Capítulo 6

Programación en el *shell* bash Parte IV

6.22. Órdenes de control de flujo 2

6.22.1. La orden `break`

Usted puede querer salir de un bucle antes de que se cumpla la condición; para esto, puede usar la orden `break`, que nos permite salir de un bucle sin abandonar el *script*. Una vez hemos salido del bucle, la ejecución continúa en la instrucción que sigue a éste.

Ejemplo:

Al ejecutar el script *6.1 se le dará un argumento, que será el directorio que se desea borrar. Antes de realizar la acción, pide confirmación.*

Si contestamos afirmativamente, se borrará el directorio y nos saldremos del bucle; si contestamos de forma negativa el directorio no se borrará y nos saldremos también; si damos una respuesta incorrecta nos dará un mensaje y nos volverá a preguntar.

La orden : *no hace nada, se utiliza para que el bucle se ejecute infinitas veces.*

***Script* 6.1** Salir de un bucle

```
#!/bin/bash

# Ejemplo de la orden break

dir=$1

while :
do
  echo "Quieres borrar el directorio $dir?"
  read respuesta
  case $respuesta in
  [sS]*)  rmdir $dir; break;;
  [nN]*)  echo "No borro $dir"; break;;
  *)      echo "Por favor conteste sí o no" ;;
  esac
done
```

La orden `break` también puede ser usada para salir de un bucle anidado usando el formato:

`break` n

donde n especifica que queremos salir del n-ésimo bucle.

Ejemplo:

Observe el código del *script* 6.2. ¿Qué salida se obtendría al ejecutarlo?

***Script* 6.2** Salir de un bucle

```
#!/bin/bash

# Ejemplo: Utilización de la orden break

for i in 1 2 3
do
  for j in 0 5
  do
    if [ $i -eq 2 -a $j -eq 0 ]
    then
      break 2
    else
      echo "$i$j"
    fi
  done
done
```

6.22.2. La orden `continue`

La orden `continue` hace que el *shell* abandone el resto del bucle actual y empiece inmediatamente la próxima iteración. Es parecida a la orden `break`, salvo que en vez de salir completamente del bucle, sólo se salta las órdenes restantes en el ciclo actual.

Formato:

`continue` [n]

Esta orden admite un argumento opcional n que, si se especifica, debe ser un número entero (o una expresión del *shell* que se evalúe a un entero). El valor de n indica qué bucle es el que debe reiniciarse cuando `continue` está inmerso en bucles anidados. Por omisión, se considera que n vale 1.

Cuando n es mayor que 1, se dan por terminados n-1 bucles como si se hubiera dado un `break` n-1. Sólo el bucle más externo designado por n se reiniciará en la próxima iteración.

Ejemplo:

El script 6.3 puede ser útil para un administrador de sistema, ya que le permite crear los directorios de casa de los usuarios listados

en `/etc/passwd` *que todavía no lo tengan, así como copiar los ficheros de arranque del shell.*

Si el directorio de casa del usuario ya existe, la orden `mkdir` *fallará y se ejecutará la orden* `continue` *haciendo que se ejecute una nueva iteración* `while`*.*

Si falla la orden `cp` *se reinicia un nuevo ciclo* `while` *ya que se ha especificado un valor para n igual a 2.*

***Script* 6.3** Uso de la orden continue

```
#!/bin/bash

# Ejemplo: Uso de la orden continue

IFS=:
while read login pass uid gid nombre home shell
do
  mkdir $home || continue
  for FICHERO in .profile .bash_logout .bashrc
  do
    if cp /usr/skel/$FICHERO $home
    then :
    else
      echo "No puedo crear $home/$FICHERO ($login)"
      continue 2
  done
done < /etc/passwd
```

6.22.3. La orden `exit`

El formato de esta orden es el siguiente:

Formato:

`exit` [n]

Hace que el *shell* o un *script* terminen con un status n. Si omitimos n el status de salida es el de la última orden ejecutada.

Ejemplo:

El *script* 6.4 posee un ejemplo de utilización del mismo.

***Script* 6.4** Uso de la orden `exit`

```
#!/bin/bash

# Ejemplo de la orden exit

case $# in
0)  echo "Uso: $0 fichero . . ."
    exit;;
*)  ;;
esac
```

6.22.4. La orden `select`

Se utiliza para mostrar un menú simple que contiene ítems numerados y un mensaje. Su sintaxis es la siguiente:

Formato:

```
select variable [in palabra1 ... palabraN]
do
  órdenes
done
```

donde *palabra1* hasta *palabraN* se muestran en la salida de errores estándar como elementos de un menú y seguidos de un indicador (por omisión `#?`). Si la respuesta está en el rango de 1 a N, entonces *variable* toma como valor la palabra correspondiente y se ejecutan las *órdenes*. La variable REPLY toma como valor la respuesta introducida. ¿Qué ocurre si nuestra respuesta no está dentro del rango válido? Se nos vuelve a mostrar el indicador hasta que introduzcamos una respuesta válida.

Si queremos utilizar un indicador diferente de `#?` tendremos que definir previamente la variable PS3. En el *script* 6.5 tenemos un ejemplo de utilización.

***Script* 6.5** Ejemplo de la orden `select`

```
#!/bin/bash

# Ejemplo de utilización de la orden select

PS3="Introduzca su selección: "

select ORDEN in "Nombre del directorio actual" \
"Listar ficheros" "Usuarios conectados" Salir
do
  case $ORDEN in
  Nombre*)  pwd;;
  Listar*)
          ( PS3="Directorio a listar: "
            select DIR in casa ejercicios actual salir
            do
              case $DIR in
              casa) ls $HOME;;
              ejercicios) ls $HOME/ejercicios;;
              actual) ls;;
              salir) break;;
              *) echo "No ha elegido una opción válida"
                 break;;
              esac
            done );;
  Usuarios*) who;;
  Salir) break;;
  *) echo "No ha elegido una opción válida";;
  esac
done
```

Un uso muy común de `select` es para que el usuario seleccione ficheros, como `select FOTO in *.jpg` o `SELECT FICH in $(find ...)`.

Si no especificamos la lista de *palabras*, los elementos del menú serán los parámetros posicionales.

6.23. La orden `eval`

Formato:

`eval` [*arg1 ... argN*]

El *shell* trabaja en tres pasos:

1. Expande los argumentos *arg1* hasta *argN*.
2. Concatena los resultados del paso 1 en una orden simple.
3. Ejecuta la orden formada en el paso 2.

Ejemplos:

1. Vamos a suponer que tenemos un *script* llamado `eval_test` al que se le pasan varios parámetros posicionales, y en un momento determinado necesitamos conocer el valor del último parámetro posicional. Un fragmento de dicho *script* podría ser el siguiente:

   ```
   echo "Número total de parámetros posicionales $#"
   echo "Valor del último parámetro posicional $$#"
   ```

   Cuando ejecutamos `eval_test` obtenemos:

   ```
   $ eval_test a b c
   Número total de parámetros posicionales 3
   Valor del último parámetro posicional 224#
   ```

   ¿Qué ha ocurrido? Al parecer no ha funcionado tal como queríamos. Lo que ha pasado es lo siguiente: el shell ha expandido `$$` *en primer lugar, y esto es el identificador del proceso actual, que ha resultado ser 224, y a esto le ha añadido el signo* `#`.

2. Para que hubiera funcionado tal como queríamos tendríamos que haber puesto:

   ```
   echo "Número total de parámetros posicionales $#"
   echo "Valor del último parámetro posicional \
   $(eval echo \$$#)"
   ```

   Si ejecutamos ahora `eval_test`:

   ```
   $ eval_test a b c
   Número total de parámetros posicionales 3
   Valor del último parámetro posicional c
   ```

Se necesita poner el carácter `\` *para que* `$` *sea ignorado en la primera expansión. Después de la primera expansión,* `$(eval echo \$$#)` *se convierte en* `$(echo $3)`*. Y esto a su vez es expandido a* `c`*.*

3. Supongamos que tenemos un fichero llamado `texto`.

```
$ cat texto
Éste es el contenido del fichero texto
```

Hacemos ahora lo siguiente:

```
$ X="<texto"
$ cat $X
<texto: No such file or directory
```

Sin embargo si usamos la orden `eval`:

```
$ eval cat $X
Éste es el contenido del fichero texto
```

6.24. La orden `getopts`

Una **opción** es un argumento de la línea de órdenes que empieza con `+` o `-` y es seguido por un carácter. La orden `getopts` se utiliza para analizar opciones. Normalmente `getopts` se utiliza como parte de un bucle `while`. El cuerpo del bucle `while` suele contener una sentencia `case`. Es realmente la combinación de las tres sentencias `getopts`, `while`, y `case` la que proporciona la forma de analizar opciones.

Vamos a ir viendo a través de ejemplos las posibilidades de esta orden.

***Script* 6.6** Tratamiento de opciones básicas

```
#!/bin/bash

# Ejemplo de script con dos opciones

while getopts xy argumentos
do
  case $argumentos in
  x)  echo "Has introducido la opción -x";;
  y)  echo "Has introducido la opción -y";;
  esac
done
```

Ejemplos:

1. Tenemos un fichero llamado `getopts1.bash`, cuyo contenido es el que aparece en el *script* 6.6. Vamos a ejecutarlo:

```
$ getopts1.bash -x
Has introducido la opción -x

$ getopts1.bash -y
Has introducido la opción -y

$ getopts1.bash -x -y
Has introducido la opción -x
Has introducido la opción -y
$ getopts1.bash -xy
Has introducido la opción -x
Has introducido la opción -y
```

   Si introducimos una opción errónea:

```
$ getopts1.bash -t -y
getopts1.bash: illegal option -- t
Has introducido la opción -y
```

2. El ejemplo previo trataba de una forma no demasiado elegante el caso de que el usuario introdujera una opción errónea. Vamos a modificar el *script* para que esto no ocurra. Las modificaciones se pueden ver en el *script* 6.7.

***Script* 6.7** Tratamiento de opciones incorrectas

```
#!/bin/bash

# Ejemplo de script con opciones y tratamiento de opciones
# incorrectas

while getopts :xy argumentos
do
  case $argumentos in
  x)  echo "Ha introducido la opción -x";;
  y)  echo "Ha introducido la opción -y";;
  \?)  echo "$OPTARG no es una opción válida";;
  esac
done
```

Hemos introducido el carácter : *al principio, lo que dice a*

`getopts` *que le dé a* `argumentos` *el valor* `?` *si el usuario especifica una opción distinta de* `x` *o* `y`.
Además establece el valor de la variable del shell OPTARG como el nombre de la opción no definida.

Si ejecutamos el nuevo *script*:

```
$ getopts2.bash -k -x
k no es una opción válida
Has introducido la opción -x

$ getopts2.bash -x -k
Has introducido la opción -x
k no es una opción válida
```

6.24.1. Opciones con argumentos

Un argumento de una opción es una palabra o número que sigue a la opción. Para decirle a `getopts` que una opción requiere un argumento, se coloca el carácter `:` después del nombre de ésta. Cuando se ejecuta el *script*, el *shell* asignará el argumento que sigue a la opción a la variable OPTARG.

***Script* 6.8** Tratamiento de opciones con argumentos

```
#!/bin/bash

# Ejemplo de script con opciones con argumentos

USO="uso: $0 [-x número] [-y número]"
while getopts :x:y: argumentos
do
  case $argumentos in
  x)  echo "Has introducido la opción x"
      arg_de_x=$OPTARG
      echo "argumento de x: $arg_de_x";;
  y)  echo "Has introducido -y como opción"
      arg_de_y=$OPTARG
      echo "argumento de y: $arg_de_y";;
  \?) echo "$OPTARG no es una opción válida"
      echo "$USO";;
  esac
done
```

Ejemplo:

Al ejecutar el *script* 6.8:

```
$ getopts4.bash -x 1024 -y 800
Has introducido -x como opción
Has introducido 1024 como argumento de x
Has introducido -y como opción
Has introducido 800 como argumento de y
```

¿Qué ocurre si se olvida dar un argumento a una opción? Ya hemos visto que el carácter `:` al principio de la lista de opciones ayuda a tratar con las opciones no válidas; además de esto, hace lo siguiente:

- Establece el valor de `argumentos` a `:` si el usuario olvida especificar el argumento.
- Establece el valor de OPTARG al nombre de la opción a la cual se ha olvidado dar su argumento.

***Script* 6.9** Tratamiento de errores con opciones

```
#!/bin/bash

# Ejemplo de tratamiento de opciones con argumentos
# y tratamiento de posibles errores

USO="uso: $0 [-y número]"

while getopts :y: argumentos
do
  case $argumentos in
  y)   altura=$OPTARG;;
  # Si el usuario olvida especificar el argumento de y,
  # el shell asigna : a argumentos
  :)   echo "No has dado un arg. para $OPTARG";;
  \?)  echo "$OPTARG no es una opción válida"
       echo "$USO";;
  esac
done
```

Ejemplo:

Al ejecutar el *script* 6.9 se obtiene:

```
$ getopts5.bash -y
No has dado un arg. para y
```

6.24.2. Análisis de líneas más complejas

La orden `getopts` también nos permite analizar líneas que contengan opciones, argumentos de opciones, y argumentos no asociados a opciones. Consideremos la siguiente línea de órdenes:

```
$ getopts6.bash -x 1024 -y 800 rojo verde azul
```

La línea de órdenes precedente contiene 2 opciones (`-x` y `-y`) y dos argumentos de opciones (`1024` y `800`). Además contiene tres valores no asociados con ninguna opción (`rojo`, `verde`, y `azul`). En una línea de órdenes mixta como ésta, el usuario debe especificar las opciones y los argumentos de éstas al principio de la línea; es decir, justo detrás del nombre del *script*.

La variable del *shell* bash OPTIND almacena el índice del argumento de la línea de órdenes que `getopts` va a evaluar actualmente. Es decir, cuando `getopts` está evaluando el primer argumento de la línea de órdenes (considerando que es una opción), el valor de OPTIND será 1.

***Script* 6.10** Líneas de órdenes complejas

```
#!/bin/bash

# Ejemplo de script con opciones con argumentos
# y argumentos simples

USO="uso: getopts6.bash [-x ancho] [-y alto] \
[color1 . . . colorN]"

while getopts :x:y: argumentos
do
  case $argumentos in
  x)  ancho=$OPTARG;;
  y)  alto=$OPTARG;;
  \?)  echo "$OPTARG no es una opción válida"
       echo "$USO";;
  esac
done
posiciones_ocupadas_por_opciones=$(( OPTIND - 1))

shift $posiciones_ocupadas_por_opciones

echo "Ancho: $ancho"
echo "Alto: $alto"
echo "Colores: $*"
```

Ejemplo:

Al ejecutar el *script* 6.10:

```
$ getopts6.bash -x 1024 -y 800 rojo verde
Ancho: 1024
Alto: 800
Colores: rojo verde
$ getopts6.bash -y 2048 amarillo azul plata
Ancho:
Alto: 2048
Colores: amarillo azul plata
```

6.25. Ejercicios

1. Escriba un *script* que muestre un menú con todos los directorios que hay por debajo de su directorio de entrada (incluidos los que se encuentran en subdirectorios). Al seleccionar uno de ellos, tiene que averiguar si el directorio seleccionado tiene activado el permiso de escritura para el grupo y para los otros. En cualquier caso debe dar el mensaje correspondiente.

2. Escribir un *script* que muestre un menú con los nombres de *login* de los usuarios conectados y al elegir uno se nos muestre el nombre completo de dicho usuario.

3. Escriba un *script* que trabaje sobre un fichero que contenga una relación de nombres con su correspondiente dirección internet, con el siguiente formato:

   ```
   nombre:DireccionInternet1
   ```

   El formato del *script* es el siguiente:

   `agenda-electronica` [`-c` | `-l` | `-a`] [*fichero*]

   `-c` Pide un nombre y muestra la dirección.

   `-l` Lista todo el fichero.

   `-a` Añade un nuevo registro.

   En caso de no darle ninguna opción, debe mostrar un menú con las operaciones anteriores. Si no se le proporciona un nombre de *fichero*, lo debe pedir al principio. El fichero debe quedar siempre ordenado después de cualquier operación. Se debe controlar e informar de todos los errores posibles.

4. Cree un *script* llamado `fotocopia`. Este *script* facilitará el manejo de fotos con el programa `convert`. Admite las siguientes opciones:

   -b hace una copia en blanco y negro de la foto (o fotos) que recibe como parámetro. Por cada foto de nombre `foto.jpg` crea una llamada `foto_BN.jpg`.

   -n <**num**> hace *num* copias de las fotos que reciba como parámetro. Por cada foto de nombre `foto.jpg` crea copias llamadas `foto_1.jpg`, `foto_2.jpg`, ..., `foto_num.jpg`.