

44 循环往复，Shell开路

更新时间：2019-08-13 11:34:05



什么是路？就是从没路的地方践踏出来的，从只有荆棘的地方开辟出来的。

—— 鲁迅

内容简介

1. 前言
2. while 循环
3. until 循环
4. for 循环
5. 总结

1. 前言

上一课 [带你玩转Linux和Shell编程 | 第五部分第五课：条件一出，Shell不服](#) 中我们学习了 Shell 中的条件语句。

有了上一课的基础，这一课将是很轻松的。

我们来学习一个几乎所有编程语言都有的要素：循环语句。

循环语句使我们得够重复一个代码块任意多次。这么有用的机制，Shell 语言当然支持。

Shell 中，主要的循环语句有三种：

- while 循环
- until 循环
- for 循环

我们一一来学习。

2. while 循环

在 **Shell** 中，我们最常用的循环是 **while** 循环。

while 是英语“当... 的时候；在... 期间”的意思。

while 循环的逻辑是这样的：

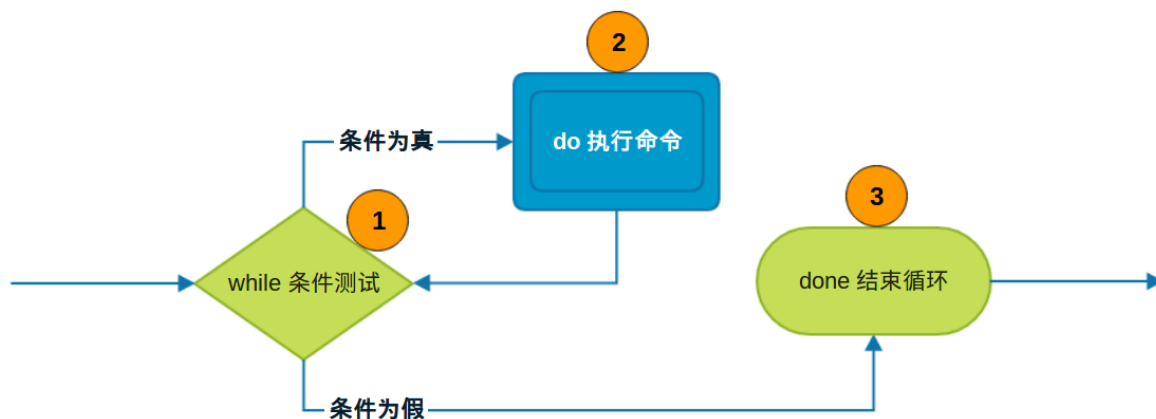
```
while [ 条件测试 ]
do
    做某些事
done
```

do 是英语“做”的意思。**done** 是英语“完成”的意思。

当然了，我们也可以像在 **if** 语句中那样，把关键字 **do** 放到与条件测试同一行上，但是之间要加分号，如下：

```
while [ 条件测试 ]; do
    做某些事
done
```

while 循环流程图：



我们来写一个程序，它会请求用户输入“yes”（英语“是”的意思），如果用户没有输入任何字符或者输入的字符串不是“yes”，那么程序就一直请求。

首先，我们创建一个名叫 **loop.sh** 的文件（**loop** 是英语“循环”的意思）。

```
vim loop.sh
```

在这个文件中输入以下内容：

```
#!/bin/bash

while [ -z $response ] || [ $response != 'yes' ]
do
    read -p 'Say yes : ' response
done
```

response 是英语“回答”的意思。**say** 是英语“说”的意思。

运行：

```
oscar@  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ chmod +x loop.sh  
oscar@oscar-laptop:~$ ./loop.sh  
Say yes : yes  
oscar@oscar-laptop:~$ ./loop.sh  
Say yes : no  
Say yes : I  
Say yes : am  
Say yes : Iron  
Say yes : Man  
Say yes : yes  
oscar@oscar-laptop:~$
```

我们做了两个条件测试：

- response 是否为空： `-z $response`
- response 是否不等于 'yes'： `$response != 'yes'`

因为这两个条件测试之间是用 `||`（逻辑或）连接的，因此只要两个条件有一个成立，整个条件测试即成立，就会执行 `do` 和 `done` 之间的语句。

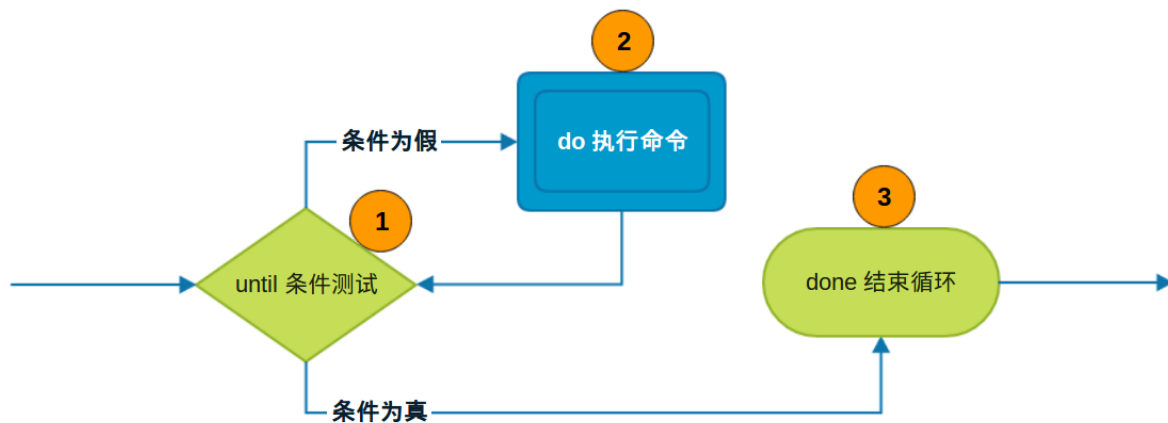
可以看到，这个程序非常“任性”，只要我们输入的不是“yes”，它就不停止输出“Say yes :”。真是“咄咄逼人”...

3. until 循环

与 `while` 这个关键字相反的有一个 `until` 关键字，`until` 在英语中是“到...为止，直到...时”的意思。

它也可以实现循环，只不过逻辑和 `while` 循环正好相反。

`until` 循环流程图：



我们可以用 **until** 来改写一下上面的程序：

```
#!/bin/bash

until [ "$response" = 'yes' ]
do
    read -p 'Say yes : ' response
done
```

运行结果和 **while** 循环那个是一样的。

4. for 循环

for 是英语“对于”的意思。

首先，我们要提醒已经学过主流编程语言（如 C 语言，Java，C++等等）的读者，Shell 中的 **for** 循环和你已经习惯的 **for** 循环方式略有不同。

我们一起来学习吧。

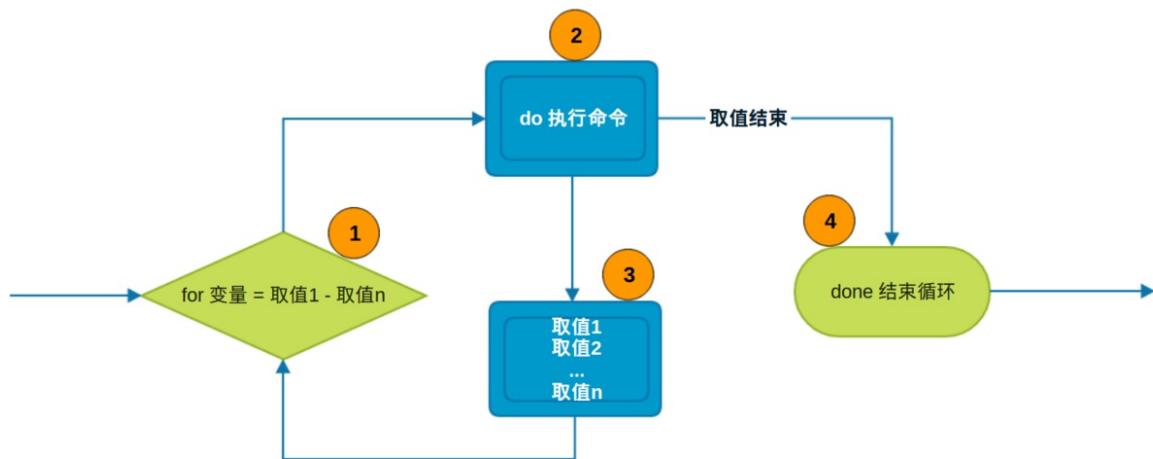
遍历列表

for 循环可以遍历一个“取值列表”，基本的逻辑如下：

```
for 变量 in '值1' '值2' '值3' ... '值n'
do
    做某些事
done
```

in 是英语“在... 之中”的意思。

for 循环流程图：



我们来写一个例子：

```
#!/bin/bash

for animal in 'dog' 'cat' 'pig'
do
    echo "Animal being analyzed : $animal"
done
```

“Animal being analyzed” 是英语“正在被分析的动物”的意思。

运行：

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./loop.sh
Animal being analyzed : dog
Animal being analyzed : cat
Animal being analyzed : pig
oscar@oscar-laptop:~$
```

可以看到，`animal` 这个变量依次取了 ‘dog’（狗），‘cat’（猫），‘pig’（猪） 这三个值。

`for` 循环的取值列表不一定要在代码中定义好，我们也可以用变量，如下例：

```
#!/bin/bash

listfile='ls`

for file in $listfile
do
    echo "File found : $file"
done
```

“File found” 是英语“找到的文件”的意思。

运行：

```
oscar@oscar-laptop
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./loop.sh
File found : chinese.txt
File found : compression
File found : condition.sh
File found : date
File found : debian-9.9.0-amd64-netinst.iso
File found : Desktop
File found : Documents
File found : Downloads
File found : emacs-26.2-copy.tar.gz
File found : emacs-26.2.tar.gz
File found : errors.log
File found : find_log
File found : grep_log
File found : helloworld
File found : helloworld.cpp
File found : htop-2.2.0
File found : htop-2.2.0.tar.gz
File found : loop.sh
File found : Music
File found : myFile
File found : name.txt
File found : new_file
File found : newly_created_file
```

如你所见，上面的程序列出了当前目录下所有的文件。

我们还可以简化上面的程序，不需要用到 `listfile` 这个变量：

```
#!/bin/bash

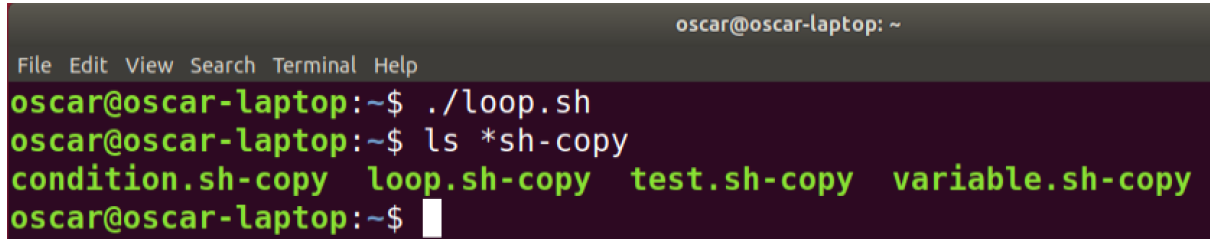
for file in `ls`
do
    echo "File found : $file"
done
```

我们可以再改进这个程序，让它复制当前目录下所有以 `.sh` 结尾的文件，并且把每个副本的名字修改为“现有名字 + `-copy`”（`copy` 是英语“拷贝”的意思）：

```
#!/bin/bash

for file in `ls *.sh`
do
    cp $file $file-copy
done
```

运行：

A terminal window titled 'oscar@oscar-laptop: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'oscar@oscar-laptop:~\$'. The user enters './loop.sh'. The prompt changes to 'oscar@oscar-laptop:~\$' and the user enters 'ls *sh-copy'. The output is 'condition.sh-copy loop.sh-copy test.sh-copy variable.sh-copy'. The prompt returns to 'oscar@oscar-laptop:~\$' with a cursor.

```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./loop.sh
oscar@oscar-laptop:~$ ls *sh-copy
condition.sh-copy loop.sh-copy test.sh-copy variable.sh-copy
oscar@oscar-laptop:~$
```

可见，Shell 非常强大。

更常规的 **for** 循环

刚才我们看到的 **for** 循环，和主流编程语言中的语法略有不同，不过我们可以借助 **seq** 命令，来实现类似主流编程语言中的 **for** 循环的语法。

seq 是 **sequence** 的缩写，是英语“序列”的意思。

来写一个例子：

```
#!/bin/bash

for i in `seq 1 10`
do
    echo $i
done
```

运行：

```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./loop.sh
1
2
3
4
5
6
7
8
9
10
oscar@oscar-laptop:~$
```

以上程序中，`seq 1 10` 会返回一个取值列表，是从 1 到 10 的整数值。因此，`echo` 就会遍历输出 1 到 10 这 10 个整数。

我们也可以修改默认的取值间隔（1），改成我们需要的数值，如下：

```
#!/bin/bash

for i in `seq 1 2 10`
do
    echo $i
done
```

运行：


```
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ ./loop.sh
1
3
5
7
9
oscar@oscar-laptop:~$
```

以上程序中，`seq 1 2 10` 会返回一个取值列表，是从 1 到 10 的整数值，但是取值间隔是 2，`echo` 就会遍历输出 1 到 10 这 10 个整数中的 1, 3, 5, 7, 9 这 5 个整数。

5. 总结

我们使用循环语句来重复执行一系列指令；

`while` 循环在条件满足的情况下可以一直重复执行。其中的条件测试语句的机制和上一课的条件语句中的一样；

`until` 循环和 `while` 循环的逻辑正好相反；

`for` 循环可以遍历一个“取值列表”，来依次执行相应的指令。在 `for` 循环内部，变量的取值会逐步变化。

今天的课就到这里，一起加油吧！

}