35

Arrays
Matrices

In the previous chapter, we looked at how the shell can manipulate strings and numbers. The data types we have looked at so far are known in computer science circles as scalar variables; that is, they are variables that contain a single value.
En el capítulo anterior, vimos cómo el shell puede manipular cadenas y números. Los tipos de datos que hemos examinado hasta ahora se conocen en los círculos de la informática como variables escalares; es decir, son variables que contienen un solo valor.

In this chapter, we will look at another kind of data structure called an array, which holds multiple values. Arrays are a feature of virtually every programming language. The shell supports them, too, though in a rather limited fashion. Even so, they can be very useful for solving some types of programming problems.
En este capítulo, veremos otro tipo de estructura de datos llamada matriz, que contiene múltiples valores. Las matrices son una característica de prácticamente todos los lenguajes de programación. El caparazón también los soporta, aunque de forma bastante limitada. Aun así, pueden resultar muy útiles para resolver algunos tipos de problemas de programación.

What Are Arrays?
¿Qué son las matrices?

Arrays are variables that hold more than one value at a time. Arrays are organized like a table. Let's consider a spreadsheet as an example. A spreadsheet acts like a two-dimensional array. It has both rows and columns, and an individual cell in the spreadsheet can be located according to its row and column address. An array behaves the same way. An array has cells, which are called elements, and each element contains data. An individual array element is accessed using an address called an index or subscript.
Las matrices son variables que contienen más de un valor a la vez. Las matrices están organizadas como una tabla. Consideremos una hoja de cálculo como ejemplo. Una hoja de cálculo actúa como una matriz bidimensional. Tiene filas y columnas, y una celda individual en la hoja de cálculo se puede ubicar de acuerdo con su dirección de fila y columna. Una matriz se comporta de la misma manera. Una matriz tiene celdas, que se denominan elementos, y cada elemento contiene datos. Se accede a un elemento de matriz individual mediante una dirección denominada índice o subíndice.

Most programming languages support multidimensional arrays. A spreadsheet is an example of a multidimensional array with two dimensions, width and height. Many languages support arrays with an arbitrary number of dimensions, though two- and three-dimensional arrays are probably the most commonly used.
La mayoría de los lenguajes de programación admiten matrices multidimensionales. Una hoja de cálculo es

un ejemplo de una matriz multidimensional con dos dimensiones, ancho y alto. Muchos lenguajes admiten matrices con un número arbitrario de dimensiones, aunque las matrices bidimensionales y tridimensionales son probablemente las más utilizadas.

Arrays in bash are limited to a single dimension. We can think of them as a spreadsheet with a single column. Even with this limitation, there are many applications for them. Array support first appeared in bash version 2.
Las matrices en bash están limitadas a una sola dimensión. Podemos pensar en ellos como una hoja de cálculo con una sola columna. Incluso con esta limitación, existen muchas aplicaciones para ellos. La compatibilidad con matrices apareció por primera vez en la versión 2 de bash.

The original Unix shell program, sh , did not support arrays at all.
El programa de shell de Unix original, sh, no admitía matrices en absoluto.

Creating an Array
Crear una matriz

Array variables are named just like other bash variables and are created automatically when they are accessed. Here is an example:
Las variables de matriz se nombran igual que otras variables de bash y se crean automáticamente cuando se accede a ellas. Aquí hay un ejemplo:

```
[me@linuxbox ~]$ a[1]=foo
[me@linuxbox ~]$ echo ${a[1]}
foo
```

Here we see an example of both the assignment and access of an array element. With the first command, element 1 of array a is assigned the value foo . The second command displays the stored value of element 1. The use of braces in the second command is required to prevent the shell from attempting pathname expansion on the name of the array element.
Aquí vemos un ejemplo de la asignación y el acceso de un elemento de matriz. Con el primer comando, al elemento 1 de la matriz a se le asigna el valor foo. El segundo comando muestra el valor almacenado del elemento 1. El uso de llaves en el segundo comando es necesario para evitar que el shell intente expandir el nombre de la ruta en el nombre del elemento de la matriz.

An array can also be created with the declare command.
También se puede crear una matriz con el comando declare.

```
[me@linuxbox ~]$ declare -a a
```

Using the -a option, this example of declare creates the array a.
Usando la opción -a, este ejemplo de declare crea la matriz a.

Assigning Values to an Array
Asignar valores a una matriz

Values may be assigned in one of two ways. Single values may be assigned using the following syntax:
Los valores se pueden asignar de dos formas. Se pueden asignar valores únicos utilizando la siguiente sintaxis:

```
name[subscript]=value
```

where name is the name of the array and subscript is an integer (or arithmetic expression) greater than or equal to zero. Note that an array's first element is subscript zero, not one. value is a string or integer assigned to the array element.
donde nombre es el nombre de la matriz y el subíndice es un número entero (o expresión aritmética) mayor o igual a cero. Tenga en cuenta que el primer elemento de una matriz es el subíndice cero, no uno. valor es una cadena o un número entero asignado al elemento de la matriz.

Multiple values may be assigned using the following syntax:
Se pueden asignar varios valores utilizando la siguiente sintaxis:

```
name=(value1 value2 ...)
```

where name is the name of the array and the value placeholders are values assigned sequentially to elements of the array, starting with element zero.
donde nombre es el nombre de la matriz y los marcadores de posición de valor son valores asignados secuencialmente a los elementos de la matriz, comenzando con el elemento cero

For example, if we wanted to assign abbreviated days of the week to the array days , we could do this:
Por ejemplo, si quisiéramos asignar días de la semana abreviados a los días de la matriz, podríamos hacer esto:

```
[me@linuxbox ~]$ days=(Sun Mon Tue Wed Thu Fri Sat)
```

It is also possible to assign values to a specific element by specifying a subscript for each value.
También es posible asignar valores a un elemento específico especificando un subíndice para cada valor.

```
[me@linuxbox ~]$ days=([0]=Sun [1]=Mon [2]=Tue [3]=Wed [4]=Thu [5]=Fri [6]=Sat)
```

Accessing Array Elements
Acceso a elementos de matriz

So, what are arrays good for? Just as many data-management tasks can be performed with a spreadsheet program, many programming tasks can be performed with arrays.
Entonces, ¿para qué sirven las matrices? Así como muchas tareas de administración de datos se pueden realizar con un programa de hoja de cálculo, muchas tareas de programación se pueden realizar con arreglos.

Let's consider a simple data-gathering and presentation example. We will construct a script that examines the modification times of the files in a specified directory. From this data, our script will output a table showing at what hour of the day the files were last modified. Such a script could be used to determine when a system is most active. This script, called hours, produces this result:
Consideremos un ejemplo sencillo de recopilación y presentación de datos. Construiremos un script que examina los tiempos de modificación de los archivos en un directorio específico. A partir de estos datos, nuestro script generará una tabla que muestra a qué hora del día se modificaron por última vez los archivos. Tal secuencia de comandos podría usarse para determinar cuándo un sistema está más activo. Este script, llamado horas, produce este resultado:

```
[me@linuxbox ~]$ hours .
Hour Files Hour Files
---- ----- ---- -----
00   0     12   11
01   1     13   7
02   0     14   1

...

Total files = 80
```

We execute the hours program, specifying the current directory as the target. It produces a table showing, for each hour of the day (0–23), how many files were last modified. The code to produce this is as follows:
Ejecutamos el programa de horas, especificando el directorio actual como destino. Produce una tabla que muestra, para cada hora del día (0-23), cuántos archivos se modificaron por última vez. El código para producir esto es el siguiente:

```
#!/bin/bash

# hours: script to count files by modification time

usage () {
    echo "usage: ${0##*/} directory" >&2
}

# Check that argument is a directory

if [[ ! -d "$1" ]]; then
```

```
    usage
    exit 1
fi

# Initialize array

for i in {0..23}; do hours[i]=0; done

# Collect data

for i in $(stat -c %y "$1"/* | cut -c 12-13); do
    j="${i#0}"
    ((++hours[j]))
    ((++count))
done

# Display data

echo -e "Hour\tFiles\tHour\tFiles"
echo -e "----\t-----\t----\t-----"
for i in {0..11}; do
    j=$((i + 12))
    printf "%02d\t%d\t%02d\t%d\n" \
        "$i" \
        "${hours[i]}" \
        "$j" \
        "${hours[j]}"
done
printf "\nTotal files = %d\n" $count
```

The script consists of one function (usage) and a main body with four sections. In the first section, we check that there is a command line argument and that it is a directory. If it is not, we display the usage message and exit. The second section initializes the array hours . It does this by assigning each element a value of zero. There is no special requirement to prepare arrays prior to use, but our script needs to ensure that no element is empty. Note the interesting way the loop is constructed. By employing brace expansion ({0..23}), we are able to easily generate a sequence of words for the for command.
El script consta de una función (uso) y un cuerpo principal con cuatro secciones. En la primera sección, verificamos que haya un argumento de línea de comando y que sea un directorio. Si no es así, mostramos el mensaje de uso y salimos. La segunda sección inicializa las horas del arreglo. Lo hace asignando a cada elemento un valor de cero. No hay ningún requisito especial para preparar matrices antes de su uso, pero nuestro script debe asegurarse de que ningún elemento esté vacío. Tenga en cuenta la forma interesante en que se construye el bucle. Al emplear la expansión de llaves ({0..23}), podemos generar fácilmente una secuencia de palabras para el comando for.

The next section gathers the data by running the stat program on each file in the directory. We use cut to extract the two-digit hour from the result. Inside the loop, we need to remove leading zeros from the hour field since the shell will try (and ultimately fail) to interpret values 00 through 09 as octal numbers (see Table 34-2). Next, we increment the value of the array element corresponding with the hour of the day. Finally, we increment a counter (count) to track the total number of files in the directory.

La siguiente sección recopila los datos ejecutando el programa stat en cada archivo del directorio. Usamos cortar para extraer la hora de dos dígitos del resultado. Dentro del ciclo, necesitamos eliminar los ceros iniciales del campo de la hora, ya que el shell intentará (y finalmente fallará) interpretar los valores del 00 al 09 como números octales (consulte la Tabla 34-2). A continuación, incrementamos el valor del elemento de matriz correspondiente a la hora del día. Finalmente, incrementamos un contador (recuento) para rastrear el número total de archivos en el directorio.

The last section of the script displays the contents of the array. We first output a couple of header lines and then enter a loop that produces four columns of output. Lastly, we output the final tally of files.
La última sección del script muestra el contenido de la matriz. Primero generamos un par de líneas de encabezado y luego ingresamos un ciclo que produce cuatro columnas de salida. Por último, generamos el recuento final de archivos.

Array Operations
Operaciones de matriz

There are many common array operations. Such things as deleting arrays, determining their size, sorting, and so on, have many applications in scripting.
Hay muchas operaciones de matriz comunes. Cosas como eliminar matrices, determinar su tamaño, ordenar, etc., tienen muchas aplicaciones en las secuencias de comandos.

Outputting the Entire Contents of an Array
Salida de todo el contenido de una matriz

The subscripts * and @ can be used to access every element in an array. As with positional parameters, the @ notation is the more useful of the two.
Los subíndices * y @ se pueden utilizar para acceder a todos los elementos de una matriz. Al igual que con los parámetros posicionales, la notación @ es la más útil de las dos.

Here is a demonstration:
Aquí hay una demostración:

```
[me@linuxbox ~]$ animals=("a dog" "a cat" "a fish")
[me@linuxbox ~]$ for i in ${animals[*]}; do echo $i; done
a
dog
a
cat
a
fish
[me@linuxbox ~]$ for i in ${animals[@]}; do echo $i; done
a
dog
a
cat
a
fish
```

```
[me@linuxbox ~]$ for i in "${animals[*]}"; do echo $i; done
a dog a cat a fish
[me@linuxbox ~]$ for i in "${animals[@]}"; do echo $i; done
a dog
a cat
a fish
```

We create the array animals and assign it three two-word strings. We then execute four loops to see the effect of word splitting on the array contents. The behavior of notations ${animals[*]} and ${animals[@]} is identical until they are quoted. The * notation results in a single word containing the array's contents, while the @ notation results in three two-word strings, which matches the array's "real" contents.
Creamos la matriz de animales y le asignamos tres cadenas de dos palabras. Luego ejecutamos cuatro ciclos para ver el efecto de la división de palabras en el contenido de la matriz. El comportamiento de las notaciones ${animales [*]} y ${animales [@]} es idéntico hasta que se citan. La notación * da como resultado una sola palabra que contiene el contenido de la matriz, mientras que la notación @ da como resultado tres cadenas de dos palabras, que coincide con el contenido "real" de la matriz.

Determining the Number of Array Elements
Determinación del número de elementos de la matriz

Using parameter expansion, we can determine the number of elements in an array in much the same way as finding the length of a string. Here is an example:
Usando la expansión de parámetros, podemos determinar el número de elementos en una matriz de la misma manera que encontramos la longitud de una cadena. Aquí hay un ejemplo:

```
[me@linuxbox ~]$ a[100]=foo
[me@linuxbox ~]$ echo ${#a[@]} # number of array elements
1
[me@linuxbox ~]$ echo ${#a[100]} # length of element 100
3
```

We create array a and assign the string foo to element 100. Next, we use parameter expansion to examine the length of the array, using the @ notation. Finally, we look at the length of element 100, which contains the string foo . It is interesting to note that while we assigned our string to element 100, bash reports only one element in the array. This differs from the behavior of some other languages in which the unused elements of the array (elements 0–99) would be initialized with empty values and counted. In bash , array elements exist only if they have been assigned a value regardless of their subscript.
Creamos la matriz ay asignamos la cadena foo al elemento 100. A continuación, usamos la expansión de parámetros para examinar la longitud de la matriz, usando la notación @. Finalmente, miramos la longitud del elemento 100, que contiene la cadena foo. Es interesante notar que mientras asignamos nuestra cadena al elemento 100, bash reporta solo un elemento en la matriz. Esto difiere del comportamiento de algunos otros lenguajes en los que los elementos no utilizados de la matriz (elementos 0–99) se

inicializarían con valores vacíos y se contarían. En bash, los elementos de la matriz existen solo si se les ha asignado un valor independientemente de su subíndice.

Finding the Subscripts Used by an Array
Encontrar los subíndices utilizados por una matriz

As bash allows arrays to contain “gaps” in the assignment of subscripts, it is sometimes useful to determine which elements actually exist. This can be done with a parameter expansion using the following forms:
Dado que bash permite que las matrices contengan "espacios" en la asignación de subíndices, a veces es útil determinar qué elementos existen realmente. Esto se puede hacer con una expansión de parámetros utilizando los siguientes formularios:

```
${!array[*]}
${!array[@]}
```

where array is the name of an array variable. Like the other expansions that use * and @ , the @ form enclosed in quotes is the most useful, as it expands into separate words.
donde matriz es el nombre de una variable de matriz. Al igual que las otras expansiones que usan * y @, la forma @ entre comillas es la más útil, ya que se expande en palabras separadas.

```
[me@linuxbox ~]$ foo=([2]=a [4]=b [6]=c)
[me@linuxbox ~]$ for i in "${foo[@]}"; do echo $i; done
a
b
c
[me@linuxbox ~]$ for i in "${!foo[@]}"; do echo $i; done
2
4
6
```

Adding Elements to the End of an Array
Agregar elementos al final de una matriz

Knowing the number of elements in an array is no help if we need to append values to the end of an array since the values returned by the * and @ notations do not tell us the maximum array index in use. Fortunately, the shell provides us with a solution. By using the += assignment operator, we can automatically append values to the end of an array. Here, we assign three values to the array foo and then append three more:
Saber el número de elementos en una matriz no es de ayuda si necesitamos agregar valores al final de una

matriz, ya que los valores devueltos por las notaciones * y @ no nos dicen el índice máximo de matriz en uso. Afortunadamente, el caparazón nos proporciona una solución. Al usar el operador de asignación + =, podemos agregar automáticamente valores al final de una matriz. Aquí, asignamos tres valores a la matriz foo y luego agregamos tres más:

```
[me@linuxbox~]$ foo=(a b c)
[me@linuxbox~]$ echo ${foo[@]}
a b c
[me@linuxbox~]$ foo+=(d e f)
[me@linuxbox~]$ echo ${foo[@]}
a b c d e f
```

Sorting an Array
Ordenar una matriz

Just as with spreadsheets, it is often necessary to sort the values in a column of data. The shell has no direct way of doing this, but it's not hard to do with a little coding.
Al igual que con las hojas de cálculo, a menudo es necesario ordenar los valores en una columna de datos. El shell no tiene una forma directa de hacer esto, pero no es difícil hacerlo con un poco de codificación.

```
#!/bin/bash

# array-sort: Sort an array

a=(f e d c b a)

echo "Original array: ${a[@]}"
a_sorted=($(for i in "${a[@]}"; do echo $i; done | sort))
echo "Sorted array: ${a_sorted[@]}"
```

When executed, the script produces this:
Cuando se ejecuta, el script produce esto:

```
[me@linuxbox ~]$ array-sort
Original array: f e d c b a
Sorted array: a b c d e f
```

The script operates by copying the contents of the original array (a) into a second array (a_sorted) with a tricky piece of command substitution. This basic technique can be used to perform many kinds of

operations on the array by changing the design of the pipeline.
El script funciona copiando el contenido de la matriz original (a) en una segunda matriz (a_sorted) con una complicada sustitución de comandos. Esta técnica básica se puede utilizar para realizar muchos tipos de operaciones en la matriz cambiando el diseño de la tubería.

Deleting an Array
Eliminar una matriz

To delete an array, use the **unset** command.
Para eliminar una matriz, use el comando **unset**.

```
[me@linuxbox~]$ foo=(a b c d e f)
[me@linuxbox~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox~]$ unset foo       #deleting array
[me@linuxbox~]$ echo ${foo[@]}
[me@linuxbox ~]$
```

unset may also be used to delete single array elements.
unset también se puede utilizar para eliminar elementos de una sola matriz.

```
[me@linuxbox~]$ foo=(a b c d e f)
[me@linuxbox~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox~]$ unset 'foo[2]' #delete single array element
[me@linuxbox~]$ echo ${foo[@]}
a b d e f
```

In this example, we delete the third element of the array, subscript 2. Remember, arrays start with subscript zero, not one! Notice also that the array element must be quoted to prevent the shell from performing pathname expansion.
En este ejemplo, eliminamos el tercer elemento de la matriz, el subíndice 2. Recuerde, las matrices comienzan con el subíndice cero, ¡no uno! Tenga en cuenta también que el elemento de matriz debe estar entre comillas para evitar que el shell realice la expansión del nombre de ruta.

Interestingly, the assignment of an empty value to an array does not empty its contents.
Curiosamente, la asignación de un valor vacío a una matriz no vacía su contenido.

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ foo=
```

```
[me@linuxbox ~]$ echo ${foo[@]}
b c d e f
```

Any reference to an array variable without a subscript refers to element zero of the array.
Cualquier referencia a una variable de matriz sin un subíndice se refiere al elemento cero de la matriz.

```
[me@linuxbox~]$  foo=(a b c d e f)
[me@linuxbox~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox~]$ foo=A
[me@linuxbox~]$ echo ${foo[@]}
A b c d e f
```

Associative Arrays
Matrices asociativas

---

bash versions 4.0 and greater support associative arrays. Associative arrays use strings rather than integers as array indexes. This capability allows interesting new approaches to managing data. For example, we can create an array called colors and use color names as indexes.
Las versiones 4.0 y posteriores de bash admiten matrices asociativas. Las matrices asociativas utilizan cadenas en lugar de enteros como índices de matriz. Esta capacidad permite nuevos enfoques interesantes para la gestión de datos. Por ejemplo, podemos crear una matriz llamada colores y usar nombres de colores como índices.

```
declare -A colors
colors["red"]="#ff0000"
colors["green"]="#00ff00"
colors["blue"]="#0000ff"
```

Unlike integer indexed arrays, which are created by merely referencing them, associative arrays must be created with the declare command using the new -A option. Associative array elements are accessed in much the same way as integer-indexed arrays.
A diferencia de las matrices indexadas de números enteros, que se crean simplemente haciendo referencia a ellas, las matrices asociativas deben crearse con el comando declare utilizando la nueva opción -A. Se accede a los elementos de la matriz asociativa de la misma manera que a las matrices indexadas con números enteros.

```
echo ${colors["blue"]}
```

In the next chapter, we will look at a script that makes good use of associative arrays to produce an interesting report.
En el próximo capítulo, veremos un script que hace un buen uso de matrices asociativas para producir un informe interesante.

Asignacion dinamica de un Array.
Dynamic assignment of a matrix

A veces es necesario crear una matriz para luego asignar o ir asignando valores conforme sea necesario, en el ejemplo siguiente vemos como hacer esto con el array **matriz**.
Sometimes it is necessary to create a matrix and then assign or assign values as necessary, in the following example we see how to do this with the array **matrix**.

```
#! /bin/bash
shopt -s nullglob
declare -a matriz
for f in ./*.mp4 ./*.MP4;
do
    name=${f%.*}
    cmd='ffmpeg -i "$f" -vf scale=640x480:flags=lanczos -c:v libx264 -
preset slow -c:a aac -strict -2 -b:a 64k -ac 2 -f mpegts "$name.ts"'
    # cmd='ffmpeg -i "$f" -c copy -bsf:v h264_mp4toannexb -f mpegts
"$name.ts"'
    echo "file '$name.ts'" >> ficheros.txt
    matriz+=("$name.ts")
    eval $cmd
    notify-send "$name.ts" "finished operation" -u normal -t 100 -i face-
laugh
done
# o también:
# printf "file '%s'\n" ./*.mp4 ./*.MP4 > ficheros.txt
cat ficheros.txt
ffmpeg -f concat -safe 0 -i ficheros.txt -c copy -bsf:a aac_adtstoasc
resultado.mp4
# borramos los ficheros subproductos
for i in "${matriz[@]}"; do rm -f "${i}"; done
rm -f ficheros.txt
# enviamos una notificacion al escritorio personal.
notify-send "resultado.mp4" "finished operation" -u normal -t 100 -i face-
laugh
```

Summing Up
Resumen

If we search the bash man page for the word array, we find many instances of where bash makes use of array variables. Most of these are rather obscure, but they may provide occasional utility in some special circumstances.
Si buscamos en la página de manual de bash la palabra matriz, encontramos muchos casos en los que bash hace uso de variables de matriz. La mayoría de estos son bastante oscuros, pero pueden proporcionar una utilidad ocasional en algunas circunstancias especiales.

In fact, the entire topic of arrays is rather under-utilized in shell programming owing largely to the fact that the traditional Unix shell programs (such as sh) lacked any support for arrays. This lack of popularity is unfortunate because arrays are widely used in other programming languages and provide a powerful tool for solving many kinds of programming problems.
De hecho, todo el tema de las matrices está bastante infrautilizado en la programación de shell debido en gran parte al hecho de que los programas tradicionales de shell de Unix (como sh) carecían de soporte para matrices. Esta falta de popularidad es lamentable porque las matrices se utilizan ampliamente en otros lenguajes de programación y proporcionan una herramienta poderosa para resolver muchos tipos de problemas de programación.

Arrays and loops have a natural affinity and are often used together.
Las matrices y los bucles tienen una afinidad natural y, a menudo, se utilizan juntos.

The following form of loop is particularly well-suited to calculating array subscripts:
La siguiente forma de bucle es particularmente adecuada para calcular subíndices de matriz:

```
for ((expr; expr; expr))
```