SCRIPT

1. INTRODUCCION

Los Shell Scripts son pequeños, pero a la vez muy útiles, archivos de texto que contienen una serie de comandos para shell que el sistema ejecuta ordenadamente, de arriba abajo. Para editarlos, tan solo hace falta un editor de textos, como vi ó gedit el primero modo texto y el segundo modo gráfico. Se guardan con extensión ".sh" y se ejecutan desde la Shell mediante el comando: **sh nombrescript.sh** ó ./nombrescript.sh. Los scripts, se comportan de igual manera que los comandos de la shell y necesitamos antes de ejcutarlos darles permisos de ejecución de la siguiente manera **chmod 777 nombrescript.sh**

chmod 777 nombrescript.sh
sh nombrescript.sh
ó
./nombrescript.sh

2. SABER QUE SHELL SE TIENE INSTALADOS

Si queremos saber que shell tenemos instaldos en el sistema

cat /etc/shells

3. RUTA DONDE BUSCAR EL INTERPRETE DE COMANDOS BASH

Esta primera línea en todo script evitará problemas a la hora de portar nuestros scripts ya que en otros sistemas puede que el bash se encuentre en ubicaciones distintas, por lo tanto al colocar esto en la primera línea de nuestro código hacemos que primero se busque en la variable de entorno PATH donde está el interprete de comandos bash.

#!/usr/bin/env bash

4. CORCHETES SIMPLES [VS DOBLES [[

- Los corchetes dobles son una mejora a los corchetes simples, por lo tanto es mas reciente su aparición.
- Tamto el corchete simple y dobles sirven para evaluar expresiones y ambos están disponibles en el estándar POXIX.
- El corchete simple [es un alias para el comando test.
- El corchete doble [[es una keyword(palabra reservada) no un programa y se trata de una versión mejorada para evaluar expresiones unicamente disponibles en Bash, KornShell y Zsh.
- Si la portalidad no es importante para tus scripts es decir que solo bas a trabajar con sistemas que tengan Bash, KornShell y Zsh es mejor usar siempre el doble corchete, por la funcionalidad extra que aporta y esta es posibilidad de usar &&, ||, expresiones regulares, pattern matching, etc).

5. MIS PRIMEROS SCRIPT

Ejemplo 0022: Mostrar el directorio actual y el usuario que se ha logueado en el sistema. El simbolo # al inicio de una línea es para mostrar comentarios de una línea dentro del script, el comando echo es para mostrar mensajes en la pantalla, el comando pwd muestra el directorio actual y el comando whoami es para mostrar el usuario que se ha logueado en el sistema.

```
#!/usr/bin/env bash
# Nuestro comentario de una línea
echo "El directorio actual es:"
pwd
echo "El usuario logueado es:"
whoami
```

6. GESTION DE CADENAS

4.1. CADENAS ENTRE COMILLAS SIMPLES

Las comillas simples nos sirven para hacer que la shell interprete lo encerrado entre ellas como una expresión literal, ya sean variables, comodines u otras expresiones, es decir, el contenido no es interpretado por la shell.
Para entenderlo mejor, imaginemos que deseamos crear una carpeta de nombre # o *. Si lo intentamos, es decir si tecleamos mkdir #, la shell indicará un error. Sin embargo si escribimos mkdir '#', el intérprete de comandos tomará el carácter # como un literal y podremos crear la carpeta con ese nombre.

- Crear la siguiente carpeta que llamaremos # al inicio. Tenemos que encerrar entre comillas dimples para que se interpreten literalmente.

mkdir '#'
mkdir '#carpeta'

- Resaltar una palabra dentro de una frase con comillas simples entonces la frase tiene que estar entre comillas dobles y la palabra a resaltar entre comillas simples y viceversa

echo "Hola 'Carlos' como estas"
echo 'Hola "Carlos" como estas'

4.2. CADENAS ENTRE COMILLAS DOBLES

Este tipo de cadenas de caracteres permiten la sustitución de parámetros, la sustitución de comandos y la evaluación de expresiones aritméticas, pero ignoran las sustituciones de tilde, expansión de caracteres comodines, aliases y la división de palabras vía delimitadores.

Es decir, se produce sustitución de variable (el signo de pesos se interpreta) por ejemplo, podremos ejecutar ls "$PWD" y nos devolverá correctamente el listado de archivos del directorio en que nos encontramos, pero si hacemos ls "*", el comodín será tomado como un literal.

- Se pueden incluir comillas dobles dentro de una cadena con comillas dobles anteponiendo un \.

echo "Hola \"Carlos\" como estas"

- Mostrar el contenido de la variable de entorno PATH acopañado de un mensaje

echo "RUTAS $PATH"

4.3. ESCAPAR CARACTERES

- Una barra inversa permite escapar al carácter que acompaña es decir que no será interpretado por el shell.

echo "RUTAS \$PATH"

7. ASIGNACION

La asignación de variables se realiza mediante el símbolo de la igualdad =. El símbolo de la igualdad no debe estar seguido de espacios.

edad=10
nombre="Hola Linux"

8. IMPRIMIR MENSAJES

- Para mostrar mensajes en la patalla se usa el comando echo

Ejemplo 0010

```
#!/usr/bin/env bash
echo Hoy como estas
edad=20
echo Creo que tienes $edad años
echo -e "Esta \nfrase \nse \nmostrará \npalabra \npor \npalabra \nen \nuna \nlínea \ndistinta
\ncada \npalabra"
echo -e "Suprimir lo que viene a continuación \cel salto de línea"
echo Imprimir todos los ficheros y carpetas a modo de comando ls
echo *
echo Imprimir todos los ficheros de un formato en concreto
echo *.sh
echo "Esta frase se direcciona a un archivo donde queda grabado" > salvar.txt
echo "Esto que sigue se añade al archivo" >> salvar.txt
echo -n "Omitimos el salto de línea"
```

- Representar caracteres unicode

```
#!/usr/bin/env bash
echo -e "\u2622"
echo -e "\U1f41e"
```

9. VARIABLES

- Las variables en bash se definen como abajo en rojo indicamos, sin espacios antes o después del símbolo = y si queremos imprimir el contenido de la variable se usa el simbolo $ delane del nombre de la variable.

NombreVariable=ValorVariable

Ejemplo 0006: Crear una variable que llamaremos edad y le asignaremos un valor de 15, luego imprimeremos el contenido de la variable en pantalla.

```
#!/usr/bin/env bash
edad=15
echo $edad
```

- Podemos asignarle diferentes valores a una misma variable

Ejemplo 0007

```
#!/usr/bin/env bash
mensaje='Hola mundo'
echo $mensaje
mensaje=5.5
echo $mensaje
mensaje=8
echo $mensaje
```

- El shell tiene una forma de evitar que se pueda modificar el valor de una variable, es decir ponerlo como sólo lectura. En el ejemplo siguiente da un error si intentamos moficar el contenido de la variable.

Ejemplo 0008

```
#!/usr/bin/env bash
mensaje='Hola mundo'
echo $mensaje
readonly mensaje
mensaje='Cambo de texto'
echo $mensaje
```

- Variables especiales

$$	Muestra el número de proceso del sistema que tiene el shell, es decir el PID.
$0	Muestra el nombre del script que se está ejecutando.
$n	El valor de n es el número de argumento que le pasamos al script, de forma que si hay varios parámetros lo definiríamos como $1, $2, $3, en el caso de que haya 3 parámetros.
$#	Muestra el número total de parámetros que le pasamos en el script.
$*	Muestra el valor de todos los parámetros que le pasamos al script.
$@	Muestra el valor de todos los parámetros que le pasamos al script.
$?	El estado de salida del último comando ejecutado. Shell devuelve un estado para saber si es correcto la ejecución o no, es decir, mostrará un 0 si ha ido todo bien y un 1 si ha fallado o ha tenido errores
$!	El número de proceso del último comando que está en segundo plano.

Ejemplo 0009

```
#!/usr/bin/env bash
echo "Nombre del script: $0"
echo "Parámetro1 : $1"
echo "Parámetro2 : $2"
echo "Valores de los parámetros: $@"
echo "Valores de los parámetros: $*"
echo "Total parámetros : $#"
echo $?
```

- Ejecutamos el script en la línea de comandos de la siguiente manera si al escript le damos el nombre s0009.sh sería de la siguiente manera:

./s0009.sh hola mundo

10. ESTRUCTURAS DE CONTROL

9.1. CONDICIONAL SIMPLE: IF

```
if [ condicion ]
then
  sentencias
fi;
```

9.2. CONDICIONAL DOBLE: IF ... ELSE

```
if [ condicion ]
then
  sentencias 1
else
  sentencias 2
fi;
```

9.3. CONDICIONAL MULTIPLE: IF ... ELIF ... ELSE

```
if [ condicion1 ]
then
  sentencias 1
elif [ condicion2 ]; then
  sentencias 2
else
  sentencias 3
fi;
```

9.4. CONDICIONAL MULTIPLE: CASE

En esta estructura cuando se encuentre un patrón que coincida, se ejecuta la lista de comandos hasta los ";;" y se termina la ejecución de case.

```
case expresion in
1)
sentencias 1
;;
2)
  sentencias 2
...

*)
  sentencias n
;;
esac
```

9.5. BUCLE: FOR

El bucle 'for' es una estructura de control iterativa que permite repetir una sección del programa, un número fijo de veces.

FORMA 1

```
for variable in [lista]
do
        sentencias
done
```

FORMA 2

```
for (( inicio ; condicion_termino ; incremento ))
do
  sentencias
done
```

9.6. BUCLE: WHILE

Sirve para repetir un bucle mientras la condición sea verdadero.

```
while [ condicion ]
do
sentencias
done
```

9.7. BUCLE: UNTIL

Sirve para repetir un bucle mientras la condición sea falso.

```
until [ condicion ]
do
        sentencias
done
```

11. SENTENCIA CONDICIONAL

Ejemplo 0023: Hacer un script que muestre el mensaje "El usuario **NombreUsuario** esta trabajando". Con > nul anulamos la salida del comando para que no se muestre en pantalla.

```
#!/usr/bin/env bash
echo FORMA 1
if whoami > nul; then
    echo "El usuario $USER esta trabajando"
fi
echo FORMA 2
if whoami; then
    echo "El usuario $USER esta trabajando"
fi
```

Ejemplo 0024: Hacer un script que verifique si un usuario existe o no existe.

```
#!/usr/bin/env bash
usuario=walter
if grep $usuario /etc/passwd > nul;
then
   echo "El usuario $usuario existe"
else
   echo "El usuario $usuario no existe"
fi
```

12. OPERADORES PARA COMPARAR CADENAS

[[	[
>	>
<	<
=	=
!=	!=

Ejemplo 0012: Comparar el contenido de 2 variables que tienen una cadena almacenada cada variable.

```
#!/usr/bin/env bash
nombre1="Luis"
nombre2="luis"
if [[ $nombre1 = $nombre2 ]]
then
   echo Son iguales
else
   echo Son distintos
fi;
```

Ejemplo 0013: Se pueden comparar las cadenas directamente si no tiene espacios en blanco sin comillas simples o dobles

```
#!/usr/bin/env bash
if [[ Luis = luis ]]
then
   echo Son iguales
else
   echo Son distintos
fi;
```

Ejemplo 0014: Si la cadena tiene espacios en blanco se tienen que poner entre apostrofes simples

```
#!/usr/bin/env bash
if [[ 'Luis estas bien' = 'Luis estas bien' ]]
then
   echo Son iguales
else
   echo Son distintos
fi;
```

Ejemplo 0015: Si la cadena va incluir variables emplear apostrofe doble.

```
#!/usr/bin/env bash
edad=15
if [[ "Tengo $edad años" = 'Tengo 15 años' ]]
then
   echo Son iguales
else
   echo Son distintos
fi;
```

Ejemplo 0028: Verificar si una variable tiene valor o no. Con –n busca un valor mayor que cero

```
#!/usr/bin/env bash
clear
variable="Hola"
if [ -n $variable ];
then
   echo "Tiene una longitud mayor que cero"
else
   echo "Tiene una longitud de cero"
fi;
```

13. OPERADORES PARA COMPARAR ENTEROS

[[	[	
-gt	-gt	mayor
-lt	-lt	menor
-ge	-ge	mayor o igual
-le	-le	menor o igual
-eq	-eq	igual
-ne	-ne	distinto

En el ejemplo 0001, debe haber un espacio al inicio y al final del corchete, es decir entre el corchete de inicio y el 2 y entre el 3 y el corchete final.

Ejemplo 0001: Determinar si 2 es mayor que 3. Usar corchete simple.

```
#!/usr/bin/env bash
if [ 2 -gt 1 ]
then
    echo "2 es mayor que 1"
else
    echo "2 no es mayor que 1"
fi;
```

Ejemplo 0002: Determinar si 2 es distinto que 3. Usar corchete simple.

```
#!/usr/bin/env bash
if [ 2 -ne 3 ]
then
    echo "2 es distinto a 3"
else
    echo "2 es igual a 3"
fi;
```

Ejemplo 0005: Determinar si 2 es distinto que 3. Usar corchete doble.

```
#!/usr/bin/env bash
if [[ 2 -ne 3 ]]
then
    echo "2 es mayor que 3"
else
    echo "2 no es mayor que 3"
fi;
```

14. RESULTADO DE UN COMANDO

Ejemplo 0025: Hacer un script que verifique si el usuario logueado puede crear un archivo.

```
#!/usr/bin/env bash
touch /root/prueba
if [ $? -eq 0 ];
then
        echo "Se ha creado el fichero"
else
        echo "No se ha creado el fichero"
fi
```

Este ejejmplo se parte con la condición que el usuario que ejecuta el script es un usuario distinto a root.
Aquí el comando touch intentará crear el fichero /root/prueba y fallará... porque no estás logueado como usuario root ¿no?
Al fallar mostrará un mensaje de error pero ¿cómo se entera nuestro script de que el programa ha fallado? Pues a través de la variable $?
$? es una variable especial que recoge el valor devuelto por el último programa ejecutado. Cuando touch funciona sin problemas devuelve el valor 0; cuando falla, como en este caso, devuelve un número distinto de cero.
Entonces bastará con comprobar si la variable $? tiene almacenado un 0 para ver si ha funcionado correctamente.

Ejemplo 0026: Script que verifica si el usuario logueado puede crear un archivo.

```
#!/usr/bin/env bash
touch prueba
if [ $? -eq 0 ];
then
        echo "Se ha creado el fichero"
else
        echo "No se ha creado el fichero"
fi
```

15. OEPRADORES BOLEANOS

[[	[
&&	-a
||	-o

Ejemplo 0003: Determinar si 4 estra entre 3 y 5. Usar corchete doble.

```
#!/usr/bin/env bash
if [[ 4 -gt 3 && 4 -lt 5 ]]
then
    echo "4 esta entre 3 y 5"
else
    echo "4 no esta entre 3 y 5"
fi;
```

Ejemplo 0004: Determinar si 4 esta entre 3 y 5. Usar corchete simple.

```
#!/usr/bin/env bash
if [ 4 -gt 3 -a 4 -lt 5 ]
then
    echo "4 esta entre 3 y 5"
else
    echo "4 no esta entre 3 y 5"
fi;
```

16. AGRUPACION

Para agrupar operaciones booleanas puedes utilizar paréntesis con los dobles corchetes, mientras que en el caso de los simples corchetes deberás utilizar los paréntesis pero escapados.

17. BUCLE FOR

Ejemplo 0016: Hacer un bucle form para iterar sobre valores simples, siendo estos valores cadenas simples.

```
#!/usr/bin/env bash
for var in Primero Segundo Tercero Cuarto Quinto; do
    echo El $var item
done
```

Ejemplo 0017: Hacer un bucle for para iterar sobre valores simples, siendo estos valores numeros.

```
#!/usr/bin/env bash
for var in 1 2 3 4 5; do
    echo Número $var
done
```

- Hacer un blucle for para iterar sobre valores complejos

Ejemplo 0018: Hacer un bucle for para iterar sobre valores complejos, siendo estos valores palabras o frases.

```
#!/usr/bin/env bash
for var in Primero "El Segundo" "El Tercero" "El Quinto soy yo"; do
    echo Esto es: $var
done
```

Ejemplo 0019: Hacer un blucle for para mostrar el contenido de un archivo.

- Crear primero el archivo walter.txt con el contenido:

Hola
esto
esta
en un
archivo

- Ahora corre el siguiente script

```
#!/usr/bin/env bash
miarchivo="walter.txt"
for var in $(cat $miarchivo); do
    echo " $var"
done
```

Ejemplo 0020: Hacer un blucle for para leer un archivo, pero que considere el carácter de nueva línea como un separador en lugar de espacios.

```
#!/usr/bin/env bash
miarchivo="/etc/passwd"
IFS=$'\n'
for var in $(cat $miarchivo); do
    echo " $var"
done
```

Ejemplo 0021: Hacer un bucle for para leer un archivo pero que considere el carácter punto y coma como separador.

- Crear primero el archivo walter21.txt con el siguiente contenido:

Luis;Ruiz;Roncal;24;1.74

- Ahora corre el siguiente script

```
#!/usr/bin/env bash
miarchivo="walter21.txt"
IFS=$';'
for var in $(cat $miarchivo); do
    echo " $var"
done
```

18. COMPARACION DE ARCHIVOS

-d my_file	Comprueba si es una carpeta.
-e my_file	Comprueba si el archivo está disponible.
-f my_file	Comprueba si es un archivo.
-r my_file	Comprueba si es legible.
my_file -nt my_file2	Comprueba si my_file es más nuevo que my_file2.
my_file -ot my_file2	Comprueba si my_file es más viejo que my_file2.
-O my_file	Comprueba si el propietario del archivo y el usuario registrado coinciden.
-G mi_archivo	Comprueba si el archivo y el usuario registrado tienen el grupo idéntico.

Ejemplo 0027: Comprobar si puedo ingresar a mi carpeta personal de trabajo.

```
#!/usr/bin/env bash
clear
micarpeta=/home/walter
if [ -d $micarpeta ];
then
    echo "Carpeta $micarpeta existe"
    cd $micarpeta
    pwd
    ls
else
    echo "No hay archivo o directorio $micarpeta"
fi
```