TEMA 5

SHELL SCRIPTING

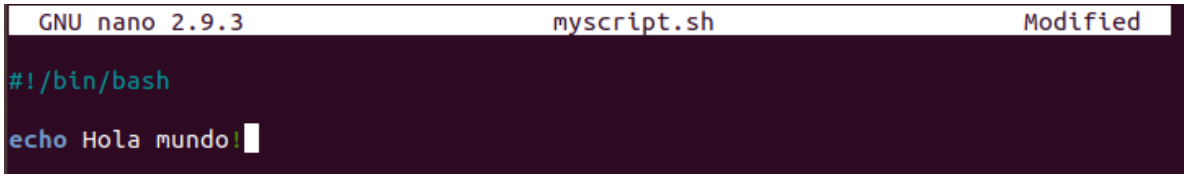
ÍNDICE

5. SHELL SCRIPTING

5.1 Introducción

Un script es un programa interpretado (no compilado) que se ejecuta paso a paso. Con los scripts de Bash podemos hacer cualquier cosa que hagamos en una consola pudiendo combinarlo con sentencias lógicas, bucles, funciones, etc…

Un script de Bash debe comenzar siempre con el 'shebang' #!/bin/bash:



Un fichero de script se lanza de la siguiente manera: ./myscript.sh

Pero la primera vez que intentéis ejecutarlo, no va a ser posible, puesto que ese fichero no tiene permisos de ejecución, es necesario dárselos con chmod:

```
aitor@aitor-virtual-machine:~$ nano myscript.sh
aitor@aitor-virtual-machine:~$ ./myscript.sh
bash: ./myscript.sh: Permission denied
aitor@aitor-virtual-machine:~$ ls -l myscript.sh
-rw-r--r-- 1 aitor aitor 30 oct 14 10:33 myscript.sh
aitor@aitor-virtual-machine:~$ chmod 755 myscript.sh
aitor@aitor-virtual-machine:~$ ls -l myscript.sh
-rwxr-xr-x 1 aitor aitor 30 oct 14 10:33 myscript.sh
aitor@aitor-virtual-machine:~$ ./myscript.sh
Hola mundo!
aitor@aitor-virtual-machine:~$
```

Los comentarios en Bash empiezan por #:

```
#!/bin/bash

echo Hola mundo! #esto es un comentario
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
Hola mundo!
```

¿Por qué usamos ./ para ejecutar los scripts?

Bash ejecuta los programas desde los directorios incluidos en el PATH (en mayusculas):

```
aitor@aitor-virtual-machine:~$ echo $path

aitor@aitor-virtual-machine:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/lo
cal/games:/snap/bin
```

Como nuestro directorio de home no está incluido en el PATH tenemos que decirle la ruta completa, y el punto si recordáis es el directorio actual.

¿Por qué hay que usar el 'shebang'?

Le estamos indicando que interprete de Bash hay que utilizar. El formato es muy importante. Debe de ser la primera línea (no vale que sea la segunda, aunque la primera esté vacía) y no puede haber espacios entre los diferentes caracteres.

5.2 Variables

Las variables se pueden declarar usando el símbolo = y se usan con el símbolo $

```
#!/bin/bash
Nombre1=Mundo
Nombre2=Foo

echo Hola $Nombre1
echo Hola $Nombre2
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
Hola Mundo
Hola Foo
```

Cuando las variables tienen espacios es necesario usar comillas o nos dará un error:

```
#!/bin/bash
var1=foo bar

echo $var1
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
./myscript.sh: line 2: bar: command not found
```

En cambio, con comillas:

```
#!/bin/bash
var1="foo bar"

echo $var1
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
foo bar
```

Las comillas dobles sustituirán las variables por su valor, mientras que las simples tomarán los caracteres literalmente.

Los valores también se le pueden proporcionar al script cuando se ejecuta, por ejemplo, el siguiente script tomara dos valores desde la línea de comandos:

```
#!/bin/bash
echo var1: $1
echo var2: $2
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh foo bar
var1: foo
var2: bar
```

Hay otra serie de variables reservadas:

- **$0** - The name of the Bash script.
- **$1 - $9** - The first 9 arguments to the Bash script. (As mentioned above.)
- **$#** - How many arguments were passed to the Bash script.
- **$@** - All the arguments supplied to the Bash script.
- **$?** - The exit status of the most recently run process.
- **$$** - The process ID of the current script.
- **$USER** - The username of the user running the script.
- **$HOSTNAME** - The hostname of the machine the script is running on.
- **$SECONDS** - The number of seconds since the script was started.
- **$RANDOM** - Returns a different random number each time is it referred to.
- **$LINENO** - Returns the current line number in the Bash script.

También es posible guardar el resultado de un comando en una variable:

```
#!/bin/bash
var1=$( ls -lh )

echo $var1
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
total 68K -rw-r--r-- 1 aitor aitor 72 oct 10 09:47 bar.txt drwxr-xr-x 2 aitor a
itor 4,0K oct 4 11:07 capitulo3 drwxr-xr-x 2 aitor aitor 4,0K oct 3 15:31 Deskt
op drwxr-xr-x 2 aitor aitor 4,0K oct 3 15:31 Documents drwxr-xr-x 2 aitor aitor
 4,0K oct 3 15:31 Downloads drwxr-xr-x 2 aitor aitor 4,0K oct 3 15:31 Music -rw
xr-xr-x 1 aitor aitor 41 oct 14 11:41 myscript.sh -rw-r--r-- 1 aitor aitor 13K
oct 10 10:21 output.html drwxr-xr-x 2 aitor aitor 4,0K oct 3 15:31 Pictures -rw
-r--r-- 1 aitor aitor 16 oct 10 09:25 prueba.txt drwxr-xr-x 2 aitor aitor 4,0K
oct 3 15:31 Public drwxr-xr-x 2 aitor aitor 4,0K oct 3 15:31 Templates -rw-r--r
-- 1 aitor aitor 98 oct 10 12:07 test_grep.txt drwxr-xr-x 2 aitor aitor 4,0K oc
t 3 15:31 Videos
```

5.3 User input

Con el comando read se le puede pedir al usuario datos de entrada:

```
#!/bin/bash

echo cual es tu nombre?
read varname

echo Tu nombre es: $varname
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh a
cual es tu nombre?
aitor
Tu nombre es: aitor
```

Con -p podemos hacer el prompt (escribir el texto) y leer el texto a la vez. Con -s podemos hacer que la entrada de texto no muestre lo que escribimos por pantalla.

```
#!/bin/bash

read -p 'Nombre de usuario:'  name
read -sp 'Contraseña:' pass

echo Tu nombre de usuario y contraseña son $name y $pass
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh a
Nombre de usuario:aitor
Contraseña:Tu nombre de usuario y contraseña son aitor y test
```

5.4 Operaciones aritméticas

El comando let permite evaluar operaciones aritméticas sencillas:

```
#!/bin/bash
let a=1+2
echo $a

let "a = 3 + 3" #para utilizar espacios usamos las comillas
echo $a

let a++
echo $a

let a=4*5
echo $a

let a=$1+10
echo $a
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh 10
3
6
7
20
20
```

El comando expr es igual que let pero imprime por pantalla:

```
#!/bin/bash
expr 3 + 4
expr 1+2
expr "4 + 6"
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
7
1+2
4 + 6
```

Otra manera de salvarlo a una variable es usar las dobles paréntesis:

```
#!/bin/bash
a=$(( 4 + 5 ))
echo $a

a=$((3+4))
echo $a
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
9
7
```

Para saber la longitud de una variable usaremos la siguiente expresión:

```
#!/bin/bash
a=3846
echo ${#a}
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
4
```

5.5 Condicionales

Las condicionales en Bash siguen el siguiente formato:

```
if [ <some test> ]
then
      <commands>
else
      <other commands>
fi
```

Por ejemplo:

```
#!/bin/bash
if [ 100 -gt 2 ]
then
        echo 100 es mas grande que 2
fi
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
100 es mas grande que 2
```

Las opciones de test y comparaciones son:

! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq TEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt NTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

Los ifs se pueden indentar:

```
#!/bin/bash
if [ 100 -gt 2 ]
then
        echo 100 es mas grande que 2
        if (( 100 % 2 == 0  ))
        then
                echo y par
        fi
fi
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
100 es mas grande que 2
y par
```

Además los ifs pueden tener elifs y elses:

```
if [ <some test> ]
then
<commands>
elif [ <some test> ]
then
<different commands>
else
<other commands>
fi
```

```
#!/bin/bash
if [ $1 -gt 100 ]
then
        echo el numero es más grande que 100
elif (( $1 % 2 == 0 ))
then
        echo el numero es par
else
        echo el numero es impar
fi
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh 20
el numero es par
```

Los tests pueden utilizar también operadores booleanos, && para AND y || para OR:

```
#!/bin/bash
if [ -r $1 ] && [ -s $1 ]
then
        echo este fichero es util
fi
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh myscript.sh
este fichero es util
```

También se puede utilizar el case:

```
case <variable> in
<pattern 1>)
<commands>
;;
<pattern 2>)
<other commands>
;;
esac
```

Por ejemplo:

```
#!/bin/bash
case $1 in
        hola)
                echo hola
                ;;
        adios)
                echo adios
                ;;
        *)
                echo no se lo que es
                ;;
esac
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh hola
hola
aitor@aitor-virtual-machine:~$ ./myscript.sh fooo
no se lo que es
```

5.6 Loops

Tenemos diferentes tipos de loops disponibles en Bash.

While

```
while [ <some test> ]

do
<commands>
done
```

Por ejemplo:

```
#!/bin/bash
counter=1
while [ $counter -le 10  ]
do
        echo $counter
        ((counter++))
done

echo se acabo
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
1
2
3
4
5
6
7
8
9
10
se acabo
```

Until

```
until [ <some test> ]
do
<commands>
done
```

Por ejemplo:

```
#!/bin/bash
counter=1
until [ $counter -gt 10  ]
do
        echo $counter
        ((counter++))
done

echo se acabo
```

```
aitor@aitor-virtual-machine:~$ ./myscript.sh
1
2
3
4
5
6
7
8
9
10
se acabo
```

For

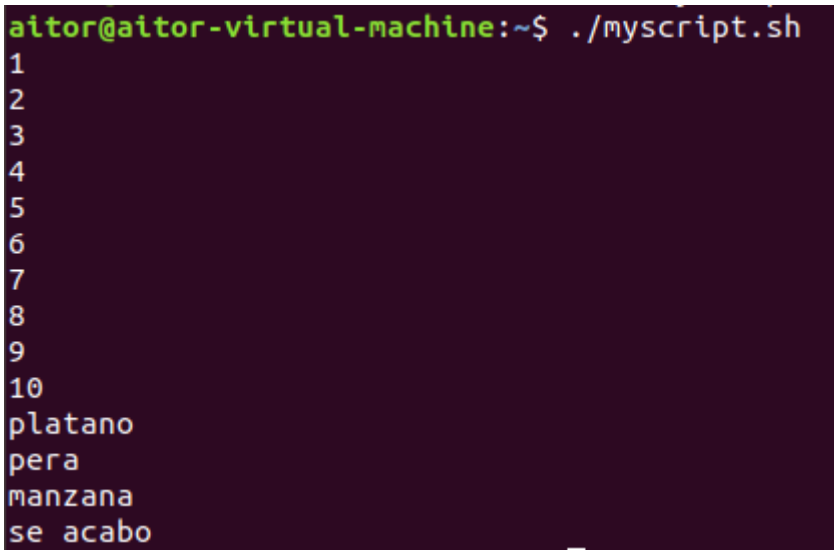
```
for var in <list>
do
<commands>
done
```

Por ejemplo:

```
#!/bin/bash
for value in {1..10}
do
        echo $value
done

frutas='platano pera manzana'
for fruta in $frutas
do
        echo $fruta
done

echo se acabo
```



Operadores	
Operador	**Descripción**
+	Suma.
-	Sustracción.
*****	Multiplicación. Como el shell reconoce la estrella en tanto que comodín, hay que cerrarla con una contrabarra: *.
/	División.
%	Módulo.
!=	Diferente. Visualiza 1 si diferente, 0 en caso contrario.
=	Igual. Visualiza 1 si igual, 0 en caso contrario.
<	Inferior. Visualiza 1 si inferior, 0 en caso contrario
>	Superior. Visualiza 1 si superior, 0 en caso contrario.
<=	Inferior o igual. Visualiza 1 si inferior, 0 en caso contrario.
>=	Superior o igual. Visualiza 1 si superior, 0 en caso contrario
:	Búsqueda en una cadena. P. ej.: expr Julio: J* devuelve 1, ya que Julio empieza por J. Sintaxis particular: expr "Julio": ".*" devuelve la longitud de la cadena