**1 #！**

是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种shell，echo命令用于向窗口输出文本

运行shell有两种方法：

1）作为可执行程序



将上面的代码保存为test.sh，并cd到相应目录，执行脚本./test.sh，而不是test.sh。linux系统会去PATH中寻找有没有叫test.sh的，而只有/bin,/sbin,/usr/bin,/usr/sbin等在PATH里，你的当前目录通常不在PATH里，所有写成test.sh是会找不到命令的，要用./test.sh告诉系统就在当前目录。



2）作为解释器

这种运行方式是，直接运行解释器，其参数就是shell脚本的文件名，如：



这种方式运行的脚本，不需要再第一行解释器信息，写了也没用

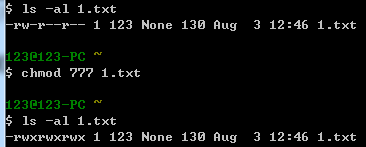
实际上就是两种方式：

1）交互式：解释执行用户的命令，用户输入一条命令，shell就解释执行一条

2）批处理：用户事先写一个shell脚本，其中有很多条命令，让shell一次把这些命令执行完，而不必一条条地敲命令

shell脚本和编程语言很相似，也有变量和流程控制语句，但shell脚本是解释执行，不需要编译，shell程序从脚本中一行一行读取并执行这些命令，相当于一个用户把脚本中的命令一行一行敲到shell提示符下执行

**2 修改文本的权限**



读写执行权限

**3 shell变量**

定义变量时，变量名不加美元符号（$，PHP语言中变量需要）如：

your\_name="runodb.com"

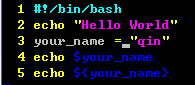
注意：变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样，同时，变量名的命名必须遵循如下规则：

（1）首个字符必须为（a-z、A-Z）

（2）中间不能有空格，可以使用下划线

（3）不能使用标点符号

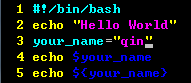
（4）不能使用bash中的关键字（可用help命令查看保留关键字）



上面的代码中在your\_name中添加了空格



修改为：



上面的代码中在your\_name中没有添加空格



使用一个定义过的变量，只要在变量名前面加美元符号即可，变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界。

变量可以被重新赋值



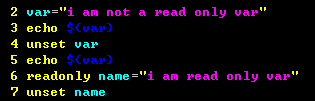
变量可以被设置为只读变量，只读变量不能被修改



显示结果为：



变量可以被删除，使用unset，unset命令不能删除只读变量：



运行shell时，会同时存在三种变量

1）局部变量：局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动

的程序不能访问局部变量

2）环境变量：所有程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变

量来正常运行，必要的时候shell脚本也可以定义环境变量

3）shell变量：shell变量是由shell程序设置的特殊变量，shell变量中有一部分是环

境变量，有一部分是局部变量，这些变量保证的shell的正常运行

shell中的字符串可以用单引号，也可以用双引号，也可以不用引号

单引号字符串的限制：

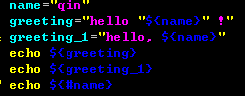
1）单引号里的任何字符都会原样输出，单引号字符串中的变量时无效的

2）单引号字符串中不能出现单引号（对单引号使用转义符后也不行）

双引号的优点：

1）双引号里可以有变量

2）双引号可以出现转义字符



获取字符串的长度要用${#string}

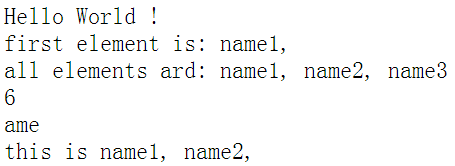
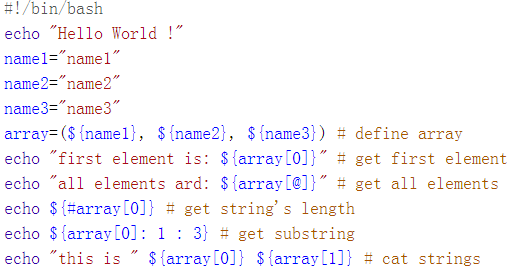
提取字符串${string:1:4}



shell数组

bash支持一维数组（不支持多维数组），并且没有限定数组大小

与c语言类似，数组的下标由0开始编号

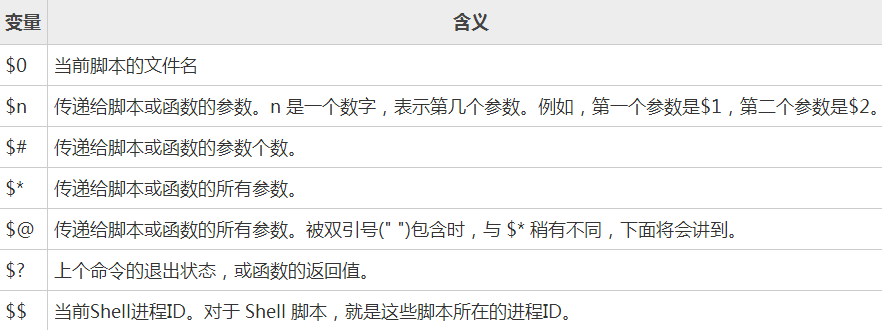


shell中以#开头的为注释，没有多行注释，只能每一行添加一个#号

如果再开发过程中，遇到大段的代码需要临时注释起来，过一会又要取消注释怎么办呢？

可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行了，达到了和注释一样的效果

Shell特殊变量



其中$\*和$@都表示传递给函数或脚本的所有参数，不被双引号包含时，都以"$1","$2"..."$n"的形式输出所有参数，但是当它们被双引号包含时，"$\*"会将所有的参数作为一个整体，以"$1 $2 ... $n"的形式输出所有参数，"$@"会将各个参数分开，以"$1","$2"..."$n"的形式输出所有参数

如果表达式中包含特殊字符，shell将会进行替换，例如，在双引号中使用变量是一种替换，转移字符也是一种替换：



这里的-e表示对转移字符进行替换，如果不使用-e选项，将会原样输出

下面的转义字符都可以用在echo中：



可以用echo命令的-E选项禁止转义，默认也是不转义的，使用-n选项可以禁止插入换行符



命令替换

命令替换是指shell可以先执行命令，将输出结果暂时保存，在适当的地方输出

命令替换的语法：

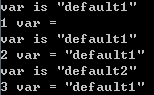
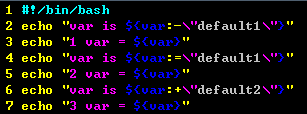
`command`

注意是反引号，不是单引号，这个键位于esc键下方



变量替换

变量替换可以根据变量的状态（是否为空、是否定义）来改变它的值



注意：在:-和:=中间不能出现空格

shell中的运算符

shell程序中的操作默认都是字符串的操作，如果要运行数学运算符的时候可能会得到意想不到的答案

(1)算数运算符

bash支持很多运算符，包括算数运算符，关系运算符，布尔运算符，字符串运算符和文件测试运算符

原生的bash不支持简单的数学运算，但是可以通过其他命令来实现，例如awk和expr，expr最常用。



有两点需要注意：

1）表达式和运算符之间要有空格，例如2+2是不对的，必须写成2 + 2

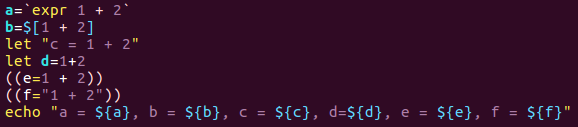
2）完整的表达式要被``包含，注意这个不是常用的单引号，在esc键下边

我们还可以有其他的几种方式

1）用let指示数学运算。可以先将运算结果赋值为变量b，运算命令是b=let 1 + 2

完后用echo${b}来输出b的值

2）用$[]表示数学运算，将一个数学运算写到$[]符号的中括号中，中括号中的内容将先进行数学运算，例如命令echo $[1 + 2]将输出结果3



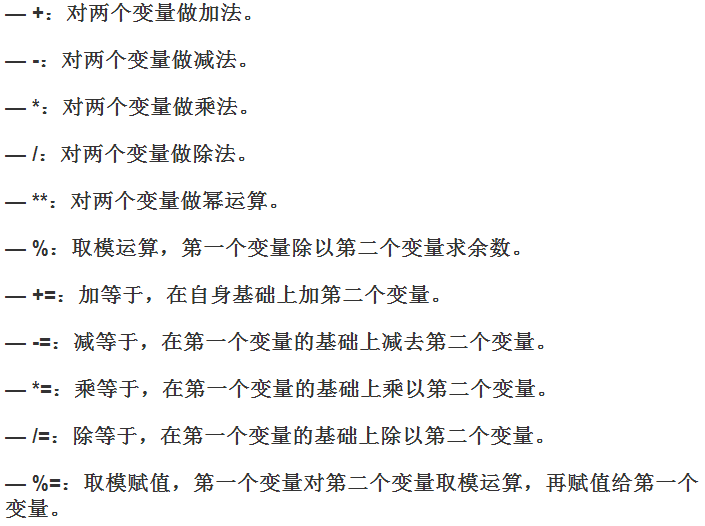
注意：let expressions执行一个或多个表达式，表达式中的变量不必有$，如果表达式中包含了空格或其他字符，则必须用引号，如上面：

let "c = 1 + 2"

let d=1+2

这其中的let可以用(())代替，let ″j=i\*6+2″等价于((j=i\*6+2)), 就像很多的循环中用法一样

**((i=$j+$k))    等价于 i=`expr $j + $k`  
((i=$j-$k))     等价于   i=`expr $j -$k`  
((i=$j\*$k))     等价于   i=`expr $j \\*$k`  
((i=$j/$k))     等价于   i=`expr $j /$k`**



http://www.cnblogs.com/yfanqiu/archive/2012/05/10/2494031.html

**shell程序中的操作默认都是字符串操作**，在要运行数学运算符的时候可能得到意想不到的答案：

var=1  
var=$var+1  
echo $var  
  
output：  
1+1

从这个例子中可以看出shell字符串连接的操作，需要注意的是空格。

在shell中可以使用let来指示下面是算术表达式，let表达式内变量不用加$

var=1  
let "var+=1" 或 let var+=1 这种写法运算符间不能有空格   
echo $var  
  
output：  
2

这其中的let可以用(())代替，let ″j=i\*6+2″等价于((j=i\*6+2)), 就像很多的循环中用法一样

注意：let必须是完整的算术表达式，即有等号两边

　　　(())、expr 可以只有等号右边的计算，由$((...))、$(expr ...)、`expr ...` 查看返回结果

var=1  
((var++)) 查看结果： echo $(())  
echo $var  
  
output:  
2

还有一种写法：

var=1  
var=$[$var+1]  
echo $var  
  
output：  
2

使用expr，需要注意的是其中的那个符号并不是单引号，而是“1”左边的那个键

注意：将需要运算的表达式写入在expr 后面，保证参数与运算符号中间有空格隔开。

var=1  
var=`expr $var + 1`  
echo $var  
  
output：  
2

**1:在shell 中$() 与 ``等效。执行中间包含的命令语句，返回执行结果。**  
**2:从效率来说let==(()) > expr > bc。let和(())运行是内建命令，使用相同的算法。**  
**3:let 和 expr 的运算是整数运算，不包括浮点预算。**  
**4:expr和bc是外部程序，expr的体积几乎等于bc的1/3，执行一次装入内存所消耗的时间就不一样。**  
**5:从运算能力来说，bc排第一位。**

上面的这几种做法都是只能处理整数运算，下面的这种做法是可以处理浮点数的：

var=1  
var=`echo "scale=1;$var\*2.0"|bc`  **bc预设输出整数，用scale 指定小数点下位数**  
echo $var  
  
output：  
2.0

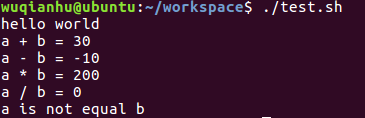
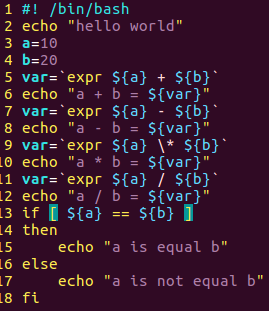
上面的`符号跟上面的一样，是1左边的那个键，而bc是linux上面的一个计算器，支持除位运算以外的所有运行。

另外一个支持浮点运算的方法是awk：

var=1  
var=`echo "$var 1"|awk '{printf("%g",$1+$2)}'`  
echo $var  
  
output：  
2

$ c=$(awk '**BEGIN**{print 7.01\*5-4.01 }')

$ echo $c



注意：

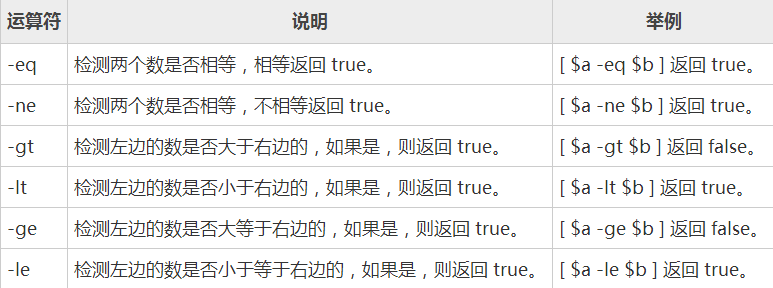
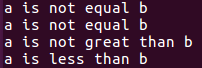
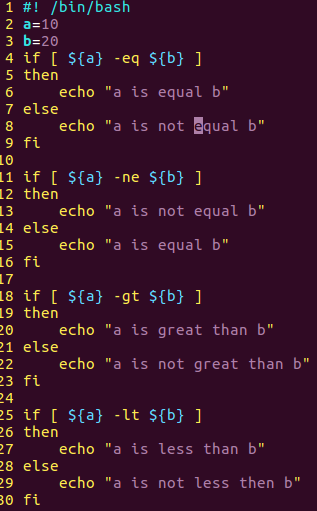
1）乘号前边必须加反斜杠才能实现乘法运算

2）if...then...else...fi语句中的条件表达式要在方括号中，并且要有空格，例如[${A}==${B}]是错误的，必须写成[ ${A} == ${B} ]

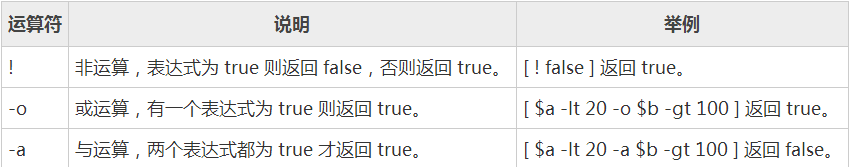
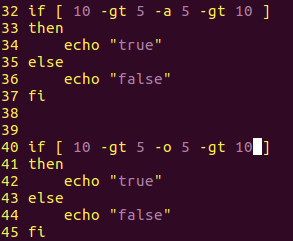


（2）关系运算符

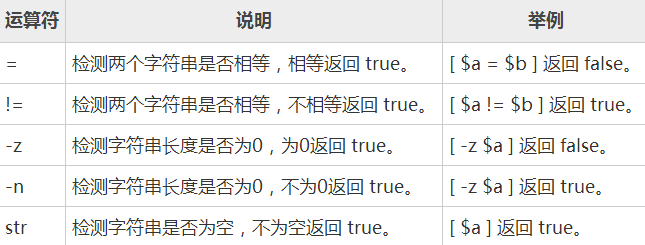
关系运算符只支持数字，不支持字符串，除非字符串的值是数字



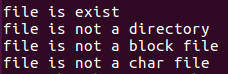
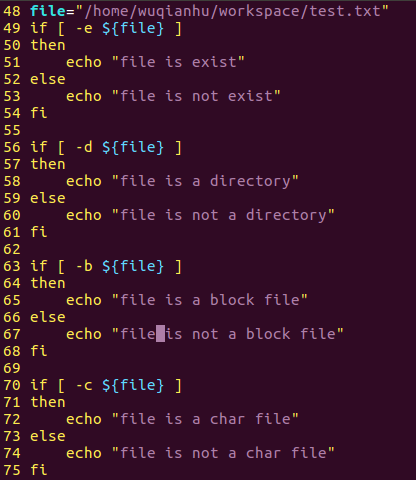
（3）布尔运算符



（4）字符串运算



（5）文件测试运算符



for循环

for循环一般格式为：

for 变量 in 列表

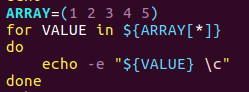
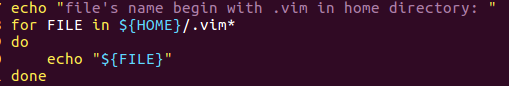
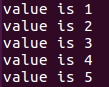
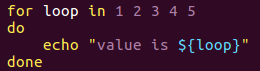
do

command1 ...

done

列表是一组值（数字、字符串等）组成的序列，每个只通过空格分隔，每循环一次，就将列表的下一个值赋给变量。

in是可选的，如果不用它，for循环使用命令行的位置参数



while循环

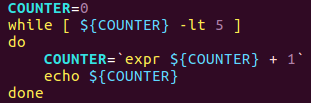
while循环用于不断执行一系列命令，也用于从输入文件中读取数据，命令通常为测试条件

while command

do

Statement(s) to be executed if comand is true

done



until循环

until循环执行一系列命令直至条件为true时停止，until循环与while循环在处理方式上刚好相反，一般while循环优于until，但在某些时候，也只是极少数情况下，until循环更加有用。

until循环格式一般为：

until command

do

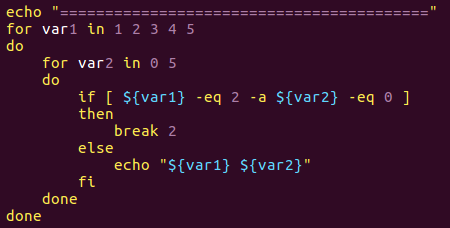
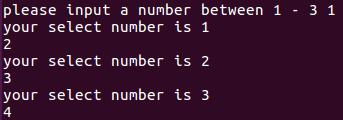
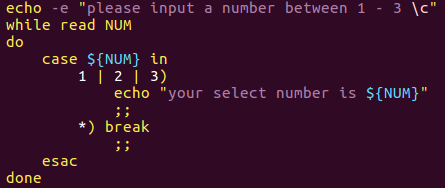
$statement(s) to be executed until command is true

done

跳出循环

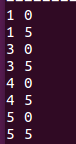
break与continue

其中break和continue后面可以跟一个数字，表示跳出第几层循环



break 2 表示跳出外层循环

如果修改为break 1则结果为：



结果重定向到文件

echo "It is a test file" > file.txt

**如果需要原样输出字符串，请用单引号**



**printf命令**

printf命令用于格式化输出，是echo命令的增强版，它是c语言printf库函数的一个有限的变形，并且在语法上有些不同

printf由POSIX标准定义，移植性比echo要好

同echo命令一样，printf命令也可以输出简单的字符串，但是它不会自动换行，必须显式添加换行符（\n）



printf命令的语法：

printf format-string [arguments...]

基本语法和c语言类似，不同点是：

1）printf命令不用加括号

2）format-string可以没有引号，但最好加上，单引号和双引号都可以

3）参数多余格式控制符（%）时,format-string可以重用，可以将所有参数都转换

4）arguments使用空格分隔，不用逗号

**条件判断语句if then ... elif then ... else ... fi**

if []

then

echo...

elif []

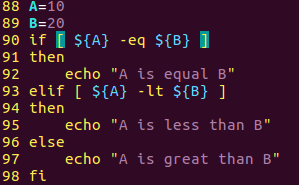
then

echo...

else

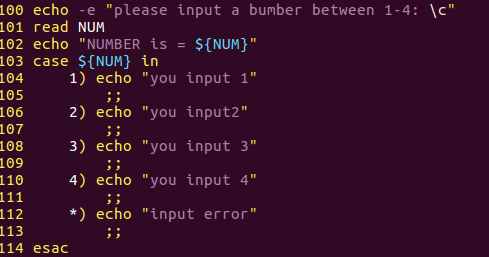
echo...

fi

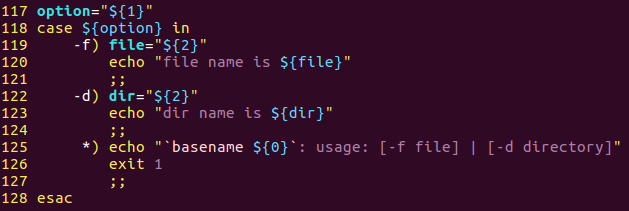


**case...esac语句**

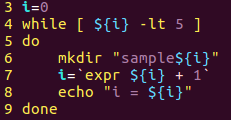
该语句和其他语言的switch...case类似，是一种多分枝选择结构



注意，上面的case语句中）相当于其他语言的：所以不要再写：了， ；；相当于break，\*）相当于default



如果将默认的修改为：



函数

函数定义格式为：

function\_name(){

list of commands

[ return value ]

}

如果愿意可以在函数名前添加关键字function

function function\_name(){

list of commands

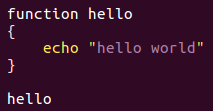
[ return value ]

}

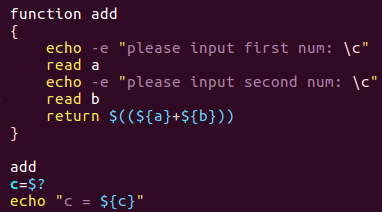
函数返回值，可以显式添加return语句；如果不加，会将最后一条命令运行结果作为返回值

shell函数返回值只能是整数，一般用来表示函数执行成功与否，0表示成功，其他值表示失败，如果return其他数据没比如一个字符串，往往会得到错误提示：“numeric argument required”

如果一定要让函数返回字符串，那么可以先定义一个变量，用来接收函数的计算结果，脚本在需要的时候访问这个便来个来获得函数的返回值



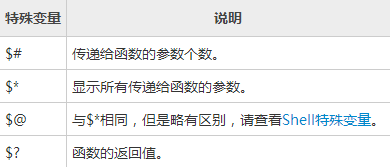
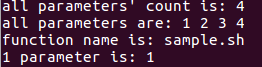
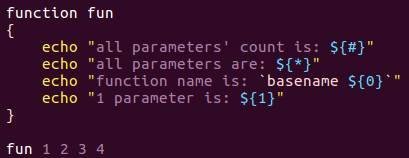
调用函数只需要给出函数名，不需要加括号



函数的返回值要在调用了add函数之后获取，如果采用c=add，最终打印出来的就是c = add

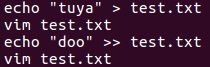
函数参数

在shell中，调用函数可以向其传递参数，在函数体内部，通过$n的形式来获取参数的值，$0表示函数本身，$1表示第一个参数，$2表示第二个参数...



输入输出重定向

1）输出重定向



在上面的两条命令中，第一条命令是覆盖，第二条命令是追加

2）输入重定向

和输出重定向一样，unix命令也可以从文件中获取输入，语法为：

command < file

这样，本来需要从键盘获取输入的命令会转移到文件读取内容

比如获取test.txt文件中的行数可以使用：



注意上面两个例子的结果不同：第一个例子，会输出文件名；第二个不会，因为它仅仅知道从标准输入读取内容

文件包含

像其他语言一样，shell也可以包含外部脚本，将外部脚本的内容合并到当前脚本中

shell包含脚本可以使用

. filename

或

source filename

两种方式的效果相同，简单起见，一般使用点号(.)，但是注意点号和文件名中间有一空格

file1.sh



file2.sh



注意，执行的脚本要有执行权限，被包含的脚本不需要有执行权限

chmod +x filename

调用环境变量：

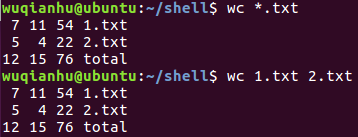
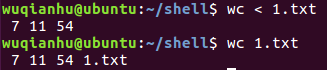


**输出重定向**

>覆盖方式

>>追加方式

**输入重定向**



还有另外一种输入重定向的方法，称为内联输入重定向。这种方法允许你在命令行而不是在文件制定输入重定向到的数据。

内联输入重定向符号是<<，除了这个符号，你必须指定一个文件标记来划分要输入数据的开始和结尾，你可以用任何字符串的值来作为文本标记，但在数据的开始和结尾必须一致

在命令行中使用内联输入重定向时，shell会用PS2环境变量中定义的提示符来提示输入数据，次提示符会一直提示输入更多数据，直到你输入了作为文本标记的那个字符串值。

