Práctica 9.
Programación en Shell.
Listas y bucles.

Autor: Francisco de Asís Conde Rodríguez / Lina García Cabrera

Índice

Introducción

En la programación de *shell script*, se usan mucho las listas. Por ejemplo: hay muchas órdenes del sistema que devuelven como resultado una lista, también los argumentos de una orden son una lista.

Por esa razón, es fundamental saber cómo se recorre una lista para realizar algún tipo de procesamiento sobre cada uno de los elementos que la componen. También es necesario saber cómo se crean listas.

En esta sesión de prácticas, se estudia qué es una lista, cómo se crea, y cómo se pueden recorrer todos los elementos de una lista de una forma sencilla.

Manipulación de listas, vectores y cadenas

En Linux, las listas, los vectores (arrays) y cadenas (strings) son estructuras de datos que puedes manejar dentro de una variable en el shell.

¿Qué es una lista?

En la programación de *shell scripts*, una lista es un conjunto de palabras separadas por espacios o saltos de renglón. Por ejemplo:

```
manzana piña pera
```

es una lista en *shell script*.

Listas en variables

Por supuesto, se puede asignar una lista a un parámetro de tipo variable para usar a lo largo de la ejecución del programa. Para ello, se usa una asignación normal:

```
frutas="manzana piña pera"
```

Fíjate que las comillas dobles son necesarias, ya que si no el espacio en blanco se interpreta como un separador de órdenes y no de los distintos elementos en la lista.

Manipular listas

Para añadir elementos a una lista, sólo hay que concatenar los nuevos elementos. Por ejemplo:

```
frutas=${frutas}" papaya"
```

nos da la lista:

```
manzana piña pera papaya
```

En este caso, la lista contiene una cadena de texto que representa múltiples palabras separadas por espacios, pero todas están almacenadas en una sola variable. Para acceder a cada elemento de esta "lista", necesitarías dividir la cadena manualmente (por ejemplo, usando IFS y `read`, o `for` en Bash como veremos en breve).

Ejemplos de uso de listas

Definir listas en *shell script* es muy sencillo. Este script, llamado `ejemlista1.sh`:

```
#!/bin/bash
# Autor: Francisco de Asís Conde Rodríguez
# Descripción: Ejemplo de uso de listas
frutas="manzana piña pera"
echo ${frutas}
frutas=${frutas}" papaya"
echo ${frutas}
```

devuelve la siguiente salida:

```
$ ./ejemlista1.sh
manzana piña pera
manzana piña pera papaya
```

Manipulación de vectores

Un modo conveniente de manipular los parámetros de un script es como si éstos se trataran de un vector de cadenas, de tal modo que luego podamos referir cualquier parámetro a través de un índice. Un vector o array en Bash es una variable que almacena varios elementos en posiciones indexadas, lo que permite acceder a cada elemento individualmente por su índice. El modo en que se crea en bash un índice es muy sencillo, basta con colocar entre paréntesis los elementos que conforman el vector, separados por un espacio:

```
$ animales_domesticos=(gato perro loro)
```

Ahora, para acceder a un elemento del vector:

```
$ echo ${animales_domesticos[1]}
perro
```

Nótese que el primer elemento está en el índice 0.

Si deseamos manipular los argumentos de un script como si de un vector se tratara todo lo que tenemos que hacer es poner los argumentos entre paréntesis.

Por defecto, si no se indica el número del vector, nos devuelve el primero. Por esa razón, si preguntamos por el número de argumentos, lo que cuenta es el número de caracteres del primer valor.

```
$ echo ${animales_domesticos}
gato
$ echo ${#animales_domesticos}
4
```

Para preguntar por el número de elementos de la lista, se debería preguntar por toda la lista, se debe poner el `*` ó `@`.

```
$ echo ${#animales_domesticos[*]}
3
```

Para mostrar TODOS los elementos de una lista debes:

```
$ echo ${animales_domesticos[*]}
gato perro loro
```

Se pueden extraer sólo algunos elementos de la lista de este modo (desde el elemento que está en la casilla 1, dos elementos de la lista):

```
$ echo ${animales_domesticos[*]:1:2}
perro loro
```

Por ejemplo, si queremos trabajar con el siguiente vector:

0	1	2	3	4	5	6	7	8
elefante	loro	gato	oso	perro	caballo	cerdo	pato	vaca

Utilizaremos la siguiente orden para crearlo:

```
$ animales=(elefante loro gato oso perro caballo cerdo pato vaca)
```

A continuación vamos a ver algunas órdenes sobre vectores y cómo las interpreta la shell bash:

```
$ echo ${animales}
elefante
```
Lo estamos usando como string y cuando hay un espacio deja de mostrar el contenido

```
$ echo ${animales[*]}
elefante loro gato oso perro caballo cerdo pato vaca
$ echo ${#animales[*]}
9
```
Muestra un 9 porque tenemos 9 animales, es decir, 9 elementos en la lista.

```
$ echo ${animales[*]:3}
oso perro caballo cerdo pato vaca
```
Muestra desde la posición 3 todos los elementos de la lista.

```
$ echo ${animales[*]:0:1}
elefante
```
Muestra el primer elemento de la lista.

```
$ echo ${animales[*]:5}
caballo cerdo pato vaca
```
Muestra desde la posición 5, todos los elementos de la lista.

```
$ echo ${animales[*]:5:2}
caballo cerdo
```
Muestra desde la posición 5, 2 elementos de la lista.

```
$ echo ${animales[*]: -3}
cerdo pato vaca
```
Muestra los 3 últimos elementos de la lista **(atención al espacio antes del menos)**.

```
$ echo ${animales[*]: -4:2}
caballo cerdo
```
Me sitúo en la posición -4 y muestro 2 elementos (atención al espacio antes del menos).

Diferencias entre * y @

`"${array[*]}"`: Expande todos los elementos del array en una sola cadena, separados por el valor del IFS (por defecto es un espacio).

`"${array[@]}"`: Expande cada elemento como una cadena independiente, preservando cada elemento tal como está.

Manipulación de cadenas

La shell bash también permite manipular una cadena mediante el **operador ":"**.

Mediante `${cadena:posicion:longitud}` podemos extraer una subcadena de otra cadena. Si omitimos `:longitud`, entonces extraerá todos los caracteres hasta el final de cadena.

Véase el siguiente ejemplo:

```
$ cadena=abcdefgh
$ echo ${cadena:1:1}
b
```

En este ejemplo, indicamos que queremos seleccionar a partir del segundo carácter (se empieza a contar desde el 0), una cadena de longitud 1.

Por ejemplo en la cadena `string=abcABC123ABCabc` tendría el siguiente orden de caracteres:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	b	c	A	B	C	1	2	3	A	B	C	a	b	c

Veamos cómo funciona el operador de subcadena:

```
$ string=abcABC123ABCabc
```

`$ echo ${string:0}` `abcABC123ABCabc`	Como no se indica tamaño, se muestra desde la posición 0 toda la cadena.
`$ echo ${string:0:1}` `a`	Desde la posición 0, 1 caracter (primer caracter).
`$ echo ${string:7}` `23ABCabc`	Desde la posición 7, toda la cadena.
`$ echo ${string:7:3}` `23A`	Desde la posición 7, 3 caracteres.
`$ echo ${string:7:-3}` `23ABC`	Desde la posición 7 quitando los 3 caracteres últimos.
`$ echo ${string: -4}` `Cabc`	Los 4 últimos de la cadena (**atención al espacio antes del menos**)
`$ echo ${string: -4:2}` `Ca`	Me sitúo en la posición -4 y muestro 2 caracteres (atención al espacio antes del menos)

Es posible conocer la longitud de una cadena precediéndola de `#`:

```
$ cadena=abcdefgh
$ echo ${#cadena}
8
```

Órdenes que devuelven listas o vectores

Muchas órdenes del sistema devuelven listas de valores. Por ejemplo: la orden **`ls`** devuelve una lista con todos los enlaces (archivos, directorios, etc.) del directorio de trabajo actual.

Si se quiere utilizar en un *shell script* la lista que devuelve una orden como resultado, es necesario usar la **sustitución de órdenes** para que la orden se interprete correctamente.

La sustitución de órdenes se expresa encerrando la orden dentro de acentos graves **`` `orden` ``** o **`$( orden )`**. Muy importante: no confundir el acento grave: `` ` `` con el apóstrofe `'` o comilla simple.

Por ejemplo, si se escribe:

```
resultado='ls'
```

lo que se asigna a la variable resultado no es la lista de archivos del directorio actual, sino la cadena `ls`. Para que se asigne el resultado de la orden `ls`, hay que hacer la sustitución de órdenes:

```
resultado=`ls`
```

o bien:

```
resultado=$( ls )
```

En el caso de la sustitución **`$()`**, no es necesario que entre los paréntesis y la orden haya espacios tal y como pasaba con los corchetes.

Si se quiere convertir el resultado en un array, se puede hacer envolviendo la sustitución entre paréntesis (). Esto divide los elementos por espacios o saltos de línea.

```
resultado=( $(ls) )
```

Ahora `resultado` es un array, donde cada archivo del directorio es un elemento separado que se puede acceder mediante índices (resultado[0], resultado[1], etc.).

Ejemplos de uso de órdenes que devuelven listas o arrays.

Para usar los resultados de una orden que devuelve listas hay que usar la sustitución de órdenes **`` `orden` ``** ó **`$( orden )`** Por ejemplo, el siguiente script `ejemlista1`:

```
#!/bin/bash
# Autor: Francisco de Asís Conde Rodríguez
# Descripción: Ejemplo de uso de listas
resultado=`ls`
echo ${resultado}
```

Da como resultado:

```
$ ./ejemlista1.sh

bin Descargas Documentos Escritorio examples.desktop
Imágenes Música Plantillas Público Vídeos
```

Para usar los resultados de una orden que devuelve listas como un array, se usa la sustitución de órdenes orden o $(orden), envolviendo la sustitución entre paréntesis.

```
#!/bin/bash
# Autor: José Ramón Balsas Almagro
# Descripción: Ejemplo de uso de arrays con
#              resultados de órdenes
# Muestra el archivo o directorio más reciente y más antigo
elementos=($(ls --sort=time))
# elementos=(`ls --sort=time`)
echo "el más reciente es ${elementos[0]}"
echo "el más antiguo es ${elementos[*]: -1}"
```

Bucles para recorrer listas o vectores

En Bash, existen diferentes formas de iterar sobre listas o vectores. Una de ellas es el bucle `for`, que permite recorrer listas, arrays y secuencias de números de diferentes maneras.

Bucle `for` con una lista de palabras o elementos

Lo normal, es que un usuario quiera recorrer todos los elementos de una lista para realizar algún procesamiento sobre ellos.

El intérprete de órdenes proporciona un bucle especial para recorrer los elementos de una lista que es muy sencillo de manejar. Es el ciclo `for`, que funciona de forma distinta al ciclo `for` de C ó C++.

En *shell script*, el ciclo `for` no comprueba una condición de parada, sino que toma como argumento una lista y recorre todos los elementos de la misma. Para cada elemento se ejecutan las órdenes que se indican en el cuerpo del bucle.

La sintaxis del ciclo `for` en shell script es:

```
for variable in lista
do
    Cuerpo del bucle
done
```

Cada elemento de la lista que se pasa como argumento se coloca por turno en la variable que se pasa como primer argumento. Una vez cargada esa variable con ese valor, se ejecutan las órdenes que componen el cuerpo del bucle usando esa variable.

Por ejemplo, el siguiente fragmento de *shell script*:

```
frutas="manzana pera piña"
for f in ${frutas}
do
    echo "Me gustan las ${f}s"
done
```

produce el siguiente resultado:

```
Me gustan las manzanas
Me gustan las peras
Me gustan las piñas
```

El cuerpo del bucle `for` se ejecuta tantas veces como elementos tenga la lista que se pasa al bucle como argumento. Para cada elemento se ejecutan las instrucciones que componen el cuerpo del bucle.

Como puede verse es una forma muy sencilla de recorrer los elementos de una lista.

Nota

En *shell script*, existen otros dos bucles que sí comprueban una condición de parada, son los bucles **`while`** y **`until`**. Estos bucles se verán en sesiones posteriores.

Para recorrer los elementos de una lista lo mejor y más fácil es usar el bucle **`for`**.

Ejemplos del uso de bucles para recorrer listas.

Borrar todos los archivos del directorio `~/bin` que no sean ejecutables.

Imaginemos que en nuestro directorio de ejecutables `~/bin` se tienen otros archivos no ejecutables, como archivos con resultados de los scripts, o logs. Se quiere escribir un *shell script* que encuentre esos archivos y los borre.

```
#!/bin/bash
# Autor: Francisco de Asís Conde Rodríguez
# Descripción: Borra aquellos archivos del directorio
#              ~/bin que no sean ejecutables.
(1) archivos=`ls ~/bin`
(2) for a in ${archivos}
    do
(3)     if [ ! -d ~/bin/${a} ]
        then
            if [ ! -x ~/bin/${a} ]
            then
(4)             rm ~/bin/${a}
            fi
        fi
done
```

(1) Se ejecuta la orden `ls ~/bin` (que lista los archivos, directorios, etc. que hay en el directorio `bin` que se encuentra en el directorio base del usuario) y la lista que da como resultado se guarda en la variable archivos.

Para que el resultado de la orden `ls` se asigne correctamente, hay que usar la sustitución de órdenes. En este ejemplo se han usado los acentos `` `ls ~/bin` `` pero también se podría haber usado la sustitución `$( ls ~/bin )`

(2) En el bucle `for` se indica una variable donde se quiere ir almacenando cada elemento de la lista en cada paso, en este caso la variable `a`; y la lista que contiene los elementos que se quieren recorrer, en este caso `${archivos}`

También se podría haber escrito directamente la orden for así:

```
for a in `ls ~/bin`
```

sin usar una variable para almacenar el resultado de la orden.

(3) Los directorios también pueden tener permisos de ejecución, pero no nos interesa borrarlos, sólo los archivos. Por eso, se comprueba primero si no es un directorio y a

continuación, para los que son archivos, se mira si no tiene permisos de ejecución.

(4) Cada vez que en el cuerpo del bucle se hace referencia a la variable que se indica como primer argumento del ciclo `for`, (en este caso la variable `a`), se obtiene el valor del elemento de la lista que se esté procesando en un momento determinado.

En este ejemplo, en cada paso por el ciclo, en la variable `a` habrá alguno de los archivos, directorios, etc. que haya dentro del directorio `~/bin`, el que se esté recorriendo en ese momento.

Bucle `for` con un array

Se utiliza para recorrer un array completo. Puedes usar **`*`** o **`@`** para referirse a todos los elementos del array.

```
#!/bin/bash
# Definir un array
frutas=("manzana" "naranja" "plátano")

# Recorrer el array usando @
for fruta in "${frutas[@]}"; do
    echo "Fruta: $fruta"
done
```

Salida:

```
Fruta: manzana
Fruta: naranja
Fruta: plátano
```

for con una secuencia de números

Existen varias posibilidades para mostrar una secuencia de números o elementos.

Formato	Ejemplo

`for` VARIABLE **`in`** 1 2 3 4 5 .. N
`do`
  orden1
  orden2
  ordenM
`done`
	`#!/bin/bash`
	
	`# Ciclo for simple`
	`for i in 1 2 3 4 5`
	`do`
	  `echo "Bienvenido $i vez"`
	`done`

Otra forma es poner un rango de números desde un comienzo a un fin entre llaves. Este tipo de `for` no admiten variables directamente dentro de las llaves **`{}`**, ya que estas llaves se expanden antes de evaluar las variables.

```
#!/bin/bash
# Ciclo for con rango
# A partir de la versión 3.0 de Bash
for i in {1..5}
do
    echo "Bienvenido $i vez o veces"
done
```

Este formato permite generar secuencias numéricas con un inicio, un fin y un paso específico. Este tipo de `for` no admiten variables directamente dentro de las llaves **`{}`**, ya que estas llaves se expanden antes de evaluar las variables.

```
#!/bin/bash
# Bucle for con una secuencia numérica
for numero in {1..10..2}
do
    echo "Número: $numero"
done
```

Salida:

```
Número: 1
Número: 3
Número: 5
Número: 7
Número: 9
```

Nota: Este formato usa **`..`** para definir el inicio, fin y paso.

Bucle `for` para recorrer los ficheros

Formato	Ejemplo

for VARIABLE **in** file1 file2
do
  orden1
  orden2
  ordenN
done
	#!/bin/bash
	# Recorrer todos los archivos
	# en el directorio actual
	for archivo in *
	do
	  echo "Archivo: $archivo"
	done
	
	Salida:
	Archivo: documento.txt
	Archivo: imagen.jpg
	Archivo: script.sh

Nota: El uso de ***** enumera todos los elementos en el directorio actual.

Bucle `for` con un rango definido usando C-style (estilo C)

En Bash, se puede usar un estilo similar al de C, que permite inicialización (EXP1), condición (EXP2) y actualización (EXP3) en una sola línea. Este formato es común para bucles controlados por conteo.

Formato:

```
for (( EXP1; EXP2; EXP3 ))
do
    orden1
    orden2
    ordenN
done
```

Ejemplo 1:

```
#!/bin/bash
# Bucle for estilo C
for ((i = 0; i < 5; i++)); do
    echo "Iteración: $i"
done
```

Salida:

```
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
```

Ejemplo 2:

```
#!/bin/bash
# Ciclo for al estilo C
for (( c=1; c<=5; c++ ))
do
    echo "Welcome $c times"
done
```

Ejemplo 3:

```
#!/bin/bash
m=7
for (( n=1; n<=$m; n++ ))
do
```

```
    echo $n
done
```

Nota: Esta sintaxis es similar al `for` de otros lenguajes como C, C++, y Java.

Bucle `for` para recorrer la lista que genera la ejecución de una orden

Formato:

```
for VARIABLE in $(Orden-de-Linux-Aquí)
do
    orden1
    orden2
    ordenN
done
```

Ejemplo 1:

```
#!/bin/bash
echo "En un lugar de la mancha" > lista.txt
for i in $(cat lista.txt)
do
    echo "Palabra: $i"
done
```

Ejemplo 2:

```
#!/bin/bash
# Autor: Francisco de Asís Conde Rodríguez
# Descripción: Ejemplo de uso de arrays con
#              resultados de órdenes
resultado=( $(ls) )
for archivo in "${resultado[@]}"
do
    echo "$archivo"
done
```

Este script generará como salida una lista de archivos y directorios en el directorio actual, impresos uno por uno.

```
$ ./ejemlista_array.sh
bin
Descargas
Documentos
Escritorio
examples.desktop
Imágenes
```

```
Música
Plantillas
Público
Vídeos
```

Listas de parámetros.

Listas de parámetros pasados a un script.

Los parámetros especiales *****, que ya conocemos, y @ cuando se sustituyen, devuelven una lista que contiene todos los parámetros posicionales que se hayan pasado al script en la llamada.

La diferencia entre ambos es muy pequeña, y además su comportamiento cambia si se encierran entre comillas dobles o no, por ello lo mejor es ver cómo funcionan con un ejemplo:

```
#!/bin/bash
# Autor:       Francisco
# Descripción: Muestra las diferencias entre los
#              parámetros especiales * y @

echo "Dólar asterisco.........:"${*}
echo "Dólar asterisco comillas:${*}"
echo "Dólar arroba............:"${@}
echo "Dólar arroba comillas...:${@}"

echo "Bucle sobre dólar asterisco"
for i in ${*}
do
    echo "Parámetro: ${i}"
done

echo "Bucle sobre dólar asterisco comillas"
for i in "${*}"
do
    echo "Parámetro: ${i}"
done

echo "Bucle sobre dólar arroba"
for i in ${@}
do
    echo "Parámetro: ${i}"
done
```

```
echo "Bucle sobre dólar arroba comillas"
for i in "${@}"
do
    echo "Parámetro: ${i}"
done
```

Da como resultado:

```
$ difasteriscoat.sh 1 2 "3   4" 5

(1) Dólar asterisco: 1 2 3 4 5
(2) Dólar asterisco comillas:1 2 3   4 5
(1) Dólar arroba: 1 2 3 4 5
(2) Dólar arroba comillas: 1 2 3   4 5

    Bucle sobre dólar asterisco
    Parámetro: 1
    Parámetro: 2
(3) Parámetro: 3
    Parámetro: 4
    Parámetro: 5

    Bucle sobre dólar asterisco comillas
(4) Parámetro: 1 2 3 4 5

    Bucle sobre dólar arroba
    Parámetro: 1
    Parámetro: 2
(5) Parámetro: 3
    Parámetro: 4
    Parámetro: 5

    Bucle sobre dólar arroba comillas
    Parámetro: 1
    Parámetro: 2
(6) Parámetro: 3   4
    Parámetro: 5
```

(1) No preserva espacios.
(2) Preserva espacios entre comillas.
(3) Ignora las comillas y cada palabra en la llamada se trata individualmente.
(4) La lista completa se trata como una sola palabra.
(5) Funciona exactamente igual que `${*}` sin comillas.
(6) Respeta que “`3  4`” sea un único argumento.

Como podemos ver, tenemos tres formas posibles de obtener la lista de argumentos del script: `${*}` ó `${@}` (que se comportan igual), `"${*}"` y `"${@}"`.

1) `${*}` ó `${@}` : En las dos, cada palabra de la entrada se trata como un argumento independiente, sin importar si se encierran o no entre comillas al hacer la llamada al script.

2) `"${*}"` : Todas las palabras de la entrada se tratan como un único parámetro. Es muy útil cuando necesitamos que el usuario del script lo llame con un único parámetro formado por varias palabras para asegurarnos de que funciona correctamente aunque el usuario no las escriba entre comillas.

3) `"${@}"` : Las palabras de la entrada que vayan entrecomilladas se tratan como un único parámetro, las demás como palabras individuales.

Ejercicios.

1) Escribe un *shell script* que renombre todos los archivos ejecutables del directorio `~/bin`, de forma que le añada los caracteres `.sh`. Así si, por ejemplo, en `~/bin` existe un archivo ejecutable llamado `programa`, debería renombrarlo para que se llame `programa.sh`

2) Escribe un *shell script* que reciba como argumentos una lista de nombres de usuario, y que compruebe si existen, indicando por pantalla si el usuario existe o no.

3) Escribe un shell script que reciba como argumentos una lista de directorios accesibles desde el directorio de trabajo actual (es decir, que los nombres que se den como argumentos deben ser rutas válidas que permitan acceder a esos directorios desde el directorio de trabajo actual) y que liste el contenido de cada directorio.

4) Escribe un script que reciba como argumentos exactamente dos nombres de directorios accesibles desde el directorio de trabajo actual, y que copie los archivos del primer directorio en el segundo, siempre y cuando no existan ya en el segundo directorio, o bien, que el archivo que se copia sea más reciente que el que se sobreescribe.

5) Crea un script que reciba como argumentos la ruta a un directorio, una fecha y una extensión. El script deberá borrar todos los archivos de dicha extensión que existan en el directorio dado como parámetro y que sean anteriores a la fecha dada.
PISTA: La orden `ls --full-time` *`fichero`* devuelve toda la información de un fichero, incluyendo la fecha en formato año-mes-dia (`YYYY-MM-DD`).