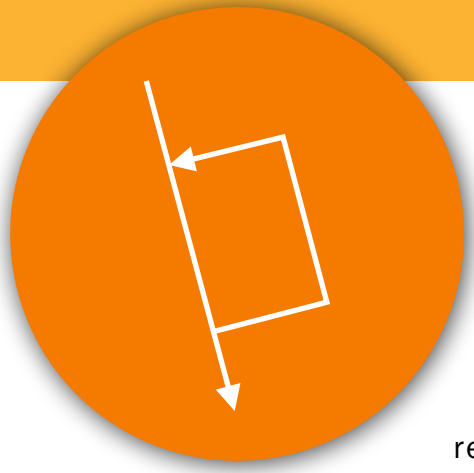
Programación en Shell
Listas y bucles sobre listas

Francisco de Asís Conde Rodríguez



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.

Listas

¿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
```

Ejemplos de uso de listas

Definir listas en shell script es muy sencillo. Este script:

```
#!/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:

```
fconde@fconde-VirtualBox:~$ ejemlista1
manzana piña pera
manzana piña pera papaya
```

Órdenes que devuelven listas

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.

Ejemplos de uso de órdenes que devuelven listas

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

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

Da como resultado:

```
fconde@fconde-VirtualBox:~$ ejemlista1
bin Descargas Documentos Escritorio examples.desktop
Imágenes Música Plantillas Público Vídeos
```

Bucles para recorrer listas

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`.

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.

archivos=`ls ~/bin`

for a in ${archivos}
do

    if [ ! -d ~/bin/${a} ]
    then

        if [ ! -x ~/bin/${a} ]
        then
            rm ~/bin/${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.

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:

```
fconde@fconde-VirtualBox:~$ difasteriscoat 1     2 "3    4" 5

Dólar asterisco:         1 2 3 4 5         No preserva espacios
Dólar asterisco comillas:1 2 3    4 5      Preserva espacios entre comillas
Dólar arroba:            1 2 3 4 5         No preserva espacios
Dólar arroba comillas:   1 2 3    4 5      Preserva espacios entre comillas

Bucle sobre dólar asterisco
Parámetro: 1
Parámetro: 2
Parámetro: 3     Ignora las comillas y cada palabra en la llamada se trata individualmente
Parámetro: 4
Parámetro: 5

Bucle sobre dólar asterisco comillas
Parámetro: 1 2 3    4 5    La lista completa se trata como una sola palabra
```

(Sigue en la página siguiente)

(Viene de la página anterior)

```
Bucle sobre dólar arroba
Parámetro: 1
Parámetro: 2
Parámetro: 3      Funciona exactamente igual que ${*} sin comillas
Parámetro: 4
Parámetro: 5

Bucle sobre dólar arroba comillas
Parámetro: 1
Parámetro: 2
Parámetro: 3    4       Respeta que "3    4" sea un único argumento
Parámetro: 5
```

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.