shell 传递参数 单双引号区别

shell 传递参数

$0 对应的是当前 Shell 脚本程序的名称，$#对应的是总共有几个参数，$* 对应的是所有位置的参数值，$? 对应的是显示上一次命令的执行返回值，而 $1、$2、$3…… 则分别对应着第 N 个位置的参数值，如图 所示。

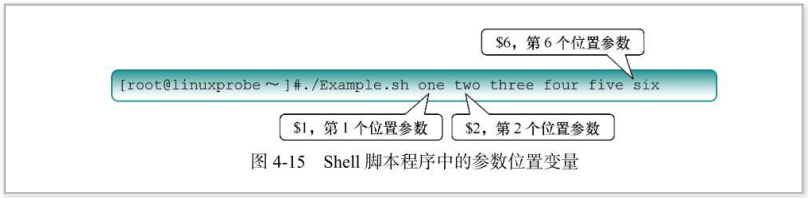

图 4-15 Shell 脚本程序中的参数位置变量

举例子说明：

```
[root@localhost home]# cat litao.sh
#!/bin/bash
echo  "这个是脚本对应的名称 $0。"
echo  "总共有$#个参数，每个参数是$*"
echo  "第一个参数为$1，第六个参数是$6"
[root@localhost home]# sh litao.sh one two therr four five six
这个是脚本对应的名称 litao.sh。
总共有6个参数，每个参数是one two therr four five six
第一个参数为one，第六个参数是six
[root@localhost home]#
```

在上面例子中，echo 内容在""（双引号）中，如果''（单引号）中。看下运行结果

```
[root@localhost home]# cat litao.sh
#!/bin/bash
echo  ' 这个是脚本对应的名称 $0。'
echo  ' 总共有$#个参数，每个参数是$* '
echo  ' 第一个参数为$1，第六个参数是$6 '
[root@localhost home]# sh litao.sh one two therr four five six
这个是脚本对应的名称 $0。
总共有$#个参数，每个参数是$*
第一个参数为$1，第六个参数是$6
[root@localhost home]#
```

其中，被赋值的变量按照脚本内容一模一样打印出来了。对比单引号和双引号可参考博文： linux shell 中的单引号与双引号的区别 (看完就不会有引号的疑问了)

在 shell prompt 敲打键盘、直到按下 enter 的时候，输入的字符就是 command line（命令行）了，然后 shell 才会以进程方式执行你所提交的命令。在 command line 输入的每一个文字，对 shell 来说，有什么类别之分呢？command line 的每一个 charactor（字符）分为两种：1.*literal（字面意义） 2.*meta（元）

① *literal：也就是普通纯文字，对 shell 来说没有特殊功能。
② *meta：对 shell 来说，具有特定功能的保留字。

凡是 abcd、123456 等这些 “文字” 都是 literal。几乎每次都会碰到的 meta 组成如下：
*IFS：由 <space> <tab> <enter> 三者之一组成（我们常用 space）。
*CR：由 <enter> 产生。

IFS 是用来拆分 command line 的每一个词（word）用的，因为 shell command line 是按词来处理的。而 CR 则是用来结束 command line 用的，这也是为何我们敲 <enter> 命令就会执行的原因。除了 IFS 和 CR 外，常用的 meta 还有：

= : 设定变量。
$: 做变量或运算替换 (请不要与 shell prompt 搞混了)。
> : 重定向 stdout。
< : 重定向 stdin。
| : 管道命令。

& ： 重定向 file descriptor ，或将命令置于后台执行。

( ): 将其内的命令置于 nested subshell 执行，或用于运算或命令替换。

{ }: 将其内的命令置于 non-named function 中执行，或用在变量替换的界定范围。

; ： 在前一个命令结束时，而忽略其返回值，继续执行下一個命令。

&& ： 在前一個命令结束时，若返回值为 true，继续执行下一個命令。

|| ： 在前一個命令结束时，若返回值为 false，继续执行下一個命令。

!: 执行 history 列表中的命令

假如我们要在 command line 中将这些保留元字符的功能关闭的话，就要用到 quoting 处理了。

在 bash 中，我们常用的 quoting 有如下三种方法:

*hard quote: ' '（单引号）：凡在 hard quote 中的所有 meta 均被关闭。

*soft quote: " "（双引号）：在 soft quote 中的大部分 meta 都会被关闭，但某些保留（如 $）。

*escape: \ （反斜线）：只有紧接在 escape（跳脱字符）之后的单一 meta 才被关闭。

下面的例子將有助于我们对 quoting 的了解:

```
$ A=B C        # 空白键未被关闭，作为 IFS 处理。
$ C: command not found.
$ echo $A

$ A="B C"        # 空白键已被关闭，仅作空白符号处理。
```

```
$ echo $A
B C
```

在第一次设定 A 变量时，由于空白键没有被关闭，command line 将被解读为：

* A=B 然后碰到 <IFS>，再执行 C 命令

在第二次设定 A 变量时，由于空白键置于 soft quote 中，因此被关闭，不再作为 IFS ：

* A=B<space>C

事实上，空白键无论在 soft quote 还是在 hard quote 中，均会被关闭。Enter 鍵亦然：

```
$ A='B
> C
> '
$ echo "$A"
B
C
```

在上例中，由于 <enter> 被置于 hard quote 当中，因此不再作为 CR 字符來处理。

这里的 <enter> 单纯只是一个断行符号 (new-line) 而已，由于 command line 并沒得到 CR 字符，因此进入第二個 shell prompt (PS2，以> 符号表示)，command line 并不会结束，

直到第三行，我们输入的 <enter> 并不在 hard quote 里面，因此并沒被关闭，

此时，command line 碰到 CR 字符，于是结束、交给 shell 來处理。

上例的 <enter> 要是被置于 soft quote 中的话，CR 也会同样被关闭：

```
$ A="B
> C
> "
$ echo $A
```

B C

然而，由于 echo $A 时的变量沒置于 soft quote 中，<enter> 没有得到 CR 字符；会被解释为 IFS ，而不是解释为 New Line 字符。

总结：

command line 的每一个 charactor 分两种，分别是 *literal *meta 。其中 literal 指的是普通纯文字，例如 abcd、123456 等这些 “文字” 都是 literal。

而 meta 对于 shell 来说，具有特定功能的保留字，我们经常碰到的 meta 分为 IFS CR；区别如下

*IFS：由 <space> <tab> <enter > 三者之一组成（我们常用 space）。 作用：IFS 是用来拆分 command line 的每一个词（word）用的
*CR：由 <enter> 产生。 作用：CR 结束 command line 用的，这也是为何我们敲 < enter > 命令就会执行的原因。

除了 IFS，CR。常用的 meta 还有：

= ： 设定变量。
$ ： 做变量或运算替换 (请不要与 shell prompt 搞混了)。
> ： 重定向 stdout。
< ： 重定向 stdin。
|： 管道命令。
& ： 重定向 file descriptor ，或将命令置于后台执行。
( )： 将其内的命令置于 nested subshell 执行，或用于运算或命令替换。
{ }： 将其内的命令置于 non-named function 中执行，或用在变量替换的界定范围。

；：　在前一个命令结束时，而忽略其返回值，继续执行下一個命令。

&& ：　在前一個命令结束时，若返回值为 true，继续执行下一個命令。

|| ：　在前一個命令结束时，若返回值为 false，继续执行下一個命令。

!:　　执行 history 列表中的命令

1、单引号、双引号用于用户把带有空格的字符串赋值给变量的分界符。

```
[root@localhost sh]# str="Today is Monday"
[root@localhost sh]# echo $str
Today is Monday
```

如果没有单引号或双引号，shell 会把空格后的字符串解释为命令。

```
[root@localhost sh]# str=Today is Monday
bash: is: command not found
```

2、单引号和双引号的区别。单引号告诉 shell 忽略所有特殊字符，而双引号忽略大多数，但不包括　$　　\　　`

双引号中的'$'（参数替换）和'`'（命令替换）是例外，所以，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 `。

如：num=3

```
echo　‘$num’

 $num
```