部变量

无需在函数中使用全局变量，函数内部使用的任何变量都可以被声明为局部变量，为了实现这一点，只需要加上local关键字即可。

local关键字保证了变量局限在函数中，即使函数外也有同名的变量，shell也会保持该变量的值是分离的。

```
[root@jenkins01 shell]# cat test.sh
#!/bin/bash
func1(){
  local temp=$[ $value + 5 ]
  result=$[ $temp * 2 ]
}
temp=4
value=6
func1
```

```
echo "The result is $result"
if [ $temp -gt $value ]
then
  echo "temp is larger"
else
  echo "temp is smaller"
fi
```

执行结果

```
[root@jenkins01 shell]# bash test.sh
The result is 22
temp is smaller
```

我们会发现，即使函数里的temp数值发生了变化，也不会影响到函数体外层的temp变量。