Flow control: Branching with case
Control de flujo: ramificación con case

In this chapter, we will continue our look at flow control. In Chapter 28, we constructed some simple menus and built the logic used to act on a user's selection. To do this, we used a series of if commands to identify which of the possible choices had been selected. This type of logical construct appears frequently in programs—so much so that many programming languages (including the shell) provide a special flow control mechanism for multiple-choice decisions.
En este capítulo, continuaremos con nuestro análisis del control de flujo. En el Capítulo 28, construimos algunos menús simples y construimos la lógica utilizada para actuar sobre la selección de un usuario. Para hacer esto, usamos una serie de comandos if para identificar cuál de las posibles opciones había sido seleccionada. Este tipo de construcción lógica aparece con frecuencia en los programas, tanto que muchos lenguajes de programación (incluido el shell) proporcionan un mecanismo de control de flujo especial para decisiones de opción múltiple.

The case Command
El comando case

In bash, the multiple-choice compound command is called case . It has the following syntax.
En bash, el comando compuesto de opción múltiple se llama case. Tiene la siguiente sintaxis.

```
case word in
    [pattern [| pattern]...) commands ;;]...
esac
```

If we look at the read-menu program from Chapter 28, we see the logic used to act on a user's selection.
Si miramos el programa de menú de lectura del Capítulo 28, vemos la lógica utilizada para actuar sobre la selección de un usuario.

```
#!/bin/bash

# read-menu: a menu driven system information program

clear
echo "
Please Select:
1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
```

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

if [[ "$REPLY" =~ ^[0-3]$ ]]; then
    if [[ "$REPLY" == 0 ]]; then
        echo "Program terminated."
        exit
    fi
    if [[ "$REPLY" == 1 ]]; then
        echo "Hostname: $HOSTNAME"
        uptime
        exit
    fi
    if [[ "$REPLY" == 2 ]]; then
        df -h
        exit
    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
        exit
    fi
else
    echo "Invalid entry." >&2
    exit 1
fi
```

Using case , we can replace this logic with something simpler.

Usando case, podemos reemplazar esta lógica con algo más simple.

```
#!/bin/bash

# case-menu: a menu driven system information program
clear
echo "
Please Select:
1. Display System Information
2. Display Disk Space
3. Display Home Space Utilization
0. Quit
"
read -p "Enter selection [0-3] > "

case "$REPLY" in
```

```
    0)  echo "Program terminated."
        exit
        ;;
    1)  echo "Hostname: $HOSTNAME"
        uptime
        ;;
    2)  df -h
        ;;
    3)  if [[ "$(id -u)" -eq 0 ]]; then
            echo "Home Space Utilization (All Users)"
            du -sh /home/*
        else
            echo "Home Space Utilization ($USER)"
            du -sh "$HOME"
        fi
        ;;
    *)  echo "Invalid entry" >&2
        exit 1
        ;;
esac
```

The case command looks at the value of word , which in our example is the value of the REPLY variable, and then attempts to match it against one of the specified patterns. When a match is found, the commands associated with the specified pattern are executed. After a match is found, no further matches are attempted.
El comando case mira el valor de word, que en nuestro ejemplo es el valor de la variable REPLY, y luego intenta compararlo con uno de los patrones especificados. Cuando se encuentra una coincidencia, se ejecutan los comandos asociados con el patrón especificado. Una vez que se encuentra una coincidencia, no se intentan más coincidencias.

Patterns
Patrones

The patterns used by case are the same as those used by pathname expansion. Patterns are terminated with a) character. Table 31-1 describes some valid patterns.
Los patrones usados por caso son los mismos que los usados por la expansión de nombre de ruta. Los patrones terminan con un) carácter. La Tabla 31-1 describe algunos patrones válidos.

Table 31-1: case Pattern Examples
Tabla 31-1: Ejemplos de patrones de casos

Pattern	Description
a)	Matches if word equals a. Coincide si la palabra es igual a a.

Pattern	Description
[[:alpha:]])	Matches if word is a single alphabetic character. Coincide si la palabra es un solo carácter alfabético.
???)	Matches if word is exactly three characters long. Coincide si la palabra tiene exactamente tres caracteres.
*.txt)	Matches if word ends with the characters .txt. Coincide si la palabra termina con los caracteres .txt.
*)	Matches any value of word . It is good practice to include this as the last pattern in a case command to catch any values of word that did not match a previous pattern, that is, to catch any possible invalid values. Coincide con cualquier valor de palabra. Es una buena práctica incluir esto como el último patrón en un comando de caso para capturar cualquier valor de palabra que no coincida con un patrón anterior, es decir, para detectar posibles valores no válidos.

Here is an example of patterns at work:
Aquí hay un ejemplo de patrones en funcionamiento

```
#!/bin/bash

read -p "enter word > "

case "$REPLY" in
    [[:alpha:]]) echo "is a single alphabetic character." ;;
    [ABC][0-9])  echo "is A, B, or C followed by a digit." ;;
    ???) echo "is three characters long." ;;
    *.txt) echo "is a word ending in '.txt'" ;;
    *) echo "is something else." ;;
esac
```

It is also possible to combine multiple patterns using the vertical bar character as a separator. This creates an “or” conditional pattern. This is useful for such thin
También es posible combinar varios patrones utilizando el carácter de barra vertical como separador. Esto crea un patrón condicional "o". Esto es útil para personas tan delgadas

```
#!/bin/bash

# case-menu: a menu driven system information program

clear
echo "
Please Select:
A. Display System Information
```

```
B. Display Disk Space
C. Display Home Space Utilization
Q. Quit
"
read -p "Enter selection [A, B, C or Q] > "
case "$REPLY" in
    q|Q)    echo "Program terminated."
            exit
            ;;
    a|A)    echo "Hostname: $HOSTNAME"
            uptime
            ;;
    b|B)    df -h
            ;;
    c|C)    if [[ "$(id -u)" -eq 0 ]]; then
                echo "Home Space Utilization (All Users)"
                du -sh /home/*
            else
                echo "Home Space Utilization ($USER)"
                du -sh "$HOME"
            fi
            ;;
    *)      echo "Invalid entry" >&2
            exit 1
            ;;
esac
```

Here, we modify the case-menu program to use letters instead of digits for menu selection. Notice how the new patterns allow for entry of both uppercase and lowercase letters.
Aquí, modificamos el programa de menú de casos para usar letras en lugar de dígitos para la selección del menú. Observe cómo los nuevos patrones permiten la entrada de letras mayúsculas y minúsculas.

Performing Multiple Actions
Realización de múltiples acciones

In versions of bash prior to 4.0, case allowed only one action to be performed on a successful match. After a successful match, the command would terminate. Here we see a script that tests a character:
En las versiones de bash anteriores a la 4.0, el caso solo permitía realizar una acción en una coincidencia exitosa. Después de una coincidencia exitosa, el comando terminaría. Aquí vemos un guión que prueba un personaje:

```
#!/bin/bash

# case4-1: test a character
read -n 1 -p "Type a character > "
echo
```

```
case "$REPLY" in
    [[:upper:]]) echo "'$REPLY' is upper case." ;;
    [[:lower:]]) echo "'$REPLY' is lower case." ;;
    [[:alpha:]]) echo "'$REPLY' is alphabetic." ;;
    [[:digit:]]) echo "'$REPLY' is a digit." ;;
    [[:graph:]]) echo "'$REPLY' is a visible character." ;;
    [[:punct:]]) echo "'$REPLY' is a punctuation symbol." ;;
    [[:space:]]) echo "'$REPLY' is a whitespace character." ;;
    [[:xdigit:]]) echo "'$REPLY' is a hexadecimal digit." ;;
esac
```

Running this script produces this:
La ejecución de este script produce esto:

```
[me@linuxbox ~]$ case4-1
Type a character > a
'a' is lower case.
```

The script works for the most part but fails if a character matches more than one of the POSIX character classes. For example, the character a is both lowercase and alphabetic, as well as a hexadecimal digit. In bash prior to version 4.0, there was no way for case to match more than one test. Modern versions of bash add the ;;& notation to terminate each action, so now we can do this:
La secuencia de comandos funciona en su mayor parte, pero falla si un personaje coincide con más de una de las clases de caracteres POSIX. Por ejemplo, el carácter a es tanto en minúsculas como alfabético, así como un dígito hexadecimal. En bash antes de la versión 4.0, no había forma de que case coincidiera con más de una prueba. Las versiones modernas de bash agregan la notación ;; & para terminar cada acción, así que ahora podemos hacer esto:

```
#!/bin/bash

# case4-1: test a character
read -n 1 -p "Type a character > "
echo

case "$REPLY" in
    [[:upper:]]) echo "'$REPLY' is upper case." ;;&
    [[:lower:]]) echo "'$REPLY' is lower case." ;;&
    [[:alpha:]]) echo "'$REPLY' is alphabetic." ;;&
    [[:digit:]]) echo "'$REPLY' is a digit." ;;&
    [[:graph:]]) echo "'$REPLY' is a visible character." ;;&
    [[:punct:]]) echo "'$REPLY' is a punctuation symbol." ;;&
    [[:space:]]) echo "'$REPLY' is a whitespace character." ;;&
```

```
    [[:xdigit:]]) echo "'$REPLY' is a hexadecimal digit." ;;&
esac
```

When we run this script, we get this:
Cuando ejecutamos este script, obtenemos esto:

```
[me@linuxbox ~]$ case4-2
Type a character > a
'a' is lower case.
'a' is alphabetic.
'a' is a visible character.
'a' is a hexadecimal digit.
```

The addition of the ;;& syntax allows case to continue to the next test rather than simply terminating.
La adición de la sintaxis ;; & permite que el caso continúe hasta la siguiente prueba en lugar de simplemente terminar.

Summing Up
Resumen

The case command is a handy addition to our bag of programming tricks. As we will see in the next chapter, it's the perfect tool for handling certain types of problems.
El comando case es una adición útil a nuestra bolsa de trucos de programación. Como veremos en el próximo capítulo, es la herramienta perfecta para manejar ciertos tipos de problemas.