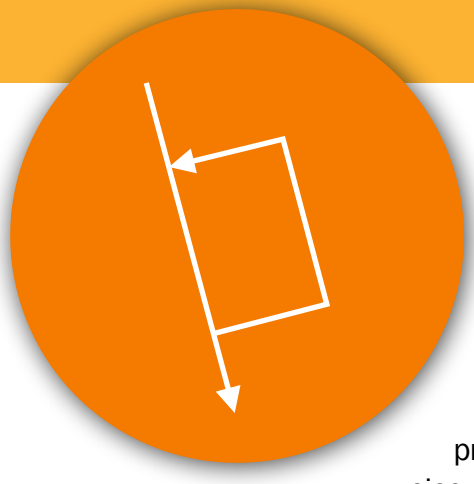
Programación en Shell Bucles

Francisco de Asís Conde Rodríguez



Además del bucle for visto en la sesión anterior de prácticas, existen otros bucles como while y until que permiten comprobar una condición de prueba y en función de ella continúan la ejecución del bucle o la detienen.

Para poder especificar condiciones de parada numéricas, es necesario realizar operaciones aritméticas. Por defecto, el intérprete interpreta todos los datos como cadenas de caracteres. Por eso es necesaria la sustitución aritmética que fuerza a intérprete a tratar los números como números y no como cadenas.

La indirección es un mecanismo muy potente del interprete de órdenes que permite acceder al contenido de una variable cuyo nombre está almacenado en otra variable.

Operaciones aritméticas en shell script

Como sabemos, en shell script, todos los valores de parámetros se interpretan como cadenas de caracteres. Así por ejemplo, la orden:

```
resultado=1+2
```

no almacena en la variable `resultado`, el valor 3 (suma de 1 mas 2), sino que almacena la cadena de caracteres `"1+2"`.

Si en un shell script hace falta realizar alguna operación aritmética, es necesario indicarlo expresamente al intérprete mediante la sustitución aritmética: `$(( expresión ))`

Por ejemplo:

```
resultado=$(( 1+2 ))
```

ahora sí almacena en la variable `resultado` el valor 3.

Como es de esperar, dentro de una expresión aritmética puede usarse sustitución de parámetros, pero no es necesario el signo $. Por ejemplo:

```
var1=2
resultado=$(( var1 * 3 ))
```

Almacena en `resultado` el valor 6.

Ejemplo de uso de operaciones aritméticas en un shell script. Escribe un shell script que elimine los archivos no ejecutables del directorio `~/bin` y al final escriba el número de archivos que ha borrado:

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Borra aquellos archivos del directorio
#              ~/bin que no sean ejecutables.

archivos=`ls ~/bin`
borrados=0
for a in ${archivos}
do
       if [ ! -d ~/bin/${a} ] && [ ! -x ~/bin/${a} ]
       then
              rm ~/bin/${a}
              borrados=$(( borrados+1 ))
       fi
done

echo "Se han borrado ${borrados} archivo(s)"
```

Operadores aritméticos

En shell script se pueden usar todos estos operadores aritméticos, todos ellos realizan operaciones enteras:

- `-` Complementario.
- `+` Suma.
- `-` Resta.
- `*` Multiplicación.
- `/` División entera. Por ejemplo: 1/3 es igual a 0.
- `%` Módulo o resto de la división.
- `**` Exponenciación. Por ejemplo: 2**4 es igual a 16.
- `&` AND a nivel de bits.
- `^` XOR a nivel de bits.
- `|` OR a nivel de bits.
- `<<` Desplazamiento de bits a izquierda. Equivale a multiplicar por dos. Por ejemplo 2<<2 es igual a 8.
- `>>` Desplazamiento de bits a derecha. Equivale a dividir por 2. Por ejemplo: 8 >> 1 es igual a 4.

Ejemplo: Dado un número de segundos, que imprima el número de días, horas, minutos y segundos equivalente.

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Cambia formato de tiempo.

seg_en_dia=86400
seg_en_hora=3600
min_en_hora=60
seg_en_minuto=60

dias=$(( ${1} / ${seg_en_dia} ))
seg_restantes=$(( ${1} % ${seg_en_dia} ))
horas=$(( ${seg_restantes} / ${seg_en_hora} ))
seg_restantes=$(( ${seg_restantes} % ${seg_en_hora} ))
minutos=$(( ${seg_restantes} / ${seg_en_minuto} ))
seg_restantes=$(( ${seg_restantes} % ${seg_en_minuto} ))

echo ${dias} ${horas} ${minutos} ${seg_restantes}
```

Bucle while

El ciclo `while` ejecuta las órdenes del cuerpo del bucle, mientras la condición de prueba sea verdadera. Su sintaxis es:

```
while condición_de_prueba
do
        cuerpo del bucle
done
```

Por ejemplo, un bucle `while` que escriba los diez primeros números naturales es:

```
i=1
while [ ${i} -le 10 ]
do
        echo "${i}"
        i=$(( ${i}+1 ))
done
```

Como en cualquier bucle, la última instrucción del cuerpo del bucle debe actualizar la condición de prueba ya que si no el bucle no termina nunca.

¿Cuando hay que usar bucles `while`?

Los bucles `while` se usan cuando no se dispone de una lista de elementos, pero sí de una condición que pueda ser comprobada.

Nota: Cuando se dispone de una lista de elementos que hay que recorrer entera, lo mejor es el bucle `for`.

Indirección

La indirección es una herramienta muy potente de la programación de shell scripts. Permite acceder al contenido de una variable cuyo nombre está almacenado en otra variable.

También se puede usar con parámetros posicionales.

Por ejemplo:

```
hola=3
nvar=hola
echo ${hola} ${nvar} ${!nvar}
```

Imprime por pantalla:

```
3 hola 3
```

La indirección `${!nvar}` indica al shell script que se quiere acceder al valor de la variable cuyo nombre se encuentra en la variable `nvar`. En este caso se quiere acceder al valor de la variable `hola` que vale `3`.

Nota: La sustitución `${${nvar}}` es errónea. En su lugar se usa la indirección.

Indirección y parámetros posicionales

Supongamos ahora que en el parámetro posicional 2 se ha pasado el argumento: `fconde`. El siguiente fragmento de código:

```
i=2
echo ${2} ${i} ${!i}
```

imprime por pantalla:

```
fconde 2 fconde
```

La indirección hace posible entre otras cosas que se pueda acceder al valor de un parámetro posicional dentro de un bucle que recorre un subconjunto de los mismos.

En la página siguiente hay un ejemplo de esto.

Ejemplos del uso de bucles para recorrer listas

Ejemplo: Escribe un shell script que reciba 2 argumentos como mínimo. El último de ellos debe ser un nombre de directorio y los primeros son nombres de archivo o directorio. El script debe copiar los archivos o directorios cuyos nombres se dan en los n-1 primeros argumentos, en el directorio que se pasa como último argumento (suponiendo que hay n argumentos).

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Recibe como mínimo dos argumentos. El
#              ultimo es un nombre de directorio. Copia
#              los primeros argumentos (archivos o
#              directorios) en el directorio dado.

if [ ${#} -lt 2 ]
then
        echo "Error. Se deben dar mínimo dos argumentos."
else
        if [ ! -d ${!#} ]
        then
                echo "Error. El último debe ser directorio."
        else
                i=1
                while [ ${i} -le $(( ${#}-1 )) ]
                do
                        if [ -e ${!i} ]
                        then
                                cp -r ${!i} ${!#}
                        fi

                        i=$(( ${i} + 1 ))
                done
        fi
fi
```

(1) Se comprueba si el nombre que se guarda en el último parámetro posicional corresponde a un directorio válido.

La indirección es necesaria ya que no se quere saber qué número tiene el último argumento sino su valor.

(2) Se recorren los n-1 primeros argumentos.

(3) Se comprueba si el i-ésimo argumento existe. De nuevo es necesaria la indirección para obtener el valor del parámetro posicional cuyo número está en la variable `i`.

(4) Se copia el elemento (archivo o directorio) en el directorio cuyo nombre está en el último argumento.

Como puede ser un directorio se usa `cp` con la opción `-r`

(5) Se incrementa la variable de control del ciclo, en este caso `i`, porque si no el bucle sería infinito ya que la condición de parada no se daría nunca.

El bucle until

Otro bucle que también puede utilizarse es el bucle `until`. Es parecido al bucle `while`. Se repite el cuerpo del bucle hasta que se cumpla una condición de prueba.

La sintaxis de bucle `until` es:

```
until condicion_de_prueba
do
        Cuerpo del bucle
done
```

Por ejemplo, un bucle `until` que escriba los diez primeros números naturales es:

```
i=1
until [ ${i} -gt 10 ]
do
        echo ${i}
        i=$(( ${i} + 1 ))
done
```

Bucles while para leer archivos

Uno de los usos más potentes de un bucle `while` es la lectura de un archivo línea a línea. Para ello se usa la sintaxis:

```
while read variable
do
        cuerpo del bucle

done < archivo
```

En este uso del bucle `while` hay tantos pasos por el bucle como líneas tenga el archivo. Cada una de las líneas se guarda en la variable que se indique y con ella se ejecutan las órdenes del cuerpo del bucle.

Por ejemplo, el siguiente fragmento de script:

```
while read linea
do
    echo ${linea}

done < "${1}"
```

Lee línea a línea el contenido del archivo que se pasa en el parámetro posicional `1`, y escribe cada línea en pantalla.

El filtro cut

`cut` es una orden que toma una línea de texto y la divide en palabras teniendo en cuenta el caracter separador que se indica como opción.

Por ejemplo:

```
echo "aa:bb:cc:dd" | cut -d: -f3
```

escribe por pantalla:

```
cc
```

La orden `cut` busca en la cadena `aa:bb:cc:dd` el carácter separador que se indica con la opción `-d` (delimiter) en este caso se usa `-d:` por lo tanto el carácter que separa las distintas palabras es `:`

La opción `-f` indica a la orden `cut`, cual o cuales de las palabras encontradas en la cadena nos interesa. En este caso se ha indicado la opción `-f3`, por tanto nos interesa la tercera palabra que en este ejemplo es: `cc`

Si se hubiese indicado la opción `-f1,3` se devolverían las palabras primera y tercera separadas por el carácter delimitador, `aa:cc`

Si se hubiera indicado la opción `-f2-4` se devolverían las palabras de la dos a la cuatro ambas incluidas, separadas por el caracter delimitador. En este caso `bb:cc:dd`

El filtro `cut` es muy útil a la hora de encontrar una palabra concreta dentro de una cadena en la que las palabras están separadas entre sí por un determinado carácter.

Ejemplo: Escribe un shell script que reciba como argumento un nombre de usuario y que compruebe si ese usuario existe en el sistema o no.

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Comprueba si existe o no un usuario
#              analizando el archivo /etc/passwd

if [ ${#} -ne 1 ]
then
        echo "Error. Hay que escribir un argumento."
else
(2)     while read linea
        do
(3)             usuario=`echo ${linea} | cut -d: -f1`

(4)             if [ ${1} = ${usuario} ]
                then
                        echo "El usuario ${1} existe"
                        exit
                fi
(1)     done < /etc/passwd
(5)     echo "El usuario ${1} no existe"
fi
```

(1) El archivo que se usa como entrada para el bucle `while` es `/etc/passwd` que tiene una línea para cada usuario del sistema con sus datos separados por `:`

(2) Las distintas líneas del archivo se leen en la variable `linea`

(3) Con el filtro `cut` se saca de cada línea la primera palabra, que en `/etc/passwd` corresponde con el nombre de usuario.

(4) Si el usuario que se pasa coincide con el que se comprueba, el usuario existe y se termina el script.

(5) Si se llega al final del bucle sin que ningún usuario del archivo `/etc/passwd` coincida con el que se pasa es porque ese usuario no existe.

Ejercicios

1) Escribe un shell script que reciba como argumento un número, el UID de un usuario y que comprueba si existe en el sistema un usuario con ese UID. Si existe debe comprobar además si ese usuario tiene el mismo UID y GID.

2) Escribe un shell script que saque una lista por pantalla de aquellos usuarios del sistema que están bloqueados. Recuerda que un usuario bloqueado tiene en el archivo `/etc/shadow` un signo `!` en lugar de una contraseña cifrada. Recuerda que para acceder a `/etc/shadow` hay que tener permisos de root.

3) Escribe un shell script que busque, en todos los directorios que cuelgan del directorio base del usuario que ejecuta el script, si hay algún archivo que tenga como nombre `core` y que borre todos los archivos `core` encontrados. Al final del script se debe imprimir el número de archivos core borrados. Recuerda que la orden `find` con la opción `-name nombrearchivo` permite buscar archivos en el disco. Para más información consulta la sintaxis de la orden `find`.

4) El archivo `/etc/group` tiene una línea por cada grupo que haya creado en el sistema. Investiga el formato de este archivo y escribe un script que escriba por pantalla cuales son los grupos que tienen usuarios asignados. Al final deberá escribir el número de grupos que tienen usuarios asignados.

5) Escribe un shell script que analice el archivo `/etc/group` y que diga si una serie de GIDs que se pasan como argumentos corresponden a grupos válidos del sistema. Al final debe escribir por pantalla cuántos de los GIDs pasados corresponden a grupos válidos y cuántos no.

6) El archivo `/proc/meminfo` contiene información relevante sobre el uso de la memoria del sistema en un instante determinado. Lista su contenido con `cat /proc/meminfo`, estúdialo y escribe un script que diga en un instante determinado, cuánta memoria en megabytes ocupan los buffers y las tablas de páginas.