\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*重要\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1 语法基本介绍  
  
1.1.1 开头  
程序必须以下面的行开始（必须放在文件的第一行）：  
#!/bin/sh  
　　符号#!用来告诉系统它后面的参数是用来执行该文件的程序。在这个例子中我们使用/bin/sh来执行程序。  
　　当编辑好脚本时，如果要执行该脚本，还必须使其可执行。  
　　要使脚本可执行：  
编译 chmod +x filename 这样才能用./filename 来运行  
  
1.1.2 注释  
  
　　在进行shell编程时，以#开头的句子表示注释，直到这一行的结束。

例子：varibleName="value"

首先有一点，变量名和等号之间不能有空格，这点千万注意

然后要知道的一些变量命名规则：

1，首个字符必须为字母（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。

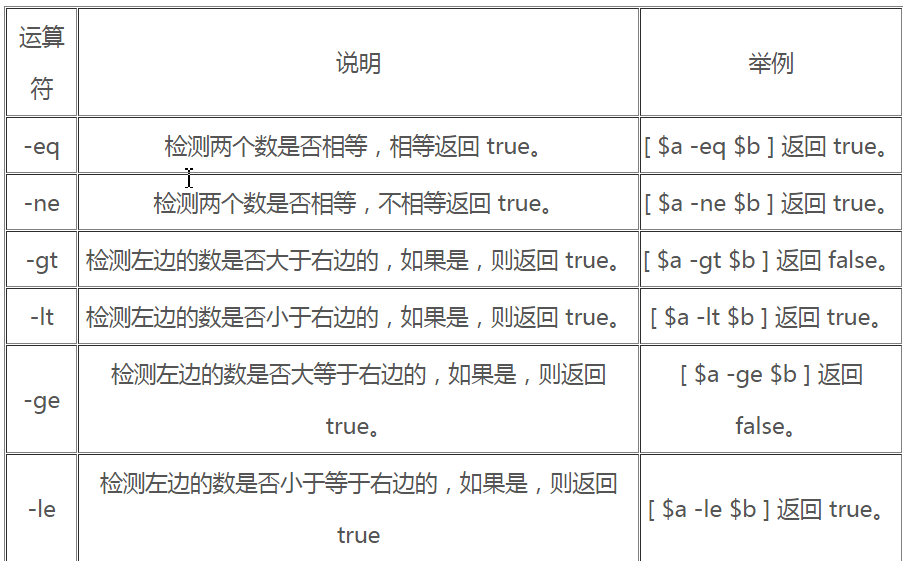
4，不能使用bash里的关键字（可用help命令查看保留关键字）。

定义好变量我们就要开始使用变量了，如何使用变量，需要在自己定义的变量名上加上大括号，然后最前面写上$字符，例如

who="dog"

echo who is ${who}?

其实变量可以直接用$who取，但是为了养成好习惯，还是加上括号吧。



**注:以上的运算符只能对数字比较使用，虽然shell里面的变量都是字符串**

**但是=可以用来比较数字的大小 < >好像不能用 比较数字的大小应该用 -gt**

**-lt 的方式**

**综上 比较字符串是否相等时 用 = 或者 == 不可以用 –eq(只能对数字比较使用)**

**而 比较数字大小的时候 用(-eq -gt -lt 当然=也可以)**

**布尔运算符**

! 非运算

-o 或运算(or)

-a 与运算(and)

获取**用户输入** read

read -p “请输入学号” –n 5 （只可以输入5个字符） -t 10 （10秒不输入自动结束输入） 变量名 (被赋予值得变量)

**------字符串**

shell中的字符串是shell中最常用的数据类型了（貌似除了他也没有其他类型了）好像**赋值的时候不论是数字还是字符串 shell都貌似会自动识别类型储存为字符串，但在比较的时候不能比较 。 可以这样理解吧 。。。 个人见解**

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；

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

双引号基本没有什么限制，一般常用双引号，双引号里面可以使用转义字符，可以带入变量

Expr a ? b 做算术计算的时候

+是不需要转义符的

-不需要转义符

\*是需要转义符的

/不需要转义符

((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`

**注：给前面的变量赋值的时候前面的变量不带$ 不然会报错**

例如 (($i=$j+$k)) X 这样是不对的

字符串的一些方法:

1. **获取字符串长度**的方法:

Eg:

#!/sh/bash

Str=”helloworld”

Echo **${#Str}**

输出为:10

(2)

Shell中的**数组**

**Array1[0]=0**

**Array[1]=1**

**Echo ${#array1[@]} -----输出结果为2 获取数组长度**

**Array1[2]=2**

**Array1[3]=3**

**Echo ${#array1[@]} ------------输出结果为4 获取数组长度**

**Echo ${array1[@]} --- 会直接输出整个数组 结果: 0 1 2 3**

**###遍历数组的方式**

**For haha in ${array1[@]}**

**do**

**echo ${haha}**

**done**

**貌似shell中的数组的长度是可扩展动态的，不像java与C是初始化时候写死长度的，由于shell中只有字符串数据类型，所以数组中可以一起存数字和字符串 (数字被识别为字符串了).**

**Case语句**

flag=3

case “$flag” in **统一一下 case后面的变量被””包括起来**

"1") ----带不带引号都行 shell中都是字符串

echo "1"**;; 注意:每一个case语句的结束 都需要两个分号**

2)

echo "2"**;;**

"3")

echo "3"**;;**

esac -----**结尾要有倒置的case匹配，即esac**

Shell中的**分支**

**if** [ expression ] ----如果此处then不换行 则[expression]后面需要＋ ; 号

**then**

**……**

**Elif** [ expression ]

**Then**

………

**Else**

……..

**fi**

**循环**

**for ((i=0;i<=100;i++))**

**do**

**…….**

**done**

**int i=0**

**while ((i<=100))**

**do**

**……**

**let i++**

**done**

**Shell函数**

**调用**

**sayHello(){**

**echo "hello there"**

**}**

**echo "begin"**

**sayHello -----调用函数 不像java一样需要函数之后的（）**

**传参数**

sayHello(){

echo $1

echo $2

}

echo "begin"

sayHello nihao lyf  **---这样就把两个参数(nihao lyf穿进去了)， 方法内调用$1 $2来访问参数**

**返回值**

**sayHello(){**

**echo "the args is $#"**

**echo $1**

**echo $2**

**echo "hello there"**

**return 0 ------返回值 ，脚本中可以使用$?调用返回值**

**}**

Shell中函数的值**只能返回整数**， **0代表成功**，**1**或者其他代表**失败**。所以，如果想通过调用函数操作返回值来做一些事会受到限制，我们**可以定义一个全局变量**，然后在**函数中操作变量**来达到我们的一些目的。

在shell脚本中定义的**变量都是全局变量** **除非**变量前**被local** 关键字**修饰**了。 比如说:在函数中定义的变量在C或者JAVA中都是局部变量，但是在shell编程中，她是全局的，在**函数外**也是**可以调用函数内部的**这个**变量**的。如果在函数中的变量加了 local修饰，则不同。

**文件操作符:**

操作符 说明

**-b** file 检测文件是否是块设备文件，如果是，则返回 true。

**-c** file 检测文件是否是字符设备文件，如果是，则返回 true

**-d** file 检测文件是否是目录，如果是，则返回 true。

**-f** file 检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。

**-g** file 检测文件是否设置了 SGID 位，如果是，则返回 true。

**-k** file 检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。

**-p** file 检测文件是否是具名管道，如果是，则返回 true。

**-u** file 检测文件是否设置了 SUID 位，如果是，则返回 true。

**-r** file 检测文件是否可读，如果是，则返回 true。

**-w** file 检测文件是否可写，如果是，则返回 true。

**-x** file 检测文件是否可执行，如果是，则返回 true。

**-s** file 检测文件是否为空（文件大小是否大于0），不为空返回 true。

**-e** file 检测文件（包括目录）是否存在，如果是，则返回 true。

shell中**特殊的变量**：

|  |  |
| --- | --- |
|  |  |
| **$0** | 当前脚本的文件名 |
| **$n** | 传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是$1，第二个参数是$2。 |
| **$#** | 传递给脚本或函数的参数个数。 |
| $\* | 传递给脚本或函数的所有参数。 |
| $@ | 传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $\* 稍有不同，下面将会讲到。 |
| **$?** | **上个**命令的退出状态，或函数的返回值。 |
| $$ | 当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID。 |

**给脚本传参数的时候 只要在 ./shellname arg1 arg2 arg3**

**然后在脚本里面直接调用就好了 $1 $2 $\* 等等**

**如果遇到了某些情况,想要正常退出shell脚本的话 exit**

**输入、输出重定向**

**>>为文件尾追加 >为覆盖**

每个 Unix/Linux 命令运行时都会打开三个文件：

**标准输入文件**(stdin)：stdin的文件描述符为0，Unix程序默认从stdin读取数据。

**标准输出文件**(stdout)：stdout 的文件描述符为1，Unix程序默认向stdout输出数据。

**标准错误文件**(stderr)：stderr的文件描述符为2，Unix程序会向stderr流中写入错误信息。

标准输入，标准输出，标准错误输出是什么呢？我们可以理解为：**键盘是标准输入，显示器是标准输出，错误输出默认仍然是显示器。**

shell的重定向用于将标准输出或标准错误输出重定向到其他位置。

比如说将本来要输出在显示器上的东西，定向到其他位置了，比如文件中

输出重定向的语法为：指令 > 文件名

我们举个例子，我们在终端执行命令 ls > test，这时候我们在屏幕上看不到输出，本来ls就是现实当前目录下的文件，这时候我们已经把这些输出重定向到一个叫test的文件中了。我们通过vim test打开，可以看到结果。确实结果输出到test文件中了。

输出重定向还有一个格式：指令 >> 文件名

这个格式会保证 若已存在文件，则结果追加到输出到文件末尾，即不会覆盖。

输入重定向的语法：指令 < 文件名

例子：比如我们有一个文件test.txt,里面有3行内容。我们要统计他的行数，可以通过执行命令：wc -l < test.txt来统计

结果输出：3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2) 概念: 管道, 重定向和 backtick  
  
　　这些不是系统命令，但是他们真的很重要。  
　　管道 (|) 将一个命令的输出作为另外一个命令的输入。  
grep "hello" file.txt | wc -l  
　　在file.txt中搜索包含有”hello”的行并计算其行数。  
　　在这里grep命令的输出作为wc命令的输入。当然您可以使用多个命令。  
　　重定向：将命令的结果输出到文件，而不是标准输出（屏幕）。  
　　> 写入文件并覆盖旧文件  
　　>> 加到文件的尾部，保留旧文件内容。  
反短斜线  
　使用反短斜线可以将一个命令的输出作为另外一个命令的一个命令行参数。  
命令：  
find . -mtime -1 -type f -print  
　　用来查找过去24小时（-mtime –2则表示过去48小时）内修改过的文件。如果您  
想将所有查找到的文件打一个包，则可以使用以下脚本：  
#!/bin/sh  
# The ticks are backticks (`) not normal quotes ('):  
tar -zcvf lastmod.tar.gz `find . -mtime -1 -type f -print`  
for-loop表达式查看一个字符串列表 (字符串用空格分隔) 然后将其赋给一个变量：  
for var in ....; do  
　 ....  
done  
在下面的例子中，将分别打印ABC到屏幕上：

复制代码代码如下:

#!/bin/sh  
for var in A B C ; do  
　 echo "var is $var"  
done