Shell是一种脚本语言，那么，就必须有解释器来执行这些脚本，常见的脚本解释器有：

**bash：**是Linux标准默认的shell。bash由Brian Fox和Chet Ramey共同完成，是BourneAgain Shell的缩写，内部命令一共有40个。

**sh：** 由Steve Bourne开发，是Bourne Shell的缩写，sh 是Unix 标准默认的shell。

另外还有：ash、 csh、 ksh等。

常见的编程语言分为两类：一个是编译型语言，如：c/c++/java等，它们远行前全部一起要经过编译器的编译。另一个解释型语言，执行时，需要使用解释器一行一行地转换为代码，如：awk, perl, python与shell等。

Shell 经过了POSIX的标准化，所以它是可以在不同的linux系统上进行移植。

关于注释的问题： 在shell中使用#进行注释，注意，sh里面没有多行注释，只能每一行加一个#号；

**第一个shell脚本程序：**

#!/bin/bash

# 上面中的 #! 是一种约定标记, 它可以告诉系统这个脚本需要什么样的解释器来执行;

echo "Hello, world!"

### 变量:

**定义变量:**

country="China"

Number=100

注意: 1,变量名和等号之间不能有空格;

2,首个字符必须为字母（a-z，A-Z）。

3, 中间不能有空格，可以使用下划线（\_）。

4, 不能使用标点符号。

5, 不能使用bash里的关键字（可用help命令查看保留关键字）。

**使用变量:**

只需要在一个定义过的变量前面加上美元符号 $ 就可以了, 另外,对于变量的{} 是可以选择的, 它的目的为帮助解释器识别变量的边界.

country="China"

echo $country

echo ${country}

echo "I love my ${country}abcd!"

#这个需要有｛｝的；

**重定义变量：** 直接把变量重新像开始定义的那样子赋值就可以了：

country="China"

country="ribenguizi"

**只读变量**: 用 readonly 命令 可以把变量字义为只读变量。

readonly country="China"

#或

country="China"

readonly country

删除变量: 使用unset命令可以删除变量，但是不能删除只读的变量。用法：

unset variable\_name

#### 变量类型

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。

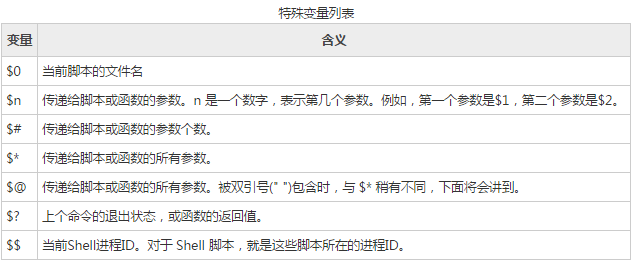
###### 2) 环境变量

所有的程序，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保证其正常运行。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，这些变量保证了shell的正常运行

**特殊变量:**



$\* 和 $@ 的区别为: $\* 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。

$? 可以获取上一个命令的退出状态。所谓退出状态，就是上一个命令执行后的返回结果。退出状态是一个数字，一般情况下，大部分命令执行成功会返回 0，失败返回 1。

### Shell中的替换

**转义符：**

在echo中可以用于的转义符有：



使用 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；

使用 echo 命令的 –e 选项可以对转义字符进行替换。

另外，注意，经过我的实验，得到：

echo "\\" #得到 \

echo -e "\\" #得到 \

echo "\\\\" #得到 \\

echo -e "\\" #得到 \

**命令替换:**

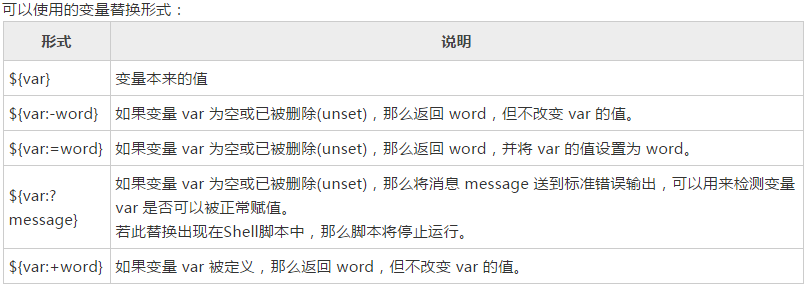
它的意思就是说我们把一个命令的输出赋值给一个变量,方法为把命令用反引号(在Esc下方)引起来.  比如:

directory=`pwd`

echo $directory

**变量替换**:

可以根据变量的状态（是否为空、是否定义等）来改变它的值.



### Shell运算符

**算数运算符:**

原生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. 下面使用expr进行；  expr是一款表达式计算工具，使用它可以完成表达式的求值操作；



比如：

a=10

b=20

expr $a + $b

expr $a - $b

expr $a \\* $b

expr $a / $b

expr $a % $b

a=$b

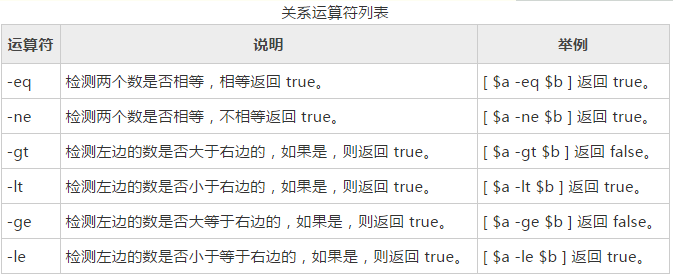
注意: 1. 在expr中的乖号为：\\*

2. 在 expr中的 表达式与运算符之间要有空格，否则错误；

3. 在[ $a == $b ]与[ $a != $b ]中，要需要在方括号与变量以及变量与运算符之间也需要有括号， 否则为错误的。（亲测过）

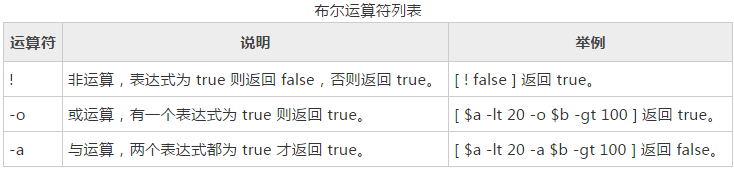
**关系运算符：**

只支持数字，不支持字符串，除非字符串的值是数字。常见的有：

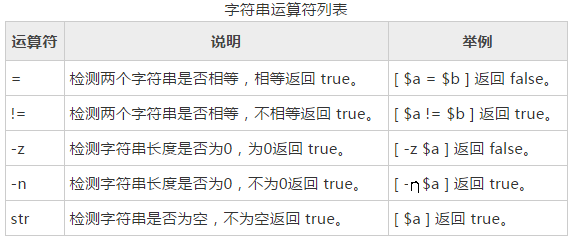


注意：也别忘记了空格；

**布尔运算符：**



**字符串运算符：**



**文件测试运算符:**

检测 Unix 文件的各种属性。



### Shell中的字符串

**单引号的限制：**

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是无效的；
2. 单引号字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. 双引号里可以出现转义字符

#### 拼接字符串：

country="China"

echo "hello, $country"

#也可以

echo "hello, "$country" "

#### 获取字符串长度:

string="abcd"

echo ${#string} #输出 4

#### 提取子字符串:

string="alibaba is a great company"

echo ${string:1:4} #输出liba

**查找子字符串:**

string="alibaba is a great company"

echo `expr index "$string" is`

#### 处理路经的字符串：

例如：当一个路径为 /home/xiaoming/1.txt时，如何怎么它的路径（不带文件) 和如何得到它的文件名？？

得到文件名使用 bashname命令：

# 参数：

# -a,表示处理多个路径；

# -s, 用于去掉指定的文件的后缀名；

basename /home/yin/1.txt -> 1.txt

basename -a /home/yin/1.txt /home/zhai/2.sh ->

1.txt

2.sh basename -s .txt /home/yin/1.txt -> 1

basename /home/yin/1.txt .txt -> 1

得到路径名（不带文件名）使用 dirname命令：

参数：没有啥参数；

//例子：

dirname /usr/bin/ -> /usr

dirname dir1/str dir2/str ->

dir1

dir2

dirname stdio.h -> .

### Shell的数组:

bash支持一维数组, 不支持多维数组, 它的下标从0开始编号. 用下标[n] 获取数组元素；

**定义数组：**

在shell中用括号表示数组，元素用空格分开。 如：

array\_name=(value0 value1 value2 value3)

也可以单独定义数组的各个分量，可以不使用连续的下标，而且下标的范围没有限制。如：

array\_name[0]=value0

array\_name[1]=value1

array\_name[2]=value2

**读取数组：**

读取某个下标的元素一般格式为:

${array\_name[index]}

读取数组的全部元素，用@或\*

${array\_name[\*]}

${array\_name[@]}

**获取数组的信息：**

取得数组元素的个数：

length=${#array\_name[@]}

#或

length=${#array\_name[\*]}

获取数组的下标：

length=${!array\_name[@]}

#或

length=${!array\_name[\*]}

取得数组单个元素的长度:

lengthn=${#array\_name[n]}

**printf函数：**

它与c语言中的printf相似，不过也有不同，下面列出它的不同的地方：

1. printf 命令不用加括号
2. format-string 可以没有引号，但最好加上，单引号双引号均可。
3. 参数多于格式控制符(%)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。

下面为例子：

# format-string为双引号

$ printf "%d %s\n" 1 "abc"

1 abc

# 单引号与双引号效果一样

$ printf '%d %s\n' 1 "abc"

1 abc

# 没有引号也可以输出

$ printf %s abcdef

abcdef

# 格式只指定了一个参数，但多出的参数仍然会按照该格式输出，format-string 被重用

$ printf %s abc def

abcdef

$ printf "%s\n" abc def

abc

def

$ printf "%s %s %s\n" a b c d e f g h i j

a b c

d e f

g h i

j

# 如果没有 arguments，那么 %s 用NULL代替，%d 用 0 代替

$ printf "%s and %d \n"

and 0

# 如果以 %d 的格式来显示字符串，那么会有警告，提示无效的数字，此时默认置为 0

$ printf "The first program always prints'%s,%d\n'" Hello Shell

-bash: printf: Shell: invalid number

The first program always prints 'Hello,0'

$

### Shell中条件语句

#### if 语句

包括：1， if [ 表达式 ] then  语句  fi

2. if [ 表达式 ] then 语句 else 语句 fi

3.  if [ 表达式] then 语句  elif[ 表达式 ] then 语句 elif[ 表达式 ] then 语句   …… fi

例子：

a=10

b=20

if [ $a == $b ]

then

echo "a is equal to b"

else

echo "a is not equal to b"

fi

另外：if ... else 语句也可以写成一行，以命令的方式来运行，像这样：

if test $[2\*3] -eq $[1+5]; then echo 'The two numbers are equal!'; fi;

其中，test 命令用于检查某个条件是否成立，与方括号([ ])类似。

**case …… esac语句**

case ... esac 与其他语言中的 switch ... case 语句类似，是一种多分枝选择结构。case语句格式如下：

case 值 in

模式1)

command1

command2

command3

;;

模式2）

command1

command2

command3

;;

\*)

command1

command2

command3

;;

esac

其中， 1. 取值后面必须为关键字 in，每一模式必须以右括号结束。取值可以为变量或常数。匹配发现取值符合某一模式后，其间所有命令开始执行直至 ;;。;; 与其他语言中的 break 类似，意思是跳到整个 case 语句的最后。2. 如果无一匹配模式，使用星号 \* 捕获该值，再执行后面的命令。

**Shell 的循环语句**

**for 循环**

  一般格式为：

for 变量 in 列表

do

command1

command2

...

commandN

done

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

顺序输出当前列表的数字：

for loop in 1 2 3 4 5

do

echo "The value is: $loop"

done

显示主目录下以 .bash 开头的文件：

#!/bin/bash

for FILE in $HOME/.bash\*

do

echo $FILE

done

**while循环**

**一般格式为：**

while command

do

Statement(s) to be executed if command is true

done

例如：

COUNTER=0

while [ $COUNTER -lt 5 ]

do

COUNTER='expr $COUNTER+1'

echo $COUNTER

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。    常用格式为：

until command

do

Statement(s) to be executed until command is true

done

command 一般为条件表达式，如果返回值为 false，则继续执行循环体内的语句，否则跳出循环。

类似地， 在循环中使用 break 与continue 跳出循环。    另外，break 命令后面还可以跟一个整数，表示跳出第几层循环。

Shell函数

Shell函数必须先定义后使用，定义如下，

function\_name () {

list of commands

[ return value ]

}

也可以加上function关键字：

function function\_name () {

list of commands

[ return value ]

}

注意:1. 调用函数只需要给出函数名，不需要加括号。

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

3. 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。

4. 函数的参数可以通过 $n  得到.如:

funWithParam(){

echo "The value of the first parameter is $1 !"

echo "The value of the second parameter is $2 !"

echo "The value of the tenth parameter is ${10} !"

echo "The value of the eleventh parameter is ${11} !"

echo "The amount of the parameters is $# !" # 参数个数

echo "The string of the parameters is $\* !" # 传递给函数的所有参数

}

funWithParam 1 2 3 4 5 6 7 8 9 34 73

5. 像删除变量一样，删除函数也可以使用 unset 命令，不过要加上 .f 选项，如下所示：

unset .f function\_name

### shell的文件包含：

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。使用：

. filename

#或

source filename

1. 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。

2. 被包含脚本不需要有执行权限。

趣谈shell系列

http://swiftlet.net/archives/category/swiftlet-shell/funny-shell