第三十章 流程控制：while/unitil循环

In the previous chapter, we developed a menu-driven program to produce various kinds of system information. The program works, but it still has a significant usability problem. It only executes a single choice and then terminates. Even worse, if an invalid selection is made, the program terminates with an error, without giving the user an opportunity to try again. It would be better if we could somehow construct the program so that it could repeat the menu display and selection over and over, until the user chooses to exit the program.

在前面的章节中，我们开发了菜单驱动程序，来产生各种各样的系统信息。虽然程序能够运行， 但它仍然存在重大的可用问题。它只能执行单一的选择，然后终止。更糟糕地是，如果做了一个 无效的选择，程序会以错误终止，而没有给用户提供再试一次的机会。如果我们能构建程序， 以致于程序能够重复显示菜单，而且能一次由一次的选择，直到用户选择退出程序，这样的程序会更好一些。

In this chapter, we will look at a programming concept called looping, which can be used to make portions of programs repeat. The shell provides three compound commands for looping. We will look at two of them in this chapter, and the third in a later one.

在这一章中，我们将看一个叫做循环的程序概念，其可用来使程序的某些部分重复。shell 为循环提供了三个复合命令。 本章我们将查看其中的两个命令，随后章节介绍第三个命令。

### 循环

Daily life is full of repeated activities. Going to work each day, walking the dog, slicing a carrot are all tasks that involve repeating a series of steps. Let’s consider slicing a carrot. If we express this activity in pseudocode, it might look something like this:

日常生活中充满了重复性的活动。每天去散步，遛狗，切胡萝卜，所有任务都要重复一系列的步骤。 让我们以切胡萝卜为例。如果我们用伪码表达这种活动，它可能看起来像这样：

1. get cutting board
2. get knife
3. place carrot on cutting board
4. lift knife
5. advance carrot
6. slice carrot
7. if entire carrot sliced, then quit, else go to step 4
8. 准备切菜板
9. 准备菜刀
10. 把胡萝卜放到切菜板上
11. 提起菜刀
12. 向前推进胡萝卜
13. 切胡萝卜
14. 如果切完整个胡萝卜，就退出，要不然回到第四步继续执行

Steps 4 through 7 form a loop. The actions within the loop are repeated until the condition, “entire carrot sliced,” is reached.

从第四步到第七步形成一个循环。重复执行循环内的动作直到满足条件“切完整个胡萝卜”。

#### while

bash can express a similar idea. Let’s say we wanted to display five numbers in sequential order from one to five. a bash script could be constructed as follows:

bash 能够表达相似的想法。比方说我们想要按照顺序从1到5显示五个数字。可如下构造一个 bash 脚本：

#!/bin/bash

# while-count: display a series of numbers

count=1

while [ $count -le 5 ]; do

echo $count

count=$((count + 1))

done

echo "Finished."

When executed, this script displays the following:

当执行的时候，这个脚本显示如下信息：

[me@linuxbox ~]$ while-count

1

2

3

4

5

Finished.

The syntax of the while command is:

while 命令的语法是：

while commands; do commands; done

Like if, while evaluates the exit status of a list of commands. As long as the exit status is zero, it performs the commands inside the loop. In the script above, the variable count is created and assigned an initial value of 1. The while command evaluates the exit status of the test command. As long as the test command returns an exit status of zero, the commands within the loop are executed. At the end of each cycle, the test command is repeated. After six iterations of the loop, the value ofcount has increased to six, the test command no longer returns an exit status of zero and the loop terminates. The program continues with the next statement following the loop.

和 if 一样， while 计算一系列命令的退出状态。只要退出状态为零，它就执行循环内的命令。 在上面的脚本中，创建了变量 count ，并初始化为1。 while 命令将会计算 test 命令的退出状态。 只要 test 命令返回退出状态零，循环内的所有命令就会执行。每次循环结束之后，会重复执行 test 命令。 第六次循环之后， count 的数值增加到6， test 命令不再返回退出状态零，且循环终止。 程序继续执行循环之后的语句。

We can use a while loop to improve the read-menu program from the previous chapter:

我们可以使用一个 while 循环，来提高前面章节的 read-menu 程序：

#!/bin/bash

# while-menu: a menu driven system information program

DELAY=3 # Number of seconds to display results

while [[ $REPLY != 0 ]]; do

clear

cat <<- \_EOF\_

Please Select:

1. Display System Information

2. Display Disk Space

3. Display Home Space Utilization

0. Quit

\_EOF\_

read -p "Enter selection [0-3] > "

if [[ $REPLY =~ ^[0-3]$ ]]; then

if [[ $REPLY == 1 ]]; then

echo "Hostname: $HOSTNAME"

uptime

sleep $DELAY

fi

if [[ $REPLY == 2 ]]; then

df -h

sleep $DELAY

fi

if [[ $REPLY == 3 ]]; then

if [[ $(id -u) -eq 0 ]]; then

echo "Home Space Utilization (All Users)"

du -sh /home/\*

else

echo "Home Space Utilization ($USER)"

du -sh $HOME

fi

sleep $DELAY

fi

else

echo "Invalid entry."

sleep $DELAY

fi

done

echo "Program terminated."

By enclosing the menu in a while loop, we are able to have the program repeat the menu display after each selection. The loop continues as long as REPLY is not equal to “0” and the menu is displayed again, giving the user the opportunity to make another selection. At the end of each action, a sleep command is executed so the program will pause for a few seconds to allow the results of the selection to be seen before the screen is cleared and the menu is redisplayed. Once REPLY is equal to “0,” indicating the “quit” selection, the loop terminates and execution continues with the line following done.

通过把菜单包含在 while 循环中，每次用户选择之后，我们能够让程序重复显示菜单。只要 REPLY 不 等于”0”，循环就会继续，菜单就能显示，从而用户有机会重新选择。每次动作完成之后，会执行一个 sleep 命令，所以在清空屏幕和重新显示菜单之前，程序将会停顿几秒钟，为的是能够看到选项输出结果。 一旦 REPLY 等于“0”，则表示选择了“退出”选项，循环就会终止，程序继续执行 done 语句之后的代码。

### 跳出循环

bash provides two builtin commands that can be used to control program flow inside loops. Thebreak command immediately terminates a loop, and program control resumes with the next statement following the loop. The continue command causes the remainder to the loop to be skipped, and program control resumes with the next iteration of the loop. Here we see a version of the while-menu program incorporating both break and continue:

bash 提供了两个内部命令，它们可以用来在循环内部控制程序流程。这个 break 命令立即终止一个循环， 且程序继续执行循环之后的语句。这个 continue 命令导致程序跳过循环中剩余的语句，且程序继续执行 下一次循环。这里我们看看采用了 break 和 continue 两个命令的 while-menu 程序版本：

#!/bin/bash

# while-menu2: a menu driven system information program

DELAY=3 # Number of seconds to display results

while true; do

clear

cat <<- \_EOF\_

Please Select:

1. Display System Information

2. Display Disk Space

3. Display Home Space Utilization

0. Quit

\_EOF\_

read -p "Enter selection [0-3] > "

if [[ $REPLY =~ ^[0-3]$ ]]; then

if [[ $REPLY == 1 ]]; then

echo "Hostname: $HOSTNAME"

uptime

sleep $DELAY

continue

fi

if [[ $REPLY == 2 ]]; then

df -h

sleep $DELAY

continue

fi

if [[ $REPLY == 3 ]]; then

if [[ $(id -u) -eq 0 ]]; then

echo "Home Space Utilization (All Users)"

du -sh /home/\*

else

echo "Home Space Utilization ($USER)"

du -sh $HOME

fi

sleep $DELAY

continue

fi

if [[ $REPLY == 0 ]]; then

break

fi

else

echo "Invalid entry."

sleep $DELAY

fi

done

echo "Program terminated."

In this version of the script, we set up an endless loop (one that never terminates on its own) by using the true command to supply an exit status to while. Since true will always exit with a exit status of zero, the loop will never end. This is a surprisingly common scripting technique. Since the loop will never end on its own, it’s up to the programmer to provide some way to break out of the loop when the time is right. In this script, the break command is used to exit the loop when the “0” selection is chosen. The continue command has been included at the end of the other script choices to allow for more efficient execution. By using continue, the script will skip over code that is not needed when a selection is identified. For example, if the “1” selection is chosen and identified, there is no reason to test for the other selections.

在这个脚本版本中，我们设置了一个无限循环（就是自己永远不会终止的循环），通过使用 true 命令 为 while 提供一个退出状态。因为 true 的退出状态总是为零，所以循环永远不会终止。这是一个 令人惊讶的通用脚本编程技巧。因为循环自己永远不会结束，所以由程序员在恰当的时候提供某种方法来跳出循环。 此脚本，当选择”0”选项的时候，break 命令被用来退出循环。continue 命令被包含在其它选择动作的末尾， 为的是更加高效执行。通过使用 continue 命令，当一个选项确定后，程序会跳过不需要的代码。例如， 如果选择了选项”1”，则没有理由去测试其它选项。

#### until

The until command is much like while, except instead of exiting a loop when a non- zero exit status is encountered, it does the opposite. An until loop continues until it receives a zero exit status. In our while-count script, we continued the loop as long as the value of the count variable was less than or equal to five. We could get the same result by coding the script with until:

这个 until 命令与 while 非常相似，除了当遇到一个非零退出状态的时候， while 退出循环， 而 until 不退出。一个 until 循环会继续执行直到它接受了一个退出状态零。在我们的 while-count 脚本中， 我们继续执行循环直到 count 变量的数值小于或等于5。我们可以得到相同的结果，通过在脚本中使用 until 命令：

#!/bin/bash

# until-count: display a series of numbers

count=1

until [ $count -gt 5 ]; do

echo $count

count=$((count + 1))

done

echo "Finished."

By changing the test expression to $count -gt 5, until will terminate the loop at the correct time. The decision of whether to use the while or until loop is usually a matter of choosing the one that allows the clearest test to be written.

通过把 test 表达式更改为 $count -gt 5 ， until 会在正确的时间终止循环。决定使用 while 循环 还是 until 循环，通常是选择一个 test 可以编写地很清楚的循环。

### 使用循环读取文件

while and until can process standard input. This allows files to be processed with while anduntil loops. In the following example, we will display the contents of the distros.txt file used in earlier chapters:

while 和 until 能够处理标准输入。这就可以使用 while 和 until 处理文件。在下面的例子中， 我们将显示在前面章节中使用的 distros.txt 文件的内容：

#!/bin/bash

# while-read: read lines from a file

while read distro version release; do

printf "Distro: %s\tVersion: %s\tReleased: %s\n" \

$distro \

$version \

$release

done < distros.txt

To redirect a file to the loop, we place the redirection operator after the done statement. The loop will use read to input the fields from the redirected file. The read command will exit after each line is read, with a zero exit status until the end-of-file is reached. At that point, it will exit with a non-zero exit status, thereby terminating the loop. It is also possible to pipe standard input into a loop:

为了重定向文件到循环中，我们把重定向操作符放置到 done 语句之后。循环将使用 read 从重定向文件中读取 字段。这个 read 命令读取每个文本行之后，将会退出，其退出状态为零，直到到达文件末尾。到时候，它的 退出状态为非零数值，因此终止循环。也有可能把标准输入管道到循环中。

#!/bin/bash

# while-read2: read lines from a file

sort -k 1,1 -k 2n distros.txt | while read distro version release; do

printf "Distro: %s\tVersion: %s\tReleased: %s\n" \

$distro \

$version \

$release

done

Here we take the output of the sort command and display the stream of text. However, it is important to remember that since a pipe will execute the loop in a subshell, any variables created or assigned within the loop will be lost when the loop terminates.

这里我们接受 sort 命令的标准输出，然后显示文本流。然而，因为管道将会在子 shell 中执行 循环，当循环终止的时候，循环中创建的任意变量或赋值的变量都会消失，记住这一点很重要。

### 总结

With the introduction of loops, and our previous encounters with branching, subroutines and sequences, we have covered the major types of flow control used in programs. bash has some more tricks up its sleeve, but they are refinements on these basic concepts.

通过引入循环，和我们之前遇到的分支，子例程和序列，我们已经介绍了程序流程控制的主要类型。 bash 还有一些锦囊妙计，但它们都是关于这些基本概念的完善。

### 拓展阅读

* The Bash Guide for Beginners from the Linux Documentation Project has some more examples of while loops:
* Linux 文档工程中的 Bash 初学者指南一书中介绍了更多的 while 循环实例：

<http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html>

* The Wikipedia has an article on loops, which is part of a larger article on flow control:
* Wikipedia 中有一篇关于循环的文章，其是一篇比较长的关于流程控制的文章中的一部分：

<http://en.wikipedia.org/wiki/Control_flow#Loops>