第6部分
Shell概述及其应用

2024年6月30日

6.3.2 变量的作用域

变量的作用域是指变量可以被引用的范围。根据变量的作用域来划分，Shell变量可以分为两类，即**本地变量**和**导出变量**(也可称为局部变量和全局变量)。

1. 本地变量

在一个Shell中定义的变量默认只在此Shell中才有意义，也就是说它们的作用是**局部**的。我们称这种变量为**本地变量**。本地变量只在本Shell中有定义，而在子Shell中是不存在的。

例6.15 本地变量的作用域:

```
$  dir=/home/jhm/memo
$  echo $dir
  /home/jhm/memo
$ bash                    #进入子Shell
$ echo $dir
                          (空串表示变量未定义)
$
```

2. 导出变量

导出变量的命令是export，格式为：

export 变量名[变量名…]

当Shell的一个子进程开始运行时，它继承了该Shell进程的全部导出变量。子进程可以修改继承来的变量的值，但修改只是对自己的变量副本进行，不影响父进程中的变量的值。

例6.16 导出变量与本地变量的使用：

```
$ name=Zhang; export name          #定义并导出变量name
$ title=Dr.; export title          #定义并导出变量title
$ greeting="Good morning"          #定义变量greeting
$ cat var_test
    name=Wang
    echo "$greeting $title $name!"
$ bash var_test                    #在子Shell中引用变量
  Dr. Wang!
$ echo "$greeting $title $name!"   #在本Shell中引用变量
  Good morning Dr. Zhang!
$
```

用命令export还可以将已导出的变量“收回”，使其变回为本地变量，不再为子进程可用。收回变量的命令格式是：

export –n 变量名

6.4 Shell表达式

Shell语言支持表达式计算。Shell表达式主要有两种形式，一是用于数值计算的**算术表达式**，其结果是**数值**；另一种是用于进行条件测试或判断的**逻辑表达式**，其结果是**真假值**。

6.4.1 数字运算表达式

与高级语言中的变量不同，Shell变量只有字符类型，它们只能存放整数数字字符串，如“127”等。Shell本身也没有数字运算的能力，必须借助某些命令来进行算术运算。expr就是用来进行数字表达式计算的命令。

expr命令

【功能】计算表达式。

【格式】expr 数值1 运算符 数值2

【参数】expr支持以下运算符：

+、－、*、/、%加、减、乘、除、取余。

&、|　　逻辑与、逻辑或。

=、==、!=　　等于、恒等于、不等于。

>、<、>=、<=　　大于、小于、大于等于、小于等于。

【输出】+、-、*、/、%运算输出结果数值；&运算当两个数值都非0时输出第1个数值，否则输出0；|运算当第1个数值非0时输出第1个数，否则输出第2个数；比较运算为真时输出1，否则输出0。

【退出状态】算术运算返回状态0；逻辑和比较运算结果为真(非0)时返回状态0，为假(0)时返回状态1；出错时返回状态2。

【说明】

(1) 运算符两侧必须留有空格(赋值符号)，与运算数分开。

(2) 算术运算的数值必须是整数，可以是数字字符串常量(如“123”)，也可以是数字字符串变量(如$a)。expr命令负责将数字字符串解释为整数，然后进行运算。

(3) 运算符如果是Shell的元字符，如*、&、|、>、<等，必须用转义符‘\’使其失去特殊意义，不被Shell解释执行。

例6.14 expr命令用法示例：

```
$ a=13
$ expr “$a” - 4 + 2        # 13-4+2，注意运算符两旁要留有空格
   11
$ expr 4 \* 5              # 4*5
   20
$ expr 5 + 7 / 3           # 5+7/3，/运算优先于+运算，结果取整
   7
$ expr `expr 5 + 7` / 3    # (5+7)/3，用命令替换改变运算次序
   4
$ expr $a \>= 3;     echo $?          # 13 >= 3?
   1
   0
```

```
$ expr 5 \& “$a”; echo $?                    # 5 and 13
   5
   0
$ expr 5 \& 0; echo $?                       # 5 and 0
   0
   1
$ expr 5 \| $a; echo $?                      # 5 or 13
   5
   0
$ a=`expr $a - 2`                            #替换后为a=11
$ echo $a
   11
```

let命令，用于整数运算

```
#!/usr/bin/env bash
a=2
echo "a init is $a"
let "a+=1"
echo "a+=1 is $a"
let "a-=1"
echo "a-=1 is $a"
```

运行结果：

```
a init is 2
a+=1 is 3
a-=1 is 2
```

6.4.2 逻辑测试表达式

test命令可对字符串、整数及文件进行各类测试。test命令并不输出任何结果，而是用退出状态来表示测试的结果：退出状态为0时表示test成功，测试结果为真；退出状态为1表示test失败，测试结果为假。test的测试结果用于在控制结构中进行条件判断。

test命令使用的表达式形式见表6-8。

表 6-8 test 命令的常用表达式

测试类型	表达式	测试含义
字符串测试	*str1* = *str2*、*str1* != *str2*	*str1* 与 *str2* 是否(相等、不等)
	str1 > *str2*、*str1* < *str2*	按字典顺序，*str1* 是否(后于、前于)*str2*
	-n *str*、**-z** *str*	*str* 长度是否(不为 0、为 0)
	str	*str* 是否非空，同**-n** *str*
整数测试	*n1* **-eq** *n2*、*n1* **-ne** *n2*	*n1* 与 *n2* 是否(相等、不等)
	n1 **-gt** *n2*、*n1* **-lt** *n2*	*n1* 是否(大于、小于)*n2*
	n1 **-ge** *n2*、*n1* **-le** *n2*	*n1* 是否(大于等于、小于等于)*n2*
文件测试	**-d** *file*、**-f** *file*	*file* 是否存在并且是(目录、普通文件)
	-r *file*、**-w** *file*、**-x** *file*	*file* 是否存在并且(可读、可写、可执行)
	-e *file*、**-s** *file*	*file* 是否(存在、存在并且长度大于 0)
	file1 **-nt** *file2*、*file1* **-ot** *file2*	*file1* 是否(新于、老于)*file2*
逻辑测试	! *exp*	*exp* 是否为假
	(*exp*)	*exp* 是否为真
	exp1 **-a** *exp2*、*exp1* **-o** *exp2*	*exp1*(与、或)*exp2* 是否为真

test命令

【功能】测试表达式的真假值。

【格式】　test 表达式

【参数】表达式的常用形式见表6-8。

【退出状态】测试结果为真时返回状态0；为假时返回状态1；出错时返回状态2。

【说明】

(1) 运算符两侧必须留有空格，与运算数分开。

(2) 表达式中若使用Shell的元字符，如>、<、(、)等，必须用转义符“\”使其失去特殊意义，不被Shell解释执行。

1. 字符串测试

例6.17 字符串测试：

```
$ user=smith
$ test “$user” = smith                    #两字符串相等？
$ echo $?                                 #显示测试结果
   0                                      (是)
$ test -z “$user”                         #字符串为空串？
$ echo $?
   1                                      (否)
```

例6.16 含有空格的字符串及空字符串的测试：

```
$ user1="Tom Smith"
$ test "$user1" = "Tom Smith"
                              # test "Tom Smith" = "Tom Smith"
$ echo $?
   0                                          (真)
$ test $user1 = "Tom Smith"     # test Tom Smith =
"Tom Smith"
    bash: test: too many arguments
$ echo $?
   2                                          (出错)
$ test "$user2" = smith          # test "" = smith
$ echo $?
   1                                          (假)
```

```
$ test $user2 = smith                    # test  = smith
    bash: test: =: unary operator expected
$ echo $?
    2                                    (出错)
$
```

当表达式中使用变量时，最好将其用双引号括起(如“$var”)。这样，Shell进行变量替换后的字符串被双引号括起，作为一个单一的字符串传递给test命令。上例中，变量user1的字符串中含有空格，“$user1”被替换为一个带有空格的字符串“Tom Smith”，而$user1替换后成为两个字符串Tom和Smith。变量user2为空，“$user2”被替换为一个空串“”，而$user2则被替换为空。所以，空串或带空格的串在没有双引号的情况下都会导致test报错。

2. 数字测试表达式

例6.17 数字比较测试：

```
$ x1=5
$ test "$x1" -eq 5; echo $?          # test "5" = 5
   0                                 (真)
```

3. 文件测试表达式

例6.18 文件测试：

```
$ test -f /home/cherry/message; echo $?    #检查指定的文件是否存在
      0                                     (真)
$ gcc myprog.c 2>err                        #编译，错误信息记入err文件
$ test -s err && echo “Errors found”  #检查编译错误文件是否存在
```

#test命令测试为真时，表示有编译错误发生，继续执行echo命令；否则不执行。

6.5 Shell控制结构

与C语言的控制结构语句类似，Shell提供了几个专门的内部命令来构造控制结构，用它们可以构造出任意的分支与循环。本章主要讲解以下几类：

分支结构：if、case。

循环结构：while、for。

循环控制：break、continue。

结束：exit。

6.5.1 条件与条件命令

控制结构需要根据某个条件作出控制转向的决策。在Shell语言中，条件是某命令的退出状态。当命令执行成功时，它返回一个0状态(即$?为0)，此时条件为真；若命令失败，返回一个非0状态(即$?不为0)，则此时条件为假。

用于进行条件判断的命令就称为条件命令。任何shell命令都具有退出状态，因而都可以作为条件命令使用。此外，Shell还提供了3个内部命令：:、true和false。它们不作任何操作，只是返回一个特定的退出状态。:和true返回0；false返回非1。它们可作为恒真和恒假条件命令使用。

6.5.2 分支控制命令

分支控制命令用于控制程序在不同的条件取值下执行不同的流程。用于分支控制的命令有if和case，if命令用于两路分支控制，case命令用于多路分支控制。

1. if命令

if命令根据条件命令执行的结果决定后续命令的执行路径。

if命令的一般形式为：

```
if 条件命令
   then 命令列表1          #若$?为0，执行此分支
   [ else 命令列表2 ]              #否则，执行此分支
fi
```

例6.19 判断变量的大小：

设有变量a=10,b=12。用以下程序来判断a和b的大小。

```
if [ "$a" –lt "$b" ]
then echo "YES, a is less than b"
else echo "NO, a is greater than b"
 fi
```

例6.20 读入姓名变量name，显示输入的内容。

```
Echo –n “enter your name.”
read name
if  [  “$name” == “”  ]
then  echo  “you did not enter any information.”
else  echo  “your name is $name.”
 fi
```

2. case命令

用case命令进行多路条件测试结构更清晰。case命令的格式为：

```
case 测试字符串 in
    模式1）命令列表1;;
    模式2）命令列表2;;
    ...
    模式n）命令列表n;;
esac
```

case命令的执行过程是：先将测试字符串与各个模式字符串逐一比较，若发现了一个匹配的模式则执行该模式对应的命令列表。注意：若有多个匹配的模式时，只执行最前面的那一个分支。

例6.21 根据参数值进行相应处理：

```
echo –n “Enter a number from 1 to 3:”
read ANS
case $ANS in
   1) echo “you select 1:” ;;
   2) echo “you select 2:” ;;
   3) echo “you select 3:” ;;
   *) echo “This is not between 1 to 3” ;;
esac
```

6.5.3 循环控制命令

循环控制命令用于重复执行某个处理过程。Shell提供了3种循环控制结构，即for、while、until结构。它们的结构控制含义与C语言中循环语句相同，只是形式上有所不同。另外，这些循环控制命令既可以用在脚本程序中作为循环控制语句，也可以以多行命令的方式在Shell下直接执行。

1. for命令

for命令常用于简单的、确定的循环处理。for命令的格式为：

```
for变量 [ in 字符串列表 ]
do
    命令列表
done
```

for命令的执行过程是：定义一个变量，它依次取字符串列表中的各个字符串的值。对每次取值都执行命令列表，直到所有的字符串都处理完。

例6.22 for循环简例1：

```
#！ /bin/bash
#forlist1
for loop in 1 2 3 4 5
do
  echo $loop
done
```

例6.23 for循环简例2：

```
#！ /bin/bash
#forlist2
for loop in  orange red blue grey
do
  echo $loop
done
```

2. while命令

while命令的作用是进行有条件的循环控制，当条件为真时执行循环体命令列表，直到条件为假时结束。while命令常用于循环次数或循环处理的对象不够明确的循环过程。

while命令的格式为：

```
while 条件命令
do
    命令列表
done
```

例6.24 while简例1：

```
#!/bin/bash
 myvar=1
 while  [ $myvar -le 10 ]
do
    echo $myvar
    myvar=$(( $myvar + 1 ))
done
```

例6.25 while简例2：

```
while read LINE
 do
    echo “$LINE”
 done <myfile
```

#read通过输入重定向，把file的所有内容按行赋值给变量line，并输入出。

6.5.4 退出循环命令

break和continue命令的作用与C语言中的break和continue语句相同，用于在必要时跳出循环。break用于终止整个循环。当执行break命令时，控制转移到循环体后(done之后)的命令执行。continue用于终止本轮循环，直接进入下一轮循环执行。当执行continue命令时，控制转移到循环体开始处(do之后)的命令执行。

另外，break和continue命令只能应用在for、while、until命令的循环体内。

例6.26 用break命令终止循环：

```
$ cat break_test
    while :                 # always true
    do
        echo “Continue? [y/n]: ”
        read replay
        if [ $replay = n ]
          then break
        fi
    done
```

6.5.5 退出命令

exit命令是Shell的退出命令，用在Shell脚本中时表示退出脚本程序，返回到其父Shell。exit命令的格式是：exit [退出码]，其中退出码是返回给父进程的退出状态值。如果程序中没有显式地使用exit命令退出，则脚本的退出状态将是其退出前执行的最后一条命令的退出状态。

6.6 Shell函数

函数是shell程序中执行特殊过程的部件，并在shell程序中可以重复调用。

1. 函数定义

Shell函数的基本格式为：

```
function name（）
  commands
```

2. 脚本中函数调用

通常将函数看成是脚本中一段代码，在使用函数之前必须先定义该函数，使用时利用函数名直接调用。

例6.28 首先创建函数，然后调用函数。

```
$cat funcexer
#In this script defining function abc ( ) and using abc( )
abc( )
{
  echo”This is the function abc”
}
echo”The abc function begin”
abc
echo”The abc function end”
##funcexer end
$
```

运行结果：

```
The abc function begin
This is the function abc
The abc function end
$
```