



# 如何书写SHELL脚本

关于Linux基础命令，可以查看另一篇博文 [Linux Shell脚本攻略笔记](#)

以下内容，主要是，了解书写shell脚本所需要的大部分知识,主要内容来自于书籍和网络

目的是，能快速书写出需要的shell脚本

开始

version 0.1 2014-01-12 基本内容, 完成度30%

---

资源

[google shell style guide](#)

[Linux Shell脚本攻略笔记](#)

[Linux Shell编程实战技巧](#)

[Bash编程易犯的错误 1234](#)

[关于shell脚本编程的10个最佳实践](#)

[Bash Pitfalls](#)

---

## 第一部分 一些概念

标准IO

### 文件描述符

- 0 标准输入 默认键盘
- 1 标准输出 默认终端
- 2 标准错误 默认终端

## 重定向

- > 输出重定向
- >> 追加到输出重定向
- < 输入重定向
- << 追加到输入重定向

```
ls -l > /tmp/a
```

```
cmd >/dev/null 2>&1 #输出到垃圾桶
```

## 管道

前后连接两个命令

```
ls -l | grep test
```

## 引号

双引号：可以除了字符\$ \外地任何字符或字符串

单引号：忽略任何引用值，将引号里的所有字符作为一个字符串 \$var 不能被解析

反引号：设置系统命令输出到变量

shell脚本识别三种基本命令：内建命令，shell函数和外部命令

基本的命令查找:shell会沿着查找路径\$PATH来寻找命令

```
echo $PATH
```

可以在.profile文件中修改

```
export PATH=$PATH:$HOME/bin
```

## and/or

```
expression1 && expression2 && expression3
```

只有前面一条命令执行成功，才执行下一条

```
expression1执行成功，才执行expression2
```

串联的

```
expression1 || expression2 || expression3
```

执行命令，直到有一条成功为止

---

## 第二部分 shell脚本

首行声明使用bash(声明脚本执行解释器)

```
#!/bin/bash

# do something

exit 0/n
```

运行

```
sh xx.sh
bash xx.sh #大部分情况下两个一样，某些命令只有bash有，只能用这个

or

chmod u+x xx.sh
./xx.sh
```

调试

```
#查看运行时，每个命令回显，执行之后回显
sh -x xx.sh

#执行之前回显
sh -v xx.sh

#检查语法错误，不执行
sh -n xx.sh

#如果使用了未定义的变量，给出错误信息
sh -u xx.sh
```

```
#调试部分脚本
echo "Hello $USER,"
set -x
echo "Today is $(date %Y-%m-%d)"
set +x
```

## 判断执行结果

```
N=$?  #0 <= N <= 255
```

0 无错误，正常执行结束

非0 异常

1-125命令不成功退出

126命令成功，但文件无法执行

127命令找不到

>128命令因收到信号而死亡

## 获取目录名和文件名

```
# To find base directory
APP_ROOT=`dirname "$0"`

# To find the file name
filename=`basename "$filepath"`

# To find the file name without extension
filename=`basename "$filepath" .html`

e.g.
BASEDIR=$(dirname $0)
cd $BASEDIR
CURRENT_DIR=`pwd`
```

## 日期

```
TODAY=`date +%Y%m%d`
DAY_1_AGO=`date -d "$TODAY 1 days ago" +%Y%m%d`
```

常用接受日期/使用默认日期处理

```
if [ -n "$1" ]
then
    TODAY="$1"
else
```

```
TODAY=`date +%Y%m%d`
```

```
fi
```

## crontab调度

查看

```
crontab -l
```

编辑

```
crontab -e
```

格式

```
* * * * * command_path
```

字段	含义	范围
1	分钟	0-59
2	小时	0-23
3	日期	1-31
4	月份	1-12
5	星期几, 0=周日	0-6
6	具体命令, 可以是调用脚本	

\*任意时刻

n1,n2 分割, n1和n2

\*/n 每隔n单位

n1-n2 时段, 一个时段内

```
0 */2 * * * sh run.sh 每隔两小时
```

```
20 7 * * * sh run.sh 每天7:20
```

```
0 1,5 * * * sh run.sh 每天1点和5点
```

```
* * * * * sh run.sh 每分钟执行一次
```

---

## 第三部分 变量

### 1. 变量赋值

```
varname="value"
```

```
varname=`expression`
```

注意, 等号两边必须不能包含空格

### 2. 分类

四种变量：环境变量、本地变量、位置变量、特定变量参数

环境变量可作用于所有子进程

本地变量在用户现在的shell 生命期的脚本中使用，仅存在于当前进程

位置变量：作为程序参数

特定变量：特殊作用

### 3.环境变量

设置

```
MYVAR="test"
```

```
export MYVAR
```

or

```
export MYVAR="test"
```

只读

```
MYVAR="test"
```

```
readonly MYVAR
```

or

```
readonly MYVAR="test"
```

显示

```
export -p
```

```
env #查看所有环境变量
```

```
$MYVAR #获取
```

消除

```
unset MYVAR
```

### 4.本地变量

设置

```
LOCAL_VAR="test"
```

or

```
LOCAL_VAR="test"
```

```
readonly LOCAL_VAR #设置只读
```

还可以使用declare命令定义

### 5.位置变量

`$0` 脚本名称

`$#` 传递到脚本参数个数

`$$` shell脚本运行当前进程ID

`$?` 退出状态

`$N N>=1`, 第n个参数

## 6. 字符串处理

### 长度

`${#VARIABLE_NAME}` 可以给出字符串的长度。

```
if [ ${#authy_api_key} != 32 ]
then
    return $FAIL
fi
```

### 拼接字符串

```
echo "$x$y"
```

### 字符串切片

`${变量名:起始:长度}` 得到子字符串

```
$ test='I love china'
$ echo ${test:5}
e china
$ echo ${test:5:10}
e china

str="hello world"
echo ${str:6} # ${var:offset:length}
```

### 字符串替换

`${变量/查找/替换值}` 一个“/”表示替换第一个, “//”表示替换所有, 当查找中出现了: “/”请加转义符“\”表示

```
echo ${str/foo/bar} #首个
echo ${str//foo/bar} #所有
```

### 正则匹配

```
if [[ $str =~ [0-9]+\.[0-9]+ ]]
```

## 7. 数值处理

### 自增

```
a=1  
a=`expr a + 1`
```

or

```
a=1  
let a++  
let a+=2
```

## let

```
no1=4  
no2=5  
let result=no1+no2
```

## expr

```
result=`expr 3 + 4`  
result=$(expr $no1 + 5)
```

## 其他

```
result=$(( no1 + no2 ))  
result=$(( $no + 5 ))  
  
result=$(( no1 + 5 ))
```

## 浮点数

```
echo "4 * 0.56" | bc  
设定精度  
echo "scale=2;3/8" | bc  
进制转换  
echo "obase=2;100" | bc  
平方  
echo "sqrt(100)" | bc
```

## 数组和map

---



## 第四部分 控制流

### 1. 条件测试

#### 语法

```
test condition
[ condition ] #注意两边加空格

 $? #获取判断结果，0表示condition=true
```

#### 条件测试中的逻辑

```
-a 与
-o 或
! 非
&&
||

if [ -n "$str" -a -f "$file" ]
if [ -n "$str" ] && [ -f "$file" ]
```

#### 字符串测试

```
= 两字符串相等
!= 两字符串不等
-z 空串 [zero]
-n 非空串 [nozero]

[ -z "$EDITOR" ]
[ "$EDITOR" = "vi" ]
```

#### 数值测试

```
-eq 数值相等 (equal)
-ne 不等 (not equal)
-gt A>B (greater than)
-lt A<B (less than)
-le A<=B (less、equal)
-ge A>=B (greater、equal)

N=130
[ "$N" -eq 130 ]
```

## 文件测试

```
-d 目录
-f 普通文件 (Regular file)

-e 文件存在
-z 文件长度=0
-s 文件长度大于0, 非空

-b 块专用文件
-c 字符专用文件
-L 符号链接

-r Readable (文件、目录可读)
-w Writable (文件、目录可写)
-x Executable (文件可执行、目录可浏览)

-g 如果文件的set-group-id位被设置则结果为真
-u 文件有suid位设置
```

## 2. 分支if-else/case

### if-else语法

```
if condition1
then
    //do thing a
elif condition2
then
    //do thing b
else
    //do thing c
fi

or

if condition; then
# do something
fi
```

### case语法

```
case $VAR in
    1)
        echo "abc"
```

```
;;  
2|3|4)  
    echo "def"  
    ;;  
*)  
    echo "last"  
    ;;  
esac
```

### 3. 循环for/while/until

#### for语法

```
for VARIABLE in 1 2 3 4 5 .. N  
do  
    //commands  
done  
  
for OUTPUT in $(Linux-Or-Unix-Command-Here)  
do  
    //commands on $OUTPUT  
done  
  
#bash  
for (( EXP1; EXP2; EXP3 ))  
do  
    //commands  
done
```

例子

```
for i in 1 2 3 4 5; do  
    echo $i  
done  
  
for i in `seq 1 5`; do  
    echo $i  
done  
  
#!/bin/bash  
echo "Bash version"  
for i in $(seq 1 2 20)  
do  
    echo "Welcome $i times"  
done
```

```
for i in {1..5}; do
    echo $i
done

#!/bin/bash
echo "Bash version"
for i in {0..10..2}
do
    echo "Welcome $i times"
done

for ((i=1; i<=10; i++)); do
    echo $i
done

#无限循环
#!/bin/bash
for (( ; ; ))
do
    echo "infinite loops [ hit CTRL+C to stop]"
done
```

## while

```
while condition
do
    //do something
done

COUNTER=0
while [ $COUNTER -lt 5 ]
do
    COUNTER=`expr $COUNTER + 1`
    echo $COUNTER
done

无限循环
while [ 1 ]
do
    //
done
```

## until

```
#执行命令，直到条件为真，至少执行一次，可以用来做监控，condition每次都回去检查
until condition
do
    //do something
done
```

## break/continue

### break

允许跳出循环，通常在进行一些列处理后退出循环或**case**语句  
若多重循环，可指定跳出的循环个数，如跳出两重循环 **break 2**

### continue

不会跳出循环，只是跳过此循环步  
命令是程序在本循体内忽略下面的语句,从循环头开始执行

---

## 第五部分 函数

### 1.函数定义

```
function func_name() {

}

func_name() {
    //do some thing
}
```

### 注意

函数名，在脚本中必须唯一  
函数必须，先定义，后使用

### return

```
function equal() {
    return 1
}
```

```
equal  
echo $? #got 1
```

## 2. 参数传递

```
#位置参数  
function copyfile() {  
    cp $1 $2  
    return $?  
}  
  
调用  
  
copyfile /tmp/a /tmp/b  
or 获取返回值  
result=`copyfile /tmp/a /tmp/b`
```

### 位置参数

\$1 - \$9, 当参数超过10个时, 需要使用\${10}  
\$# 参数个数  
\$\* 将所有参数视为一个字符串="\$1 \$2 ..."  
\$@ 将所有参数视为个体="\$1" "\$2" "\$3"

## 3. 返回值和退出状态

```
#返回值  
function func_a() {  
    return 1  
}  
  
result=`func_a`  
if [ result != 0 ]  
then  
    echo "Error"  
fi  
  
#退出状态  
function func_b() {  
    //do something  
}  
  
func_b  
if [ $? -eq 0 ]  
then
```

```

    echo "Success"
else
    echo "Error"
fi

#更简洁
if func_b; then
    echo "Success"
else
    echo "Error"
fi

func_b && echo "Success" || echo "Error"

```

## 第四部分 高级

### bash中参数展开-展开运算符

`${varname:-word}` 如果变量未定义，返回默认值。 `${noexist:-0}`返回0  
`${varname:=word}` 如果变量未定义，设置变量为默认值 `${noexists:=0}`; echo  
`${noexists}`; 得到0  
  
`${varname:?message}` 若未定义，显示varname:message并退出当前的命令或脚本  
`${varname:+word}` 若存在且非null，返回word，否则返回null

### 模式匹配

```

${variable##pattern}
${variable%pattern}

```

## 第五部分 其他

### 读文件

```

while read -r line; do
    echo $line
done < file

保留首尾字符
while IFS= read -r line; do
    echo $line
done

```

### 一些内置命令

**:**  
空命令，类似python的pass

**.**  
相当于source

**\**  
用于跨行命令

**echo**  
输出，类似println

**exec**

**exit n**  
脚本以n作为退出码退出

**export**  
设置或显示环境变量

**expr**  
简单计算  
`x=`expr $x + 1``  
`x=$((expr $x + 1))`

**let**  
`d=111`  
`let d=$((d+1)); echo $d`  
`112`

**printf**  
格式化输出

**return**  
函数返回

**set**

**shift**  
所有参数变量左移一个位置

**unset**  
从环境变量中删除变量或函数

BP:



使用`$()` 代替反引号```  
`$(())` 代替`expr`运算符

## bash

GNU Bash 主页  
<http://www.gnu.org/software/bash/>  
 GNU Bash 手册  
<http://www.gnu.org/software/bash/manual/>

## 更多的特性

分享

`$(3 + 4)`                    而不需要 `expr 3 + 4`, 算术展开  
`/usr/{bin,local/bin}`    而不需要 `/usr/bin /usr/local/bin`  
`${str/src/dst}`            而不需要 `echo $str | sed "s/$src/$dst/"`

## 更方便的语法

```
for (( expr1; expr2; expr3 )); do
    commands
done
for (( i = 0; i < 100; i++ )); do ... done
echo a{b,c,d}e ==> abe ace ade
```

## 表达式求值

`[$]`      `[$]`中间可以加表达式 eg: `echo [$a+$b]`  
`$(( ))`    `(( ))`中间可以加表达式。Eg: `total=$((a*b))`

## 正则

```
bash的正则表达式
str='hello, world'
if [[ $str =~ '\s+world$' ]]; then
    echo match!
fi
if echo "$str" | grep -E '[ ]+world$'; then
    echo match!
fi
```

## 获取软连接指向的真实文件名

```
#注:有些系统没有这个命令
readlink /usr/bin/python
```

## 增加debug

🔍 search this website

```
function debug() {
    if [[ $DEBUG ]]
    then
        echo ">>> $"
    fi
}
```

```
# For any debug message
debug "Trying to find config file"
```

还有来自于一些很酷的Geeks的单行debug函数:

```
function debug() { ((DEBUG)) && echo ">>> $*"; }
function debug() { [ "$DEBUG" ] && echo ">>> $*"; }
```

## 将执行日志全部写到某个文件

```
exec >>"$LOGPATH"/xx.log.$TODAY 2>&1
#begin of code
```

---

版权声明：自由转载-非商用-非衍生-保持署名 | [Creative Commons BY-NC-ND 3.0](https://creativecommons.org/licenses/by-nc-nd/3.0/)



如果你觉得我的文章或项目对你有所帮助, You can buy me a coffee:)

上一篇: 读书笔记-程序员的思维训练 下一篇: 读书笔记——追随你的心，用思想改变世界

DISQUS

多说

2 条评论

wklken's blog

 登录 ▾

按评分高低排序 ▾

分享  收藏 ★

加入讨论...



Lotaku · 1年前

又有新内容了！^\_^ 谢谢！

^ | ▾ · 回复 · 分享 ▾



wklken 管理员 ↗ Lotaku · 1年前

多谢支持。一年多攒了N多笔记什么的，最近在逐步整理，发上来也方便查

^ | ▾ · 回复 · 分享 ▾

在 WKLKEN'S BLOG 上还有.....

[这是什么？](#)

一些nginx配置

3 条评论 · 11天前



GeQi — .me的域名解析昨晚有问题.. 还以为过期了呢

Vim相关资源

2 条评论 · 3个月前



One — 用了你的k-vim，节省了很多时间，所以捐了一杯咖啡。 ...

Python招聘需求与技能体系

7 条评论 · 1年前




daemon\_zhang — 吊啊，膜拜学习中。。。

读书笔记——用追随你的心，思想改变世界

1条评论 · 1年前



xgan — 看了很多篇你的日志对我很有帮助~

 订阅 在您的网站上使用Disqus 隐私

DISQUS

COPYRIGHT © 2014 WKLKEN

HOSTED ON [DIGITALOCEAN](#) AND [GITCAFE \(CHINA\)](#). POWERED BY [PELICAN](#). SEARCH POWERED BY [SWIFTYPE](#)  
SOCIAL ICONS BY [FONT-AWESOME](#).