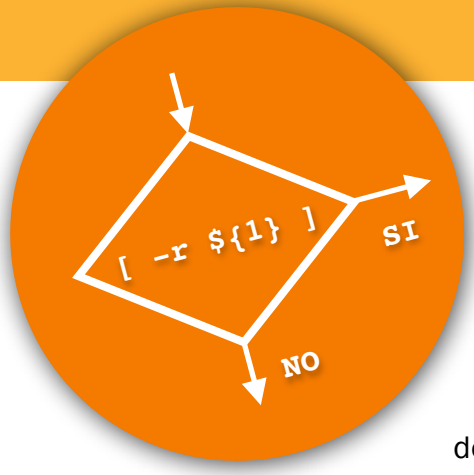
Programación en Shell
Ejecución condicional de órdenes

Francisco de Asís Conde Rodríguez



En ocasiones, un shell script deberá ejecutar un conjunto de órdenes si se da una condición y otro distinto si no se da. Es lo que se denomina ejecución condicional de órdenes.

Como en la mayoría de lenguajes de programación, en la programación de shell scripts existe la estructura `if-then-else`, que son palabras clave o reservadas del intérprete de órdenes, y que permiten programar la ejecución condicional de órdenes.

En esta sesión de prácticas se estudia esta estructura y cómo se evalúan las expresiones lógicas que nos permitirán construir programas más flexibles y útiles.

La estructura condicional `if - then - else`

Si se escribe en un terminal `type -a if`, el terminal responde:

```
fconde@fconde-VirtualBox:~$ type -a if
if es una palabra clave del shell
```

Es decir, `if` no es una orden, sino que es una palabra clave que indica al intérprete de órdenes que debe interpretar lo que sigue al `if` como un condicional. La sintaxis correcta para escribir un condicional en shell script es:

```
(1) → if [ condicion ]
      then
            sentencias si condición verdadera
      else
            sentencias si condicion falsa
(2) → fi
```

(1) Los corchetes `[ ]` y los espacios que hay entre ellos y la condición son **obligatorios.** Es un fallo muy común no poner espacio entre el primer corchete y el primer carácter de la condición. Por ejemplo, al ejecutar:

`if [-d ${HOME}/bin]` (El primer corchete `[` va pegado a `-d` sin espacio)

el intérprete de órdenes da el error:

`[-d: no se encontró la orden`

También es frecuente olvidar el espacio entre el final de la condición y el último corchete `]`. Por ejemplo al ejecutar:

`if [ -d ${HOME}/bin]` (El último corchete `]` va pegado sin espacio a `bin`)

el intérprete de órdenes da el error:

`bash: [: falta un `]'`

(2) Las estructuras condicionales en shell script comienzan con la palabra reservada `if` y terminan con la palabra reservada `fi`.

Condiciones y operadores

La condición dentro de una instrucción `if-then-else`, siempre se escribe entre corchetes y entre los corchetes y la condición debe existir un espacio, de lo contrario el intérprete de órdenes dará un error.

Para construir las condiciones, se usan valores, sustituciones de parámetros y operadores.

Por ejemplo:

```
[ -r ${1} ]
```

es una condición sintácticamente bien construída que comprueba si el valor del parámetro posicional 1, es el nombre de un archivo que existe y es legible (readable).

En la programación shell, existen muchos operadores que podemos utilizar para construir nuestras condiciones.

Operadores para aplicar a archivos:

Los ocho primeros son unarios, es decir sólo necesitan un operando que se escribe a continuación del operador:

```
[ operador archivo ]
```

Los dos últimos son binarios, es decir, requieren dos operandos que se escriben uno a cada lado del operador:

```
[ archivo1 operador archivo2 ]
```

- **-r** Comprueba si lo que va a continuación es el nombre de un archivo que existe y es legible.
- **-x** Comprueba si el archivo existe y es ejecutable.
- **-h** Comprueba si el archivo existe y es un enlace simbólico.
- **-d** Comprueba si el archivo existe y es un directorio.
- **-s** Comprueba si el archivo existe y no está vacío (su tamaño es mayor que cero).
- **-e** Comprueba si el archivo existe.
- **-O** Comprueba si el archivo es propiedad de quien ejecuta el shell script.
- **-G** Comprueba si el archivo pertenece al grupo de quien ejecuta el shell script.
- **-nt** Se coloca entre dos nombres de archivos y devuelve verdadero si el primero tiene fecha de modificación más reciente que el segundo.
- **-ot** Comprueba si el primer archivo tiene fecha de modificación más antigua que el segundo.

Operadores para aplicar a números:

Todos ellos son binarios. Es decir, se aplican a dos argumentos que se escriben uno a cada lado del operador.

- **-eq** Comprueba si los dos números que se pasan como argumentos son iguales.
- **-ne** Comprueba si los dos números son distintos.
- **-ge** Comproeba si el primer número es mayor o igual que el segundo.
- **-gt** Comprueba si el primer número es mayor estricto que el segundo.
- **-le** Comprueba si el primer número es menor o igual que el segundo.
- **-lt** Comprueba si el primer número es menor estricto que el segundo.

Operadores para aplicar a cadenas de caracteres:

Los dos primeros son unarios mientras que los dos últimos son binarios. Además, los dos últimos (< y >) requieren dobles corchetes para que se puedan interpretar bien, ya que también son los símbolos que usa el intérprete de órdenes para indicar la redirección.

- **-z** Indica si la longitud de la cadena que se pasa como argumento es cero (cadena vacía).
- **-n** Indica si la longitud de la cadena es mayor que cero.
- **=** Comprueba si las dos cadenas de caracteres son iguales.
- **!=** Comprueba si las dos cadenas de caracteres son distintas.
- **<** Indica si la primera cadena es menor que la segunda. Hace una comparación lexicográfica (es decir, comparando carácter a carácter). Por ejemplo, la cadena adiós es menor que la cadena hola.
- **>** Indica si la primera cadena es mayor que la segunda.

Condiciones compuestas:

Se pueden usar condiciones compuestas, por ejemplo comprobar a la vez si un archivo es legible y ejecutable. Para ello se usan los operadores lógicos `&&` (and) y `||` (or) para separar dos condiciones simples.

```
[ -r ${file} ] && [ -x ${file} ]
```

También se puede usar el operador unario `!` para negar el resultado de una condición.

Importante

Como en cualquier lenguaje de programación, si en un shell script no se escribe la sintaxis de las órdenes correctamente se producen errores.

Para aprender a programar shell scripts es necesario estudiar la sintaxis y practicarla con diversos ejemplos hasta que la aprendas.

Comprobar si se han pasado todos los argumentos

Uno de los usos de los condicionales en programas shell script es comprobar si se han pasado todos los argumentos al programa. En aquellos programas que requieran argumentos es muy importante que se compruebe que se han pasado los parámetros posicionales adecuados, ya que un parámetro que no existe siempre se sustituye por la cadena vacía y esto podría causar errores en el programa.

Ejemplo: Escribir un shell script que reciba dos argumentos y comprobar si se han pasado justamente dos argumentos.

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Comprueba si al script se le han pasado
#              justamente dos argumentos.

if [ ! ${#} -eq 2 ]
then
        echo "Error, se han pasado ${#} argumento(s)"
fi
```

(1) Como sabemos de la sesión anterior, el parámetro especial # contiene el número de parámetros posicionales que se han pasado al shell script cuando se ejecutó. Por tanto si comprobamos si es distinto de 2, habremos comprobado si a nuestro script se le han pasado, o no, justamente dos argumentos.

(2) La palabra reservada `then` tiene que ir escrita **en la línea siguiente** a la línea donde está la palabra reservada `if`. Si se escribe en la misma línea se produce un error al interpretar la línea.

(3) El sangrado es opcional, pero si se incluye, el código es mucho más legible.

Comprobar si existe o no un archivo

Como administrador del sistema operativo, en muchas ocasiones tendrás que hacer comprobaciones sobre archivos, qué permisos tienen o a quien pertenecen, y en función de esas comprobaciones realizar unas tareas de administración u otras.

Ejemplo: Escribir un shell script que reciba un argumento (el nombre de un archivo ejecutable) y compruebe si ese archivo está en el directorio bin del usuario que ejecuta el shell script y realmente es ejecutable. Si no está debe escribir por pantalla que no existe ese archivo, y si no es ejecutable debe escribir por pantalla que no es ejecutable.

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Comprueba si un nombre que se pasa como
#              argumento corresponde a un programa
#              ejecutable situado en el directorio ~/bin

if [ ${#} -ne 1 ]
then
        echo "Error. Se debe pasar un argumento"
        exit
fi

if [ ! -e ${HOME}/bin/${1} ]
then
        echo No existe el script: ${1}
else
        if [ ! -x ${HOME}/bin/${1} ]
        then
                echo ${1} no es un ejecutable
        fi
fi
```

(1) La variable `HOME` contiene la ruta completa del directorio base del usuario que ejecuta el script. Fíjate la importancia de siempre usar las llaves `{}` en la sustitución de parámetros. Si no se usara, la sustitición `${HOME}/bin/${1}` sería imposible.

Comparar cadenas de caracteres

En shell script, todos los valores se tratan como cadenas de caracteres a menos que se indique lo contrario, por tanto, es muy importante saber escribir condicionales en los que los argumentos de las condiciones sean cadenas.

Ejemplo: Escribir un shell script que reciba exactamente dos argumentos y que compruebe que son distintos. Si no se escriben dos argumentos, o si éstos son iguales debe escribir un error en la pantalla.

```
#!/bin/bash

# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Comprueba si los dos argumentos que se
#              dan son iguales.

if [ ${#} -ne 2 ]
then
       echo "Error. No se han dado dos argumentos"
else
       if [ ${1} = ${2} ]
       then
              echo "Error. Los argumentos son iguales"
       fi
fi
```

La comprobación de la igualdad de argumentos es útil por ejemplo a la hora de renombrar un archivo. En este caso, si el nombre del archivo que se renombra y el nuevo nombre que se quiere dar al archivo son iguales, no se debería ejecutar la orden de renombrar.

Ejercicios

1) Reescribe el shell script:

```
#!/bin/bash
# Autor:       Francisco de Asís Conde Rodríguez
# Descripción: Comprueba si al script se le han pasado
#              justamente dos argumentos.

if [ ! ${#} -eq 2 ]
then
          echo "Error, se han pasado ${#} argumento(s)"
fi
```

para que realice la misma comprobación, pero usando otros operadores.

2) Escribe un shell script que compruebe si se le han pasado entre dos y cuatro argumentos, en caso de que no se le hayan pasado ese número de argumentos que escriba un error, y en caso de que sí se le hayan pasado que los escriba por pantalla.

3) Escribe un shell script que reciba exactamente un argumento, un nombre de usuario, y compruebe si en el directorio base de ese usuario existe un archivo llamado `.profile`. Si no existe, que lo copie del directorio `/etc/skel` en el directorio del usuario y le dé permisos `-rw-r--r--`

4) Escribe un shell script que reciba exactamente un argumento que es el nombre de un directorio. A continuación compruebe si ya existe, y si no existe que lo cree. Por último, tanto si existía previamente como si no, que le dé permisos `drwx------`

5) Escribe un shell script que reciba exactamente dos argumentos que sean cadenas de caracteres y diga si la primera está ordenada alfabéticamente con respecto a la segunda o no.

6) La tarea de escribir un nuevo shell script requiere varios pasos: comprobar si el nombre del script ya existe, escribir el archivo, darle permisos de ejecución y copiarlo al directorio ~/bin. Escribe un shell script que reciba como argumento el nombre de un nuevo shell script que se quiere escribir y simplifique todos esos pasos.

   Nota: El parámetro especial `?` devuelve el resultado de la última orden (si se ejecutó con o sin errores).