PRÁCTICA 2

INTRODUCCIÓN A LA PROGRAMACIÓN DE SHELL-SCRIPTS

Objetivos

Programación de shell-scripts. Operadores de control de flujo del shell.

Shell-Script

Un shell-script es un fichero que contiene órdenes del shell. Asociado a la confección de shell-scripts se tiene un lenguaje de programación que veremos a continuación a través de varios ejemplos.

El flujo de trabajo para la creación y ejecución de un shell-script es el siguiente:

- Editar el shell-script mediante alguno de los editores instalados en el servidor linux: nano, mcedit, vi, joe, ...
- Dar permiso de ejecución al fichero que contiene el shell-script.
- Ejecutar el shell-script. Si nuestro shell-script se llama `prog` y admite por ejemplo 2 parámetros de llamada (`par1` y `par2`), la ejecución la haremos mediante `./prog par1 par2`

Construcción `if-then-else`

La construcción `if-then-else` del shell *bash* tiene la sintaxis siguiente:

```
if comando
   then comando
   [else comando]
fi
```

donde los saltos de línea pueden sustituirse por puntos y coma en caso de que queramos escribir la expresión en una sola línea.

Si el `comando` del `if` devuelve un *errorlevel* 0, entonces se ejecuta la parte `then`; en caso contrario se ejecuta la parte `else` (que puede no existir).

En este tipo de construcciones (y en las siguientes) pueden utilizarse los comandos `true` que devuelve 0 como *errorlevel* y `false` que devuelve un valor distinto de 0.

Ejemplo 1

```
#!/bin/bash
# Comentario: la línea anterior indica sobre qué shell se ejecuta el script
#
if  ls -l existe.txt >/dev/null
    # ls -l devuelve errorlevel=0 al listar un fichero que existe.
    # Por tanto entrara en la seccion del then
then echo "Esto funciona bien"
fi
```

La orden `test`

Su función es evaluar una expresión condicional. Si ésta es cierta devuelve un *errorlevel* 0 y si es falsa devuelve un *errorlevel* distinto de 0. La sintaxis es: `test expresión` o bien `[ expresión ]`

Ejemplo: `test -f fichero` o bien `[ -f fichero ]`

Algunas de las expresiones admitidas por esta orden son las siguientes:

-r fichero	Evalúa a cierto si el fichero existe y tiene permiso de lectura para el usuario que accede.
-w fichero	Evalúa a cierto si el fichero existe y tiene permiso de escritura para el usuario que accede.
-x fichero	Evalúa a cierto si el fichero existe y tiene permiso de ejecución para el usuario que accede.
-f fichero	Evalúa si el fichero existe y es un fichero regular.
-d fichero	Evalúa si es un directorio.
-c fichero	Evalúa si el fichero existe y es un fichero especial de caracteres .
-b fichero	Evalúa si el fichero existe y es un fichero especial de bloques.
-h fichero	Evalúa si el fichero existe y es una ligadura simbólica.
-s fichero	Evalúa si el fichero existe y tiene tamaño mayor que 0 .
-z $s1$	Evalúa a cierto si la longitud de la cadena $s1$ es 0.
-n $s1$	Evalúa a cierto si la longitud de la cadena $s1$ es distinta de 0.
$s1 = s2$	Evalúa a cierto si ambas cadenas son iguales.
$s1 != s2$	Evalúa a cierto si ambas cadenas no son iguales.
$s1$	Evalúa a cierto si $s1$ no es la cadena vacía.
$n1$ -eq $n2$	Evalúa a cierto si los dos enteros ($n1$,$n2$) son exactamente iguales.
$n1$ -lt $n2$	Evalúa a cierto si el entero $n1$ es estrictamente menor que $n2$.
$n1$ -gt $n2$	Evalúa a cierto si el entero $n1$ es estrictamente mayor que $n2$.
$n1$ -le $n2$	Evalúa a cierto si el entero $n1$ es menor o igual que $n2$.
$n1$ -ge $n2$	Evalúa a cierto si el entero $n1$ es mayor o igual que $n2$.
$n1$ -ne $n2$	Evalúa a cierto si el entero $n1$ es distinto de $n2$.

Si se utiliza `[ expresión ]` hay que tener cuidado de dejar un espacio en blanco entre los corchetes y la expresión. También hay que dejar espacios en blanco en las comparaciones de cadenas entre cada cadena y el símbolo de comparación (=, != , etc).

Además pueden emplearse los siguientes operadores para construir expresiones condicionales compuestas:

!	Operador “not”
-a	Operador “and”
-o	Operador “or”
()	Construcción de subexpresiones

Ejemplo: `test -f fichero -o -d fichero` o bien `[ -f fichero -o -d fichero ]`

Sentencia `exit`

La orden `exit` tiene la sintaxis:

```
exit [n]
```

Su efecto es abandonar el shell con un *errorlevel* igual a n. Si no se indica el argumento, el *errorlevel* devuelto es el de la última orden ejecutada.

Variables del shell

Hay algunas variables predefinidas en el shell que están muy relacionadas con la ejecución de shell scripts:

$#	Número de argumentos posicionales en un *shell-script*.
$@	Lista de todos los argumentos posicionales en un *shell-script*.
$0	Nombre del *shell-script* en ejecución.
$i ($i > 0$)	Argumento posicional i en un *shell-script*: $1, $2, ...
$?	Valor de *errorlevel* devuelto por el programa anterior (entre 0 y 255).
$$	Número de identificación del proceso (PID) del shell.
$!	Número de identificación del último proceso lanzado en *background*.

Ejemplo 2

```
#!/bin/bash
#
# Indica si un fichero pasado como parametro existe o no
#
if [ $# -ne 1 ]
then echo "Numero de parametros incorrecto"
     exit
fi

if ls -l $1 >/dev/null 2>/dev/null
then echo "Existe"
else echo "No existe"
fi
```

Ejemplo 3

```
#!/bin/bash
#
# Indica si un fichero pasado como parámetro existe o no.
# En caso de que exista se indica si el usuario tiene permiso de ejecucion
#
if [ $# -ne 1 ]
then echo "Numero de parametros incorrecto"
     exit
fi


if ls -l $1 >/dev/null 2>/dev/null
then if [ -x $1 ] # cuidado con los espacios
     then echo "Existe y tienes permiso de ejecucion"
     else echo "Existe pero no tienes permiso de ejecucion"
     fi
else echo "No existe"
fi
```

Ejemplo 4

```
#!/bin/bash
# Compara el tamaño de dos ficheros regulares y dice cual es mas grande
# Controla que se haya pasado dos ficheros como parametros y que existan
#
if [ $# -ne 2 ]
then echo "Numero de parametros incorrecto";   exit
fi

if [ ! -f $1 ] || [ ! -f $2 ]
then echo "Algun fichero no es regular"; exit
fi

t1=$(ls -l $1 | awk '{print $5}')  # la orden awk se vera en la practica siguiente
t2=`ls -l $2 | awk '{print $5}'` # la notación $(ls) y `ls` (operador grave) es equivalente

if [ $t1 -eq $t2 ]
then echo "Son iguales"
else if [ $t1 -gt $t2 ]
     then echo "$1 es mayor que $2"
```

```
     else echo "$2 es mayor que $1"
     fi
fi
```

Ejercicio 1 Escribir un script que admita un parámetro posicional y que realice lo siguiente:

1. Comprobar que se le ha pasado un argumento como parámetro posicional. Si no es así se mostrará el mensaje correspondiente.
2. El parámetro proporcionado debe ser el nombre de un fichero. Comprobar si existe o no en el directorio actual. En caso de no existir dar el mensaje correspondiente.
3. En el caso de que el parámetro proporcionado sea un fichero regular comprobar si está vacío o no y dar el mensaje correspondiente.
4. En el caso de que el parámetro proporcionado sea un directorio, listar los ficheros de dicho directorio.

Construcciones de control iterativas: Construcción `for`.

La sintaxis es:

```
for nombre [in lista_de_palabras]
do
   comandos
done
```

La lista de palabras puede construirse de muy diversas formas. Puede ser una lista escrita explícitamente, puede ser un patrón de nombre de fichero, o puede ser un comando del shell (usando acentos graves) que genera una lista (p. ej. `for i in` `` `ls` ``). Si no se indica ninguna lista se considera que está formada por los argumentos pasados al shell-script.

Ejemplo 5

```
#!/bin/bash
#
for  VAL in 1 2 3 4     # Para VAL en 1, 2, 3 y 4
     do
      echo $VAL     # Imprimir VAL: 1, 2, 3 y 4
     done
```

Ejemplo 6

```
#!/bin/bash
#
# Simula un ls con un for y numera cada fichero
#
num=0;
for i in *
do
    echo $num $i
    num=‘expr $num + 1‘
done
```

Ejemplo 7

```
#!/bin/bash
#
```

```
# Imprime los parametros de llamada
#
for i in $@ #(puede ser "for i")
do
    echo $i
done

# otra forma.
num=$#
for ((i=0; i<num; i++ ))
do
    echo $1
    shift
done
```

Construcciones de control iterativas: Construcción `while`.

La sintaxis es:

```
while comando
do
   comandos
done
```

En el comando que acompaña al `while` suele ser necesario utilizar la orden `test`. Mientras el `comando` devuelva 0 (`true`) se seguirá en el bucle.

Construcciones de control iterativas: Construcción `until`.

La sintaxis es:

```
until comando
do
   comandos
done
```

Es básicamente igual a `while` salvo que en este caso el bucle se ejecuta hasta que la condición es cierta.

Ejemplo 8

```
#!/bin/bash
# Solicita una cadena a introducir por el teclado y la imprime

while [ -z $cadena ]
do
    echo "Introduce una cadena"
    read cadena
done
echo $cadena
```

Sentencia `break`

La orden `break` tiene la sintaxis:

```
break [n]
```

Se emplea para abandonar un bucle (`for` o `while`) en tantos niveles de anidamiento como especifique el parámetro `n`. Por defecto `n=1`.

Ejemplo 9

```
#!/bin/bash
#
# Imprime "1 0" y "1 5". Despues abandona los dos bucles (break 2)
#
for i in 1 2 3
do for j in 0 5
    do if (test $i -eq 2 -a $j -eq 0)
       then break 2
       else echo "$i $j"
       fi
   done
done
```

Ejercicio 2 Escribir un script que tomando como valores los ficheros del directorio HOME, organice una estructura de directorios de forma que:

- Los ficheros con extensión *.c* se muevan a un directorio que se llamará *prog_c*
- Los ficheros con extensión *.txt* se muevan a un directorio que se llamará *textos*

Los directorios indicados sólo se crearán si existen archivos para ser movidos a ese directorio.

Sentencia `case`

Su sintaxis es la siguiente:

```
case palabra in
patrón [ | patrón ...]) comandos ;;
...
esac
```

Se busca el primer patrón en que encaje la palabra y se ejecutan los comandos que le siguen. La barra vertical es el operador lógico OR.

Ejemplo 10

```
#!/bin/bash
#
# Borra el directorio pasado como parametro  tras una confirmacion
# del usuario
#
while true
do
    echo "Quieres borrar el directorio $1"
    read respuesta
    case $respuesta in
             [sS]*) rmdir $1; break;;
             [nN]*) echo "No borro $1"; break;;
             *) echo "Por favor conteste si o no";;
    esac
done
```

Ejemplo 11

```
#!/bin/bash
# Menú de selección
```

```
# Pide opción hasta que se elija una de las válidas

echo "Selecciona una opción de entre las siguientes:"
echo "    Primera opción: 1 ó Uno"
echo "    Segunda opción: 2 ó Dos"
echo "    Tercera opción: 3 ó Tres"

while true
do
    echo -n "> "   # -n evita salto de línea
    read respuesta
    case $respuesta in
             1|Uno) echo "Primera opción"; break;;
             2|Dos) echo "Segunda opción"; break;;
             3|Tres) echo "Tercera opción"; break;;
             *) echo "Por favor introduce una opción correcta";;
    esac
done
```

Ejemplo 12

```
#!/bin/bash
# Menú de selección
# Ejecuta una de las opciones pásadas como parámetro
# Si no se pasa ningún parámetro se muestra el menú de opciones

if [ $# -ne 1 ]
then
echo "Introduce como parámetro una opción de entre las siguientes:"
echo "    Primera opción: 1 ó Uno"
echo "    Segunda opción: 2 ó Dos"
echo "    Tercera opción: 3 ó Tres"
exit
fi

case $1 in
        1|[Uu]no) echo "Primera opción";;
        2|[Dd]os) echo "Segunda opción";;
        3|[Tt]res) echo "Tercera opción";;
        *) echo "Opción incorrecta";;
esac
```

Funciones

Pueden utilizarse funciones para ejecutar varias veces un mismo conjunto de comandos. Las funciones admiten parámetros de llamada con la misma sintaxis que el propio script principal.

Ejemplo 13

```
#!/bin/bash
#
# Ejemplo de utilización de funciones dentro de un script
#

muestra()
{
```

```
    echo $1          # $1 hace referencia al parámetro
                     # pasado a la función
}

for i     # recorre todos los parámetros de la llamada
do
    muestra $i       # $i hace referencia a cada parámetro
                     # pasado al script
done
```

Ejercicio 3 Escribir un script *copia* que admita un nombre de fichero como parámetro de llamada y que realice una copia de ese fichero a otro cuyo nombre sea el mismo pero seguido de una extensión que indique la fecha en la que se hizo la copia.
Por ejemplo, si hoy es 15 de Mayo de 2019 y el fichero se llama *texto*, el nuevo fichero debe llamarse *texto.15052019*.
Nota: debe controlarse que el fichero pasado como parámetro existe y en caso de que no exista debe darse el mensaje adecuado.
Ayuda: consultar las opciones de la orden *date*.