Capítulo 6

Programación en el *shell* bash Parte III

6.17. Operaciones con cadenas

Los operadores de cadenas permiten la manipulación de los valores de las variables de una forma sencilla. Permiten hacer las siguientes operaciones:

- Comprobar que una variable está definida.
- Establecer valores por omisión.
- Mostrar mensajes de error si una variable no está establecida.
- Eliminar subcadenas que coincidan con patrones.

6.17.1. Concordancia con patrones

Los operadores empleados para eliminar subcadenas son:

- `${`*parámetro*`#`*patrón*`}`

  Esta construcción se expande al valor del *parámetro*, borrándole la porción más pequeña que concuerde con el *patrón* por la izquierda.

  Ejemplo:

```
$ X=datos.tar.gz
$ echo ${X#*.}
tar.gz
```

- `${`*parámetro*`##`*patrón*`}`

  Se expande al valor del *parámetro*, borrándole la porción más grande que concuerde con el *patrón* por la izquierda.

  Ejemplo:

  ```
  $ X=$(pwd)
  $ echo $X
  /home/usuario/pepe
  $ echo ${X##*/}
  pepe
  ```

  Compare este resultado con el obtenido al ejecutar la orden `basename`.

  Estos dos tipos de expansión por la izquierda son útiles para extraer los distintos directorios que componen el nombre de un fichero, para identificar sus permisos, etc.

- `${`*parámetro*`%`*patrón*`}`

  Se expande al valor del *parámetro*, borrándole la porción más pequeña que concuerde con el *patrón* por la derecha.

  Ejemplos:

  ```
  $ X=hola.c
  $ echo ${X%.*}
  hola
  ```

- `${`*parámetro*`%%`*patrón*`}`

  Esta forma es expandida al valor del *parámetro*, borrándole la porción más grande que concuerde con el *patrón* por la derecha.

  Ejemplo:

  ```
  $ X=usr/spool/mail
  $ echo ${X%%/*}
  usr
  ```

Estos dos tipos de expansiones por la derecha permite eliminar las diferentes extensiones del nombre de un fichero, etc.

6.17.2. Operadores de sustitución

Uso de valores por omisión

La construcción `${`*parámetro*`:-`*palabra*`}` se expande al valor del *parámetro* si éste está establecido y no es nulo; en caso contrario, se expande a *palabra*.

Ejemplo:

```
$ DIRECTORIO=~/temporal
$ echo ${DIRECTORIO:-/tmp}
/home/pepe/temporal
$ unset DIRECTORIO
$ echo ${DIRECTORIO:-/tmp}
/tmp
$ echo $DIRECTORIO

$
```

En este ejemplo, si la variable `DIRECTORIO` *no se establece se utilizará como directorio predeterminado el directorio* `/tmp`.

Este tipo de expansión se suele utilizar para utilizar una serie de valores predeterminados si alguna variable no se declara.

Asignación de valores por omisión

`${`*parámetro*`:=`*palabra*`}` se expande al valor del *parámetro* si éste está establecido y no es nulo; en caso contrario, se establece a *palabra* y se expande. Este tipo de expansión se emplea igual que el anterior, pero en este caso nos interesa que la variable disponga de tal valor.

Ejemplo:

```
$ unset A
$ echo ${A:=abc}
abc
$ echo $A
abc
```

Mostrar un mensaje de error

`${`*parámetro*`:?`*palabra*`}` es expandido al valor del *parámetro* si éste está establecido y no es nulo; en caso contrario, muestra *palabra* en la salida de errores estándar y hace que termine la ejecución del *script*.

Ejemplo:

```
$ unset A
$ echo ${A:?Variable no establecida}
/bin/bash: A: Variable no establecida
```

Se emplea para comprobar que las variables necesarias para el funcionamiento de nuestro *script* están definidas.

Uso de un valor alternativo

`${`*parámetro*`:+`*palabra*`}` se expande a *palabra* si el parámetro está establecido y no es nulo; en caso contrario, no se sustituye nada.

Ejemplo:

```
$ Y=abc
$ echo ${Y:+def}
def
$ unset Y
$ echo ${Y:+def}

$
```

Este tipo se expansión se suele utilizar cuando queremos mostrar mensajes de qué acciones va realizando el *script*.

6.18. La orden `[...]`

Nos permite evaluar expresiones condicionales con atributos de ficheros, cadenas, enteros, etc.

Formato:

[*expresión*]

donde *expresión* es la condición que se va a evaluar. Deben dejarse blancos justo después de [y antes de]. Los argumentos de la expresión deben separarse de los operadores también mediante blancos. Si la expresión que se evalúa es verdadera, se devuelve un status de salida 0, y si es falsa, distinto de 0.

6.18.1. Comprobación de atributos de ficheros

En la tabla 6.1 se dan algunos de los operadores que se pueden utilizar para comprobar atributos de ficheros.

`-r` *fichero*	*fichero* tiene permiso de lectura.
`-w` *fichero*	*fichero* tiene permiso de escritura.
`-x` *fichero*	*fichero* tiene permiso de ejecución.
`-f` *fichero*	*fichero* es un fichero regular.
`-d` *fichero*	*fichero* es un directorio.
`-b` *fichero*	*fichero* es un fichero especial de bloques.
`-c` *fichero*	*fichero* es un fichero especial de caracteres.
`-s` *fichero*	El *fichero* no está vacío.
`-O` *fichero*	El *fichero* existe y es propiedad del UID efectivo del proceso actual.
fichero1 `-nt` *fichero2*	*fichero1* es más nuevo (ha sido modificado más recientemente) que *fichero2*.
fichero1 `-ot` *fichero2*	*fichero1* es más viejo que *fichero2*.

Cuadro 6.1: Operadores de ficheros que se pueden usar con [...]

Ejemplo:

```
[ -f tmp ] && echo "tmp existe y es regular"
```

Si `tmp` *es un fichero regular imprime el mensaje, si no existe no imprime nada*

6.18.2. Comprobación de cadenas

Podemos usar la orden [...] para comprobar si una variable está establecida a un determinado valor y si su longitud es o no cero.

En la tabla 6.2 se muestran los operadores de cadena usados más frecuentemente con `[...]`.

`-n` *cadena*	Verdad si la longitud de la cadena no es 0.
`-z` *cadena*	Verdad si la longitud de *cadena* es 0.
cadena1 `=` *cadena2*	Verdad si *cadena1* es igual a *cadena2*.
cadena1 `!=` *cadena2*	Verdad si *cadena1* es distinta de *cadena2*.

Cuadro 6.2: Operadores de cadena de la orden `[...]`

Ejemplo:

```
[ $X = abcd ] && echo "X vale abcd"
```

Si la variable `X` *vale* `abcd` *se imprimirá el mensaje; si no es así, no se imprimirá nada*

6.18.3. Comparación de enteros

La orden `[...]` proporciona una serie de operadores que nos permiten comparar enteros.

La tabla 6.3 muestra los operadores de enteros más comúnmente usados de la orden `[...]`. *exp1* y *exp2* pueden ser enteros positivos o negativos.

exp1 `-eq` *exp2*	Verdadero si *exp1* es igual a *exp2*.
exp1 `-ne` *exp2*	Verdadero si *exp1* no es igual a *exp2*.
exp1 `-le` *exp2*	Verdadero si *exp1* es menor o igual que *exp2*.
exp1 `-lt` *exp2*	Verdadero si *exp1* es menor que *exp2*.
exp1 `-ge` *exp2*	Verdadero si *exp1* es mayor o igual que *exp2*.
exp1 `-gt` *exp2*	Verdadero si *exp1* es mayor que *exp2*.

Cuadro 6.3: Operadores de enteros para la orden `[...]`

Ejemplos:

1. `[ $# -gt 2 ] && echo "Demasiados argumentos"`
   Si el número de parámetros posicionales que se han pasado al llamar al script es superior a 2, nos da el mensaje: `Demasiados argumentos`.

2. `[ $(who | wc -l) -gt 10 ] &&`
   `echo "Hay más de 10 usuarios en el sistema"`
   Obtenemos el número de usuarios conectados al sistema y si éste es superior a 10 nos muestra un mensaje.

6.18.4. Otros operadores de la orden `[...]`

La orden `[...]` también admite el uso del operador `!` (negación), y permite combinar expresiones mediante los operadores `-a` (Y lógico) y `-o` (O lógico).

Ejemplo:

```
[ -r $FICH -o -w $FICH ]
```

Mediante esta expresión comprobamos si `$FICH` *tiene activado el permiso de lectura o el de escritura.*

6.19. Órdenes de control de flujo

Al igual que los lenguajes de programación de alto nivel, el *shell* bash proporciona varias órdenes de control de flujo.

6.19.1. La construcción `if`

El tipo más simple de construcción de control de flujo es la condicional. Se usa cuando queremos elegir hacer o no algo, o elegir entre un pequeño número de acciones, dependiendo de que se cumplan o no una serie de condiciones.

La construcción `if` tiene la siguiente sintaxis:

```
if orden1
then
    órdenes
[ elif orden2
then órdenes-1 ... ]
[ else
    órdenes-2 ]
fi
```

Podemos usar la construcción `if` de varias formas:

1. La forma más simple (sin las claúsulas `elif` ni `else`) ejecuta las *órdenes* indicadas si al ejecutar *orden1* obtiene un status de salida 0.

   Ejemplo:

   ```
   if [ ! -f $1 ]
   then
      echo "El fichero $1 no existe"
   fi
   ```

   Si el fichero que se ha pasado como primer parámetro posicional existe, no hace nada; si no existe, muestra un mensaje.

2. Otra forma de `if` se utiliza para ejecutar una serie de órdenes si el status de salida de la condición es 0, y otras órdenes si es distinto de 0.

   Ejemplo:

   ```
   if [ ! -f $1 ]
   then
       echo "El fichero $1 no existe"
   else
       echo "El fichero $1 existe"
   fi
   ```

   En este caso, tanto si el fichero existe como si no, muestra el mensaje correspondiente.

3. También puede comprobar con `if` si una serie de condiciones son verdaderas. Dependiendo de cual sea la verdadera, se ejecutará el conjunto de órdenes asociadas a la misma.

   Ejemplo:

   ```
   if [ ! -f $1 ]
   then
       echo "El fichero $1 no existe"
   elif [ ! -r $1 ]
   then
       echo "El fichero $1 no tiene permiso de lectura"
   elif [ ! -w $1 ]
   then
       echo "El fichero $1 no tiene permiso de escritura"
   elif [ ! -x $1 ]
   then
       echo "El fichero $1 no tiene permiso de ejecución"
   ```

```
else
    echo "El fichero $1 existe y tiene todos los permisos"
fi
```

Se comprueba si no existe el fichero y si no posee algún permiso, mostrando el mensaje correspondiente.

Hay que hacer notar que la percepción de verdad o falsedad es al revés que en el lenguaje de programación C.

6.19.2. La construcción `case`

Se utiliza para comparar un valor simple frente a un cierto número de otros valores. Las órdenes asociadas con ese valor se ejecutan cuando hay una concordancia. La sintaxis de `case` es:

```
case valor in
patrón1 )   orden
            ...
            orden;;
patrón2 )   orden
            ...
            orden;;
...
patrónN )   orden
            ...
            orden;;
esac
```

donde *valor* se compara con *patrón1*, *patrón2*, ... Cuando concuerda con uno de ellos se ejecutan las órdenes asociadas a ese patrón. La lista de órdenes asociada a cada patrón debe terminar con `;;`. Una vez que el *shell* ha encontrado una concordancia entre *valor* y uno de los patrones, ejecuta las órdenes asociadas y se salta el resto de la sentencia (al contrario del `switch` de C, que necesita la órden `break` para ir al final del bloque).

Ejemplo:

Si ejecutamos el script 6.1, dependiendo del valor introducido como primer parámetro posicional, se realizará la acción correspondiente. Si se introduce algo distinto de `-a`, `-b` *o* `-c`, *nos dirá que la opción introducida no es correcta.*

***Script* 6.1** Utilización de la orden `case`

```
#!/bin/bash

# Ejemplo: Utilización de la orden case

case $1 in
    -a) echo "Has introducido la opción -a";;
    -b) echo "Has introducido la opción -b";;
    -c) echo "Has introducido la opción -c";;
     *) echo "No has introducido una opción correcta";;
esac
```

6.20. Bucles

6.20.1. El bucle `for`

Se utiliza para ejecutar un conjunto de órdenes un número especificado de veces. Su sintaxis es:

```
for variable [in palabra1 palabra2 ...]
do
    órdenes
done
```

donde *variable* va tomando los valores *palabra1*, *palabra2*, etc., y para cada uno de estos valores se ejecutan las *órdenes*.

Ejemplo:

Al ejecutar el *script* 6.2 obtendremos:

```
Ciclo 1: X=rojo
Ciclo 2: X=amarillo
Ciclo 3: X=azul
```

***Script* 6.2** Utilización de la orden `for`

```
#!/bin/bash

# Ejemplo: Utilización de la orden for

declare -i NUMCICLO=1

for X in rojo amarillo azul
do
    echo "Ciclo $NUMCICLO: X=$X"
    let "NUMCICLO+=1"
done
```

También se pueden utilizar la sustitución de nombres de ficheros, la sustitución de órdenes y la sustitución de variables para generar una lista de palabras para la orden `for`.

Ejemplos:

1. `for FICHERO in ejer[1-3]`
2. `for FICHERO in $(ls ejer[1-3])`
3. `EJERCICIOS=$(ls ejer[1-3])`
   `for FICHERO in $EJERCICIOS`

Si existen los ficheros `ejer1`, `ejer2` *y* `ejer3`, *la variable* `FICHERO` *tomará en cada ciclo uno de los tres valores.*

También podemos utilizar la construcción `for` sin especificar una lista de palabras.

```
for variable
do
    órdenes
done
```

en cuyo caso, es equivalente a:

```
for variable in "$@"
do
    órdenes
done
```

6.20.2. Los bucles `while` y `until`

La sintaxis de estas órdenes es la siguiente:

```
while orden1
do
    órdenes
done
```

Si el status de salida obtenido después de ejecutar *orden1* es 0, se ejecutarán las órdenes que aparecen entre `do` y `done`. Esto se hará así hasta que el status de salida de *orden1* deje de ser 0.

```
until orden1
do
    órdenes
done
```

La orden `until` es como `while` excepto en que se ejecuta hasta que la condición sea verdadera.

Ejemplos:

1. ```
   while [ $# != 0 ]
   do
       echo $1
       shift
   done
   ```
   Si al ejecutar el script le pasamos varios parámetros posicionales, nos imprimirá los valores de todos ellos.
2. ```
   until [ $# == 0 ]
   do
        echo $1
        shift
   done
   ```
   Hace lo mismo que el ejemplo anterior. Observe que se ha cambiado la condición a evaluar.

6.21. Ejercicios

1. Escriba un *script* que reciba un fichero (puede ser con una ruta). Si el nombre de dicho fichero no comienza por punto cámbie su nombre para que así sea. Y si lo pueden leerlo, modificarlo o borrarlo intente cambiar los permisos para que no sea posible.

2. Escriba un *script* que se llame `quien`. Si se invoca sin parámetros funcionará como la orden `who`. Si recibe un parámetro mostrará las líneas del comando `who` correspondientes a él. Realice este ejercicio sin el operadores de comparación.

3. Escriba un *script* que permita que le pasen cuatro parámetros posicionales como máximo. Los parámetros que se le pasan deben ser nombres de ficheros y se debe comprobar si estos ficheros existen y si es así, si están vacíos o no. Como resultado debe dar la información correspondiente acerca de cada fichero.

4. Escriba un *script* que se llame `mata` y que mate un proceso por el nombre. Si no se introduce el nombre como argumento, debe pedirlo. Si existen más de un proceso con el mismo nombre debe matar a todos.

5. Escriba un *script* llamado `busca-multi` que admita uno o más parámetros. El script buscará, con `find`, los ficheros que tienen los nombres indicados a partir del directorio donde se invoque. En caso de que no se reciba ningún parámetro informará de su sintaxis y terminará.

6. Escriba un *script* de nombre `permisos` que reciba diez parámetros. Los primeros nueve serán podrán valer r, w, x o - y el décimo será un fichero. La orden podrá los permisos dados al fichero en concreto. Sólo se podrá usar la orden `chmod` una vez, en modo simbólico. En caso de que el número de parámetros no sea el adecuado informe de la sintaxis. Si algún permiso no es correcto o no se puede cambiar los permisos del fichero, informe del error y termine, pero evite que el sistema genere mensajes de error que lleguen al usuario.

7. Escriba un *script* llamado `permisos2` que funcione igual que el anterior pero use `chmod` con números octales.

8. Escriba un *script* que se llame `fecha` y que admita una fecha como parámetro con el formato `dd-mm-aa` y compruebe si es correcta. Si no lo es, debe dar los mensajes correspondientes. Si no se le introduce la fecha como parámetro debe pedirla.

9. Cree un fichero que tenga una columna de palabras. Una vez hecho, escriba un *script* con el siguiente formato:

`palabras -l | -w` *expresion_ regular*

- La opción `-l` hace que se genere un fichero que contiene la lista de palabras junto con el número de caracteres de cada una, separados por el carácter `-`.
- La opción `-w` hace que se genere un fichero con aquellas palabras que concuerden con la expresión regular dada.

10. Escriba un *script* que se llame `cuenta` que simule el comportamiento de la orden `wc`. Debe aceptar múltiples ficheros y si no se especifica ninguno debe leer de la entrada estándar.

11. Cree un *script* llamado `thumb` que cree imágenes a tamaño 100 por 100 puntos de todos los parámetros que reciba. En concreto, por cada fichero de nombre `foto.jpg` creará uno llamado `foto_thumb.jpg`. Para ello use el programa `convert` (incluido en el paquete `ImageMagick`) que tiene la siguiente sintaxis (para redimensionar a 100 por 100):

convert -resize 100x100 <origen> <destino>