第三十四章 流程控制：for循环

In this final chapter on flow control, we will look at another of the shell’s looping constructs. The for loop differs from the while and until loops in that it provides a means of processing sequences during a loop. This turns out to be very useful when programming. Accordingly, the for loop is a very popular construct in bash scripting.

在这关于流程控制的最后一章中，我们将看看另一种 shell 循环构造。for 循环不同于 while 和 until 循环，因为 在循环中，它提供了一种处理序列的方式。这证明在编程时非常有用。因此在 bash 脚本中，for 循环是非常流行的构造。

A for loop is implemented, naturally enough, with the for command. In modern versions of bash, for is available in two forms.

实现一个 for 循环，很自然的，要用 for 命令。在现代版的 bash 中，有两种可用的 for 循环格式。

### for: 传统 shell 格式

The original for command’s syntax is:

原来的 for 命令语法是：

for variable [in words]; do

commands

done

Where variable is the name of a variable that will increment during the execution of the loop, words is an optional list of items that will be sequentially assigned to variable, and commands are the commands that are to be executed on each iteration of the loop.

这里的 variable 是一个变量的名字，这个变量在循环执行期间会增加，words 是一个可选的条目列表， 其值会按顺序赋值给 variable，commands 是在每次循环迭代中要执行的命令。

The for command is useful on the command line. We can easily demonstrate how it works:

在命令行中 for 命令是很有用的。我们可以很容易的说明它是如何工作的：

[me@linuxbox ~]$ for i in A B C D; do echo $i; done

A

B

C

D

In this example, for is given a list of four words: “A”, “B”, “C”, and “D”. With a list of four words, the loop is executed four times. Each time the loop is executed, a word is as- signed to the variable i. Inside the loop, we have an echo command that displays the value of i to show the assignment. As with the while and until loops, the done keyword closes the loop.

在这个例子中，for 循环有一个四个单词的列表：“A”，“B”，“C”，和 “D”。由于这四个单词的列表，for 循环会执行四次。 每次循环执行的时候，就会有一个单词赋值给变量 i。在循环体内，我们有一个 echo 命令会显示 i 变量的值，来演示赋值结果。 正如 while 和 until 循环，done 关键字会关闭循环。

The really powerful feature of for is the number of interesting ways we can create the list of words. For example, through brace expansion:

for 命令真正强大的功能是我们可以通过许多有趣的方式创建 words 列表。例如，通过花括号展开：

[me@linuxbox ~]$ for i in {A..D}; do echo $i; done

A

B

C

D

or pathname expansion:

或者路径名展开：

[me@linuxbox ~]$ for i in distros\*.txt; do echo $i; done

distros-by-date.txt

distros-dates.txt

distros-key-names.txt

distros-key-vernums.txt

distros-names.txt

distros.txt

distros-vernums.txt

distros-versions.txt

or command substitution:

或者命令替换：

#!/bin/bash

# longest-word : find longest string in a file

while [[ -n $1 ]]; do

if [[ -r $1 ]]; then

max\_word=

max\_len=0

for i in $(strings $1); do

len=$(echo $i | wc -c)

if (( len > max\_len )); then

max\_len=$len

max\_word=$i

fi

done

echo "$1: '$max\_word' ($max\_len characters)"

fi

shift

done

In this example, we look for the longest string found within a file. When given one or more filenames on the command line, this program uses the strings program (which is included in the GNU binutils package) to generate a list of readable text “words” in each file. The for loop processes each word in turn and determines if the current word is the longest found so far. When the loop concludes, the longest word is displayed.

在这个示例中，我们要在一个文件中查找最长的字符串。当在命令行中给出一个或多个文件名的时候， 该程序会使用 strings 程序（其包含在 GNU binutils 包中），为每一个文件产生一个可读的文本格式的 “words” 列表。 然后这个 for 循环依次处理每个单词，判断当前这个单词是否为目前为止找到的最长的一个。当循环结束的时候，显示出最长的单词。

If the optional in words portion of the for command is omitted, for defaults to pro- cessing the positional parameters. We will modify our longest-word script to use this method:

如果省略掉 for 命令的可选项 words 部分，for 命令会默认处理位置参数。 我们将修改 longest-word 脚本，来使用这种方式：

#!/bin/bash

# longest-word2 : find longest string in a file

for i; do

if [[ -r $i ]]; then

max\_word=

max\_len=0

for j in $(strings $i); do

len=$(echo $j | wc -c)

if (( len > max\_len )); then

max\_len=$len

max\_word=$j

fi

done

echo "$i: '$max\_word' ($max\_len characters)"

fi

done

As we can see, we have changed the outermost loop to use for in place of while. By omitting the list of words in the for command, the positional parameters are used instead. Inside the loop, previous instances of the variable i have been changed to the variable j. The use of shift has also been eliminated.

正如我们所看到的，我们已经更改了最外围的循环，用 for 循环来代替 while 循环。通过省略 for 命令的 words 列表， 用位置参数替而代之。在循环体内，之前的变量 i 已经改为变量 j。同时 shift 命令也被淘汰掉了。

**Why i?**

**为什么是 i？**

You may have noticed that the variable i was chosen for each of the for loop examples above. Why? No specific reason actually, besides tradition. The variable used with for can be any valid variable, but i is the most common, followed by j and k.

你可能已经注意到上面所列举的 for 循环的实例都选择 i 作为变量。为什么呢？ 实际上没有具体原因，除了传统习惯。 for 循环使用的变量可以是任意有效的变量，但是 i 是最常用的一个，其次是 j 和 k。

The basis of this tradition comes from the Fortran programming language. In For- tran, undeclared variables starting with the letters I, J, K, L, and M are automati- cally typed as integers, while variables beginning with any other letter are typed as real (numbers with decimal fractions). This behavior led programmers to use the variables I, J, and K for loop variables, since it was less work to use them when a temporary variable (as loop variables often are) was needed. It also led to the following Fortran-based witticism:

“GOD is real, unless declared integer.”

这一传统的基础源于 Fortran 编程语言。在 Fortran 语言中，以字母 I，J，K，L 和 M 开头的未声明变量的类型 自动设为整形，而以其它字母开头的变量则为实数类型（带有小数的数字）。这种行为导致程序员使用变量 I，J，和 K 作为循环变量， 因为当需要一个临时变量（正如循环变量）的时候，使用它们工作量比较少。这也引出了如下基于 fortran 的俏皮话：

“神是真实的，除非是声明的整数。”

### for: C 语言格式

Recent versions of bash have added a second form of for command syntax, one that resembles the form found in the C programming language. Many other languages support this form, as well:

最新版本的 bash 已经添加了第二种格式的 for 命令语法，该语法相似于 C 语言中的 for 语法格式。 其它许多编程语言也支持这种格式：

for (( expression1; expression2; expression3 )); do

commands

done

where expression1, expression2, and expression3 are arithmetic expressions and com- mands are the commands to be performed during each iteration of the loop. In terms of behavior, this form is equivalent to the following construct:

这里的 expression1，expression2，和 expression3 都是算术表达式，commands 是每次循环迭代时要执行的命令。 在行为方面，这相当于以下构造形式：

(( expression1 ))

while (( expression2 )); do

commands

(( expression3 ))

done

expression1 is used to initialize conditions for the loop, expression2 is used to determine when the loop is finished, and expression3 is carried out at the end of each iteration of the loop.

expression1 用来初始化循环条件，expression2 用来决定循环结束的时间，还有在每次循环迭代的末尾会执行 expression3。

Here is a typical application:

这里是一个典型应用：

#!/bin/bash

# simple\_counter : demo of C style for command

for (( i=0; i<5; i=i+1 )); do

echo $i

done

When executed, it produces the following output:

脚本执行之后，产生如下输出：

[me@linuxbox ~]$ simple\_counter

0

1

2

3

4

In this example, expression1 initializes the variable i with the value of zero, expression2 allows the loop to continue as long as the value of i remains less than 5, and expression3 increments the value of i by one each time the loop repeats.

在这个示例中，expression1 初始化变量 i 的值为0，expression2 允许循环继续执行只要变量 i 的值小于5， 还有每次循环迭代时，expression3 会把变量 i 的值加1。

The C language form of for is useful anytime a numeric sequence is needed. We will see several applications for this in the next two chapters.

C 语言格式的 for 循环对于需要一个数字序列的情况是很有用处的。我们将在接下来的两章中看到几个这样的应用实例。

### 总结

With our knowledge of the for command, we will now apply the final improvements to our sys\_info\_page script. Currently, the report\_home\_space function looks like this:

学习了 for 命令的知识，现在我们将对我们的 sys\_info\_page 脚本做最后的改进。 目前，这个 report\_home\_space 函数看起来像这样：

report\_home\_space () {

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

cat <<- \_EOF\_

<H2>Home Space Utilization (All Users)</H2>

<PRE>$(du -sh /home/\*)</PRE>

\_EOF\_

else

cat <<- \_EOF\_

<H2>Home Space Utilization ($USER)</H2>

<PRE>$(du -sh $HOME)</PRE>

\_EOF\_

fi

return

}

Next, we will rewrite it to provide more detail for each user’s home directory, and include the total number of files and subdirectories in each:

下一步，我们将重写它，以便提供每个用户家目录的更详尽信息，并且包含用户家目录中文件和目录的总个数：

report\_home\_space () {

local format="%8s%10s%10s\n"

local i dir\_list total\_files total\_dirs total\_size user\_name

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

dir\_list=/home/\*

user\_name="All Users"

else

dir\_list=$HOME

user\_name=$USER

fi

echo "<H2>Home Space Utilization ($user\_name)</H2>"

for i in $dir\_list; do

total\_files=$(find $i -type f | wc -l)

total\_dirs=$(find $i -type d | wc -l)

total\_size=$(du -sh $i | cut -f 1)

echo "<H3>$i</H3>"

echo "<PRE>"

printf "$format" "Dirs" "Files" "Size"

printf "$format" "----" "-----" "----"

printf "$format" $total\_dirs $total\_files $total\_size

echo "</PRE>"

done

return

}

This rewrite applies much of what we have learned so far. We still test for the superuser, but instead of performing the complete set of actions as part of the if, we set some vari- ables used later in a for loop. We have added several local variables to the function and made use of printf to format some of the output.

这次重写应用了目前为止我们学过的许多知识。我们仍然测试超级用户（superuser），但是我们在 if 语句块内 设置了一些随后会在 for 循环中用到的变量，来取代在 if 语句块内执行完备的动作集合。我们添加了给 函数添加了几个本地变量，并且使用 printf 来格式化输出。

### 拓展阅读

* The Advanced Bash-Scripting Guide has a chapter on loops, with a variety of examples using for:
* 《高级 Bash 脚本指南》有一章关于循环的内容，其中列举了各种各样的 for 循环实例：

<http://tldp.org/LDP/abs/html/loops1.html>

* The Bash Reference Manual describes the looping compound commands, including for:
* 《Bash 参考手册》描述了循环复合命令，包括了 for 循环：

<http://www.gnu.org/software/bash/manual/bashref.html#Looping-Constructs>