

引号和转义

网道 (WangDoc.com) , 互联网文档计划

Bash 只有一种数据类型，就是字符串。不管用户输入什么数据，Bash 都视为字符串。因此，字符串相关的引号和转义，对 Bash 来说就非常重要。

目录 [隐藏]

- 1. 转义
- 2. 单引号
- 3. 双引号
- 4. Here 文档
- 5. Here 字符串

1. 转义

某些字符在 Bash 里面有特殊含义（比如 `$`、`&`、`*`）。

```
$ echo $date
```

```
$
```

上面例子中，输出 `$date` 不会有任何结果，因为 `$` 是一个特殊字符。

如果想要原样输出这些特殊字符，就必须在它们前面加上反斜杠，使其变成普通字符。这就叫做“转义”（escape）。

📖 Bash 脚本教程

- 📖 1. 简介
- 📖 2. 基本语法
- 📖 3. 模式扩展
- 📖 4. 引号和转义
- 📖 5. 变量
- 📖 6. 字符串操作
- 📖 7. 算术运算
- 📖 8. 操作历史
- 📖 9. 行操作
- 📖 10. 目录堆栈
- 📖 11. 脚本入门
- 📖 12. read 命令
- 📖 13. 条件判断
- 📖 14. 循环
- 📖 15. 函数
- 📖 16. 数组
- 📖 17. set 命令, shopt 命令

```
$ echo \$date
$date
```

上面命令中，只有在特殊字符 `$` 前面加反斜杠，才能原样输出。

反斜杠本身也是特殊字符，如果想要原样输出反斜杠，就需要对它自身转义，连续使用两个反斜线（`\\`）。

```
$ echo \\
\
```

上面例子输出了反斜杠本身。

反斜杠除了用于转义，还可以表示一些不可打印的字符。

- `\a`：响铃
- `\b`：退格
- `\n`：换行
- `\r`：回车
- `\t`：制表符


如果想要在命令行使用这些不可打印的字符，可以把它们放在引号里面，然后使用 `echo` 命令的 `-e` 参数。


```
$ echo a\tb
atb


$ echo -e "a\tb"
a      b
```


上面例子中，命令行直接输出不可打印字符 `\t`，Bash 不能正确解释。必须把它们放在引号之中，然后使用 `echo` 命令的 `-e` 参数。

换行符是一个特殊字符，表示命令的结束，Bash 收到这个字符以后，就会对输入的命令进行解释执行。换行符前面加上反斜杠转义，就使得换行符变成一个普通字符，Bash 会将其当作长度为 0 的空字符处理，从而可以将一行命令写成多行。

 **18.** 脚本除错

 **19.** mktemp 命令，trap 命令

 **20.** 启动环境

 **21.** 命令提示符

链接

 本文源码

 代码仓库

 反馈

```
$ mv \  
/path/to/foo \  
/path/to/bar  
  
# 等同于  
$ mv /path/to/foo /path/to/bar
```

上面例子中，如果一条命令过长，就可以在行尾使用反斜杠，将其改写成多行。这是常见的多行命令的写法。

2. 单引号

Bash 允许字符串放在单引号或双引号之中，加以引用。

单引号用于保留字符的字面含义，各种特殊字符在单引号里面，都会变为普通字符，比如星号（`*`）、美元符号（`$`）、反斜杠（`\`）等。

```
$ echo '*'  
*  
  
$ echo '$USER'  
$USER  
  
$ echo '$((2+2))'  
$((2+2))  
  
$ echo '$(echo foo)'  
$(echo foo)
```

上面命令中，单引号使得 Bash 扩展、变量引用、算术运算和子命令，都失效了。如果不使用单引号，它们都会被 Bash 自动扩展。

由于反斜杠在单引号里面变成了普通字符，所以如果单引号之中，还要使用单引号，不能使用转义，需要在外层的单引号前面加上一个美元符号（`$`），然后再对里层的单引号转义。

```
# 不正确
$ echo it's

# 不正确
$ echo 'it\'s'

# 正确
$ echo $'it\'s'
```

不过，更合理的方法是改在双引号之中使用单引号。

```
$ echo "it's"
it's
```

3. 双引号

双引号比单引号宽松，大部分特殊字符在双引号里面，都会失去特殊含义，变成普通字符。

```
$ echo "*"
*
```

上面例子中，通配符 `*` 是一个特殊字符，放在双引号之中，就变成了普通字符，会原样输出。这一点需要特别留意，这意味着，双引号里面不会进行文件名扩展。

但是，三个特殊字符除外：美元符号（`$`）、反引号（```）和反斜杠（`\`）。这三个字符在双引号之中，依然有特殊含义，会被 Bash 自动扩展。

```
$ echo "$SHELL"
/bin/bash

$ echo "`date`"
Mon Jan 27 13:33:18 CST 2020
```

上面例子中，美元符号（`$`）和反引号（```）在双引号中，都保持特殊含义。美元符号用来引用变量，反引号则是

执行子命令。

```
$ echo "I'd say: \"hello.\""  
I'd say: "hello."  
  
$ echo "\\ "  
\  

```

上面例子中，反斜杠在双引号之中保持特殊含义，用来转义。所以，可以使用反斜杠，在双引号之中插入双引号，或者插入反斜杠本身。

换行符在双引号之中，会失去特殊含义，Bash 不再将其解释为命令的结束，只是作为普通的换行符。所以可以利用双引号，在命令行输入多行文本。

```
$ echo "hello  
world"  
hello  
world
```

上面命令中，Bash 正常情况下会将换行符解释为命令结束，但是换行符在双引号之中就失去了这种特殊作用，只用来换行，所以可以输入多行。`echo` 命令会将换行符原样输出，显示的时候正常解释为换行。

双引号的另一个常见的使用场合是，文件名包含空格。这时就必须使用双引号（或单引号），将文件名放在里面。

```
$ ls "two words.txt"
```

上面命令中，`two words.txt` 是一个包含空格的文件名，如果不放在双引号里面，就会被 Bash 当作两个文件。

双引号会原样保存多余的空格。

```
$ echo "this is a      test"  
this is a      test
```

双引号还有一个作用，就是保存原始命令的输出格式。

```
# 单行输出
$ echo $(cal)
一月 2020 日 一 二 三 四 五 六 1 2 3 ... 31
```

```
# 原始格式输出
$ echo "$(cal)"
      一月 2020
日 一 二 三 四 五 六
      1  2  3  4
5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

上面例子中，如果 `$(cal)` 不放在双引号之中，`echo` 就会将所有结果以单行输出，丢弃了所有原始的格式。

4. Here 文档

Here 文档（here document）是一种输入多行字符串的方法，格式如下。

```
<< token
text
token
```

它的格式分成开始标记（`<< token`）和结束标记（`token`）。开始标记是两个小于号 + Here 文档的名称，名称可以随意取，后面必须是一个换行符；结束标记是单独一行顶格写的 Here 文档名称，如果不是顶格，结束标记不起作用。两者之间就是多行字符串的内容。

下面是一个通过 Here 文档输出 HTML 代码的例子。

```
$ cat << _EOF_
<html>
<head>
  <title>
    The title of your page
  </title>
```

```
</head>

<body>
    Your page content goes here.
</body>
</html>
_EOF_
```

Here 文档内部会发生变量替换，同时支持反斜杠转义，但是不支持通配符扩展，双引号和单引号也失去语法作用，变成了普通字符。

```
$ foo='hello world'
$ cat << _example_
$foo
"$foo"
'$foo'
_example_

hello world
"hello world"
'hello world'
```

上面例子中，变量 `$foo` 发生了替换，但是双引号和单引号都原样输出了，表明它们已经失去了引用的功能。

如果不希望发生变量替换，可以把 Here 文档的开始标记放在单引号之中。

```
$ foo='hello world'
$ cat << '_example_'
$foo
"$foo"
'$foo'
_example_

$foo
"$foo"
'$foo'
```

上面例子中，Here 文档的开始标记（`_example_`）放在单引号之中，导致变量替换失效了。

Here 文档的本质是重定向，它将字符串重定向输出给某个命令，相当于包含了 `echo` 命令。

```
$ command << token
string
token

# 等同于

$ echo string | command
```

上面代码中，Here 文档相当于 `echo` 命令的重定向。

所以，Here 字符串只适合那些可以接受标准输入作为参数的命令，对于其他命令无效，比如 `echo` 命令就不能用 Here 文档作为参数。

```
$ echo << _example_
hello
_example_
```

上面例子不会有任何输出，因为 Here 文档对于 `echo` 命令无效。

此外，Here 文档也不能作为变量的值，只能用于命令的参数。

5. Here 字符串

Here 文档还有一个变体，叫做 Here 字符串（Here string），使用三个小于号（`<<<`）表示。

```
<<< string
```

它的作用是将字符串通过标准输入，传递给命令。

有些命令直接接受给定的参数，与通过标准输入接受参数，结果是不一样的。所以才有了这个语法，使得将字符串通过

标准输入传递给命令更方便，比如 `cat` 命令只接受标准输入传入的字符串。

```
$ cat <<< 'hi there'
# 等同于
$ echo 'hi there' | cat
```

上面的第一种语法使用了 Here 字符串，要比第二种语法看上去语义更好，也更简洁。

```
$ md5sum <<< 'ddd'
# 等同于
$ echo 'ddd' | md5sum
```

上面例子中，`md5sum` 命令只能接受标准输入作为参数，不能直接将字符串放在命令后面，会被当作文件名，即 `md5sum ddd` 里面的 `ddd` 会被解释成文件名。这时就可以用 Here 字符串，将字符串传给 `md5sum` 命令。

◀ 模式扩展

变量 ▶

本教程采用知识共享 署名-相同方式共享 3.0协议。

分享本文      

联系：contact@wangdoc.com