MANUAL 1

SHELL SCRIPTS

INTRODUCCIÓN

1. ¿Qué es la Shell de Linux?

La shell es el término usado para referirse al intérprete de comandos, que consiste en la interfaz de usuario tradicional de los sistemas operativos basados en Unix y similares, como GNU/Linux.

Utiliza órdenes o comandos propios del sistema, así como instrucciones de un lenguaje de programación en modo intérprete. Mediante las instrucciones que aporta el intérprete, el usuario puede comunicarse con el kernel (núcleo) y por extensión, ejecutar dichas órdenes.

De este modo, la shell permite al usuario interactuar con el sistema a través de la interpretación de los comandos que el usuario ingresa en la línea de comandos (o también a través de los Shell scripts, archivos que ejecutan un conjunto de comandos).

Existen diferentes shells en Linux:

- Bourne Shell (sh)
- C Shell (csh)
- Korn Shell (ksh)
- Bourne Again Shell (bash): Es la shell por defecto en la mayoría de las distribuciones GNU/Linux y será la que utilicemos.

2. ¿Qué es un Shell script?

Se conoce como shell script cuando almacenamos una secuencia de comandos en un archivo de texto y le decimos a la shell que ejecute este archivo de texto.

De este modo, la shell interpretará línea a línea los comandos que encuentre en el archivo de texto ejecutable.

Al proceso de crear shell scripts se le denomina programación shell o en inglés, shell programming o shell scripting.

La programación shell es muy útil para resolver tareas repetitivas, típicas de los administradores de sistemas, como configuraciones específicas, tareas de automatización, etc. que desde un entorno gráfico se tardaría más tiempo en realizar.

3. Cómo escribir y ejecutar un script

Los pasos necesarios para escribir y ejecutar un shell script son:

(1) Usar cualquier editor como vi o **gedit** para escribir el script.

Sintaxis:

```
gedit nombre-script
```

(2) Después de escribir el script, hay que darle permisos de ejecución de la siguiente manera:

Sintaxis:

```
chmod permiso nombre-script
```

Ejemplos:

```
$ chmod +x nombre-script
$ chmod 755 nombre-script
```

Nota: Esto establecerá el permiso de lectura, escritura y ejecución (7) para el propietario, y para el grupo y el resto de usuarios solo permisos de lectura y ejecución (5).

(3) Ejecuta el script de la siguiente forma:

Sintaxis:

```
./nombre-script
```

Ejemplos:

```
$ ./miscript.sh
```

Consejo: Los archivos de shell script guárdalos con la extensión .sh, de manera que facilite la identificación de este tipo de archivos.

4. Mi primer script

Ya estamos listo para escribir nuestro primer shell script (`holamundo.sh`) que imprimirá en pantalla el mensaje “Hola mundo!”.

Ejecuta el siguiente comando para abrir el editor de texto:

```
$ gedit holamundo.sh
```

Escribe el siguiente shell script:

```
#!/bin/bash
# Mi primer shell script

clear
echo "Hola mundo!"
```

Explicación del código del script línea a línea:

Comando	Significado
`#!/bin/bash`	El sha-bang (#!) al comienzo de un script indica al sistema que el archivo va a ser interpretado por un determinado intérprete. En este caso por /bin/bash. Todos los scripts deben empezar por esta línea. Hay otros intérpretes como /bin/sh, /bin/ksh pero nosotros utilizaremos /bin/bash para nuestros scripts.
`# Mi primer shell script`	# seguido de cualquier texto se considera un comentario y no será interpretado. Los comentarios sirven para proporcionar más información sobre el script, explicar partes del código, etc.
`clear`	Limpia la pantalla
`echo "Hola mundo!"`	Imprime el mensaje “Hola mundo” en pantalla. Para imprimir un texto en pantalla se utiliza el comando echo.

Después guarda el script y dale permisos de ejecución

```
$ chmod 755 holamundo.sh
```

Finalmente ejecuta el script:

```
$ ./holamundo.sh
```

5. Variables

Hay tres tipos de variables:

- **Variables del sistema**: creadas y mantenidas por el propio Linux. Este tipo de variable se definen en mayúsculas.
- **Variables especiales**: hay un conjunto de variables que ya están configuradas y la mayoría de ellas no se le puede asignar un valor (solo lectura). Contienen información útil.
- **Variables definidas por el usuario**: creadas y mantenidas por el usuario.

5.1. Variables del sistema

Algunas de las variables del sistema más importantes son:

Variable	Significado
HOME	Su directorio principal. Ej: /home/juan
LOGNAME	Su nombre de inicio de sesión. Ej: juan
PATH	La ruta de búsqueda del shell. Los directorios se separan mediante dos puntos.
PWD	El directorio actual del shell
SHELL	El nombre de la shell que estamos utilizando
USER	Su nombre de inicio de sesión
UID	Identificador del usuario que ejecuta el script (0 para root)

Para imprimir el contenido de cualquiera de las variables anteriores hay que utilizar el comando `echo` y anteponer al nombre de la variable el símbolo del dólar (`$`):

Ejemplos:

```
$ echo $HOME     #Imprime el valor de la variable HOME
```

```
$ echo HOME      #Esto imprime literalmente HOME y no el
                 #contenido de la variable. Hay que usar el
                 #carácter $ seguido del nombre de la variable
                 #para imprimir el valor de la variable
```

5.2. Variables especiales

Variable	Significado
$0	el nombre del script.
$1…$9	Corresponde a los primeros 9 parámetros con los que se llamó el script.
$#	Número de parámetros con los que se ha invocado el script. Es muy útil para comprobar el número de argumentos pasados en la ejecución de un Shell script.
$*	Los parámetros pasados al script.
$@	Los parámetros pasados al script.
$$	El PID de la shell.
$?	Estado de salida del último comando o script ejecutado.

5.3. Variables definidas por el usuario

Para definir variables y asignar sus valores se utiliza la siguiente sintaxis:

```
nombre_variable=3
```

Para imprimir o acceder al valor de una variable, basta con colocar el símbolo del dólar ($) por delante del nombre de la variable

```
echo $nombre_variable
```

5.4. Reglas para nombrar variables y asignarles valor

A. El nombre de la variable debe comenzar con un carácter alfanumérico o un carácter de subrayado (_), seguido de uno o más caracteres alfanuméricos. Ejemplos: `nombre`, `miVar`, etc.

B. No poner espacios a ambos lados del signo igual (=) cuando asigne un valor a la variable. Las siguientes asignaciones son erróneas:

```
var =10
var= 10
var = 10
```

C. Las variables distinguen entre mayúsculas y minúsculas. Las siguientes variables son diferentes:

```
var=10
Var=11
vAr=2
```

6. Comillas

Hay tres tipos de comillas

Comillas	Nombre	Significado
"	Comillas dobles	Dentro de las comillas dobles todos los caracteres son interpretados literalmente. Excepto algunos de ellos que si son interpretados ($, \, `).
'	Comillas simples	Dentro de las comillas simples todos los caracteres son interpretados literalmente. Es decir, ninguno de los caracteres especiales conserva su significado dentro de las comillas simples.
`	Comilla invertida	Se utilizan para ejecutar comandos. Es decir, ejecuta el comando y sustituye la cadena por su salida. Ejemplo: `echo `date``

7. Comodines

Hay una serie de caracteres que se pueden utilizar como comodines y que pueden sustituir a otros caracteres o conjuntos de caracteres.

Comodín	Significado	Ejemplos	
*	Coincide con cualquier cadena o grupo de caracteres.	`ls *`	Muestra todos los archivos
		`ls a*`	Muestra todos los archivos cuya primera letra empiece por 'a'
		`ls *.c`	Muestra todos los archivos que tengan la extensión .c
		`ls ut*.c`	Muestra todos los archivos que tengan la extensión .c y cuyo nombre empiece por las letras 'ut'.
?	Coincide con cualquier carácter individual.	`ls ?`	Muestra todos los archivos cuyo nombre consta de un solo carácter.
		`ls fo?`	Muestra todos los archivos cuyo nombre tienen tres caracteres y comienzan por 'fo'
[...]	Coincide con cualquiera de los caracteres entre corchetes	`ls [abc]*`	Muestra todos los archivos cuyo nombre empiece por las letras a, b y c.

Nota:

[..-..] Un par de caracteres separados por el signo menos (-) indica un rango.

Ejemplo: Muestra todos los archivos cuyo nombre empieza por las letras a, b o c

```
$ ls /bin/[a-c]*
```

/bin/arch /bin/awk /bin/bsh /bin/chmod /bin/cp

/bin/ash /bin/basename /bin/cat /bin/chown /bin/cpio

/bin/ash.static /bin/bash /bin/chgrp /bin/consolechars /bin/csh

Ejemplo: Si el primer carácter después de [es una ! o un ^, entonces cualquier carácter que no coincida con el rango entre corchetes se mostrará.

```
$ ls /bin/[!a-o]
$ ls /bin/[^a-o]
```

/bin/ps /bin/rvi /bin/sleep /bin/touch /bin/view

/bin/pwd /bin/rview /bin/sort /bin/true /bin/wcomp

/bin/red /bin/sayHello /bin/stty /bin/umount /bin/xconf

/bin/remadmin /bin/sed /bin/su /bin/uname /bin/ypdomainname

/bin/rm /bin/setserial /bin/sync /bin/userconf /bin/zcat

/bin/rmdir /bin/sfxload /bin/tar /bin/usleep

/bin/rpm /bin/sh /bin/tcsh /bin/vi

8. Comandos echo y read

8.1. echo

El comando `echo` imprime en pantalla el texto que se le pasa.

Sintaxis:

```
echo [opciones] cadenas
```

Ejemplo:

```
$ echo "Hola ¿Qué tal?"
Hola ¿Qué tal?
```

Opciones

`-n`	No imprime carácter final de nueva línea
`-e`	Interpreta los siguientes caracteres de escape:

`\a`	Alerta (reproduce un pitido)
`\b`	Retroceso
`\c`	No imprime la nueva línea final (igual que -n)
`\n`	Salto de línea (nueva línea)
`\r`	Retorno de carro
`\t`	Tabulación horizontal
`\v`	Tabulación vertical
`\\`	Una barra invertida
`\'`	Comillas simples
`\"`	Comillas dobles

Ejemplo:

```
#Imprime de forma literal lo que hay entre las comillas
$ echo "\tHola"
\tHola
```

```
#Imprime Hola con una tabulación horizontal
$ echo -e "\tHola"
     Hola
```

8.2. read

Se usa para leer datos del usuario desde teclado y almacenarlos en variable.

Sintaxis:

```
read variable1
```

Ejemplo: El siguiente script primero pregunta al usuario el nombre y queda a la espera de que lo introduzca por teclado. Cuando el usuario introduce el nombre por teclado y pulsa la tecla intro, se almacena en la variable `varnombre`.

```
#Ejemploread.sh
#Script para leer tu nombre desde teclado
#
echo –n "Introduzca su nombre: "
read varnombre
echo "¡Hola $varnombre!"
```

Al ejecutarlo:

```
$ ./Ejemploread.sh
Introduzca su nombre: Javier
¡Hola Javier!
```

NOTA: Se puede utilizar el comando `read` con la opción `–p` para indicar en la misma línea el texto que se mostrará por pantalla.

Ejemplo:

```
read –p “Introduzca su nombre: ” varnombre

#Es equivalente a:
# echo –n "Introduzca su nombre: "
# read varnombre
```

9. Operaciones aritméticas

Se utiliza para realizar operaciones aritméticas

Sintaxis:

```
expr op1 operador op2
```

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto (módulo)

Ejemplos:

```
$ expr 1 + 3
$ expr 2 - 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 \* 3
$ echo `expr 6 + 3`
```

Nota:
La multiplicacion usa * y no *, para escapar el carácter.

Nota:

En el caso de utilizar `expr` como argumento dentro del comando echo hay que tener en cuenta que hay que usar las comillas ` para que `expr` sea evaluada y el comando `echo` imprima el resultado de la expresión.

Si usas las comillas dobles (") o las comillas simples (') no funcionará.

Ejemplo: (De este modo NO se realiza la operación)

```
$ echo "expr 5 - 2" # Imprimirá literalmente el texto expr 5 - 2
```

```
$ echo 'expr 5 - 2' # Imprimirá literalmente el texto expr 5 - 2
```

También podemos realizar operaciones aritméticas sin utilizar el comando `expr`. Para ello tenemos que definir variables numéricas utilizando el comando `let`.

Ejemplo:

```
#!/bin/bash
let A=100
let B=200
let C=$A+$B
echo "C= $C"
```

10. Redirección de Entrada/Salida

La mayoría de los comandos proporcionan la salida por pantalla o toman la entrada desde el teclado, pero es posible enviar la salida a un archivo o leer la entrada de un archivo.

Es decir, el shell puede redirigir la entrada estándar (stdin), la salida estándar (stdout) y el error estándar (stderr) desde y hacia archivos.

Símbolo redirección	Sintaxis	Significado
<	`comando < archivo`	`comando` obtiene la entrada de `archivo` y no de teclado
>	`comando > archivo`	Redirige la salida estándar a `archivo`. (Si archivo existe se sobrescribe y si no se crea)
>>	`comando >> archivo`	Redirige la salida estándar a `archivo`. (Si archivo existe se añaden los datos al final del archivo.)
2>	`comando 2> archivo`	Redirige el error estándar a `archivo`
&>	`comando &> archivo`	Redirige tanto la salida estándar como el error estándar a `archivo`

Ejemplos:

Redirecciona la salida del comando `ls` y la vuelca en `archivo.txt`

```
$ ls > archivo.txt
```

Sin embargo, cada vez que ejecutemos ese comando el contenido de `archivo.txt` será reemplazado por la salida del comando `ls`. Si queremos agregar la salida del comando al final del archivo, en lugar de reemplazarla, entonces ejecutamos:

```
$ date >> archivo.txt
```

11. Tuberías (pipes)

Una tubería (pipe) permite redirigir la salida estándar de un comando para que se convierta en la entrada estándar de otro comando por medio del operador `|`

Sintaxis:

```
comando1 | comando2
```

Ejemplos:

Comandos con tuberías	Significado de la tubería
`$ ls | more`	La salida del comando ls se envía como entrada al comando more, que la muestra de forma paginada.
`$ who | sort`	Envía la salida del comando who al comando sort e imprime una lista de usuarios conectados ordenada alfabéticamente.
`$ ls -l | wc -l`	La salida del comando ls se proporciona como entrada al comando wc para que imprima la cantidad de archivos en el directorio actual.

12. Estado de salida de un comando

Cuando se ejecuta un comando o un shell script, este devuelve un valor que se usa para determinar si se ha ejecutado correctamente o no.

1. Si el valor de retorno es cero (0), el comando es exitoso.
2. Si el valor de retorno es distinto de cero, el comando no ha tenido éxito o se ha producido algún tipo de error al ejecutar el comando / shell script.

Este valor se conoce como estado de salida.

Pero, ¿cómo averiguar el estado de salida de un comando o shell script? Para determinar este estado de salida se puede usar la variable especial `$?`

Ejemplo: (Este ejemplo asume que `miArchivo` no existe)

```
$ rm miArchivo
```

Esto mostrará el siguiente error: rm: cannot remove 'miArchivo': No such file or directory

Si justo después ejecutamos `echo $?` mostrará un valor distinto de cero, lo que indicará error.

```
$ echo $?
1
```

Ejemplo: Esto mostrará los archivos y directorios que hay en el directorio actual.

```
$ ls
```

Si justo después ejecutamos `echo $?` mostrará cero (0), lo que indicará que el comando se ha ejecutado satisfactoriamente.

```
$ echo $?
0
```

13. Comando test ([ ])

El comando test se usa para evaluar expresiones booleanas con números, cadenas y archivos, y establece su estado de salida en 0 (true) o 1 (false)

Expresiones de archivos

`-d nombre`	El nombre del archivo es un directorio
`-f nombre`	El nombre del archivo es un archivo estándar
`-r nombre`	El nombre del archivo existe y se puede leer
`-w nombre`	El nombre del archivo existe y se puede escribir
`-x nombre`	El nombre del archivo existe y se puede ejecutar
`f1 -nt f2`	El archivo f1 es más reciente que archivo f2
`f1 -ot f2`	El archivo f1 es más antiguo que archivo f2

Ejemplo:

```
$ test -d misFotos          #¿Es misFotos un directorio?
$ echo $?
0                           #Sí
```

Expresiones de cadenas

`s1 = s2`	La cadena s1 es igual a la cadena s2
`s1 == s2`	La cadena s1 es igual a la cadena s2
`s1 != s2`	La cadena s1 no es igual a la cadena s2
`-z s1`	La cadena s1 tiene longitud cero
`-n s1`	La cadena s1 NO tiene longitud cero

Ejemplo:

```
$ test -z "hola"            #¿La cadena "hola" tiene longitud cero?
$ echo $?
1                           #No
```

Expresiones numéricas

`a -eq b`	Los enteros a y b son iguales
`a -ne b`	Los enteros a y b son distintos
`a -gt b`	El entero a es mayor que el entero b
`a -ge b`	El entero a es mayor o igual que el entero b
`a -lt b`	El entero a es menor que el entero b
`a -le b`	El entero a es menor o igual que el entero b

Ejemplo:

```
$ test 10 -lt 5             #¿Es 10 menor que 5?
$ echo $?
1                           #No
```

Combinar y negar expresiones

`t1 -a t2`	Las expresiones t1 y t2 son true
`t1 -o t2`	O expresión t1 o expresión t2 son true
`!t1`	Cierto si la expresión t1 es false

El comando `test` tiene un alias, `[]` (corchetes) como abreviatura para usarlo con condicionales y bucles. Las siguientes dos expresiones son equivalentes:

```
$ test 10 -lt 5

$ [ 10 -lt 5 ]
```

14. Estructuras de control de flujo

14.1. if

La instrucción `if` elige entre alternativas. La forma más sencilla es la instrucción `if-then`:

```
if comando       #Si el estado de salida de comando es 0
then
      cuerpo
fi
```

Ejemplo:

```
#!/bin/bash
if [ `whoami` = root ]
then
      echo "Tú eres el superusuario"
fi
```

Si queremos distinguir entre dos alternativas, se utiliza la instrucción `if-then-else`:

```
if comando       #Si el estado de salida de comando es 0
then
      cuerpo1
else
      cuerpo2
fi
```

Ejemplo:

```
#!/bin/bash
if [ `whoami` = root ]
then
      echo "Tú eres el superusuario"
else
      echo "NO eres el superusuario"
fi
```

Por último, la contrucción `if-then-elif-else`, puede considerar todas las alternativas que se desee:

```
if comando1
then
      cuerpo1
elif comando2
then
      cuerpo2
elif ...
then
      ...
else
      cuerpoN
fi
```

14.2. while

El bucle `while` repite una serie de comandos mientras una condición sea cierta.

```
while comando    #Mientras el estado de salida de comando sea 0
do
      cuerpo
done
```

Ejemplo:

```
#!/bin/bash
i=0
while [ $i -lt 3 ]
do
      echo "$i"
      i=`expr $i +1`
done
```

Al ejecutarlo obtenemos:

```
$ ./while1.sh
0
1
2
```

14.3. until

El bucle `until` se repite hasta que la condición sea cierta:

```
until comando   #Hasta que el estado de salida de comando sea 0
do
      cuerpo
done
```

Ejemplo:

```
#!/bin/bash
i=0
until [ $i –ge 3 ]
do
      echo “$i”
      i=`expr $i +1`
done
```

Al ejecutarlo obtenemos:

```
$ ./until1.sh
0
1
2
```

14.4. for

El bucle `for` itera por los valores de una lista:

```
for variable in lista
do
      cuerpo
done
```

Ejemplo:

```
#!/bin/bash
for nombre in Alberto Carlos Juan
do
      echo “Se llama $nombre”
done
```

Al ejecutarlo obtenemos:

```
$ ./for1.sh
Se llama Alberto
Se llama Carlos
Se llama Juan
```

El bucle `for` tiene también otra sintaxis posible mucho más parecida a la de los lenguajes de programación convencionales (Java,C,C++,etc.)

```
for (( inicialización; condición; incremento ))
do
     cuerpo
done
```

Ejemplo: Imprime los números del 1 al 10

```
#!/bin/bash
for (( i=1; i<=10; i++ ))
do
     echo "$i"
done
```

En la mayoría de los casos utilizaremos la primera sintaxis (la de iterar por los valores de una lista), aunque en ciertas ocasiones puede ser útil utilizar esta segunda sintaxis del bucle `for`.

14.5. case

La instrucción case es una buena alternativa a la instrucción if-else multinivel. Permite evaluar los diferentes valores que puede tomar una cadena y realizar diferentes acciones en cada caso.

```
case cadena in
    expr1)
      cuerpo1 ;;
    expr2)
      cuerpo2 ;;
    .....
    exprN)
      cuerpoN ;;
    *)
      cuerpoPorDefecto ;;
esac
```

donde `cadena` se compara con los diferentes patrones (`expr1, expr2, …`) hasta que se encuentra una coincidencia. El shell luego ejecuta todas las sentencias hasta los dos puntos y coma. El valor predeterminado es *) y se ejecuta si no se encuentra ninguna coincidencia.

Ejemplo:

```
#!/bin/bash
read –p "Introduzca un carácter alfanumérico: " caracter

case $caracter in
     [A-Z])
          echo "$caracter es una letra mayúscula"
          ;;
     [a-z])
          echo "$caracter es una letra mayúscula"
          ;;
     [0-9])
          echo "$caracter es un dígito"
          ;;
     *)
          echo "carácter no identificado"
          ;;
esac
```

15. Funciones

Se pueden utilizar funciones para agrupar varios comandos o sentencias. Para definir una función dentro de un script existen dos formas:

Forma 1:

```
function nombrefn
{
      ...
      comandos bash
      ...
}
```

Forma 2:

```
nombrefn()
{
      ...
      comandos bash
      ...
}
```

Para llamar a una función únicamente es necesario indicar el nombre de la función.

Ejemplo:

```
#!/bin/bash

fecha()
{
      echo “La fecha es: `date`”
}

misDatos()
{
      echo "Nombre de usuario: $USER"
      echo "UID (Identificador de usuario): $UID"
      echo "Directorio home: $HOME"
}

misDatos

fecha
```