33

Flow control: Looping with For
Control de flujo: bucle con For

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 popular construct in bash scripting.
En este capítulo final sobre el control de flujo, veremos otra de las construcciones de bucle del shell. El bucle for se diferencia de los bucles while y until en que proporciona un medio para procesar secuencias durante un bucle. Esto resulta muy útil a la hora de programar. En consecuencia, el bucle for es una construcción popular en los scripts bash.

A for loop is implemented, naturally enough, with the for compound command. In bash , for is available in two forms.
Un bucle for se implementa, naturalmente, con el comando compuesto for. En bash, for está disponible en dos formas.

for: Traditional Shell Form
para: Forma de concha tradicional

The original for command's syntax is as follows:
El original de la sintaxis del comando es el siguiente:

```
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.
donde variable es el nombre de una variable que se incrementará durante la ejecución del ciclo, palabras es una lista opcional de elementos que se asignarán secuencialmente a la variable y los comandos son los comandos que se ejecutarán en cada iteración del ciclo.

The for command is useful on the command line. We can easily demonstrate how it works.
El comando for es útil en la línea de comandos. Podemos demostrar fácilmente cómo funciona.

```
[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 assigned 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.
En este ejemplo, se proporciona una lista de cuatro palabras: A, B, C y D. Con una lista de cuatro palabras, el ciclo se ejecuta cuatro veces. Cada vez que se ejecuta el ciclo, se asigna una palabra a la variable i. Dentro del ciclo, tenemos un comando echo que muestra el valor de i para mostrar la asignación. Al igual que con los bucles while y until, la palabra clave done cierra el bucle.

The really powerful feature of for is the number of interesting ways we can create the list of words. For example, we can do it through brace expansion, like so:
La característica realmente poderosa de for es la cantidad de formas interesantes en las que podemos crear la lista de palabras. Por ejemplo, podemos hacerlo mediante expansión de llaves, así:

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

or we could use a pathname expansion, as follows:
o podríamos usar una expansión de nombre de ruta, como sigue:

```
[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
```

Pathname expansion provides a nice, clean list of pathnames that can be processed in the loop. The one precaution needed is to check that the expansion actually matched something. By default, if the expansion fails to match any files, the wildcards themselves (distros*.txt in the preceding example) will be returned. To guard against this, we would code the preceding example in a script this way:
La expansión de nombre de ruta proporciona una lista limpia y agradable de nombres de ruta que se pueden procesar en el ciclo. La única precaución necesaria es comprobar que la expansión realmente coincide con algo. De forma predeterminada, si la expansión no coincide con ningún archivo, se devolverán los propios comodines (distros * .txt en el ejemplo anterior). Para evitar esto, codificaríamos el ejemplo anterior en un script de esta manera:

```
for i in distros*.txt; do
    if [[ -e "$i" ]]; then
        echo "$i"
    fi
done
```

By adding a test for file existence, we will ignore a failed expansion.
Al agregar una prueba de existencia de archivos, ignoraremos una expansión fallida.

Another common method of word production is command substitution.
Otro método común de producción de palabras es la sustitución de comandos.

```
#!/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 -n "$i" | wc -c)"
            if (( len > max_len )); then
                max_len="$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 whether the current word is the longest found so far. When the loop concludes, the longest word is displayed.
En este ejemplo, buscamos la cadena más larga que se encuentra dentro de un archivo. Cuando se le da uno o más nombres de archivo en la línea de comando, este programa usa el programa de cadenas (que se incluye en el paquete GNU binutils) para generar una lista de "palabras" de texto legibles en cada archivo. El bucle for procesa cada palabra por turno y determina si la palabra actual es la más larga encontrada hasta ahora. Cuando concluye el ciclo, se muestra la palabra más larga.

One thing to note here is that, contrary to our usual practice, we do not surround the command substitution $(strings "$1") with double quotes. This is because we actually want word splitting to occur to give us our list. If we had surrounded the command substitution with quotes, it would produce only a single word containing every string in the file. That’s not exactly what we are looking for.
Una cosa a tener en cuenta aquí es que, contrariamente a nuestra práctica habitual, no rodeamos la sustitución de comandos $ (cadenas "$ 1") entre comillas dobles. Esto se debe a que realmente queremos que se produzca la división de palabras para obtener nuestra lista. Si hubiéramos rodeado la sustitución del comando con comillas, produciría solo una palabra que contenga todas las cadenas del archivo. Eso no es exactamente lo que estamos buscando.

If the optional in words portion of the for command is omitted, for defaults to processing the positional parameters. We will modify our longest-word script to use this method:
Si se omite la parte opcional en palabras del comando for, por defecto se procesan los parámetros posicionales. Modificaremos nuestro script de palabras más largas para usar este método:

```
#!/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="$(echo -n "$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.

Como podemos ver, hemos cambiado el bucle más externo para usar for en lugar de while. Al omitir la lista de palabras en el comando for, se utilizan en su lugar los parámetros posicionales. Dentro del ciclo, las instancias anteriores de la variable i se han cambiado a la variable j. También se ha eliminado el uso de turno.

Why i?
¿Por qué yo?

You may have noticed that the variable i was chosen for each of the previous for loop examples. 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.
Es posible que haya notado que la variable i fue elegida para cada uno de los ejemplos anteriores de bucle for. ¿Por qué? En realidad, no hay una razón específica además de la tradición. La variable usada con for puede ser cualquier variable válida, pero i es la más común, seguida de j y k.

The basis of this tradition comes from the Fortran programming language. In Fortran, undeclared variables starting with the letters I, J, K, L, and M are automatically typed as integers, while variables beginning with any other letter are typed as reals (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.
La base de esta tradición proviene del lenguaje de programación Fortran. En Fortran, las variables no declaradas que comienzan con las letras I, J, K, L y M se escriben automáticamente como números enteros, mientras que las variables que comienzan con cualquier otra letra se escriben como reales (números con fracciones decimales). Este comportamiento llevó a los programadores a usar las variables I, J y K para las variables de ciclo, ya que era menos trabajo usarlas cuando se necesitaba una variable temporal (como suelen ser las variables de ciclo).

It also led to the following Fortran-based witticism: “GOD is real, unless declared integer.”
También condujo al siguiente ingenio basado en Fortran: "DIOS es real, a menos que se declare entero".

for: C Language Form
for: Formulario en lenguaje C

Recent versions of bash have added a second form of the for command syntax, one that resembles the form found in the C programming language.
Las versiones recientes de bash han agregado una segunda forma de sintaxis de comandos for, una que se parece a la forma que se encuentra en el lenguaje de programación C.

Many other languages support this form, as well.
Muchos otros idiomas también admiten este formulario.

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

```
done
```

Here, expression1 , expression2 , and expression3 are arithmetic expressions, and ***commands*** are the commands to be performed during each iteration of the loop.
Aquí, expression1, expression2 y expression3 son expresiones aritméticas, y ***comandos*** son los comandos que se ejecutarán durante cada iteración del ciclo.

In terms of behavior, this form is equivalent to the following construct.
En términos de comportamiento, esta forma es equivalente a la siguiente construcción.

```
(( 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.
expresión1 se usa para inicializar las condiciones del ciclo, ***expresión2*** se usa para determinar cuándo finaliza el ciclo y ***expresión3*** se lleva a cabo al final de cada iteración del bucle.

Here is a typical application:
Esta es una aplicación típica:

```
#!/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:
Cuando se ejecuta, produce la siguiente salida:

```
[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 1 each time the loop repeats.
En este ejemplo, expression1 inicializa la variable i con el valor de cero, expression2 permite que el ciclo continúe siempre que el valor de i permanezca menor que 5, y expression3 incrementa el valor de i en 1 cada vez que el ciclo se repite.

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.
La forma en lenguaje C de for es útil siempre que se necesite una secuencia numérica. Veremos varias aplicaciones para esto en los próximos dos capítulos.

Summing Up
Resumen

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:
Con nuestro conocimiento del comando for, ahora aplicaremos las mejoras finales a nuestro script sys_info_page. Actualmente, la función report_home_space tiene este aspecto:

```
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.
A continuación, lo reescribiremos para proporcionar más detalles para el directorio de inicio de cada usuario e incluir el número total de archivos y subdirectorios en cada uno.

```
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 variables 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.
Esta reescritura aplica mucho de lo que hemos aprendido hasta ahora. Seguimos probando para el superusuario, pero en lugar de realizar el conjunto completo de acciones como parte del if, establecemos algunas variables que se usarán más adelante en un bucle for. Hemos agregado varias variables locales a la función y hemos utilizado printf para formatear parte de la salida.