Shell 参数扩展及各类括号在 Shell 编程中的应用

原创 大数据之路 awk/shell 2015/01/17 01:56 阅读数 1.1W

今天看有人总结了 shell 下的参数扩展，但不是很全，恰好以前整理过放在百度空间，但百度空间目前半死不活的情况下对 Google 非常不友好，索性一并转过来方便查阅。

1、bash 中的大括号参数扩展（Parameter Expansion）

假设我们定义了一个变量为：

```
file=/dir1/dir2/dir3/my.file.txt
```

1.1 bash 下的 split 取“数组”的首、尾：

```
${file#*/}：拿掉第一条 / 及其左边的字符串：dir1/dir2/dir3/my.file.txt
${file##*/}：拿掉最后一条 / 及其左边的字符串：my.file.txt
${file#*.}：拿掉第一个 .  及其左边的字符串：file.txt
${file##*.}：拿掉最后一个 .  及其左边的字符串：txt
${file%/*}：拿掉最后条 / 及其右边的字符串：/dir1/dir2/dir3
${file%%/*}：拿掉第一条 / 及其右边的字符串：(空值)
${file%.*}：拿掉最后一个 .  及其右边的字符串：/dir1/dir2/dir3/my.file
${file%%.*}：拿掉第一个 .  及其右边的字符串：/dir1/dir2/dir3/my
```

Tips：

记忆的方法为：

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（类似贪婪匹配）。

1.2 bash 下的 substring 按字符位置、长度截取

```
${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}：计算出字符串的长度，/dir1/dir2/dir3/my.file.txt 字符串长度 27
${file: -4}：提取最后四个字符串（空格是为了避免冲突，注意不同于echo ${file:-4}，也可以用(-4)代替空格），类似用法
```

1.3 bash 下的 replace 与 replaceAll

我们也可以对变量值里的字符串作替换：

```
${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
```

1.4 bash 下的变量空值检测与初始化

利用 ${ } 还可针对不同的变量状态赋值(没设定、空值、非空值)：

```
${file-my.file.txt} ：假如 $file 没有设定，则使用 my.file.txt 作传回值。(空值及非空值时不作处理)
${file:-my.file.txt} ：假如 $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）
```

关于作者

大数据之
架构师
关注

文章 500
经验值 2.9K

作者的专辑

java (35)
awk/shell (23)
scala (4)
perl (3)

推荐关注

志成就 文章 242 访问
ywp1 文章 1 访问 6
欧德 文章 53 访问
hi懒喵 文章 44 访问
naughty 文章 97 访问

```
${file=my.file.txt} ：若 $file 没设定，则使用 my.file.txt 作传回值，同时将 $file 赋值为 my.file.txt 。（s
${file:=my.file.txt} ：若 $file 没设定或为空值，则使用 my.file.txt 作传回值，同时将 $file 赋值为 my.file.t
${file?my.file.txt} ：若 $file 没设定，则将 my.file.txt 输出至 STDERR。（空值及非空值时不作处理）
${file:?my.file.txt} ：若 $file 没设定或为空值，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。

而 - 和 = 的区别在于是否把传回值赋给引用变量，例如：

```
${parameter:-word}      word is only substituted.
${parameter:=word}      word is substituted and assigned to parameter.
root@localhost ~ $ echo "$var"


root@localhost ~ $ echo "${var:-hello}"
hello
root@localhost ~ $ echo "$var"


root@localhost ~ $ echo "${var:=hello}"
hello
root@localhost ~ $ echo "$var"
hello
```

1.5 bash 下的数组和关联数组

Bash4中可以使用两种容器。

一种是数组，另一种是关联数组，类似于其他语言中的Map/Hash/Dict。

声明数组的常用语法： declare -a ARY或者ARY=(1 2 3)

声明关联数组的唯一语法： declare -A MAP（bash4以下不支持）

赋值的语法：

直接ARY[N]=VALUE，N可以是数字索引也可以是键。关联数组可以使用MAP=([x]=a [y]=b)进行多项赋值，注意这是赋值的语句而不是声明。

亲测数组中的索引不一定要按顺序来，你可以先给2和3上的元素赋值。（同样算是弱类型的Javascript也支持这种无厘头赋值，这算通病么？）

```
往现有数组批量添加元素:
ARY+=(a b c)
MAP+=([a]=1 [b]=2)
取值:
${ARY[INDEX]}
${MAP[KEY]}
注意花括号的使用
${A[@]} 展开成所有的变量，而获取数组长度使用 ${#A[@]}
切片:
${ARY[@]:N:M} N是offset而M是length
返回索引，相当于keys():
${!MAP[@]}
试试下面的代码:
declare -a ARY
declare -A MAP
MAP+=([a]=1 [b]=2)
ARY+=(a b c)

echo ${ARY[1]}
echo ${MAP[a]}
echo "${ARY[@]}"
echo "${MAP[@]}"
echo "${ARY[@]:0:1}"
echo ${#ARY[@]}
echo "${!MAP[@]}"

ARY[4]=a
echo ${ARY[@]}
echo ${ARY[3]}
```

1.6 bash 下的大小写变换

```
echo "$HI" # Hello
echo ${HI^} # Hello
echo ${HI^^} # HELLO
echo ${HI,} # hello
echo ${HI,,} # hello
echo ${HI~} # hello
echo ${HI~~} #hELLo
^大写，,小写， ~大小写切换
重复一次只修改首字母，重复两次则应用于所有字母。

混着用会怎样?
echo ${HI^,^} # Hello
看来是不行的x_x
```

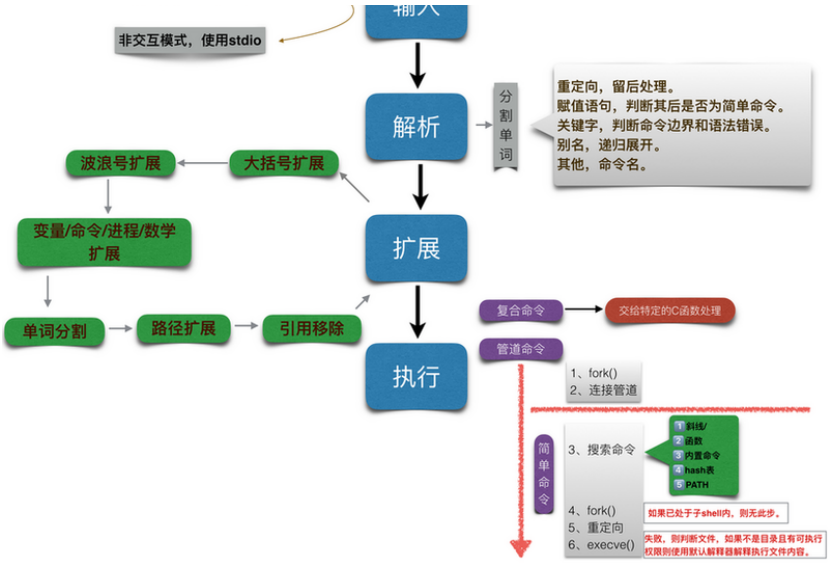
2、各类括号在 shell/bash 编程中的应用

上面应该见识到了 shell 中大括号的强大功能，其实 shell 下有很多种括号，不像其它高级语言括号只起到语法和意义的作用，而 shell 下的每种括号除了语法、语义的作用之外，还对 shell 编程起到了功能上的扩展。

```
2.1 () 在子shell中运行
    (a=1);echo $a，结果是空，因为a=1不是在当前shell中运行的(a=1);(echo $a)也是空的。
    小技巧：(cd $path, do something) 可以让不切换当前目录而在其它目录干点别的事儿~
    () 还有个功能是数组的赋值：比如a=(1 3 5)，那么${a[0]}=1;${a[1]}=3;${a[2]}=5，需要注意的是，下标是从0开始

2.2 (()) 表达式计算
    a=1;((a++));echo $a，这时a就是2了。

2.3 <() 和 >() 进程代入，可以把命令的执行结果当成文件一样读入
    比如comm前一般需要sort，那就可以这样comm <(sort 1.lst) <(sort 2.lst)
    或者是paste <(cut -t2 file1) <(cut -t1 file1)，和管道差不多，但是支持多个输入。

2.4 $() $(cmd) 执行cmd的结果，
    比如cmd是echo ls，那么就是执行ls，比如file $(which bash)，which bash的结果是/bin/bash，
    所以file $(which bash)等于file /bin/bash。如果你$(ls)，而且你的当前目录下只有a b两个文件，
    那么就是执行a b，然后系统会提示，命令没找到。$() 基本和 `` 等价。

2.5 $(()) 表达式扩展，
    和(())很相似，但是这个是有点不同，$(())不能直接$((b++))，例如：b=1;echo $((++b))
    这时b等于2，显示的也是2，b=1; echo $((b++))这时b等于2，显示的是1.

2.6 [] 和 [[]]，[] 就是 test，[]和[[]]都是条件表达式，不过[[]]有比[]高的容错性，
    如果a为空，那么[ $a -eq 0 ]会报错，但是[[ $a -eq 0 ]]不会，所以一般都会使用[[]]或者是
    [ "$a" -eq 0 ]，[[]]支持的功能也比 [] 多，比如[[ aaa =~ a{3} ]]，[] 还有一种用途，
    如果你的当前目录下有a1-a9九个文件，你可以用a[1-9]来替代这九个文件。
    有点需要注意，你不能用a[1-20]来代替a1- a20，必须要a[1-9] a1[0-9] a20。
    但是需要注意的是 [[]] 数字进制转换的坑~

2.7 $[] 是 $(()) 的过去形式，现在已经不建议使用。

2.8 {n..m} {1..30} 就是1-30，或者是/{,s}bin/表示/bin/和/sbin/，ab{c,d,e}表示abc、abd、abe，
    小技巧：文件备份：cp a.sh{,.bak}
    而 { cmd1; cmd2; } 的作用是定义一个命令组，一般用在单行的条件表达式中：
    [[ 1 -eq 2 ]] && echo True || { echo False; echo "Program will exit! "; }
    其实 shell 函数的语法也是它的变体：
    a(){ i=$1; echo $((i++)); echo $((++i)); } && a 1

2.9 ${} 变量的Parameter Expansion，
    用法很多，最基本的 ${var}1，防止变量扩展冲突，具体可以查看man bash。
    或者参考我之前的博文链接：http://hi.baidu.com/leejun_2005/blog/item/ebfee11a4177ddc1ac6e751d.html
```

3、bash命令执行流程：

执行分为四大步骤：输入、解析、扩展和执行。



4、Refer：

[1] shell十三问之大括号参数扩展（Parameter Expansion）

http://hi.baidu.com/leejun_2005/item/138c09343aaddff6e6bb7a49

[2] shell 十三問?

http://bbs.chinaunix.net/forum.php?mod=viewthread&tid=218853&page=7#

[3] shell/bash编程中各类括号的应用

http://hi.baidu.com/leejun_2005/item/6f9eb7345e5f4f302f20c453

[4] Bash Hackers Wiki Frontpage » Syntax » Parameter expansion

http://wiki.bash-hackers.org/syntax/pe

[5] 玩转Bash变量

http://segmentfault.com/blog/spacewander/1190000002539169

[6] Bash快速入门指南

http://blog.jobbole.com/85183/

[7] SHELL(bash)脚本编程六：执行流程

https://segmentfault.com/a/1190000008215772

 举报

作者的其它热门文章

关于 python ImportError: No module named 的问题
Flume NG 简介及配置实战
python基础（5）：深入理解 python 中的赋值、引用、拷贝、作用域
Fiddler 高级用法：Fiddler Script 与 HTTP 断点调试

云台煮酒

1.4节有问题.. 真确的是这里http://bbs.51cto.com/thread-1158753-1.html

2018/03/09 18:33 回复 举报

云台煮酒

好文章 谢谢

2018/03/09 15:21 回复 举报

更多评论