

## 一、小括号，圆括号 ( )

### 1、单小括号 ( )

①命令组。括号中的命令将会新开一个子shell顺序执行，所以括号中的变量不能够被脚本余下的部分使用。括号中多个命令之间用分号隔开，最后一个命令可以没有分号，各命令和括号之间不必有空格。

②命令替换。等同于`cmd`，shell扫描一遍命令行，发现了\$(cmd)结构，便将\$(cmd)中的cmd执行一次，得到其标准输出，再将此输出放到原来命令。有些shell不支持，如tcsh。

③用于初始化数组。如：array=(a b c d)

### 2、双小括号 (( ))

①整数扩展。这种扩展计算是整数型的计算，不支持浮点型。((exp))结构扩展并计算一个算术表达式的值，如果表达式的结果为0，那么返回的退出状态码为1，或者是"假"，而一个非零值的表达式所返回的退出状态码将为0，或者是"true"。若是逻辑判断，表达式exp为真则为1,假则为0。

②只要括号中的运算符、表达式符合C语言运算规则，都可用在\$(exp)中，甚至是三目运算符。作不同进位(如二进制、八进制、十六进制)运算时，输出结果全都自动转化成了十进制。如：echo \$((16#5f)) 结果为95 (16进位转十进制)

③单纯用(( ))也可重定义变量值，比如 a=5; ((a++)) 可将 \$a 重定义为6

④常用于算术运算比较，双括号中的变量可以不使用\$符号前缀。括号内支持多个表达式用逗号分开。只要括号中的表达式符合C语言运算规则,比如可以直接使用for((i=0;i<5;i++)), 如果不使用双括号, 则为for i in `seq 0 4` 或者for i in {0..4}。再如可以直接使用if ((\$i<5)), 如果不使用双括号, 则为if [ \$i -lt 5 ]。

## 二、中括号，方括号 []

### 1、单中括号 []

①bash 的内部命令，[和test是等同的。如果我们不用绝对路径指明，通常我们用的都是bash自带的命令。if/test结构中的左中括号是调用test的命令标识，右中括号是关闭条件判断的。这个命令把它的参数作为比较表达式或者作为文件测试，并且根据比较的结果来返回一个退出状态码。if/test结构中并不是必须右中括号，但是新版的Bash中要求必须这样。

②Test和[]中可用的比较运算符只有==和!=，两者都是用于字符串比较的，不可用于整数比较，整数比较只能使用-eq, -gt这种形式。无论是字符串比较还是整数比较都不支持大于号小于号。如果实在想用，对于字符串比较可以使用转义形式，如果比较"ab"和"bc": [ ab \< bc ], 结果为真，也就是返回状态为0。[]中的逻辑与和逻辑或使用-a 和-o 表示。

③字符范围。用作正则表达式的一部分，描述一个匹配的字符范围。作为test用途的中括号内不能使用正则。

④在一个array 结构的上下文中，中括号用来引用数组中每个元素的编号。

## 2、双中括号[[ ]]

①[[是 bash 程序语言的关键字。并不是一个命令，[[ ]] 结构比[ ]结构更加通用。在[[和]]之间所有的字符都不会发生文件名扩展或者单词分割，但是会发生参数扩展和命令替换。

②支持字符串的模式匹配，使用=~操作符时甚至支持shell的正则表达式。字符串比较时可以把右边的作为一个模式，而不仅仅是一个字符串，比如[[ hello == hell? ]]，结果为真。[[ ]] 中匹配字符串或通配符，不需要引号。

③使用[[ ... ]]条件判断结构，而不是[ ... ]，能够防止脚本中的许多逻辑错误。比如，&&、||、<和> 操作符能够正常存在于[[ ]]条件判断结构中，但是如果出现在[ ]结构中的话，会报错。比如可以直接使用if [[ \$a != 1 && \$a != 2 ]]，如果不适用双括号，则为if [ \$a -ne 1 ] && [ \$a != 2 ]或者if [ \$a -ne 1 -a \$a != 2 ]。

④bash把双中括号中的表达式看作一个单独的元素，并返回一个退出状态码。

例子：

```
1  if ($i<5)
2  if [ $i -lt 5 ]
3
4  if [ $a -ne 1 -a $a != 2 ]
5  if [ $a -ne 1 ] && [ $a != 2 ]
6  if [[ $a != 1 && $a != 2 ]]
7
8  for i in $(seq 0 4); do echo $i; done
9  for i in `seq 0 4`; do echo $i; done
   for ((i=0;i<5;i++)); do echo $i; done
   for i in {0..4}; do echo $i; done
```

## 三、大括号、花括号 {}

### 1、常规用法

①大括号拓展。(通配(globbing))将对大括号中的文件名做扩展。在大括号中，不允许有空白，除非这个空白被引用或转义。第一种：对大括号中的以逗号分割的文件列表进行拓展。如 touch {a,b}.txt 结果为a.txt b.txt。第二种：对大括号中以点点(..)分割的顺序文件列表起拓展作用，如：touch {a..d}.txt 结果为a.txt b.txt c.txt d.txt

```
1  # ls {ex1,ex2}.sh
2  ex1.sh ex2.sh
3
4  # ls {ex{1..3},ex4}.sh
5  ex1.sh ex2.sh ex3.sh ex4.sh
6  # ls {ex[1-3],ex4}.sh
   ex1.sh ex2.sh ex3.sh ex4.sh
```

②代码块，又被称为内部组，这个结构事实上创建了一个匿名函数。与小括号中的命令不同，大括号内的命令不会新开一个子shell运行，即脚本余下部分仍可使用括号内变量。括号内的命令间用分号隔开，最后一个也必须要有分号。{}的第一个命令和左括号之间必须要有一个空格。

## 2、几种特殊的替换结构

```
1 | | ${var:-string},${var:+string},${var:=string},${var:?string}
```

① `${var:-string}`和`${var:=string}`：若变量var为空，则用在命令行中用string来替换`${var:-string}`，否则变量var不为空时，则用变量var的值来替换`${var:-string}`；对于`${var:=string}`的替换规则和`${var:-string}`是一样的，所不同之处是`${var:=string}`若var为空时，用string替换`${var:=string}`的同时，把string赋给变量var：`${var:=string}`很常用的一种用法是，判断某个变量是否赋值，没有的话则给它赋上一个默认值。

② `${var:+string}`的替换规则和上面的相反，即只有当var不是空的时候才替换成string，若var为空时则不替换或者说是替换成变量var的值，即空值。（因为变量var此时为空，所以这两种说法是等价的）

③ `${var:?string}`替换规则为：若变量var不为空，则用变量var的值来替换`${var:?string}`；若变量var为空，则把string输出到标准错误中，并从脚本中退出。我们可利用此特性来检查是否设置了变量的值。

补充扩展：在上面这五种替换结构中string不一定是常值的，可用另外一个变量的值或是一种命令的输出。

## 3、四种模式匹配替换结构

模式匹配记忆方法：

# 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)

#和%中的单一符号是最小匹配，两个相同符号是最大匹配。

```
1 | | ${var%pattern},${var%%pattern},${var#pattern},${var##pattern}
```

第一种模式：`${variable%pattern}`，这种模式时，shell在variable中查找，看它是否一给的模式pattern结尾，如果是，就从命令行把variable中的内容去掉右边最短的匹配模式

第二种模式：`${variable%%pattern}`，这种模式时，shell在variable中查找，看它是否一给的模式pattern结尾，如果是，就从命令行把variable中的内容去掉右边最长的匹配模式

第三种模式：`${variable#pattern}`这种模式时，shell在variable中查找，看它是否一给的模式pattern开始，如果是，就从命令行把variable中的内容去掉左边最短的匹配模式

第四种模式：`${variable##pattern}`这种模式时，shell在variable中查找，看它是否一给的模式pattern结尾，如果是，就从命令行把variable中的内容去掉右边最长的匹配模式

这四种模式中都不会改变variable的值，其中，只有在pattern中使用了\*匹配符号时，%和%%，#和##才有区别。结构中的pattern支持通配符，\*表示零个或多个任意字符，?表示仅与一个任意字符匹配，[...]表示匹配中括号里面的字符，[!...]表示不匹配中括号里面的字符。

```

1 | # var=testcase
2 | # echo $var
3 |
4 | testcase
5 | # echo ${var%s*e}
6 | testca
7 |
8 | # echo $var
9 | testcase
10 | # echo ${var%%s*e}
11 | te
12 |
13 | # echo ${var#?e}
14 | stcase
15 |
16 | # echo ${var##?e}
17 | stcase
18 | # echo ${var##*e}
    # echo ${var##*s}
    e
    # echo ${var##test}
    case

```

#### 4、字符串提取和替换

```

1 | ${var:num},${var:num1:num2},${var /pattern/pattern},${var //pattern/patte

```

第一种模式： `${var:num}`，这种模式时，shell在var中提取第num个字符到末尾的所有字符。若num为正数，从左边0处开始；若num为负数，从右边开始提取字符串，但必须使用在冒号后面加空格或一个数字或整个num加上括号，如`${var: -2}`、`${var:1-3}`或`${var:(-2)}`。

第二种模式： `${var:num1:num2}`，num1是位置，num2是长度。表示从\$var字符串的第\$num1个位置开始提取长度为\$num2的子串。不能为负数。

第三种模式： `${var/pattern/pattern}` 表示将var字符串的第一个匹配的pattern替换为另一个pattern。

第四种模式： `${var//pattern/pattern}` 表示将var字符串中的所有能匹配的pattern替换为另一个pattern。

```

1 | [root@centos ~] # var=/home/centos
2 | [root@centos ~] # echo $var
3 | /home/centos
4 | [root@centos ~] # echo ${var:5}
5 | /centos
6 | [root@centos ~] # echo ${var: -6}
7 | centos
8 |
9 |
10 |

```

```
11 [root@centos ~]# echo ${var:(-6)}
12 centos
13
14 [root@centos ~]# echo ${var:1:4}
15 home
[root@centos ~]# echo ${var/o/h}
/hhme/centos
[root@centos ~]# echo ${var//o/h}
/hhme/cenhs
```

#### 四、符号\$后的括号

- (1) \${a} 变量a的值, 在不引起歧义的情况下可以省略大括号。
- (2) \$(cmd) 命令替换, 和`cmd`效果相同, 结果为shell命令cmd的输, 过某些Shell版本不支持\$()形式的命令替换, 如tcsh。
- (3) \$((expression)) 和`expexpression`效果相同, 计算数学表达式exp的数值, 其中exp只要符合C语言的运算规则即可, 甚至三目运算符和逻辑表达式都可以计算。

#### 五、使用

##### 1、多条命令执行

- (1) 单小括号, (cmd1;cmd2;cmd3) 新开一个子shell顺序执行命令cmd1,cmd2,cmd3, 各命令之间用分号隔开, 最后一个命令后可以没有分号。
- (2) 单大括号, { cmd1;cmd2;cmd3;} 在当前shell顺序执行命令cmd1,cmd2,cmd3, 各命令之间用分号隔开, 最后一个命令后必须有分号, 第一条命令和左括号之间必须用空格隔开。

对{}和()而言, 括号中的重定向符只影响该条命令, 而括号外的重定向符影响到括号中的所有命令。

#### 总结

以上所述是小编给大家介绍的shell中各种括号的作用()、(( ))、[]、[[ ]]、{}(推荐), 希望对大家有所帮助, 如果大家有任何疑问请给我留言, 小编会及时回复大家的。在此也非常感谢大家对脚本之家网站的支持!

#### 您可能感兴趣的文章:

[shell中的各种括号的使用方法](#)

原文链接: <http://www.cnblogs.com/Q--T/archive/2017/09/06/7486872.html>