链滴

Shell 编程简介

作者：nianxingyan
原文链接：https://ld246.com/article/1493362425158
来源网站：链滴

Shell编程简介

以下示例都在Mac系统中进行，因为大家用Mac居多，而且Mac系统也是在Linux基础上开发的，所很多东西和Linux都是一样的。

Mac下的终端程序Terminal是很好用的，工欲善其事必先利其器，所以我们可以先来配置一下终端程。

什么是Shell编程

什么是Shell

直译就是外壳，可以理解成是Linux系统内核的外壳，我们通过这个外壳和Linux系统进度交互。再具点，可以理解成是终端。

一般情况下，我们在操作Linux系统时，就是在终端上搞各种指令。而Shell编程，或者通俗点说叫Shel脚本，其实就是把一堆指令集中在一个脚本文件里，然后通过运行这个脚本文件，来执行这一大堆的令。所以它的目的就是为了**方便我们进行一些批量操作。**

Shell编程，其实指的就是我们写的这些Shell脚本。

Hello world!

一个Shell脚本其实就是一个普通的文本文件，所以可以使用任意的文本编辑器来编辑Shell脚本，比如ublimeText。不过既然脚本是在终端下运行的，所以如果能直接在终端下编辑脚本的话，就最舒服不了。所以如果有兴趣的话，可以学习一下vim的使用。

接下来写个Hello World吧。将以下内容写到脚本里，并保存为hello.sh：

```
bash
echo "Hello world!"
```

接下来在终端里进到刚刚保存文件的目录里，并确认文件存在的话，就可以用以下命令来运行了：

```
[nianxingyan@NianxyMBP shell]$ls
hello.sh
[nianxingyan@NianxyMBP shell]$sh hello.sh
Hello world!
[nianxingyan@NianxyMBP shell]$
```

上边$符号之后的部分是我们输入的内容，$及其之前的部分是命令行提示符，显示一些当前状态。

sh hello.sh的作用就是执行hello.sh。

解释一下以上出现的一些命令：

- **echo**: 在终端输出字符串，相当于print
- **cd**: 改变当前目录
- **ls**: 显示当前目录下的文件列表

一个稍微复杂点的例子

下面玩个稍微复杂点的，它的功能是这样的：

1. 在当前目录下创建一个 db.20170412目录，其中20170412是执行脚本时的当前日期，注意要替掉

2. 将当前目录下所有以 dat结尾的文件都转移到刚刚创建的这个目录中

3. 将当前目录下一个 db.latest的软链接指向刚刚创建的目录

*上面描述的这个功能可以用在如下场景：*假如有个服务程序，每小时在当前目录下生成一个以dat尾的数据文件，这样每天就会生成24个数据文件。这些数据文件如果不加整理，那么一段时间后目录的文件数量就是非常多，不利于管理。

要完成以上功能，可以编写如下一段脚本：

```
# 获取当天日期
today=`date +%F`
# 创建目录
mkdir db.$today
# 转移dat文件
mv *.dat db.$today/
# 删除之前的软链接，并创建一个新的软链接
ln -f -s db.$today db.latest
```

逐行解释一下：

- 第2行，**反引号“`”**的作用是将其中的命令输出做为字符串进行返回
而date +%F就是将当前日期以yyyyMMdd的格式进行显示，所以，变量today此时的值就是当前日了。
- mkdir的功能是创建目录，后边跟目录名。此时目录名中的$today就是对today变量的引用。
- mv是move的缩写，也就是转移文件。
- ln的作用就是创建软链接

以上命令保存为cleanup.sh，执行后是如下效果：

```
[nianxingyan@NianxyMBP shell]$touch test1.dat test2.dat
[nianxingyan@NianxyMBP shell]$ls -l
total 16
-rw-r--r--  1 nianxingyan  staff  196  4 12 15:48 cleanup.sh
-rw-r--r--  1 nianxingyan  staff   20  4 12 15:21 hello.sh
-rw-r--r--  1 nianxingyan  staff    0  4 12 15:49 test1.dat
-rw-r--r--  1 nianxingyan  staff    0  4 12 15:49 test2.dat
[nianxingyan@NianxyMBP shell]$sh cleanup.sh
[nianxingyan@NianxyMBP shell]$ls -l
total 24
-rw-r--r--  1 nianxingyan  staff  196  4 12 15:48 cleanup.sh
drwxr-xr-x  4 nianxingyan  staff  136  4 12 15:49 db.2017-04-12
lrwxr-xr-x  1 nianxingyan  staff   13  4 12 15:49 db.latest -> db.2017-04-12
-rw-r--r--  1 nianxingyan  staff   20  4 12 15:21 hello.sh
[nianxingyan@NianxyMBP shell]$
```

命令解释：

- **touch**: 暂时可以理解为生成了test1.dat和test2.dat这两个空文件。
- **ls -l**: 刚才的ls命令已经用过了，加-l参数目的就相当于以列表的形式显示当前目录中的文件，而只是以排列的方式显示文件名

流程控制

Shell脚本中也是可以进行分支、循环等流程控制的。

if语句

举例如下：

```
var1=Alice
if [ $var1 == "Alice" ] ; then
    echo "Hello, Alice!"
else
    echo "Who are you?"
fi
```

以上代码应该不难能懂，主要是记住书写格式，关键点列举一下：

1. []中间是条件判断，且每个括号两边要留空格
2. then如果和if写在同一行，就需要中间的;分隔，如果要then写在下一行，就不要写;
3. else也是一样，如果和其它命令写在一行，就需要命令之间用;分隔，否则不需要;
4. 最后一定不要忘记写 fi，否则分支判断没有结束（fi就是if反过来写，意为结束）

上边说到的;问题，其实和if命令本身没什么关系，在Linux的终端里，多个命令如果想写在同一行，可以用;分隔，这时系统就会顺序执行该行内的每条指令。比如：ifdown eth1; ifup eth1，意思就是停用网终接口eth1，然后再把它启动起来，其实就是重启该接口。

还有一点要注意，每个分支内部必须有指令，如果是一个空的分支，执行会报错，比如将上边第3行释掉，再运行脚本时就是如下效果：

```
[nianxingyan@NianxyMBP shell]$sh alice.sh
alice.sh: line 4: syntax error near unexpected token `else'
alice.sh: line 4: `else'
[nianxingyan@NianxyMBP shell]$
```

深层意思就是“这个分支里我还什么都没干，怎么就发现了else”。虽然我不理解为什么要这样规定但仍然不得不接受这个事实。

接下来要重点介绍一下我们的条件判断格式，也就是[]中间的部分。

条件判断格式

上面的示例中，用==来判断两个字符串是否相等，注意==两边要留空格。类似==，还有如下几种断：

- -eq：两个数字相等
- -ne：两个数字不相等
- -lt：左边的数字小于右边的数字
- -le：左边的数字小于或等于右边的数字
- -gt：左边的数字大于右边的数字

- -ge：左边的数字大于或等于右边的数字
- !=：两个字符串不相等

注意以上这5个判断参数只能用来判断数字，而==和!=只能用来判断字符串。所以，Shell里边是不比较两个字符串谁大谁小的。

如果想对条件判断取“反”，即“非xxx”，写法是这样的：

```
if [ ! 1 -eq 1 ] ; then
```

就是在判断最开始加!限定。

多个条件组合的写法：

```
if [ 1 -eq 1 ] && [ 2 -eq 2 ] ; then
或是
if [ 1 -eq 1 ] || [ 2 -eq 2 ] ; then
```

即将&&或是||写在两个条件判断中间

引号的作用

在Shell里，引号并不是用来区分字符串的，而是用来强调一个整体的。所以，即使是数字也可以用引括起来，比如这样：["1" -eq "1"]。不过对于字符串，还是强烈建议都用引号括起来，以免发生意，比如下面这段代码：

```
var1="Alice Chen"
if [ $var1 == "Alice Chen" ] ; then
echo "Hello, Alice!"
else
echo "Who are you?"
fi
```

这段代码就会报错，因为Shell脚本在执行时，是先将变量进行替换，再执行指令，于是上边代码的第行在替换完后就变成了这个样子：

```
if [ Alice Chen == "Alice Chen" ]; then
```

这显然是不对的。
仍然是刚才那段代码，再看一个示例：

```
var1=""
if [ $var1 == "Alice Chen" ] ; then
echo "Hello, Alice!"
else
echo "Who are you?"
fi
```

变量var1此时是一个空串，按刚才的解释，代码执行前，第2行会被替换成这个样子：

```
if [  == "Alice Chen" ] ; then
```

这样显然也是不对的。
所以，我们在写Shell脚本的时候，对于字符串变量，或者说所有的**变量**，在引用时，最好都加上**双引号**

双引号和单引号的区别

这两种引号的区别在于：双引号忽略除$ \ 反引号以外的所有特殊字符，而单引号则没有例外。
特殊字符比如有：空格 & ! () []等等。
看下面的代码：

```
var1="Alice Chen"
echo "\$var1 is $var1"
echo '\$var1 is $var1'
```

执行结果如下：

```
$var1 is Alice Chen
\$var1 is $var1
```

第一个echo用了双引号，所以\$转义后变成了$，$var1变成了Alice Chen；第二个echo用了单引号相当于直接输出了内部的原始内容。

while语句

解释了if语句后，再看while就容易很多了。直接看示例：

```
test=0
while [ $test -ne 1 ] ; then

done
```

for语句

for语句有两种写法

- 第一种 for 是一种简单的for循环，和平时用的for循环一样

```
for (( i=1 ; $i <= 100; i++ ))
do
echo $i;
done
```

- 第二种 for in 是一种简单的字符串枚举遍历法，利用for in格式对字符串按空格切份的功能

```
for i in $(seq 1 100)
```

```
do
echo $i
done
```

基本使用

基本参数

在终端输入 ***.sh 后，紧跟着输入的数据，中间可以用空格隔开，将作为参数传入到 ***.sh中

- $0 这个程序的执行名字
- $n 这个程序的第n个参数值，n=1...9
- $* 这个程序的所有参数
- $# 这个程序的参数个数

例 test.sh

```
echo $0
echo $1
echo $*
echo $#
```

执行

```
admindeMac-mini:~ xc$ /Users/xlc/Desktop/test.sh a b c d
```

输出为

```
/Users/xlc/Desktop/test.sh
a
a b c d
4
```

方法的使用

- 创建

```
function test(){
    echo $1
    echo $2
}
```

- 使用

```
test a b
```

- 输出结果

```
a
b
```

读取用户输入的字符

```
read style
sleep 0.5
buildstyle="$style"
echo $buildstyle
```

XCode打包脚本的实现原理

```
function build_xcodeproj(){
  /usr/libexec/PlistBuddy -c 'Set :CFBundleShortVersionString '$2'' "$Project_Path/$1".plist || er
exit "版本号设置失败"
  /usr/libexec/PlistBuddy -c 'Set :CFBundleVersion '$2'' "$Project_Path/$1".plist || errexit "版本
设置失败"

  if [ `ls build/ipa/"$1"/"$2"/|wc -l` -eq 0 ]; then

    if [ -e build/archives/"$2"/"$1".xcarchive ]; then
      echo "'$1' '$2' 版本archive包已存在 如需重新打包,请删除旧包"
    else
      xcodebuild -project "$Project_Path/$Project_Name" -scheme "$1" -configuration "$Confi
uration" clean archive build -archivePath "$Project_Path"/build/archives/"$2"/"$1".xcarchive ||
errexit "'$1' '$2' archive失败"
    fi

      create_export_plist $ProjectExportPlistFile

      xcodebuild -exportArchive -archivePath "$Project_Path"/build/archives/"$2"/"$1".xcarchiv
 -exportOptionsPlist $ProjectExportPlistFile -exportPath "$Project_Path"/build/ipa/"$1"/"$2"

      if [ $? -ne 0 ] ; then
        rm -f $ProjectExportPlistFile
        errexit "'$1' '$2' archive失败"
      fi
      rm -f $ProjectExportPlistFile

  else
    echo "'$1' '$2' 版本ipa包已存在 如需重新打包,请删除旧包"
  fi
}
```