

# Shell 编程学习笔记

Author: Marlous

E-mail: [goonecat@foxmail.com](mailto:goonecat@foxmail.com)

Date: 2018/10/26

[参考教程 1](#) [参考博文 1](#) [参考博文 2](#) [参考博文 3](#) [慕课网课程 shell 编程系列](#)

## 一 shell 注释模板与运行

### 1 注释模板

```
#!/bin/bash
# -----
# name: login.sh
# version: 1.0
# createTime: 2018/10/12
# description: shell 脚本的功能描述
# author: Marlous
# email: example@gmail.com
# github: https://github.com/test
# -----
```

### 2 运行 shell 脚本

```
cd 到该脚本目录下
chmod +x ./test.sh    #给脚本执行权限
./test.sh             #执行该脚本

或
sh test.sh
```

## 二 shell 编程

### 1 变量

分为用户自定义变量、环境变量、预定义变量。shell 不 declare 话中默认为字符串类型。

1. 变量赋值与引用:

给变量赋值: num=12 或 string="zifuchuan"

引用变量: \${num} 用花括号把变量包裹起来, 看情况也可以不加, 推荐加。

只读变量: readonly num

删除变量: unset num

读取变量: (可加参数)

```
read -p "Please enter:" name
echo $name
```

显示环境变量:

```
set
```

增加变量值:

```
PATH="$PATH":值
```

# 系统环境变量使用

```
curTime=$(date "+%H%M%S") #将时间 时 分 秒 赋给变量 curTime, 注意 date 和 双引号之间有空格。
```

```
curDate=$(date "+%Y%m%d") #将时间 年 月 日 赋给变量 curDate。
```

补充:

查看语系:

```
locale
```

-a 查看支持的语系

设置主语系变量:

```
LANG
```

第三方插件支持中文: 如 zhcon

2. 变量声明 (可进行数值运算的类型):

```
declare [+|-] 选项 变量名
```

- - : 给变量设定类型属性
- + : 取消变量的类型属性
- -a : 将变量声明为数组型
- -i : 将变量声明为整数型 (integer)
- -x : 将变量声明为环境变量
- -r : 将变量声明为只读变量
- -p : 显示指定变量的被声明的类型

export 声明变量 (加在变量前), 可以让别的程序或脚本调用。等于 `declare -x`。

3. 字符串:

可以不用引号，也可以用单引号或双引号（用双引号，其中可以有变量、转义字符；不用双引号则原样输出）。

#### 4. 数组:

给数组赋值: `array_name=(value0 value1 value2 value3)` 或 `array_name[0]=value0`

读取数组元素: 单个元素 `${array_name[n]}` 全部值用 `${array_name[@]}`

获取数组长度: `${#array_name[@]}`

#### 5. 注释:

- 单行注释: `#`
- 多行注释:

`:<<符号`

符号

# 例: EOF 也可以是 `!`、`'` (感叹号、反引号等) 。

`:<<EOF`

EOF

## 2 接收参数

- `$0` 为执行的文件名, `$1` 到 `$n` 为第一个参数到第 `n` 个参数。
- 其他特殊参数:

`$#` 传递到脚本的参数个数。  
`$*` 以一个单字符串显示所有向脚本传递的参数。  
`$$` 脚本运行的当前进程 ID 号。  
`$!` 后台运行的最后一个进程的 ID 号。  
`$@` 与 `$*` 相同, 但是使用时加引号, 并在引号中返回每个参数。  
`$-` 显示 Shell 使用的当前选项, 与 `set` 命令功能相同。  
`$?` 显示最后命令的退出状态。0 表示没有错误, 其他任何值表明有错误。

## 3 运算符

#### 1. 算术运算:

加、减、乘、除、取模、赋值、相等、不相等。

`+` `-` `*` `/` `%` `=` `==` `!=`

## 2. 关系运算符:

- eq 检测两个数是否相等, 相等返回 true。
- ne 检测两个数是否相等, 相等返回 true。
- gt 检测左边的数是否大于右边的, 如果是, 则返回 true。
- lt 检测左边的数是否小于右边的, 如果是, 则返回 true。
- ge 检测左边的数是否大于等于右边的, 如果是, 则返回 true。
- le 检测左边的数是否小于等于右边的, 如果是, 则返回 true。

## 3. 布尔 (逻辑) 运算符:

- ! 非运算, 表达式为 true 则返回 false, 否则返回 true。
- o 或运算, 有一个表达式为 true 则返回 true。
- a 与运算, 两个表达式都为 true 才返回 true。

## 4. 逻辑运算符:

- && 逻辑与
- || 逻辑或

## 5. 字符串运算符:

- = 检测两个字符串是否相等, 相等返回 true。
- != 检测两个字符串是否相等, 不相等返回 true。
- z 检测字符串长度是否为 0, 为 0 返回 true。
- n 检测字符串长度是否为 0, 不为 0 返回 true。
- str 检测字符串是否为空, 不为空返回 true。

## 6. 文件测试运算符:

- b file 检测文件是否是块设备文件, 如果是, 则返回 true。 例: [-b \$file] 或 test -b \$file
- c file 检测文件是否是字符设备文件, 如果是, 则返回 true。
- d file 检测文件是否是目录, 如果是, 则返回 true。
- f file 检测文件是否是普通文件 (既不是目录, 也不是设备文件), 如果是, 则返回 true。
- g file 检测文件是否设置了 SGID 位, 如果是, 则返回 true。
- k file 检测文件是否设置了粘着位(Sticky Bit), 如果是, 则返回 true。
- p file 检测文件是否是有名管道, 如果是, 则返回 true。
- u file 检测文件是否设置了 SUID 位, 如果是, 则返回 true。
- r file 检测文件是否可读, 如果是, 则返回 true。
- w file 检测文件是否可写, 如果是, 则返回 true。
- x file 检测文件是否可执行, 如果是, 则返回 true。
- s file 检测文件是否为空 (文件大小是否大于0), 不为空返回 true。
- e file 检测文件 (包括目录) 是否存在, 如果是, 则返回 true。

## 7. shell 中的运算小结:

# 用 expr 进行整数运算

```

echo `expr 1 + 2`
dd=$(expr $a + $b)
dd=$(( $a+$b ))

# 用 $[] 表示数学运算
echo $[ $a+$b ]

# 用 let 指示整数数学运算
let b=1+2
echo $b

# 补充：使用字符截取命令间接操作。浮点数计算用 awk
echo "1 13" | awk '{printf("%0.3f\n", $1/$2)}'
```

## 4 一些命令

命令详见 Linux 命令。

- read 命令： `read variable`，从标准输入中读取一行,并把输入行的每个字段的值指定给 shell 变量。
- echo 命令：输出。
- 显示命令执行结果：用反引号括起来。如 `echo 反引号 某命令 反引号`
- printf 命令，格式化输出字符：与 C 语言相同。

```
printf format-string [arguments...]
```

例：

```

printf "%-10s %-8s %-4s\n" 姓名 性别 体重kg
printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234
printf "%-10s %-8s %-4.2f\n" 杨过 男 48.6543
printf "%-10s %-8s %-4.2f\n" 郭芙 女 47.9876
```

输出结果：

姓名	性别	体重kg
郭靖	男	66.12
杨过	男	48.65
郭芙	女	47.99

- test 命令：判断某个条件是否成立。使用和关系运算符相同。

例：

```

num1=100
num2=100
if test $[num1] -eq $[num2]
then
    echo '两个数相等！'
else
    echo '两个数不相等！'
fi
```

## 5 一些符号

### 参考资料

- 表达式中包含了空格或其他特殊字符，则必须引起来。
- 分号：多个语句之间的分割符号，当只有一个语句（或多行）的时候，末尾无需分号。
- 一些转义字符：`\c` 为显示不换行。
- 原样输出字符串，不进行转义或取变量：用单引括起来。
- 单小括号 `()`：

命令组：

括号中的命令将会新开一个子 shell 顺序执行，所以括号中的变量不能够被脚本余下的部分使用。

括号中多个命令之间用分号隔开，最后一个命令可以没有分号。

命令替换：

等同于 ``cmd``，执行命令，用输出替换原命令位置。

用于初始化数组：

如：`array=(a b c d)`

- 双小括号 `(( ))`：

整数扩展：

`((exp))` 结构扩展并计算一个算术表达式的值（整型）。

表达式的结果为零，返回假或状态码 1；表达式的结果为非零，返回 true 或状态码 0。

括号中的运算符、表达式符合 C 语言运算规则，都可用在 `$((exp))` 中。

如 `echo $((16#5f))` 结果为 95，十六进制转十进制输出。

用 `(( ))` 也可重定义变量值。如 `((a++))`

用于算术运算比较：

双括号中的变量可以不使用 `$` 符号前缀。括号内支持多个表达式用逗号分开。

只要括号中的表达式符合 C 语言运算规则，可以直接使用。如，直接使用 `if (($i<5))` 如果不使用双括号，则为 `if [$i -lt 5]`

- 单中括号 `[]`：

bash 内部命令：

等同于 test 命令。

[ ] 中关系（整数）比较只能使用 -eq 这种形式。

[ ] 中的逻辑与和逻辑或使用 -a 和 -o 表示。

[ ] 中可用字符串比较运算符只有 == 和 != 两种。

无论是字符串比较还是整数比较都不支持大于号小于号；

如果实在想用，对于字符串比较可以使用转义形式，如果比较 "ab" 和 "bc" 为 [ ab \< bc ]，结果为真，也就是返回状态为 0。

字符范围：

用作正则表达式的一部分，test 中不能用正则。

引用数组元素：

引用数组中每个元素的编号。

- 双中括号 `[[ ]]`：

`[[ ]]` 结构比 `[ ]` 结构更加通用。

在 `[[ ]]` 之间所有的字符都不会发生文件名扩展或者单词分割，但是会发生参数扩展和命令替换。

`[[ ]]` 中匹配字符串或通配符，不需要引号。

字符串比较时可以把右边的作为一个模式，不仅是一个字符串，如 `[[hello == hell?]]`，结果为真。

使用 `[[ ]]` 条件判断结构，能防止脚本中的许多逻辑错误。

可直接如 C 语言，`if [[ $a != 1 && $a != 2 ]]`

bash 把双中括号中的表达式看作一个单独的元素，并返回一个退出状态码。

- 大括号 `{ }`：

大括号拓展：

对以逗号，分割的文件列表进行拓展，如 `touch {a,b}.txt` 结果为 `a.txt b.txt`

对以点点 .. 分割的顺序文件列表起拓展作用，如 `touch {a..d}.txt` 结果为 `a.txt b.txt c.txt d.txt`

代码块：

这个结构事实上创建了一个匿名函数。

与小括号中的命令不同，大括号内的命令不会新开一个子 shell 运行，即脚本余下部分仍可使用括号内变量。

括号内的命令间用分号隔开，最后一个也必须有分号。`{ }` 的第一个命令和左括号之间必须要有一个空格。

用来对字符串做处理。

## 6 符号使用小结

1. \$ 后的括号：

`${a}` 变量 `a` 的值，在不引起歧义的情况下可以省略大括号。

`$(cmd)` 命令替换，和 ``cmd`` 效果相同，结果为 shell 命令 `cmd` 的输出，某些 Shell 版本不支持 `$()` 形式的命令替换，如 `tcsh`。

`$((expression))` 和用反引号包裹效果相同，计算数学表达式 `exp` 的数值，其中 `exp` 只要符合 C 语言的运算规则即可，甚至三目运算符和逻辑表达式都可以计算。

## 2. 多条命令执行：

单小括号，`(cmd1;cmd2;cmd3)` 新开一个子 shell 顺序执行命令 `cmd1 cmd2 cmd3`，各命令之间用分号隔开，最后一个命令后可以没有分号。

单大括号，`{ cmd1;cmd2;cmd3;}` 在当前 shell 顺序执行命令 `cmd1 cmd2 cmd3`，各命令之间用分号隔开，最后一个命令后必须有分号，第一条命令和左括号之间必须用空格隔开。

对 `{}` 和 `()` 而言，括号中的重定向符只影响该条命令，而括号外的重定向符影响到括号中的所有命令。

# 7 控制结构

不可为空，else 无操作则不写。

## 1. 选择：(写成一行需要在条件、命令结尾加分号)

- if 选择：

```
if condition1
then
    command1
elif condition2
then
    command2
else
    commandN
fi
```

- case 选择：



```
case 变量名 in
值1)
    command1
    command2
    commandN
    ;;
值2)
    command1
    command2
    commandN
    ;;
esac
```

## 2. 循环:

- for 循环:

```
# 将 item1 item2 等依次赋给变量 var。
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done

# 将字符串中的每个字符分别赋给变量 str。
for str in 'This is a string'
do
    echo $str
done
```

- while 循环:

```
while condition
do
    command
done

# 无限循环:
while true
do
    command
done
```

- until 循环: 循环执行一系列命令直至条件为 true 时停止。

```
until condition
do
    command
done
```

### 3. break 与 continue

## 8 输入输出与重定向

```
# 文件描述符 0 通常是标准输入 (STDIN), 1 是标准输出 (STDOUT), 2 是标准错误输出 (STDERR)。
```

command > file	将输出重定向到 file。
command < file	将输入重定向到 file。
command >> file	将输出以追加的方式重定向到 file。
n > file	将文件描述符为 n 的文件重定向到 file。
n >> file	将文件描述符为 n 的文件以追加的方式重定向到 file。
n >& m	将输出文件 m 和 n 合并。
n <& m	将输入文件 m 和 n 合并。

-----

```
<< tag          将开始标记 tag 和结束标记 tag 之间的内容作为输入。
```

例:

```
command << tag
    document
tag
```

-----

```
重定向到 /dev/null
```

```
> /dev/null
```

```
command > /dev/null 2>&1 # 屏蔽 stdout 和 stderr
```

## 9 函数

调用函数时可以向其传递参数。在函数体内部, 通过 \$n 的形式来获取参数的值。

```
funname()
{
    action;
    return int;
}
```

## 10 调用脚本

- fork: 如果脚本有执行权限的话为 `path/to/test.sh`; 如果没有为 `sh path/to/test.sh`, 子 Shell 中执行, 子从父 Shell 单向继承环境变量, 执行完返回。(类似于批处理中的 start)
- source: `source path/to/test.sh`, 同一个 Shell 中执行, 环境变量可双向获取。(类似于批处理中的 call)
- exec: `exec path/to/test.sh`, 同一个 Shell 内执行, 调用一个新脚本以后, 父脚本中 exec 行之后的内容就不会再执行。

# 11 正则表达式

用于字符串的分割、匹配、查找、替换。

正则用来匹配文件中的字符串，是包含匹配。通配符用来匹配文件名，是完全匹配。

1. shell 正则表达式与通配符：

- 通配符：

- \* 星号代表任意多个或零个字符
- ? 问号代表任意一个字符
- [] 匹配括号中的一个字符

- 基础正则表达式（非扩展）：

元字符	作 用
*	前一个字符匹配0次或任意多次。
.	匹配除了换行符外任意一个字符。
^	匹配行首。例如：^hello会匹配以hello开头的行。
\$	匹配行尾。例如：hello&会匹配以hello结尾的行。
[]	匹配中括号中指定的任意一个字符，只匹配一个字符。 例如：[aoeiu] 匹配任意一个元音字母，[0-9] 匹配任意一位数字，[a-z][0-9] 匹配小写字母和一位数字构成的两位字符。
[^]	匹配除中括号的字符以外的任意一个字符。例如：[^0-9] 匹配任意一位非数字字符，[^a-z]表示任意一位非小写字母。
\	转义符。用于取消讲特殊符号的含义取消。
\{n\}	表示其前面的字符恰好出现n次。例如：[0-9]\{4\} 匹配4位数字，[1][3-8][0-9]\{9\} 匹配手机号码。
\{n, \}	表示其前面的字符出现不小于n次。例如：[0-9]\{2,\} 表示两位及以上的数字。
\{n, m\}	表示其前面的字符至少出现n次，最多出现m次。例如：[a-z]\{6,8\} 匹配6到8位的小写字母。

2. 字符截取命令：cut、printf、awk、sed

```
# 行提取命令 cut
cut 选项 文件名
-f 列号      # 提取第几列
-d 分隔符    # 按指定分隔符分割列

例：
cut -d ":" -f 1,3 /etc/passwd

-----

# 列提取命令 grep

-----

# 格式化输出命令 printf
printf '输出类型输出格式' 输出内容
%ns      # 输出字符串。n 是数字指代输出几个字符。
%ni      # 输出整数。n 是数字指代输出几个数字。
%m.nf    # 输出的整数位和小数位。
```

-----  
# awk

见图。

-----  
# 字符选取、替换、删除、增加命令 sed

见图。

awk 命令：

**# awk ‘条件1{动作1} 条件2{动作2}...’ 文件名**

**条件 ( Pattern ) :**

- 一般使用关系表达式作为条件
- $x > 10$  判断变量  $x$  是否大于 10
- $x \geq 10$  大于等于
- $x \leq 10$  小于等于

**动作 ( Action ) :**

- 格式化输出
- 流程控制语句

sed 命令：

**sed [选项] ‘[动作]’ 文件名**

**选项：**

- **-n**：一般 sed 命令会把所有数据都输出到屏幕，如果加入此选择则只会把经过 sed 命令处理的行输出到屏幕。
- **-e**：允许对输入数据应用多条 sed 命令编辑
- **-i**：用 sed 的修改结果直接修改读取数据的文件，而不是由屏幕输出

**动作：**

- **a**：追加，在当前行后添加一行或多行
- **c**：行替换，用 c 后面的字符串替换原数据行
- **i**：插入，在当期行前插入一行或多行。 **d**：删除，删除指定的行
- **p**：打印，输出指定的行。
- **s**：字符串替换，用一个字符串替换另外一个字符串。格式为“行范围 s/旧字符串/新字符串/g”（和 vim 中的替换格式类似）。

3. 字符处理命令：  
sort、wc 命令。

## 五 shell 脚本示例

---

```
#!/bin/bash
#模拟 linux 登录 shell
echo -n "login:"
read name
echo -n "password:"
read passwd
if [ $name = "cht" -a $passwd = "abc" ];then
echo "the host and password is right!"
else echo "input is error!"
fi
```

```
#!/bin/bash
#删除当前目录下大小为 0 的文件
for filename in `ls`
do
    if test -d $filename
    then b=0
    else
        a=$(ls -l $filename | awk '{ print $5 }')
        if test $a -eq 0
        then rm $filename
        fi
    fi
done
```