Flow Control: looping with while/until.
Flujo de control: Bucle instrucción con mientras/hasta

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 executes only 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. 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 chapter.
En el capítulo anterior, desarrollamos un programa basado en menús para producir varios tipos de información del sistema. El programa funciona, pero todavía tiene un problema de usabilidad significativo. Ejecuta solo una opción y luego termina. Peor aún, si se realiza una selección no válida, el programa termina con un error, sin darle al usuario la oportunidad de volver a intentarlo. Sería mejor si pudiéramos construir de alguna manera el programa de manera que pudiera repetir la visualización y selección del menú una y otra vez, hasta que el usuario elija salir del programa. En este capítulo, veremos un concepto de programación llamado bucle, que puede usarse para hacer que partes de programas se repitan. El shell proporciona tres comandos compuestos para realizar un bucle. Veremos dos de ellos en este capítulo y el tercero en un capítulo posterior.

Looping
Bucle

Daily life is full of repeated activities. Going to work each day, walking the dog, and 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:
La vida diaria está llena de actividades repetidas. Ir a trabajar todos los días, pasear al perro y cortar una zanahoria son tareas que implican repetir una serie de pasos. Consideremos cortar una zanahoria. Si expresamos esta actividad en pseudocódigo, podría verse así:

1. Get cutting board.
   Obtener tabla de cortar
2. Get knife.
   Conseguir un cuchillo
3. Place carrot on cutting board.
   Coloque la zanahoria en la tabla de cortar.
4. Lift knife.
   Levantar cuchillo
5. Advance carrot.
   Avance zanahoria.
6. Slice carrot.
   Cortar la zanahoria.
7. If entire carrot sliced, then quit; else go to step 4.
   Si se corta la zanahoria entera, deje de hacerlo; si no, vaya al paso 4

Steps 4 through 7 form a loop. The actions within the loop are repeated until the condition, "entire carrot sliced," is reached.
Los pasos 4 a 7 forman un bucle. Las acciones dentro del ciclo se repiten hasta que se alcanza la condición "zanahoria entera en rodajas".

while
Mientras

bash can express a similar idea. Let's say we wanted to display five numbers in sequential order from 1 to 5. A bash script could be constructed as follows:
bash puede expresar una idea similar. Supongamos que queremos mostrar cinco números en orden secuencial del 1 al 5. Un script bash podría construirse de la siguiente manera:

```
#!/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:
Cuando se ejecuta, este script muestra lo siguiente:

```
[me@linuxbox ~]$ while-count
1
2
3
4
5
Finished.
```

The syntax of the while command is as follows:
La sintaxis de el comando while es la siguiente:

```
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 previous script, the variable count is created and assigned an initial value of 1. The while command evaluates the exit status of the [[]] compound command. As long as the [[]] command returns an exit status of zero, the commands within the loop are executed. At the end of each cycle, the [[]] command is repeated. After 5 iterations of the loop, the value of count has increased to 6, the [[]] command no longer returns an exit status of zero, and the loop terminates. The program continues with the next statement following the loop.
Como si, while evalúa el estado de salida de una lista de comandos. Siempre que el estado de salida sea cero, ejecuta los comandos dentro del bucle. En el script anterior, la variable count se crea y se le asigna un valor inicial de 1. El comando while evalúa el estado de salida del comando compuesto [[]]. Siempre que el comando [[]] devuelva un estado de salida de cero, se ejecutan los comandos dentro del ciclo. Al final de cada ciclo, se repite el comando [[]]. Después de 5 iteraciones del ciclo, el valor de count ha aumentado a 6, el comando [[]] ya no devuelve un estado de salida de cero y el ciclo termina. El programa continúa con la siguiente declaración siguiendo el ciclo.

We can use a while loop to improve the read-menu program from the previous chapter.
Podemos usar un ciclo while para mejorar el programa de menú de lectura del capítulo anterior.

```
#!/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.
Al encerrar el menú en un ciclo while, podemos hacer que el programa repita la visualización del menú después de cada selección. El bucle continúa mientras REPLY no sea igual a 0 y el menú se muestre nuevamente, dando al usuario la oportunidad de hacer otra selección. Al final de cada acción, se ejecuta un comando de suspensión, por lo que el programa se detendrá durante unos segundos para permitir que los resultados de la selección se vean antes de que se borre la pantalla y se vuelva a mostrar el menú. Una vez que REPLY es igual a 0, lo que indica la selección "salir", el ciclo termina y la ejecución continúa con la siguiente línea hecha.

Breaking Out of a Loop
Saliendo de un Bucle

bash provides two builtin commands that can be used to control program flow inside loops. The break command immediately terminates a loop, and program control resumes with the next statement following the loop. The continue command causes the remainder of 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 proporciona dos comandos integrados que pueden usarse para controlar el flujo del programa dentro de los bucles. El comando break termina inmediatamente un bucle y el control del programa se reanuda con la siguiente instrucción que sigue al bucle. El comando continue hace que se omita el resto del ciclo y el control del programa se reanuda con la siguiente iteración del ciclo. Aquí vemos una versión del programa de menú while que incorpora tanto romper como continuar:

```
#!/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 an 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.
En esta versión del script, configuramos un bucle sin fin (uno que nunca termina por sí solo) usando el comando true para proporcionar un estado de salida a while. Dado que true siempre saldrá con un estado de salida de cero, el ciclo nunca terminará. Esta es una técnica de creación de scripts sorprendentemente común. Dado que el ciclo nunca terminará por sí solo, depende del programador proporcionar alguna forma de salir del ciclo cuando sea el momento adecuado. En este script, el comando break se usa para salir del bucle cuando se elige la selección 0. El comando continue se ha incluido al final de las otras opciones de script para permitir una ejecución más eficiente. Al usar continuar, el script saltará el código que no es necesario cuando se identifica una selección. Por ejemplo, si se elige e identifica la selección 1, no hay razón para probar las otras selecciones.

until
Hasta

The until compound 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 5. We could get the same result by coding the script with until.
El comando compuesto hasta es muy parecido a while, excepto que en lugar de salir de un bucle cuando se encuentra un estado de salida distinto de cero, hace lo contrario. Un bucle hasta que continúa hasta que recibe un estado de salida cero. En nuestro script while-count, continuamos el ciclo siempre que el valor de la variable count fuera menor o igual a 5. Podríamos obtener el mismo resultado codificando el script con until.

```
#!/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.
Al cambiar la expresión de prueba a $ count -gt 5, hasta terminará el ciclo en el momento correcto. La decisión de utilizar el bucle while o hasta suele ser cuestión de elegir el que permite escribir la prueba más clara.

Reading Files with Loops
Leyendo archivos con bucles

while and until can process standard input. This allows files to be processed with while and until loops. In the following example, we will display the contents of the distros.txt file used in earlier chapters:
while y until pueden procesar la entrada estándar. Esto permite que los archivos se procesen con bucles while y until. En el siguiente ejemplo, mostraremos el contenido del archivo distros.txt utilizado en capítulos anteriores:

```
#!/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.
Para redirigir un archivo al bucle, colocamos el operador de redirección después de la instrucción done. El ciclo usará read para ingresar los campos del archivo redirigido. El comando de lectura saldrá después de leer cada línea, con un estado de salida cero hasta que se alcance el final del archivo. En ese punto, saldrá con un estado de salida distinto de cero, terminando así el bucle. También es posible canalizar la entrada estándar en un bucle.

```
#!/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.
Aquí tomamos la salida del comando sort y mostramos el flujo de texto. Sin embargo, es importante recordar que dado que una tubería ejecutará el bucle en una subcapa, cualquier variable creada o asignada dentro del bucle se perderá cuando el bucle termine.

Summing Up
Resumen

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.
Con la introducción de bucles y nuestros encuentros anteriores con ramificaciones, subrutinas y secuencias, hemos cubierto los principales tipos de control de flujo utilizados en los programas. bash tiene algunos trucos más bajo la manga, pero son mejoras en estos conceptos básicos