SCRIPTING EN LINUX

¿Qué es un shell?

El **shell** gestiona la interacción entre el usuario y el sistema operativo solicitándole la entrada, interpretando dicha entrada **para** el sistema operativo y gestionando cualquier resultado **de** salida procedente del sistema operativo. Los **shells** ofrecen un método **para** comunicarse con el sistema operativo.

¿Que son los scripts?

En Linux los procedimientos de comandos se conocen como scripts. Linux dispone de varios Shell diferentes csh, bash, sh, ksh, etc. Cualquiera de ellos es mucho más potente que cmd.

¿Cómo crear un script en Linux?

Para crear un script utilizaremos cualquiera de los editores de texto plano como vi, vim, nano, joe, etc.

Una vez escrito el script, para ejecutarlo hay que añadirle el permiso de ejecución

chmod u+x script.sh

y en ese momento ya podremos ejecutarlo. Para esto habrá que poner el nombre del script precedido del directorio en el que se encuentra, incluso si es el directorio en el que estamos. En este caso bastará con poner ./ delante del nombre del script.

./script.sh

Si se quisiese ver cómo se va ejecutando los comandos y que valor va tomando las variables durante la ejecución a efectos de depuración del programa, se puede ejecutar el script con la sentencia bash -x delante del nombre del script. **bash -x ./script.sh**

Estructura del lenguaje. (bash)

Es recomendable poner en la primera línea, a partir del primer carácter la sentencia:

#!/bin/bash

Esto hace que el sistema operativo sepa con qué intérprete de comandos va a ejecutar las instrucciones siguientes. Aquí se puede añadir -x o -v para que se vean las líneas de comandos en tiempo de ejecución. Con -x sustituye las variables por su valor y con -v no.

Por supuesto, se pueden utilizar todos los comandos del Shell, y en la siguiente tabla aparecen algunos comandos y opciones para scripts.

#	Comentario. No interpreta lo que hay a partir de este símbolo
read	Para que el usuario introduzca un dato
echo	Muestra el mensaje que viene a continuación por pantalla
&	Puesto después del comando, ejecuta este en segundo plano
orden1 && orden2	Ejecuta la orden 1 y si va bien, ejecuta la orden 2
orden1 || orden2	Ejecuta la orden 1 y si va mal, ejecuta la orden2
;	Ejecuta todas las órdenes una tras otra ls -l; catprac.c; date

Para mostrar por pantalla con **echo**, podemos utilizar las siguientes secuencias de escape.

\n	Salto de línea
\r	Return
\t	Tabulador
\v	Tabulador vertical
\b	Retroceso
\a	Pitido
\"	Comillas dobles (dentro de otras comillas dobles)
\$	El carácter $
\\	La barra \

Además, hay que saber cómo funcionan las comillas en bash. Las comillas dobles permiten que se evalúen las variables que contienen, mientras que las comillas simples ponen todo literalmente.

Crear un script, método paso por paso.

Comenzaremos creando un Simple y clásico "Hola Mundo", escribiendo en tu plantilla el siguiente código:

```
#!/bin/bash
# declaramos la variable cadena
cadena="Hola Mundo"
# mostramos la variable por pantalla
echo $cadena
```

Variable=	Para definir una variable
$variable	Para coger el valor de la variables

Para ejecutar recuerde:

chmod u +x hola.sh

./hola.sh

Script que lee un valor del usuario y lo muestra por pantalla

```
#!/bin/bash
# leemos un valor del usuario
echo "Introduce un texto: "
read texto
echo "El texto introducido fue:  $texto"
```

Script que se le pasa diferentes parámetros.

```
#!/bin/bash
# paso de parámetros
echo "Este script muestra los parámetros que has pasado"
echo "El primer parámetro es $1"
echo "El segundo parámetro es $2"
echo "El tercero parámetro es $3"
echo "En total has pasado $#"parámetros"
echo "Y la lista de parámetros total es $* "
```

	Nombre del script
$1	1 º parámetro
$2	2 º parámetro
$3	3 º parámetro
$#	total parámetros
$*	Número de parámetros

Definición de operadores

Símbolo	Operación
+	Suma
-	Resta
*****	Multiplicación
/	División
%	Resto de la división entera
()	Sirven para agrupar operaciones
+=	Acumulación de suma
-=	Acumulación de resta
***=**	Acumulación de producto
/=	Acumulación de división
%=	Resto de la división
++	Incremento de 1
--	Decremento de 1

Definición de las estructuras de control

Sentencia If

if ["codicion"]; **then**

"comandos"

[elif "condicion" **then**

"comandos"**]**

[else "comandos"**]**

fi

Operadores de comparación para números:

Operador	Verdad (TRUE) si:
x -lt y	x menor que y
x -le y	x menor o igual que y
x -eq y	x igual que y
x -ge y	x mayor o igual que y
x -gt y	x mayor que y
x -ne y	x no igual que y

Operadores de comparación para cadenas:

Operador	Verdad (TRUE) si:
cadena	Cadena contiene algo
-n string	La longitud de la cadena no es 0
-z cadena	La longitud de la cadena es 0
cadena1 = cadena2	Las dos cadena son iguales
cadena1 !=cadena2	Las dos cadenas son distintas
cadena1 < cadena2	Cadena1 es menor que cadena2
cadena1 > cadena2	Cadena1 es mayor que cadena 2

Operadores de comparación para ficheros:

Operador	Verdad (TRUE) si:
-d fichero	Fichero existe y es un directorio
-e fichero	Fichero existe
-f fichero	Fichero existe y es un fichero regular (ni directorio ni fichero especial)
-b fichero	Es un dispositivo de bloque (disco duro, etc.)
-c fichero	Es un dispositivo de carácter (ratón, teclado, etc, etc.)
-p fichero	Es una tubería
-s fichero	Fichero existe y no está vacío
-w fichero	Tienes permiso de escritura en fichero
-x fichero	Tienes permiso x en fichero (o de búsqueda si es un directorio)
-O fichero	Eres el dueño del fichero
-G fichero	El grupo del fichero es igual al tuyo
fichero1 -nt fichero2	Fichero1 es más reciente que fichero2
fichero1 -ot fichero2	Fichero1 es más antiguo que fichero2
!	Niega la condición

Ejemplos:

a) Script que compara un número con 1000 y nos dice si es mayor o no.

```
#!/bin/bash
#Compara valor numérico echo "Introduce un número: "
read numero
if [ $numero -gt 1000 ]; then
      echo "El número es mayor de 1000"
else fi
echo " El número es menor que 1000 "
```

b) Script que pide un nombre de fichero y nos dice si existe o no.

```
#!/bin/bash
#Fichero existe o no
echo "Introduce un nombre del fichero: "
read fichero
if [ -e  $fichero ] ; then
    echo "El fichero existe"
else fi
echo " El fichero no existe "
```

Sentencias repetitivas

Sentencia repetitiva FOR, tenemos dos opciones:

) For realizará las acciones una vez por cada elemento de la lista que se le pasa. Las variables del for irá recibiendo un elemento de esto en cada vuelta. La sintaxis es:

for variable [[**in** lista]] **do**

instrucciones **done**

a) En esta otra acepción de for, se ejecuta el bucle, un número fijo de veces y hay una ariable que va cambiando en cada vuelta desde un valor inicial hasta que llega o sobrepasa un valor final y se dice también el valor del incremento de la variable en cada vuelta. La sintaxis es:

for ((expr1;expr2;expr3)) **do**

instrucciones

done

Sentencia repetitiva While y Until.

a) While ejecuta una serie de instrucciones mientras se cumpla la condición que se dice. Se ejecutan hasta que la condición da un valor de falso, o distinto de 0. La sintaxis es:

```
while  codicion
do
    comandos
done
```

b) Until hace exactamente lo mismo, pero con la condición contraria, es decir, hace el bucle hasta que se cumple la condición que se indica. La sintaxis es:

```
until codicion
do
    comandos
done
```

Sentencia Select

Sentencia SELECT. Permite seleccionar al usuario una opción de una lista de opciones en un menú. La sintaxis de esta instrucción es la siguiente:

```
select nombre [in lista]
do
    comandos que pueden utilizar $nombre
done
```

Ejercicios

Realizar los siguientes scripts:

a) Pedir un número y comprobar si es mayor o menor que 500.
b) Pedir un número y realizar la suma de los 10 números posteriores a él.
c) Mostrar los números del 0 al 100.
d) Mostrar los números del 0 al 100 de 5 en 5.
e) Mostrar los números del 0 al 100 de 7 en 7 con while.
f) Pedir una palabra y mostrarla, hasta que escribamos “quit”.
g) Dado el nombre de un archivo por parámetro, comprobar que existe el mismo en el directorio.