SCRIPTS IN THE SHELL

¿Qué es un script de consola de bash?
Es un fichero de comandos que la consola interpreta y ejecuta (análogo a un .bat de MSDOS)

¿Para qué sirve?
Aumenta la comodidad en la ejecución de procesos.
Automatiza tareas complejas.
Se utilizan durante el proceso de arranque/parada del sistema.
Los utilizan muchas otras herramientas (Pe.: cron, programas de instalación).

Creación:
Abrimos con un editor un fichero de texto y escribimos los comandos en su interior.
vi hola
echo Hola mundo
Guardamos el fichero.

Ejecución:
Existen dos métodos
1. bash <fichero script>
Este método funciona siempre
2. /ruta/<fichero script>
Esto solo funciona si hemos realizado unos pasos previos sobre este fichero script:
1. La primera línea del script (para orientar al sistema sobre que programa se empleará sobre él) será:
#!/bin/bash
2. Cambiaremos los permisos del fichero para darle el de ejecución:
chmod a+x <fichero script>
3. Ahora podremos ejecutarlo:
a. /ruta-absoluta/<fichero script>
Pe.: /root/script1
b. ./<fichero script> si nos hallamos en el directorio donde esta el script.
Pe.: ./script1 nos hallamos en /root .

Ejemplo:
Contenido del script para realizar un backup
#!/bin/bash
tar cfz /var/backup/root-bk.tgz /root
echo Backup completado
· Comentarios y líneas en blanco
Podemos insertar comentarios en el script anteponiendo el carácter #.

Ejemplo:
#!/bin/bash
#Esto es un comentario
Tanto los comentarios como las líneas en blanco no efectúan ninguna operación durante
la ejecución del programa.

· Definición de variables
Una variable es un nombre simbólico que almacena un valor

Ejemplo:
#!/bin/bash
STR=hola

echo $STR
La definición de una variable . nombre=valor
El $ indica que se va a usar el contenido de la variable o expresión.
No usar para nombre de una variable una palabra reservada del sistema como un comando. Una forma de evitarlo es emplear mayúsculas para los nombres de las variables.

Ejemplo:
#!/bin/bash
CMD=tar
ARG1=/var/backup/root-bk.tgz
ARG2=/root
$CMD cfvz $ARG1 $ARG2
Al final ejecuta tar cfvz /var/backup/root-bk.tgz /root
Nota: Podríamos pasar los argumentos junto con el comando empleando comillas (").
CMD="tar cvfz"

· Argumentos
$0 nombre del script .
$1,$2,$3,...,$9 Los 9 primeros argumentos .
$# número de argumentos pasados al script .
$@,$* todos los argumentos, empezando por $1 .

Ejemplo:
#!/bin/bash
echo Nombre del script: $0
echo Número de argumentos: $#
echo Argumento 1: $1
echo Argumento 2: $2
echo Argumento 3: $3
echo Parte de argumentos: $@
echo Parte de argumentos: $*

Ejemplo:
#!/bin/bash
#Programa backup
tar cvfz $1 $2
Hemos cogido el origen de los datos y el nombre del nuevo fichero como argumentos al
llamar al programa:
backup /root/etc-bk.tgz /etc

· Estructura condicional:
if <condición>;
then
<comandos>
fi

Ejemplo:
if ["foo" = "foo"];
then
echo Sólo se ve si la condición es cierta
fi
echo Esto se verá siempre
if <condición>;

```
then
<comandos>
else
<comandos>
fi

Ejemplo:
#!/bin/bash
if [ "$1" = "hola" ];
then
echo Esto se vera si la condición es VERDAD
else
echo Esto se vera si la condición es FALSA
fi
echo FIN
if <condición 1>;
then
<bloque 1>
elif <condición 2>;
then
<bloque 2>
elif <condición 3>;
then
<bloque 3>
fi

Ejemplo:
if [ "$1" = "hola" ];
then
echo Es HOLA
elif [ "$1" = "adiós" ];
then
echo Es ADIOS
elif [ "$1" = "linux" ];
then
echo Es LINUX
else
echo Es OTRA COSA
fi
case <variable>
in
expresión 1)
<bloque 1>
;;
expresión 2)
<bloque 2>
;;
expresión 3)
<bloque 3>
;;
*)
<bloque por defecto>
;;
esac
Ejemplo:
case $1
```

```
in
"hola")
echo Es HOLA
;;
"adiós")
echo Es ADIOS
;;
"linux")
echo Es LINUX
;;
*)
echo Es OTRA COSA
;;
esac
```

· Condiciones:
Cadena de texto (strings)
["$a" = "$b"] Igualdad .
["$a" == "$b"] Igualdad .
["$a" != "$b"] Desigualdad .
["$a" \< "$b"] Orden alfabético. Pe.: . a=abc es menor que b=zxy
["$b" \> "$b"] Orden alfabético inverso .
[-z "$a"] es verdadero si la variable esta vacía .
[-n "$a"] es verdadero si la variable contiene algo .

Condiciones sobre números enteros
["$a" -eq "$b"] Igualdad .
["$a" -ne "$b"] Desigualdad .
["$a" -gt "$b"] Mayor que .
["$a" -lt "$b"] Menor que .
["$a" -ge "$b"] Mayor o igual que .
["$a" -le "$b"] Menor o igual que .

Condiciones sobre ficheros
[-e nombre fichero] Existencia .
[-f nombre fichero] es fichero ordinario .
[-s nombre fichero] es fichero vacío (0 bytes) .
[-d nombre fichero] es directorio .
[-b nombre fichero] es dispositivo de bloques .
[-c nombre fichero] es dispositivo de caracteres .
[-L nombre fichero] es enlace simbólico .
[-r nombre fichero] tiene permiso de lectura .
[-x nombre fichero] tiene permiso de ejecución .
[-w nombre fichero] tiene permiso de escritura .
[-O nombre fichero] somos propietarios del fichero (UID) .
[-G nombre fichero] somos propietarios del fichero (GID) .
[-N nombre fichero] el fichero se modifico desde la ultima vez que fue leído .
[-h nombre fichero] es enlace duro .

Negación:
Se usa el carácter !
[! -e fichero]
[! -f fichero]
Operadores lógicos

["expr1" -a "expr2"] AND Verdadera si expr1 Y expr2 . .

[“expr1” -o “expr2”] OR Verdadera si expr1 O expr2 . .
[expr1] || [expr2] OR .

· read
read acepta datos desde el teclado

Ejemplo:
echo Introduzca un dato
read DATO
echo $DATO

· Valor de retorno
Ejecución correcta devuelve 0 .
Ejecución no correcta devuelve valores de 1 a 255 .
El valor de retorno se almacena en la variable $?

· Bucles
for <variable> in <recorrido>
do
<comandos>
done

Ejemplo:
for i in $(ls)
do
echo Fichero: $i
done
Con seq <valor inicial> <valor final> incrementaremos de 1 en 1 el valor de la variable
desde el valor inicial hasta el valor final

Pe.:
seq 1 100 de 1 en 1 desde 1 hasta 100 .
Con seq <valor inicial> <incremento> <valor final> incrementaremos el valor de incremento en la variable desde el valor inicial hasta el valor final.

Pe.:
seq 1 2 10 de 2 en 2 desde 1 hasta 10 .

Ejemplo:
for i in $(seq 1 100)
do
echo Valor: $i
done
while <condición>
do
<comandos>
done
Ejemplo:
a=0
while [$a -lt 10]
do
echo Valor: $a
let a=a+1
done
until <condición>

do
<comandos>
done
Ejemplo:
a=20
until [$a -lt 10]
do
echo Valor: $a
let a=a-1
done

· Usar el valor de un comando
Podemos usar el resultado de un comando como valor de otro comando.

Pe.:
a=$(pwd) # En a almacena el valor de pwd
ls $a # Muestra un listado de a

· Salir de un script
exit

· Operaciones aritméticas
Suma (+) . 2+3
Resta (-) . 2-3
Multiplicación (*) . 3*5
División entera (/) . 3/2
Resto de división entera (%) . 3%2
Exponenciación (**) . 3**2

· let
Asigna un nuevo valor a una variable
let <variable>=<expresión>

Pe.:
let a=1+3
let a=$a+1
Auto incremento . let <variable>++ let a++ .
Auto decremento . let <variable>-- let a-- .
Otros:
let a+=2 . let a=$a+2
let a-=5 . let a=$a-5

· expr y similares
Pe.:
echo $(expr 1 + 3) muestra 4 .
expr muestra el valor de una expresión.
$((expresión)) o $[expresión]

Pe.:
echo 1 + 1 . muestra 1 + 1
echo $((1 + 1)) . muestra 2

· bc
bc es una extensión de la consola que nos permite operaciones matemáticas mas complejas. Para más información man bc

Pe.:
echo 3/4 | bc -l

· Rango de los números
El rango de los números soportado por los scripts en las operaciones matemáticas es de
números enteros de 32 bits: de -2147483648 a 2147483647

· break
Rompe la ejecución de un bucle, nos manda al final del bucle.

· continue
Continua la ejecución de un bucle, nos manda al principio de este.
Tanto break como continue actúan sobre el bucle en el que se están ejecutando. Si este
es un bucle anidado y queremos interrumpir la ejecución normal de varios de los bucles
usaremos break N y continue N, siendo N el número de bucles que queremos alterar.

Pe.:
```
for externo in 1 2 3 4 5
do
echo "Externo: $externo"
for interno in 1 2 3 4 5
do
echo "Interno: $interno"
if [ "$interno" -eq 3 ];
then
break
fi
done
done
```

Nota: para imprimir $ como carácter usaremos las comillas simples (‘), de otra forma
piensa que es una variable.

· Funciones
```
function <nombre>
{
<comandos>
}
```

Pe.:
```
#!/bin/bash
function hola {
echo Hola
}
function salir {
exit
}
hola #esto invoca a la función hola
salir #esto invoca a la función salir
```

Nota: el carácter } lo podemos lograr con AltGr + 0

Para pasar un argumento a una función lo escribimos seguido de la función separado por
un espacio.

Pe.:
function imprimir {
echo $1
}
imprimir Hola
Nos referimos a los argumentos dentro de la función como si fueran argumentos del script ($1, $2, ...)

· Variables globales y locales
Pueden existir variables globales del script y locales de la función con el mismo nombre.

Pe.:
function saludar {
local saludo=hola # le damos un valor a la variable local saludo de la función
echo $saludo # mostramos el valor local de saludo en la función
}
saludo=hello # Esta es la variable global saludo que se usa en todo el script
saludar
echo $saludo # Nos muestra el valor del hola global porque esta fuera de la función
Nos mostrará:
hola
hello

· select
Sirve para crear menús interactivos.
#!/bin/bash
OPTIONS="Hola Salir"
select opt in $OPTIONS
do
if ["$opt" = "Salir"];
then
echo Fin
exit
elif ["$opt" = "Hola"];
then
echo Hola
else
clear
echo Opción incorrecta
fi
done
Por defecto para pedir la opción nos aparecerá un prompt con este formato #?. Podemos
cambiarlo pasándole los valores nuevos a la variable PS3.
PS3=">>"