**用法一**：**显示脚本参数（$0、$?、$\*、$@、$#、$$、$!）（本质上属于变量替换）**

$0：就是该bash文件名,个位数的，可直接使用数字，但两位数以上，则必须使用 {} 符号来括住，如${10}.

$?：是上一指令的返回值，成功是0，不成功是1。一般来说，UNIX(linux) 系统的进程以执行系统调用exit() 来结束的。这个回传值就是status值。回传给父进程，用来检查子进程的执行状态。一般指令程序倘若执行成功，其回传值为 0；失败为 1。

$\*：所有脚本参数的内容：就是调用调用本bash shell的参数。

$@：基本上与上面相同。只不过是“$\*”返回的是一个字符串，字符串中存在多外空格。 “$@”返回多个字符串。

$#：返回所有脚本参数的个数。

$$ :Shell本身的PID（ProcessID），即当前进程的PID。

$! ：Shell最后运行的后台Process的PID

$- ：使用Set命令设定的Flag一览

示例：

# touch variable   
# vi variable   
脚本内容如下：   
#!/bin/sh   
echo "number:$#"   
echo "scname:$0"   
echo "first :$1"   
echo "second:$2"   
echo "argume:$@"   
保存退出   
赋予脚本执行权限   
# chmod +x variable   
执行脚本   
# ./variable aa bb   
number:2   
scname:./variable   
first: aa   
second:bb   
argume:aa bb   
通过显示结果可以看到：   
$# 是传给脚本的参数个数   
$0 是脚本本身的名字   
$1是传递给该shell脚本的第一个参数   
$2是传递给该shell脚本的第二个参数   
$@ 是传给脚本的所有参数的列表

**用法二** ：**获取变量与环境变量的值**

如：path=2，则echo $path 或者echo${path}显示的就是path的值。

在linux及unix的sh中，以$开头的[字符串](http://www.so.com/s?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&ie=utf-8&src=internal_wenda_recommend_textn)表示的是sh中定义的[变量](http://www.so.com/s?q=%E5%8F%98%E9%87%8F&ie=utf-8&src=internal_wenda_recommend_textn)，这些变量可以是系统自动增加的，也可以是用户自己[定义](http://www.so.com/s?q=%E5%AE%9A%E4%B9%89&ie=utf-8&src=internal_wenda_recommend_textn)的$PATH表示的是系统的[命令](http://www.so.com/s?q=%E5%91%BD%E4%BB%A4&ie=utf-8&src=internal_wenda_recommend_textn)搜索[路径](http://www.so.com/s?q=%E8%B7%AF%E5%BE%84&ie=utf-8&src=internal_wenda_recommend_textn)，和windows的%path%是一样的$HOME则表示是用户的主目录，也就是用户登录后工作目录

示例：echo $PATH.

# 用法三：shell中$(( ))、$( )、``与${ }的区别

说明：

${ }这种形式其实与用法一和二是一样的，属于变量替换的范畴，只不过在变量替换中可以加上大括号，也可以不加大括号。

简而言之：$(( ))属于执行计算公式，$( )和` `属于命令替换，${ }属于变量替换

**（1）$( )和` `：属于命令替换**

在bash中，$( )与` `（反引号）都是用来作命令替换的。  
**命令替换**与变量替换差不多，都是用来重组命令行的，先完成引号里的命令行，然后将其结果替换出来，再重组成新的命令行

示例：命令：$ echo today is $(date "+%Y-%m-%d")，显示：today is 2014-07-01

##### $( )与｀｀(反引号)：返回括号中命令的结果

在操作上，这两者都是达到相应的效果，但是建议使用$( )，理由如下：

1）｀｀很容易与''搞混乱，尤其对初学者来说。

2）在多层次的复合替换中，｀｀必须要额外的跳脱处理（反斜线），而$( )比较直观。

3）最后，$( )的弊端是，并不是所有的类unix系统都支持这种方式，但反引号是肯定支持的。

示例：

1. *# 将cmd1执行结果作为cmd2参数，再将cmd2结果作为cmd3的参数*
2. cmd3 $(cmd2 $(cmd1))
3. *# 如果是用反引号，直接引用是不行的，还需要作跳脱处理*
4. cmd3 `cmd2 \`cmd1\``

**（2）${ }变量替换**

一般情况下，$var与${var}是没有区别的，但是用${ }会比较精确的界定变量名称的范围

1. $ A=B
2. $ echo ${A}B
3. BB

**取路径、文件名、后缀**  
先赋值一个变量为一个路径，如下：  
file=/dir1/dir2/dir3/my.file.txt

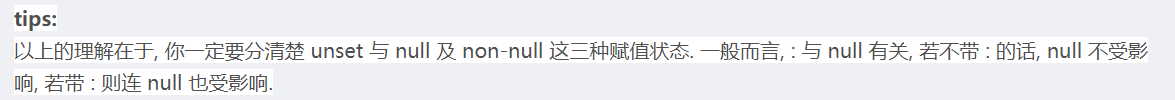


记忆方法如下：

* # 是去掉左边(在键盘上 # 在 $ 之左边)
* % 是去掉右边(在键盘上 % 在 $ 之右边)
* 单一符号是最小匹配;两个符号是最大匹配
* \*是用来匹配不要的字符，也就是想要去掉的那部分
* 还有指定字符分隔号，与\*配合，决定取哪部分

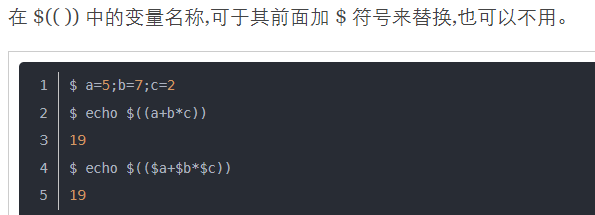


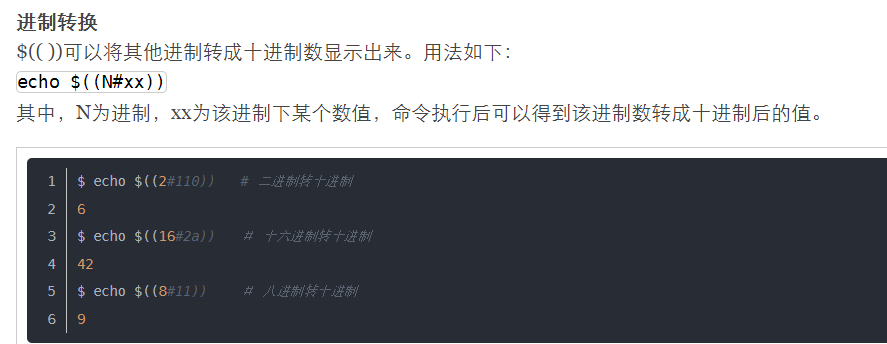


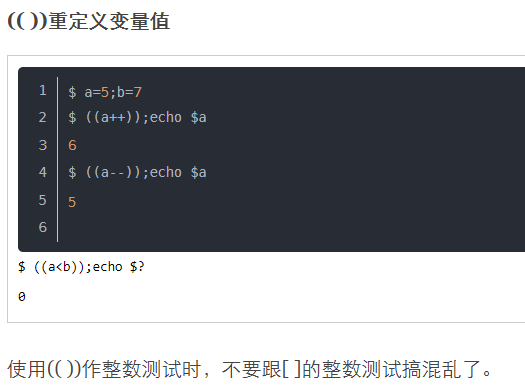


**(3) $(( ))与算术运算**









# 用法四：$PWD显示环境变量

PWD 是环境变量，$PWD 与 $(pwd)就是等价的。