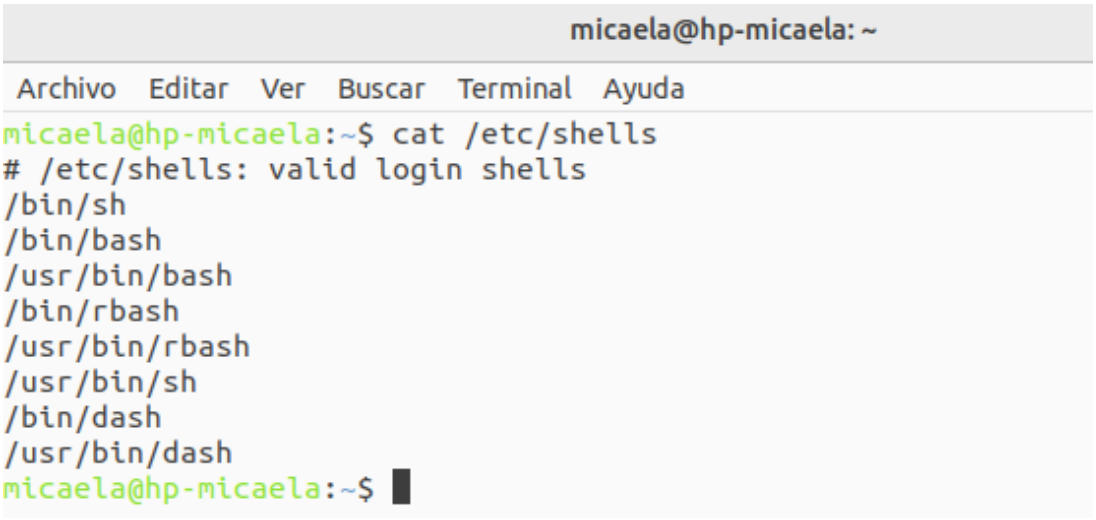
1. BASH

¿Qué shells tengo disponibles?

En /etc/shells puedo visualizar los shells disponibles en mi sistema. Con bash debe de ser más que suficiente para las tareas habituales que necesitemos.

Los repositorios de Linux contienen otros shells que podríamos añadir a nuestro sistema.

Lo vemos en el siguiente listado:



¿Por qué necesitamos scripts en bash?

Porque necesitamos AUTOMATIZAR tareas complejas, habituales y que requieran ser simplificadas.

Al crear un script que contiene una serie de instrucciones podemos reducir tareas repetitivas a la ejecución de una sola orden.

Ejemplos : Copia de seguridad, envío de mensajes, comprobación parámetros sistema, tareas complejas, ..

Además de las funcionalidades básicas de ejecución de una serie de órdenes bash nos proporciona una serie de características que nos va a permitir crear automatizaciones de cierta complejidad.

Entre esas características tenemos:

- Variables.
- Estructuras de control.
- Parámetros.
- Funciones.

Para crear un script debemos realizar los siguientes pasos:

1. Crear un fichero con un editor de texto, Geany o cualquier programa similar. La extensión de dicho fichero será sh
2. Dar permisos de ejecución bien a través del botón derecho y la pestaña permisos o bien a través del terminal mediante la orden chmod +x fichero.sh
3. Ejecutar el script bien a través del terminal o en el propio Geany. Si es a través del terminal pondremos ./
4. Dentro del fichero, nuestro script siempre empezará con #!/bin/bash

2. VARIABLES

Una variable es un “lugar” donde podemos almacenar un valor y recuperarlo cuando queramos, siempre que nuestro script este en ejecución.

Para poder hacer referencia a una variable utilizaremos un nombre. Sobre una variable podemos realizar operaciones de guardar su valor y leer su valor.

Por ejemplo:

variable.sh

```
1  #!/bin/bash
2  nombre=Micaela
3  echo "Hola $nombre"
4
5  echo Hola $nombre
6
```

Como resultado quedaría:

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Hola Micaela
Hola Micaela


------------------
(program exited with code: 0)
Press return to continue
```

Cuando las variables sean texto y contengan espacios en blanco, estas deberán estar definidas entre comillas:

nombre=Micaela

nombre=”Micaela Alfonso”

Comillas dobles y comillas simples

Si escribo echo “Hola $nombre”, el resultado será el mostrado en el ejemplo. Por contra si escribimos echo ‘Hola $nombre’ el resultado será el literal Hola $nombre, no sustituyendo la variable nombre en la ejecución.

Lectura de variables por teclado:

variable.sh

```
1  #!/bin/bash
2  echo "Dime tu nombre"
3  read nombre
4  echo "Hola $nombre"
5
6
```

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Dime tu nombre
Micaela
Hola Micaela


------------------
(program exited with code: 0)
Press return to continue
```

O también:

variable.sh

```
1  #!/bin/bash
2  echo "Dime tu nombre"
3  read nombre
4  echo "Hola ${nombre}"
5
6
```

Cuyo resultado es el mismo que el anterior.

También podemos pasar el valor de la variable directamente por el terminal al ejecutar el script:

variable.sh

```
1  #!/bin/bash
2  echo "Nombre pasado directamente al script por el terminal"
3
4  echo "Hola $1"
5
6
```

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
micaela@hp-micaela:~/Documentos/curso23-24/1bat/scripts_teoria$ ./variable.sh micaela
Nombre pasado directamente al script por el terminal
Hola micaela
micaela@hp-micaela:~/Documentos/curso23-24/1bat/scripts_teoria$
```

Variables especiales en bash

- $0 : Nombre del script
- $1 – $9 : los primeros nueve argumentos que se pasan a un script en Bash
- $# : el número de argumentos que se pasan a un script
- $@ : todos los argumentos que se han pasado al script
- $? : la salida del último proceso que se ha ejecutado
- $$: ID del proceso del script
- $USER : el nombre del usuario que ha ejecutado el script.
- $HOSTNAME : hostname de la máquina en la que se está ejecutando el script
- $SECONDS : tiempo transcurrido desde que se inició el script, contabilizado en segundos.
- $RANDOM : número aleatorio cada vez que se lee esta variable.
- $LINENO : indica el número de líneas que tiene nuestro script.

Variables accesibles en bash

Utilizando el comando env en un terminal podemos ver todas las variables accesibles de nuestro shell.

Variables avanzado

Veamos algunos ejemplos de uso avanzado de variables:

- Asignación de órdenes a variables. En el siguiente ejemplo nos contará los usuarios del sistema operativo.

variable.sh

```
1  #!/bin/bash
2  num=$(cat /etc/passwd|wc -l)
3
4  echo "Hay $num usuarios en el sistema"
5
```

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Hay 48 usuarios en el sistema


------------------
(program exited with code: 0)
Press return to continue
```

variable.sh

```
1  #!/bin/bash
2
3  num=`cat /etc/passwd|wc -l`
4
5  echo "Hay $num usuarios en el sistema"
6
```

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Hay 48 usuarios en el sistema


------------------
(program exited with code: 0)
Press return to continue
```

- Variables con más de una línea generada:

variable.sh

```
1  #!/bin/bash
2
3  num=`cat /etc/passwd`
4  echo $num
5
6
```

El resultado sería el listado de todo el fichero passwd de contraseñas encriptadas del sistema operativo.

- Operaciones aritméticas:

Operador	Operación	Ejemplo
+	Suma	echo $((2 + 2))
-	Resta	echo $((2 - 2))
/	División	echo $((2 / 2))
*	Multiplicación	echo $((2 * 2))
%	Módulo	echo $((2 % 2))
++	Incremento	x = 5 echo $((x++))
--	Decremento	x = 5 echo $((x--))
**	Exponente	echo $((2 ** 2))

- Operaciones con cadenas de texto. Extraer texto

variable.sh

```
1  #!/bin/bash
2  # Extraer texto
3  frase="Esto es una cadena de texto"
4  echo $frase
5  echo ${frase:6:10}
6
```

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Esto es una cadena de texto
s una cade


------------------
(program exited with code: 0)
Press return to continue
```

- Operaciones con cadenas de texto. Sustituir texto.

variable.sh

```
#!/bin/bash
# Extraer texto
frase="Esto es una cadena de texto"
echo $frase
echo ${frase/texto/caracteres}

```

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Esto es una cadena de texto
Esto es una cadena de caracteres


------------------
(program exited with code: 0)
Press return to continue
```

- Operaciones con cadenas de texto. Eliminar texto.

variable.sh

```
#!/bin/bash
# Extraer texto
frase="Esto es una cadena de texto"
echo $frase
echo ${frase//cadena}

```

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Esto es una cadena de texto
Esto es una de texto


------------------
(program exited with code: 0)
Press return to continue
```

- Operaciones con cadenas de texto. Eliminación, sustitución….

variable.sh

```
#!/bin/bash
# Extraer texto
echo "Introduce una frase"
read frase
echo "La frase introducida es: $frase"
echo "Introduce alguna palabra de la frase para eliminarla"
read palabra

echo ${frase//${palabra}}

```

Terminal

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Introduce una frase
Clase de informática de Bachillerato
La frase introducida es: Clase de informática de Bachillerato
Introduce alguna palabra de la frase para eliminarla
de Bachillerato
Clase de informática


-----------------
(program exited with code: 0)
Press return to continue
```

3. CONDICIONALES

Ejecución condicional de una lista de órdenes:

Ejecutar una orden si la primera ejecución es correcta:

cd mi_carpeta && pwd

El operador && indica que se ejecute la segunda orden sólo en el caso de que la ejecución de la primera orden haya sido correcta. En el ejemplo sólo mostrará la ruta si la carpeta existe.

Ejecutar una orden si la primera ejecución es errónea:

cd my_directory || exit

El operador || indica que se ejecute la segunda orden sólo en el caso de que la ejecución de la primera orden haya sido errónea.

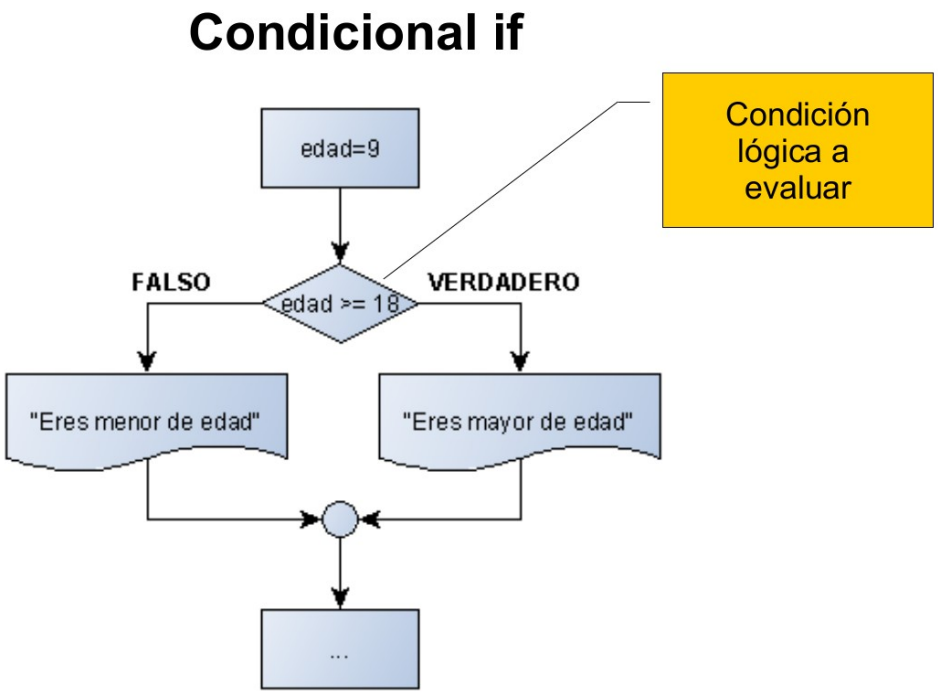
En el ejemplo en el caso de no existir la carpeta lanza un exit para para la ejecución del script. Así evitaremos situaciones desastrosas por utilizar una carpeta que no era la prevista.

Condicional if

```
if [[ $1 -eq 1 ]]; then
        echo “Hola”
else
        echo “Adios”
fi
```

Si el parámetro pasado al script es 1 muestra “Hola” y si no muestra “Adios”.

Así funciona un condicional:



Condicionales lógicas de ficheros

- -e “$fichero” : Devuelve verdadero si el fichero existe
- -d “$fichero” : Devuelve verdadero si el fichero existe y es una carpeta
- -f “$fichero” : Devuelve verdadero si el fichero existe y es un fichero regular
- -h “$fichero” : Devuelve verdadero si el fichero existe y es un enlace
- -r “$fichero” : Devuelve verdadero si el fichero existe y tiene permiso de lectura
- -w “$fichero” : Devuelve verdadero si el fichero existe y tiene permiso de escritura
- -x “$fichero” : Devuelve verdadero si el fichero existe y tiene permiso de ejecución

Condicionales lógicas de cadenas de caracteres

- -z “$cadena” : Devuelve verdadero si la longitud de la cadena es cero.
- -n “$cadena” : Devuelve verdadero si la longitud de la cadena no es cero.
- “$cadena1” = “$cadena2” : Devuelve verdadero si las dos cadenas son iguales.
- “$cadena1” != “$cadena2” : Devuelve verdadero si las dos cadenas son diferentes.

Ejemplo:

```
if [[ $1 = “Micaela” ]]; then
        echo “Hola Micaela”
else
        echo “¿Dónde está Micaela?”
fi
```

Condicionales lógicas de números enteros

- “$entero1” -eq “$entero2” : Devuelve verdadero si los dos enteros son iguales.
- “$entero1” -ne “$entero2” : Devuelve verdadero si los dos enteros son diferentes.
- “$entero1” -gt “$entero2” : Devuelve verdadero si entero1 es mayor que entero2.
- “$entero1” -ge “$entero2” : Devuelve verdadero si entero1 es mayor o igual que entero2.
- “$entero1” -lt “$entero2” : Devuelve verdadero si entero1 es menor que entero2.
- “$entero1” -le “$entero2” : Devuelve verdadero si entero1 es menor o igual que entero2.

Ejemplo:

```
if [[ $edad -ge 18 ]]; then
        echo “Eres mayor de edad”
else
        echo “Eres menor de edad”
fi
```

Expresiones más complejas:

Ejemplo: Si el número es mayor o igual que 200 Y menor o igual que 300 muestra OK

```
if [[ $num -ge 200 ]] && {[ $num -le 300 ]];
then
        echo “OK”
fi
```

Condicional case

Dada una expresión la comparamos con diferentes supuestos y sólo ejecutamos el que se cumpla :

```
case <expresión> in
        <patrón 1>)
                órdenes
                ;;
        <patrón 2>)
                órdenes
                ;;
        *)
                órdenes
                ;;
esac
```

Ejemplo 1:

```
case $1 in
        1)
                echo "Uno"
                ;;
        2)
                echo "Dos"
                ;;
        *)
                echo "Otro número"
                ;;
esac
```

Ejemplo 2:

```
case $1 in
        s|S)
                echo "Has pulsado Sí"
                ;;
        n|N)
                echo "Has pulsado No"
                ;;
        *)
                echo "Desconozco la opción"
                ;;
esac
```

4. BUCLES

Un bucle es una estructura que permite que un grupo de órdenes en Bash se repita varias veces. Es una estructura habitual en cualquier lenguaje de programación y permite realizar tareas repetidas varias veces determinadas por una lista o una condición.

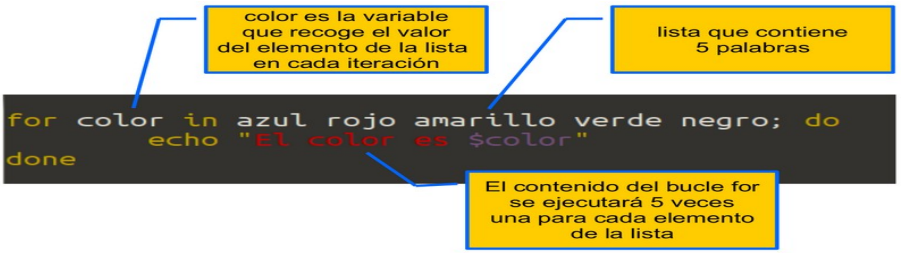
En Bash tenemos dos formas de realizar bucles mediante las sentencias for y while.

Ambas sentencias tienen sus peculiaridades y utilizaremos una u otra en función del número de repeticiones, si está predeterminado antes de empezar, cuál es el caso inicial, cuándo termina el bucle, ...

Incluso en determinadas situaciones se puede resolver el mismo problema utilizando for o while.

Bucle for para recorrer listas o secuencias

Ejemplo 1:



Ejemplo 2:

```
#!/bin/bash

for i in 1 2 3 4 5 6 7 8 9 10; do
        echo $i
done
```

```
#!/bin/bash

for i in {1..10}; do
        echo $i
done
```

```
#!/bin/bash

for i in `seq 1 10`; do
        echo $i
done
```

Las 3 formas son equivalentes para indicar una secuencia del 1 al 10

```
1
2
3
4
5
6
7
8
9
10
```

Ejemplo 3:

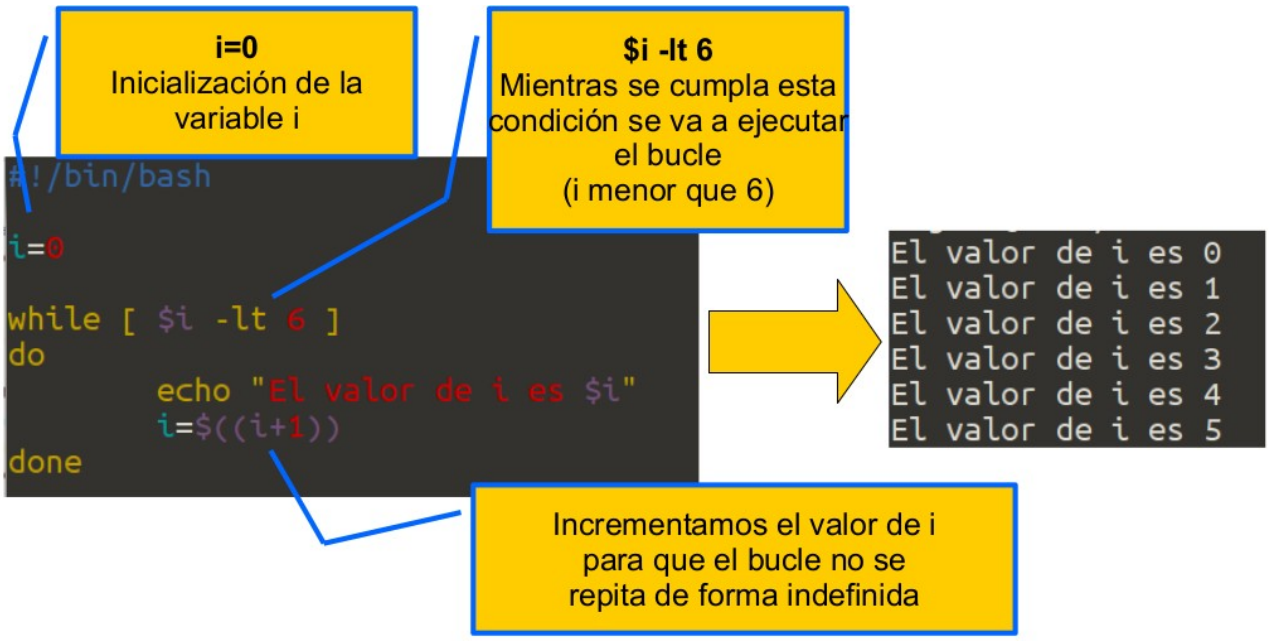
```
#!/bin/bash

for (( i=1 ; i<=10 ; i++ ))
do
        cuadrado=$(( i * i ))
        echo "El cuadrado de $i es $cuadrado"
done
```

```
El cuadrado de 1 es 1
El cuadrado de 2 es 4
El cuadrado de 3 es 9
El cuadrado de 4 es 16
El cuadrado de 5 es 25
El cuadrado de 6 es 36
El cuadrado de 7 es 49
El cuadrado de 8 es 64
El cuadrado de 9 es 81
El cuadrado de 10 es 100
```

while

El bucle while ejecuta iteraciones mientras se cumpla la condición lógica que se indica al principio.



Las condiciones lógicas que se pueden indicar en un bucle while son las mismas que se vieron en el condicional if:

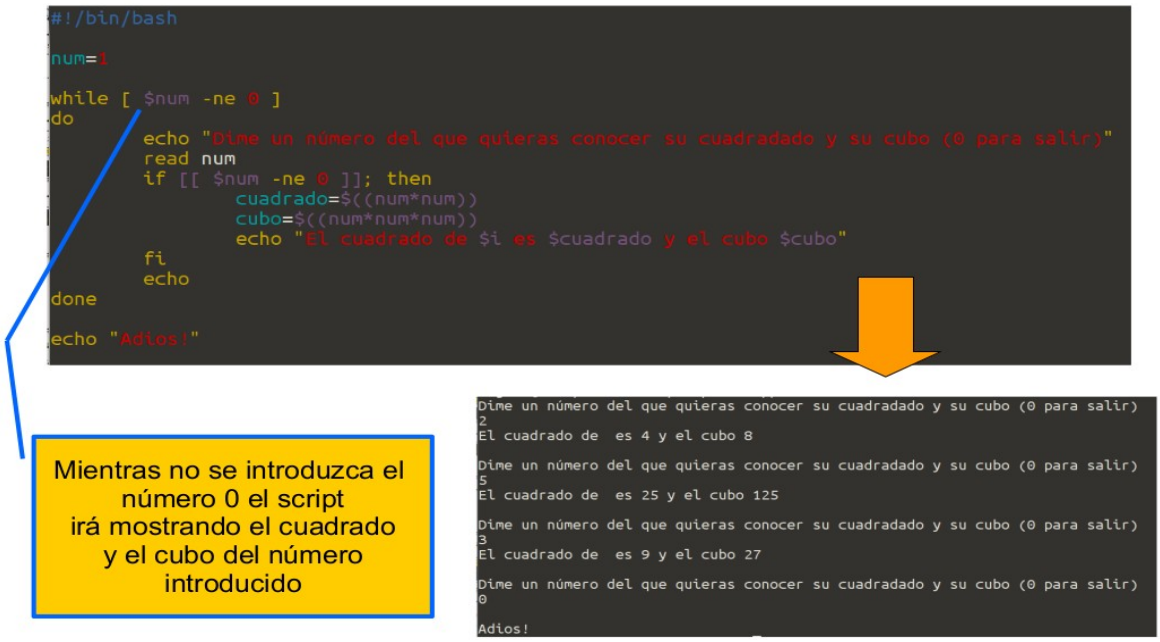
- ficheros

  Ejemplo: -e “/etc/network/interfaces”
- enteros

  Ejemplo: num -ne 6
- cadenas de caracteres

  Ejemplo: $cadena1 = “hola”

Ejemplo:



Break y continue

Tanto en los bucles for como en los bucles while existen dos instrucciones que nos permiten romper el orden lógico de la ejecución del bucle:

- break → Sale de la ejecución del bucle y el bucle ya no se ejecuta más.
- continue → Sale de la ejecución de la iteración y vuelve a comenzar una nueva iteración.

5. DOCUMENTACIÓN

- Miguel Ángel Ferrer. IES Salvador Gadea.

- Micaela Alfonso. IES Salvador Gadea.