Linux—shell中$(( ))、$( )、``与${ }的区别

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。

exp 1

```
[root@localhost ~]# echo today is $(date "+%Y-%m-%d")
today is 2017-11-07
[root@localhost ~]# echo today is `date "+%Y-%m-%d"`
today is 2017-11-07
```

$( )与` `
在操作上，这两者都是达到相应的效果，但是建议使用$( )，理由如下：

` `很容易与''搞混乱，尤其对初学者来说，而$( )比较直观。
最后，$( )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。

exp 2

```
[root@localhost ~]#  echo Linux `echo Shell `echo today is `date "+%Y-%m-%d"```
Linux Shellecho today is 2017-11-07     #过多使用``会有问题
[root@localhost ~]# echo Linux `echo Shell $(echo today is $(date "+%Y-%m-%d"))`
Linux Shell today is 2017-11-07     ``和$()混合使用
[root@localhost ~]# echo Linux $(echo Shell $(echo today is $(date "+%Y-%m-%d")))
Linux Shell today is 2017-11-07    #多个$()同时使用也不会有问题
```

${ }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

exp 1

```
[root@localhost ~]# A=Linux
[root@localhost ~]# echo $AB    #表示变量AB

[root@localhost ~]# echo ${A}B    #表示变量A后连接着B
LinuxB
```

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
```

命令　　解释　　结果

```
${file#*/}    拿掉第一条 / 及其左边的字符串    dir1/dir2/dir3/my.file.txt
[root@localhost ~]# echo ${file#*/}
dir1/dir2/dir3/my.file.txt

${file##*/}    拿掉最后一条 / 及其左边的字符串    my.file.txt
[root@localhost ~]# echo ${file##*/}
my.file.txt

${file#*.}    拿掉第一个 . 及其左边的字符串    file.txt
[root@localhost ~]# echo ${file#*.}
file.txt

${file##*.}    拿掉最后一个 . 及其左边的字符串    txt
[root@localhost ~]# echo ${file##*.}
txt

${file%/*}    拿掉最后一条 / 及其右边的字符串    /dir1/dir2/dir3
[root@localhost ~]# echo ${file%/*}
/dir1/dir2/dir3

${file%%/*}    拿掉第一条 / 及其右边的字符串    (空值)
[root@localhost ~]# echo ${file%%/*}
(空值)

${file%.*}    拿掉最后一个 . 及其右边的字符串    /dir1/dir2/dir3/my.file
[root@localhost ~]# echo ${file%.*}
/dir1/dir2/dir3/my.file
```

```
${file%%.*}     拿掉第一个 . 及其右边的字符串      /dir1/dir2/dir3/my
[root@localhost ~]# echo ${file%%.*}
/dir1/dir2/dir3/my
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
```

取子串及替换

命令	解释	结果
${file:0:5}	提取最左边的 5 个字节	/dir1
${file:5:5}	提取第 5 个字节右边的连续 5 个字节	/dir2
${file/dir/path}	将第一个 dir 提换为 path	/path1/dir2/dir3/my.file.txt
${file//dir/path}	将全部 dir 提换为 path	/path1/path2/path3/my.file.txt
${#file}	获取变量长度	27

根据状态为变量赋值

命令	解释	备注
${file-my.file.txt}	若 $file 没设定,则使用 my.file.txt 作传回值	空值及非空值不作处理
${file:-my.file.txt}	若 $file 没有设定或为空值,则使用 my.file.txt 作传回值	非空值时不作处理
${file+my.file.txt}	若$file 设为空值或非空值,均使用my.file.txt作传回值	没设定时不作处理
${file:+my.file.txt}	若 $file 为非空值,则使用 my.file.txt 作传回值	没设定及空值不作处理
${file=txt}	若 $file 没设定,则回传 txt ,并将 $file 赋值为 txt	空值及非空值不作处理
${file:=txt}	若 $file 没设定或空值,则回传 txt ,将 $file 赋值为txt	非空值时不作处理

命令	解释	备注
${file?my.file.txt}	若 $file 没设定,则将 my.file.txt 输出至 STDERR	空值及非空值不作处理
${file:?my.file.txt}	若 $file没设定或空值,则将my.file.txt输出至STDERR	非空值时不作处理

tips:

以上的理解在于, 你一定要分清楚 unset 与 null 及 non-null 这三种赋值状态. 一般而言, : 与 null 有关, 若不带 : 的话, null 不受影响, 若带 : 则连 null 也受影响.

数组

```
A="a b c def"    # 定义字符串
A=(a b c def)    # 定义字符数组
```

命令	解释	结果
${A[@]}	返回数组全部元素	a b c def
${A[*]}	同上	a b c def
${A[0]}	返回数组第一个元素	a
${#A[@]}	返回数组元素总个数	4
${#A[*]}	同上	4
${#A[3]}	返回第四个元素的长度，即def的长度	3
A[3]=xzy	则是将第四个组数重新定义为 xyz	

$(( ))与整数运算

bash中整数运算符号

符号	功能
+ - * /	分别为加、减、乘、除
%	余数运算
& | ^ !	分别为“AND、OR、XOR、NOT”

在 $(( )) 中的变量名称,可于其前面加 $ 符号来替换,也可以不用。

```
[root@localhost ~]# echo $((2*3))
6
[root@localhost ~]# a=5;b=7;c=2
[root@localhost ~]# echo $((a+b*c))
19
[root@localhost ~]# echo $(($a+$b*$c))
19
```

进制转换

$(( ))可以将其他进制转成十进制数显示出来。用法如下：

echo $((N#xx))

其中，N为进制，xx为该进制下某个数值，命令执行后可以得到该进制数转成十进制后的值。

```
[root@localhost ~]# echo $((2#110))
6
[root@localhost ~]# echo $((16#2a))
42
[root@localhost ~]# echo $((8#11))
9
```

(())重定义变量值

```
[root@localhost ~]# a=5;b=7
[root@localhost ~]# ((a++))
[root@localhost ~]# echo $a
6
[root@localhost ~]# ((a--));echo $a
5
[root@localhost ~]# ((a<b));echo $?
0
[root@localhost ~]# ((a>b));echo $?
1
```

转： s h e l l $(( ))、$( )、``与${ }的区别

****

学习永远不晚。——高尔基

****