Shell各种符号之含义

#!：符号能够被内核识别成是一个脚本的开始，这一行必须位于脚本的首行

$0：当前脚本的名字

$#：输入<调用>参数(脚本或函数的位置参数) 的个数，如 NumArg=$#；echo"\$#: $#;\$NumArg: $NumArg"

$@：传递给脚本或函数的所有参数，按空格划分成不同的部分。注意: 如果一个参数中有空格，这个参数将会被从空格($IFS默认)处分尸

"$@"：分隔成单个参数，如"$1” “$2” “$3”, $@以IFS（默认为空格）来划分字段，如果空格在“”里面，不划分即不会被分尸

$*：传递给脚本或函数的所有参数，按空格划分成不同的部分，与$@没有区别

"$*"：扩展成一个参数，如"$1$2 $3"，不论调用时传入了多少个参数，都将被按原样处理成一个整体性的参数使用。$@ $*只在被双引号包起来的时候才会有差异

$?：上个命令或函数的返回值

$n：第几个参数，n=3则$n是$3 ，注意，$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10时，需要使用${n}来获取参数。

$$：当前shell脚本的进程ID

$!：用于保存运行的最后一个进程的PID 号。

$(command)：内置命令替换操作符。Spid=$(pidof NmsServer) 或 echo $(pidof NmsServer) 等同于反引号： `pidof NmsServer`

`command`：反斜杠——同上：$(command)

注意：这两种命令的执行方式，是在当前shell环境中，开启了一个子进程，在这个子进程中执行命令并完成替换。所以当语句执行完回到主脚本中时，再访问子进程中定义、修改的数据，注意徒劳！

${PIDLIST[@]}：数组PIDLIST的全部值

${#PIDLIST[@]}：获取数组PIDLIST=(${Spid} ${Mpid} ${ProbePid})中的元素个数

${}：shell变量的引方式<更加明确的变量名称的界定>；匹配与替换操作；

变量引用

pid=$(pgrep-f java)

echo ${pid}

输出：3758564369 79366

echo ${pid}|awk '{print NF}'

输出结果：6

echo "${pid}"

输出:

37585

64369

79366

特殊的替换结构

${var:-string},${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。

匹配替换结构

${var%pattern},${var%%pattern},${var#pattern},${var##pattern}

第一种模式：${variable%pattern}，这种模式时，shell在variable中查找，看它是否一给定的模式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，看它是否一给定]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，看它是否一给定的模式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。

字符串提取和替换

${var:num},${var:num1:num2},${var/pattern/pattern},${var//pattern/pattern}

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。

$[] $(()) :它们功能一样，都是进行数学运算的。支持+ - * / %：分别为“加、减、乘、除、取模”。但是注意，bash只能作整数运算，对于浮点数是当作字符串处理的。

$IFS : shell的内部域分隔符。当 shell处理"命令替换"和"参数替换"时，shell 根据 IFS 的值，默认是 space, tab, newline来拆解读入的变量，然后对特殊字符进行处理，最后重新组合赋值给该变量。IFS的默认值为：空白（包括：空格，tab,和新行).

将其ASSII码用十六进制打印出来：

$ echo$IFS

$ echo"$IFS" | od -b

0000000040 011 012 012

0000004

直接输出IFS是看不到的，把它转化为二进制就可以看到了，"040"是空格，"011"是Tab，"012"是换行符"\n"。最后一个 012 是因为 echo 默认是会换行的

expr : 命令对算术表达式求值 如：c=`expr $a + $b`

0 表示标准输入

1 表示stdout标准输出

2 表示stderr标准错误

& 表示等同于的意思，2>&1，表示2的输出重定向等同于1

> ：将执行结果输出到文件、设备上

> /dev/null：不可回收垃圾箱 ，命令等同于1>/dev/null

Example: ./yunNanAlertTransferTest.sh 1>/dev/null 2>yunNanAlertTransferTest.log &

>>：将执行结果追加到文件、设备上

<：从文件、设备、输入点读入内容

<<：从文件、设备、输入点追加读入内容

|：一个命令的输出传递给另一个命令当做输入

test命令，等价于[ ]：内置test命令常用操作符号，将表达式写在[]中, 如：[expression]#注意：expression首尾都有个空格。表达式结果为真，则test返回值为0，否则为非0

==：两个值的比较，如A1是否与A2相等，真：返回0，非：返回非0 [string1 == sting2]

!=：两个值的比较，如A1是否与A2不相等。 注：[string1]——string1是否是个空串。

=~：用于匹配或侧表达式，即左侧是否包含右侧。if [[$1 =~ stop || $1 =~ 'status']]；[] 这种结构不支持该操作。

XXX()：表示这是一个函数

{ command1;command2;command3; }：表示这个主体中的内容是个执行体，这个执行体不开子进程，在当前脚本进程中按序执行。开关与结尾的空格，最后一个命令必须有“；”

./：表示当前目录——fork调用，在当前父进程中新起一个子进程执行

../：表示当前目录的上级目录

\：用作转义字符，或称逃脱字符，echo要让转义字符发生作用，就要使用-e选项，且转义字符要使用双引号 。另一作用，反斜杠用于一行的最后一个字符时，把行尾的反斜杠作为续行，这种结构在分几行输入长命令时经常使用。

''：单引号——表示纯情字符串；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；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）。

单引号将剥夺其中的所有字符的特殊含义，阻止对所有字符的转义， 将所有的字符看做其字面的意思。

""：双引号——表示纯情字符串或变量的值。如果变量需多行呈现，需要双引号，否则单行呈现。见下图示。双引号里可以有变量，双引号里可以出现转义字符。

双引号会处理字符串中的'$'（参数替换），'``' or ' () '（命令替换），'\' 转义字符 和算数等。

两者基本上没有什么区别，除非在内容中遇到了参数替换符$和命令替换符`(我不是单引号)。

sync：刷新内存缓存

exit：Shell内置命令，用来退出当前 Shell进程，并返回一个退出状态；使用$?可以接收这个退出状态(0~255 之间的整数，其中只有 0 表示成功，其它值都表示失败) 注意，exit 表示退出当前 Shell 进程，我们必须在新进程中运行test.sh，否则当前 Shell 会话（终端窗口）会被关闭，我们就无法看到输出结果了。为了脚本执行完不会退出终端，脚本的执行需要使用 fork 方式。