Shell脚本编程30分钟入门

什么是Shell脚本

示例

看个例子吧：

```
#!/bin/sh
cd ~
mkdir shell_tut
cd shell_tut

for ((i=0; i<10; i++)); do
    touch test_$i.txt
done
```

示例解释

- 第1行：指定脚本解释器，这里是用/bin/sh做解释器的
- 第2行：切换到当前用户的home目录
- 第3行：创建一个目录shell_tut
- 第4行：切换到shell_tut目录
- 第5行：循环条件，一共循环10次
- 第6行：创建一个test_1…10.txt文件
- 第7行：循环体结束

cd, mkdir, touch都是系统自带的程序，一般在/bin或者/usr/bin目录下。for, do, done是sh脚本语言的关键字。

shell和shell脚本的概念

shell是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。Ken Thompson的sh是第一种Unix Shell，Windows Explorer是一个典型的图形界面Shell。

shell脚本（shell script），是一种为shell编写的脚本程序。业界所说的shell通常都是指shell脚本，但读者朋友要知道，shell和shell script是两个不同的概念。由于习惯的原因，简洁起见，本文出现的“shell编程”都是指shell脚本编程，不是指开发shell自身（如Windows Explorer扩展开发）。

环境

shell编程跟java、php编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

OS

当前主流的操作系统都支持shell编程，本文档所述的shell编程是指Linux下的shell，讲的基本都是POSIX标准下的功能，所以，也适用于Unix及BSD（如Mac OS）。

Linux

Linux默认安装就带了shell解释器。

Mac OS

Mac OS不仅带了sh、bash这两个最基础的解释器，还内置了ksh、csh、zsh等不常用的解释器。

Windows上的模拟器

windows出厂时没有内置shell解释器，需要自行安装，为了同时能用grep, awk, curl等工具，最好装一个cygwin或者mingw来模拟linux环境。

- cygwin
- mingw

脚本解释器

sh

即Bourne shell，POSIX（Portable Operating System Interface）标准的shell解释器，它的二进制文件路径通常是/bin/sh，由Bell Labs开发。

本文讲的是sh，如果你使用其它语言用作shell编程，请自行参考相应语言的文档。

bash

Bash是Bourne shell的替代品，属GNU Project，二进制文件路径通常是/bin/bash。业界通常混用bash、sh、和shell，比如你会经常在招聘运维工程师的文案中见到：熟悉Linux Bash编程，精通Shell编程。

在CentOS里，/bin/sh是一个指向/bin/bash的符号链接:

```
[root@centosraw ~]# ls -l /bin/*sh
-rwxr-xr-x. 1 root root 903272 Feb 22 05:09 /bin/bash
-rwxr-xr-x. 1 root root 106216 Oct 17  2012 /bin/dash
lrwxrwxrwx. 1 root root      4 Mar 22 10:22 /bin/sh -> bash
```

但在Mac OS上不是，/bin/sh和/bin/bash是两个不同的文件，尽管它们的大小只相差100字节左右:

```
iMac:~ wuxiao$ ls -l /bin/*sh
-r-xr-xr-x  1 root  wheel  1371648  6 Nov 16:52 /bin/bash
-rwxr-xr-x  2 root  wheel   772992  6 Nov 16:52 /bin/csh
-r-xr-xr-x  1 root  wheel  2180736  6 Nov 16:52 /bin/ksh
-r-xr-xr-x  1 root  wheel  1371712  6 Nov 16:52 /bin/sh
-rwxr-xr-x  2 root  wheel   772992  6 Nov 16:52 /bin/tcsh
-rwxr-xr-x  1 root  wheel  1103984  6 Nov 16:52 /bin/zsh
```

高级编程语言

理论上讲，只要一门语言提供了解释器（而不仅是编译器），这门语言就可以胜任脚本编程，常见的解释型语言都是可以用作脚本编程的，如：Perl、Tcl、Python、PHP、Ruby。Perl是最老牌的脚本编程语言了，Python这些年也成了一些linux发行版的预置解释器。

编译型语言，只要有解释器，也可以用作脚本编程，如C shell是内置的（/bin/csh），Java有第三方解释器Jshell，Ada有收费的解释器AdaScript。

如下是一个PHP Shell Script示例（假设文件名叫test.php）：

```
#!/usr/bin/php
<?php
```

```
for ($i=0; $i < 10; $i++)
        echo $i . "\n";
```

执行：

```
/usr/bin/php test.php
```

或者：

```
chmod +x test.php
./test.php
```

如何选择**shell**编程语言

熟悉 **vs** 陌生

如果你已经掌握了一门编程语言（如PHP、Python、Java、JavaScript），建议你就直接使用这门语言编写脚本程序，虽然某些地方会有点啰嗦，但你能利用在这门语言领域里的经验（单元测试、单步调试、IDE、第三方类库）。

新增的学习成本很小，只要学会怎么使用shell解释器（Jshell、AdaScript）就可以了。

简单 **vs** 高级

如果你觉得自己熟悉的语言（如Java、C）写shell脚本实在太啰嗦，你只是想做一些备份文件、安装软件、下载数据之类的事情，学着使用sh，bash会是一个好主意。

shell只定义了一个非常简单的编程语言，所以，如果你的脚本程序复杂度较高，或者要操作的数据结构比较复杂，那么还是应该使用Python、Perl这样的脚本语言，或者是你本来就已经很擅长的高级语言。因为sh和bash在这方面很弱，比如说：

- 它的函数只能返回字串，无法返回数组
- 它不支持面向对象，你无法实现一些优雅的设计模式
- 它是解释型的，一边解释一边执行，连PHP那种预编译都不是，如果你的脚本包含错误(例如调用了不存在的函数)，只要没执行到这一行，就不会报错

环境兼容性

如果你的脚本是提供给别的用户使用，使用sh或者bash，你的脚本将具有最好的环境兼容性，perl很早就是linux标配了，python这些年也成了一些linux发行版的标配，至于mac os，它默认安装了perl、python、ruby、php、java等主流编程语言。

第一个**shell**脚本

编写

打开文本编辑器，新建一个文件，扩展名为sh（sh代表shell），扩展名并不影响脚本执行，见名知意就好，如果你用php写shell 脚本，扩展名就用php好了。

输入一些代码，第一行一般是这样：

```
#!/bin/bash
#!/usr/bin/php
```

“#!”是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行。

运行

运行Shell脚本有两种方法：

作为可执行程序

```
chmod +x test.sh
./test.sh
```

注意，一定要写成./test.sh，而不是test.sh，运行其它二进制的程序也一样，直接写test.sh，linux系统会去PATH里寻找有没有叫test.sh的，而只有/bin, /sbin, /usr/bin，/usr/sbin等在PATH里，你的当前目录通常不在PATH里，所以写成test.sh是会找不到命令的，要用./test.sh告诉系统说，就在当前目录找。

通过这种方式运行bash脚本，第一行一定要写对，好让系统查找到正确的解释器。

这里的"系统"，其实就是shell这个应用程序（想象一下Windows Explorer），但我故意写成系统，是方便理解，既然这个系统就是指shell，那么一个使用/bin/sh作为解释器的脚本是不是可以省去第一行呢？是的。

作为解释器参数

这种运行方式是，直接运行解释器，其参数就是shell脚本的文件名，如：

```
/bin/sh test.sh
/bin/php test.php
```

这种方式运行的脚本，不需要在第一行指定解释器信息，写了也没用。

变量

定义变量

定义变量时，变量名不加美元符号（$），如：

```
your_name="qinjx"
```

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。

除了显式地直接赋值，还可以用语句给变量赋值，如：

```
for file in `ls /etc`
```

使用变量

使用一个定义过的变量，只要在变量名前面加美元符号即可，如：

```
your_name="qinjx"
echo $your_name
echo ${your_name}
```

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

```
for skill in Ada Coffe Action Java do
    echo "I am good at ${skill}Script"
done
```

如果不给skill变量加花括号，写成echo "I am good at $skillScript"，解释器就会把$skillScript当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。

推荐给所有变量加上花括号，这是个好的编程习惯。IntelliJ IDEA编写shell script时，IDE就会提示加花括号。

重定义变量

已定义的变量，可以被重新定义，如：

```
your_name="qinjx"
echo $your_name

your_name="alibaba"
echo $your_name
```

这样写是合法的，但注意，第二次赋值的时候不能写$your_name="alibaba"，使用变量的时候才加美元符。

注释

以“#”开头的行就是注释，会被解释器忽略。

多行注释

sh里没有多行注释，只能每一行加一个#号。就像这样：

```
#--------------------------------------------
# 这是一个自动打ipa的脚本，基于webfrogs的ipa-build书写：https://github.com/webfrogs/xcode
```

```
# 功能：自动为etao ios app打包，产出物为14个渠道的ipa包
# 特色：全自动打包，不需要输入任何参数
#-------------------------------------------

##### 用户配置区 开始 #####
#
#
# 项目根目录，推荐将此脚本放在项目的根目录，这里就不用改了
# 应用名，确保和Xcode里Product下的target_name.app名字一致
#
##### 用户配置区 结束  #####
```

如果在开发过程中，遇到大段的代码需要临时注释起来，过一会儿又取消注释，怎么办呢？每一行加个#符号太费力了，可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，达到了和注释一样的效果。

字符串

字符串是shell编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了，哈哈），字符串可以用单引号，也可以用双引号，也可以不用引号。单双引号的区别跟PHP类似。

单引号

```
str='this is a string'
```

单引号字符串的限制：

- 单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的
- 单引号字串中不能出现单引号（对单引号使用转义符后也不行）

双引号

```
your_name='qinjx'
str="Hello, I know your are \"$your_name\"! \n"
```

- 双引号里可以有变量

- 双引号里可以出现转义字符

字符串操作

拼接字符串

```
your_name="qinjx"
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"

echo $greeting $greeting_1
```

获取字符串长度：

```
string="abcd"
echo ${#string} #输出：4
```

提取子字符串

```
string="alibaba is a great company"
echo ${string:1:4} #输出：liba
```

查找子字符串

```
string="alibaba is a great company"
echo `expr index "$string" is`#输出：8，这个语句的意思是：找出单词is在这名话中的位置
```

更多

参见本文档末尾的参考资料中Advanced Bash-Scripting Guid Chapter 10.1

数组

管道

条件判断

流程控制

和Java、PHP等语言不一样，sh的流程控制不可为空，如：

```
<?php
if (isset($_GET["q"])) {
    search(q);
}
else {
    //do nothing
}
```

在sh/bash里可不能这么写，如果else分支没有语句执行，就不要写这个else。

还要注意，sh里的if [$foo -eq 0]，这个方括号跟Java/PHP里if后面的圆括号大不相同，它是一个可执行程序（和cd, ls, grep一样），相不到吧？在CentOS上，它在/usr/bin目录下：

```
ll /usr/bin/[
-rwxr-xr-x. 1 root root 33408 6月  22 2012 /usr/bin/[
```

正因为方括号在这里是一个可执行程序，方括号后面必须加空格，不能写成if [$foo -eq 0]

if else

if

```
if condition
then
    command1
    command2
    ...
    commandN
fi
```

写成一行（适用于终端命令提示符）：

```
if `ps -ef | grep ssh`;  then echo hello; fi
```

末尾的fi就是if倒过来拼写，后面还会遇到类似的

if else

```
if condition
then
    command1
    command2
    ...
    commandN
else
    command
fi
```

if else-if else

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

for while

for

在开篇的示例里演示过了：

```
for var in item1 item2 ... itemN
do
    command1
```

```
    command2
    ...
    commandN
done
```

写成一行：

```
for var in item1 item2 ... itemN; do command1; command2… done;
```

C风格的for

```
for (( EXP1; EXP2; EXP3 ))
do
    command1
    command2
    command3
done
```

while

```
while condition
do
    command
done
```

无限循环

```
while :
do
    command
done
```

或者

```
while true
do
    command
```

```
done
```

或者

```
for (( ; ; ))
```

until

```
until condition
do
    command
done
```

case

```
case "${opt}" in
    "Install-Puppet-Server" )
        install_master $1
        exit
    ;;

    "Install-Puppet-Client" )
        install_client $1
        exit
    ;;

    "Config-Puppet-Server" )
        config_puppet_master
        exit
    ;;

    "Config-Puppet-Client" )
        config_puppet_client
        exit
    ;;

    "Exit" )
        exit
    ;;
```

```
    * ) echo "Bad option, please choose again"
esac
```

case的语法和C family语言差别很大，它需要一个esac（就是case反过来）作为结束标记，每个case分支用右圆括号，用两个分号表示break

函数

定义

调用

文件包含

可以使用source和.关键字，如：

```
source ./function.sh
. ./function.sh
```

在bash里，source和.是等效的，他们都是读入function.sh的内容并执行其内容（类似PHP里的include），为了更好的可移植性，推荐使用第二种写法。

包含一个文件和在执行一个文件一样，也要写这个文件的路径，不能光写文件名，比如上述例子中:

```
. ./function.sh
```

不可以写作：

```
. function.sh
```

如果function.sh是用户传入的参数，如何获得它的绝对路径呢？方法是：

```
real_path=`readlink -f $1`#$1是用户输入的参数，如function.sh
```

```
. $real_path
```

用户输入

执行脚本时传入

脚本运行中输入

select菜单

stdin和**stdout**

常用的命令

sh脚本结合系统命令便有了强大的威力，在字符处理领域，有grep、awk、sed三剑客，grep负责找出特定的行，awk能将行拆分成多个字段，sed则可以实现更新插入删除等写操作。

ps

查看进程列表

grep

排除**grep**自身

查找与**target**相邻的结果

awk

sed

插入

替换

删除

xargs

curl