Chapter 5. bash引用-Quoting详细介绍

概述

- 引用的字面意思就是，用引号括住一个字符串。这可以保护字符串中的特殊字符不被shell或shell脚本重新解释或扩展。(如果一个字有不同于其字面意思的解释，它就是“特殊的”。例如：星号*除了本身代表*号以外还表示文件通配和正则表达式中的通配符)。

```
[root@centos8 ~]$ ls -l P*
Pictures:
total 0
Public:
total 0
[root@centos8 ~]$ ls -l 'P*'
ls: cannot access 'P*': No such file or directory
```

- 在生活中用语或者书写，当我们使用双引号"引用"一个句子时，我们会区别对待该句子并赋予其特殊意义；在Bash脚本中，当我们使用双引号"string"引用一个字符串时，我们同样区别对待并保护其字面意思（一般性的意思）。
- 在涉及到命令替换时，引用可以让echo输出带格式的命令结果,保留变量所带的空白符号。

```
bash$ echo $(ls -l)                 # 无引号命令替换
total 8 -rw-rw-r-- 1 bo bo 13 Aug 21 12:57 t.sh -rw-rw-r-- 1 bo bo 78 Aug 21 12:57
u.sh
bash$ echo "$(ls -l)"               # 被引用的命令替换
total 8
 -rw-rw-r--  1 bo bo  13 Aug 21 12:57 t.sh
 -rw-rw-r--  1 bo bo  78 Aug 21 12:57 u.sh
```

5.1. 引用变量(Quoting Variables)

- 当我们引用一个变量时，比较建议的做法是使用双引号将其引用起来。这样做可以避免bash再次解析双引号中的特殊字符（只不过：$、反引号`、和反斜杠\仍然会被bash解析）。在双引号中的$符号仍然被视为特殊字符，这样做的好处是可以进行变量替换("$variable"),也就是使用变量的值替换掉变量名。
- 使用双引号的另一个用途是防止单词分割。在双引号中的参数表现为单个词语(即使其包含空白字符)。

```
List="one two three"
for a in $List     # 使用空格符作为分隔符分割变量的值(字符串).
do
  echo "$a"
done
```

```
# 输出
# one
# two
# three
echo "---"
for a in "$List"   # Preserves whitespace in a single variable.
do #     ^     ^
  echo "$a"
done
# 输出
# one two three
```

- 下面是一个更加精心设计的例子

```
variable1="a variable containing five words"
COMMAND This is $variable1    # 执行COMMAND命令时会被认为带了7个参数如下:
# "This" "is" "a" "variable" "containing" "five" "words"
COMMAND "This is $variable1"  # # 执行COMMAND命令时会被认为带了1个参数如下:
# "This is a variable containing five words"
variable2=""    # 空变量.
COMMAND $variable2 $variable2 $variable2
               # COMMAND命令没带参数执行.
COMMAND "$variable2" "$variable2" "$variable2"
               # 带了三个空参数执行COMMAND命令.
COMMAND "$variable2 $variable2 $variable2"
               # COMMAND命令带一个参数执行(该参数为2个空格).
# 谢谢这个大佬指出:Stéphane Chazelas.
```

- 使用echo给标准输出打印奇怪的字符

例 5-1. echo输出奇怪的变量

```
#!/bin/bash
# weirdvars.sh: Echoing weird variables.
echo
var="'(]\\{}\$\""
echo $var        # '(]\{}$"
echo "$var"      # '(]\{}$"     结果相同.
echo
IFS='\'
echo $var        # '(] {}$"     \ 变为了空格，为什么?(IFS为内部域分割符，临时使用'\'作为分隔符)
echo "$var"      # '(]\{}$"
# 以上例子来自Stephane Chazelas.
echo
var2="\\\\\""
echo $var2       #   "
echo "$var2"     # \\"
echo
```

```
# 但是 ... var2="\\\\"" 是非法的？（此处四个\，位置2,4的两个刚好被转义，而剩余3个"符，
所以不行）
var3='\\\\'
echo "$var3"     # \\\\
# 但是强引用可以.
# ************************************************************ #
# 变量嵌套替换也可以
echo "$(echo '"')"           # "
#    ^           ^
# 某些情况下嵌套很有用
var1="Two bits"
echo "\$var1 = "$var1""      # $var1 = Two bits
#    ^               ^
# Or, as Chris Hiestand points out ...
if [[ "$(du "$My_File1")" -gt "$(du "$My_File2")" ]]
#     ^     ^         ^ ^     ^     ^         ^ ^
then
  ...
fi
# ************************************************************ #
```

- 单引号(' ')的工作机制类似于双引号，但是在单括号中不允许变量替换，因为$符的特殊意义被关闭了。在单引号中任何特殊的符号都按照字面意思解释(除了'单引号自身)。
- 由于转义符(逃逸符)在单引号中都失去了转义的意义，所以试图在单引号中括单引号是不行的，下面的例子可以实现输出单引号。

```
echo "Why can't I write 's between single quotes"
echo
# The roundabout method.
echo 'Why can'\''t I write '"'"'s between single quotes'
#    |-------|  |----------|   |----------------------|
# 上面使用了三个单引号组；一个转义的单引号和一个双引号引起的单引号.
```

5.2. 转义/逃逸(Escaping)

- 转义是一种用来引用单个字符的方法。在字符前的反斜杠\告诉shell以字面意思解析该字符。

> 注意：在echo或者sed这些程序命令中，转义某个字符可能有相反的作用，可以触发某种特殊意义。

> 特定的被转义的字符所具有的意义如下:

```
used with echo and sed
\n 新行(means newline)
\r 回车(means return)
\t tab键
\v 垂直tab键(means vertical tab)
\b 退格(means backspace)
```

```
\a 报警(means alert (beep or flash))
\0xx 将反斜杠后的数字视为八进制的ASCII码值
```

- $' ... ' :该符号结构的机制是使用转义的八进制或者十六进制值将ASCII码赋给变量；例如：quote=$'\042'.

例5-2. 被转义的字符(Escaped Characters)

```
#!/bin/bash
# escaped.sh: escaped characters
###############################################################
### First, let's show some basic escaped-character usage. ###
###############################################################
# Escaping a newline.
# ------------------
echo ""
echo "This will print
as two lines."
# 上面的写法将会打印两行
echo "This will print \
as one line."
# 上面的写法将会打印一行
echo; echo
echo "=============="
echo "\v\v\v\v"      # 按照字面意思打印 \v\v\v\v .
# 使用 -e 选项 打印转义字符所代表的字符
echo "=============="
echo "VERTICAL TABS" # 垂直tab键
echo -e "\v\v\v\v"   # 此时将会打印四个垂直tab
echo "=============="
echo "QUOTATION MARK"
echo -e "\042"       # 打印一个双引号 " (quote, 八进制的 ASCII 值:42;代表一个双引号).
echo "=============="
# 使用该结构 $'\X' 使得-e选项不在需要
echo; echo "NEWLINE and (maybe) BEEP"
echo $'\n'           # 新行.
echo $'\a'           # 警告音(beep).
                     # May only flash, not beep, depending on terminal.
# We have seen $'\nnn" string expansion, and now . . .
# ============================================================== #
# $'\nnn' 该种字符串展开的机制在bash2.0中引进
# ============================================================== #
echo "Introducing the \$\' ... \' string-expansion construct . . . "
echo ". . . featuring more quotation marks."
echo $'\t \042 \t'   # 打印左右两边分别有个tab键的双引号(").
# '\nnn' 为八进制数.
echo
# 将一个ASCII字符赋值给一个变量
# ------------------------------------
quote=$'\042'        # " 赋值给一个变量
echo "$quote Quoted string $quote and this lies outside the quotes."
echo
# Concatenating ASCII chars in a variable.
```

```
triple_underline=$'\137\137\137'  # 137 是八进制的ASCII值，代表'_'.
echo "$triple_underline UNDERLINE $triple_underline"
echo
ABC=$'\101\102\103\010'           # 101, 102, 103 分别代表 A, B, C.
echo $ABC
echo
escape=$'\033'                    # 033 是escape的八进制表示.
echo "\"escape\" echoes as $escape"
#                                   并无可视化的输出.
echo
exit 0
```

- 一个更加精心设计的例子

例5-3. 检测按键(Detecting key-presses)

```
#!/bin/bash
# Author: Sigurd Solaas, 20 Apr 2011
# Used in ABS Guide with permission.
# Requires version 4.2+ of Bash.
key="no value yet"
while true; do
  clear
  echo "Bash Extra Keys Demo. Keys to try:"
       #Bash 识别按键的demo.可以识别一下按键:
  echo
  echo "* Insert, Delete, Home, End, Page_Up and Page_Down"
  echo "* The four arrow keys"
       #四个方向键
  echo "* Tab, enter, escape, and space key"
       #tab,回车，返回，空格键
  echo "* The letter and number keys, etc."
       #标点按键
  echo
  echo "    d = show date/time"
  echo "    q = quit"
  echo "================================="
  echo
 # Convert the separate home-key to home-key_num_7:
 if [ "$key" = $'\x1b\x4f\x48' ]; then
  key=$'\x1b\x5b\x31\x7e'
  #   Quoted string-expansion construct.
 fi
 # Convert the separate end-key to end-key_num_1.
 if [ "$key" = $'\x1b\x4f\x46' ]; then
  key=$'\x1b\x5b\x34\x7e'
 fi
 case "$key" in
  $'\x1b\x5b\x32\x7e')  # Insert
   echo Insert Key
  ;;
  $'\x1b\x5b\x33\x7e')  # Delete
```

```
    echo Delete Key
   ;;
   $'\x1b\x5b\x31\x7e')  # Home_key_num_7
    echo Home Key
   ;;
   $'\x1b\x5b\x34\x7e')  # End_key_num_1
    echo End Key
   ;;
   $'\x1b\x5b\x35\x7e')  # Page_Up
    echo Page_Up
   ;;
   $'\x1b\x5b\x36\x7e')  # Page_Down
    echo Page_Down
   ;;
   $'\x1b\x5b\x41')  # Up_arrow
    echo Up arrow
   ;;
   $'\x1b\x5b\x42')  # Down_arrow
    echo Down arrow
   ;;
   $'\x1b\x5b\x43')  # Right_arrow
    echo Right arrow
   ;;
   $'\x1b\x5b\x44')  # Left_arrow
    echo Left arrow
   ;;
   $'\x09')  # Tab
    echo Tab Key
   ;;
   $'\x0a')  # Enter
    echo Enter Key
   ;;
   $'\x1b')  # Escape
    echo Escape Key
   ;;
   $'\x20')  # Space
    echo Space Key
   ;;
   d)
    date
   ;;
   q)
   echo Time to quit...
   echo
   exit 0
   ;;
   *)
    echo You pressed: \'"$key"\'
   ;;
  esac
  echo
  echo "================================"
  unset K1 K2 K3
  read -s -N1 -p "Press a key: "
```

```
  K1="$REPLY"
  read -s -N2 -t 0.001
  K2="$REPLY"
  read -s -N1 -t 0.001
  K3="$REPLY"
  key="$K1$K2$K3"
done
exit $?
```

\" 还原双引号的字面意思(就是双引号，不在用于引用)

```
echo "Hello"                      # Hello
echo "\"Hello\" ... he said."     # "Hello" ... he said.
```

$ 还原$符的字面意思，意思是在$后的变量不会被替换

```
echo "\$variable01"           # $variable01
echo "The book cost \$7.98."  # The book cost $7.98.
```

\\ 还原\的字面意思

```
echo "\\"  # Results in \
# Whereas . . .
echo "\"   # Invokes secondary prompt from the command-line.
           # In a script, gives an error message.
# However . . .
echo '\'   # Results in \
```

- 反斜杠\在不同的情况下表现出不一样的行为(是否被转义；是否位于强引用中；是否在弱引用中；是否在命令替换中；是否在“here document”)

```
                    #  简单的转义和引用
echo \z             #  z
echo \\z            # \z
echo '\z'           # \z
echo '\\z'          # \\z
echo "\z"           # \z
echo "\\z"          # \z
                    #  命令替换
echo ècho \z`       #  z
echo ècho \\z`      #  z
echo ècho \\\z`     # \z
echo ècho \\\\z`    # \z
echo ècho \\\\\\z`  # \z
echo ècho \\\\\\\z` # \\z
```

```
echo ècho "\z"`       # \z
echo ècho "\\z"`      # \z
                       # 此处文本(Here document)
cat <<EOF
\z
EOF                    # \z
cat <<EOF
\\z
EOF                    # \z
# These examples supplied by Stéphane Chazelas.
```

- 赋给某个变量的字符串中的某些元素可能会被(能够被)转义，但是单个被转义的字符不一定可以赋给变量。

```
variable=\
echo "$variable"
# 不可行，打印一个错误提示:
# test.sh: : command not found
# 单独一个转义字符不可赋给变量
#
#  事实上此处转义字符 "\" 转义了新行（在编写比较长的命令时就可以使用一个转义符来将命令写成多行）
#+ 真正的行为是:          variable=echo "$variable"
#+                       把命令赋值给变量是不允许的
variable=\
23skidoo
echo "$variable"         #  23skidoo
                         #  可行，因为第二行是合法的变量，且实际上第二行被转义成了第一。
variable=\
#        \^    转义符后跟一个空格
echo "$variable"         # 输出空格
variable=\\
echo "$variable"         # \
variable=\\\
echo "$variable"
# 行不通，出错:
# test.sh: \: command not found
#
#  上面三个转义符中，第一个转义可第二个，但是第三个转义符还在，类似于第一个例子。
variable=\\\\
echo "$variable"         # \\
                         # Second and fourth escapes escaped.
                         # This is o.k.
```

- 转义空格可以防止命令的参数列表发生词语分割。
- Escaping a space can prevent word splitting in a command's argument list.

```
file_list="/bin/cat /bin/gzip /bin/more /usr/bin/less /usr/bin/emacs-20.7"
# List of files as argument(s) to a command.

# Add two files to the list, and list all.
ls -l /usr/X11R6/bin/xsetroot /sbin/dump $file_list
echo "------------------------------------------------------------------------"
# What happens if we escape a couple of spaces?
ls -l /usr/X11R6/bin/xsetroot\ /sbin/dump\ $file_list
输出:
################################
[root@centos8 ~]#ls -l /usr/X11R6/bin/xsetroot\ /sbin/dump\ $file_list
ls: cannot access '/usr/X11R6/bin/xsetroot /sbin/dump /bin/cat': No such file or
directory
################################
# Error: the first three files concatenated into a single argument to 'ls -l'
#        because the two escaped spaces prevent argument (word) splitting.
```

- 转义也有"等待多行命令"的意思。一般的，不同的行会是不同的命令，但是行末的转义字符转义了新行的字符，命令可以一直写到下一行。

```
(cd /source/directory && tar cf - . ) | \
(cd /dest/directory && tar xpvf -)
# 该命令为Alan Cox写的拷贝命令，作两行写，增加了易读性。
# 下面的命令同样功能:
tar cf - -C /source/directory . |
tar xpvf - -C /dest/directory
# See note below.
# (Thanks, Stéphane Chazelas.)
```

- 如果脚本行结束跟一个|,一个管道符号，那么转义字符\，就不是那么严格的需要了。但是后面跟上转义字符是比较好的习惯。

```
echo "foo
bar"
##两行
#foo
#bar
echo
echo 'foo
bar'    # 仍然是两行
#foo
#bar
echo
echo foo\
bar     # 换行符被转义输出一行.
#foobar
echo
echo "foo\
```

```
bar"     # 同样是一行，转义字符在弱引用中("\")不会丢掉其转义字符的特殊意义。
#foobar
echo
echo 'foo\
bar'     # 两行，因为转义字符在强引用('\')中失去了转义字符的意义，被bash按照字面意义解
释。
#foo\
#bar
# Examples suggested by Stéphane Chazelas.
```