

Unidad 5. Shell Scripts



- Un shell script es un **fichero de texto plano que contiene comandos para ser interpretados** por el terminal o shell en orden secuencial.
- Imprescindibles para un **administrador de sistemas Unix/Linux**, son también de mucha utilidad para **usuarios avanzados y programadores**.

- Un sistema operativo Unix/Linux ya contiene decenas de shell scripts que se encargan de **arrancar el sistema y configurar el entorno de usuario.**

- `/etc/rc*.d/`
- `$HOME/.bashrc`
- `$HOME/.profile`
- Etc.

```
rgonzalez@rgonzalez-ubuntu:/etc/rc5.d$ ls
K01apache-htcacheclean  S01cups          S01pulseaudio-enable-autospawn
K01speech-dispatcher    S01cups-browsed  S01rsync
K01sysstat              S01dbus          S01rsyslog
S01acpid                S01gdm3          S01saned
S01anacron              S01grub-common   S01spice-vdagent
S01apache2              S01irqbalance    S01sssd
S01appport              S01kerneloops    S01unattended-upgrades
S01avahi-daemon         S01odoo           S01uuidd
S01bluetooth            S01openvpn        S01whereami
S01console-setup.sh    S01plymouth       S01whoopsie
S01cron                 S01postgresql
```



- En sistemas operativos Windows también existe el equivalente a estos shell scripts: son los llamados **batch scripts**, almacenados en archivos con **extensión .BAT** y que contienen una serie de instrucciones a ejecutar por el **terminal CMD o PowerShell**.

```
com_bkup.bat - Notepad2
File Edit View Settings ?
1 @ECHO OFF
2 REM com_bkup.bat 2/11/12 maw
3 REM Backup com file to c:\111\com
4
5 CLS
6 DIR c:\windows\system32\*.com /W
7 COPY c:\windows\system32\*.com c:\111\com
8 DIR c:\111\com /W
9 ECHO Copy complete
```

- Otros lenguajes de scripting de amplia utilización son:

- **PHP**
- **AWK**
- **Lisp**
- **Python**
- **Perl**
- **Ruby**
- **PowerShell**



Need Scripting?

I will do it for you...



- Los shell script estándar en GNU/Linux (y también MacOS) suelen desarrollarse para **Bash (Bourne-Again Shell)**, si bien con nulos o pocos cambios pueden ejecutarse en otros terminales como Korn Shell o C Shell.
- Para identificarlos con mayor facilidad, suelen ser ficheros con **extensión .sh**, si bien esto es totalmente opcional.

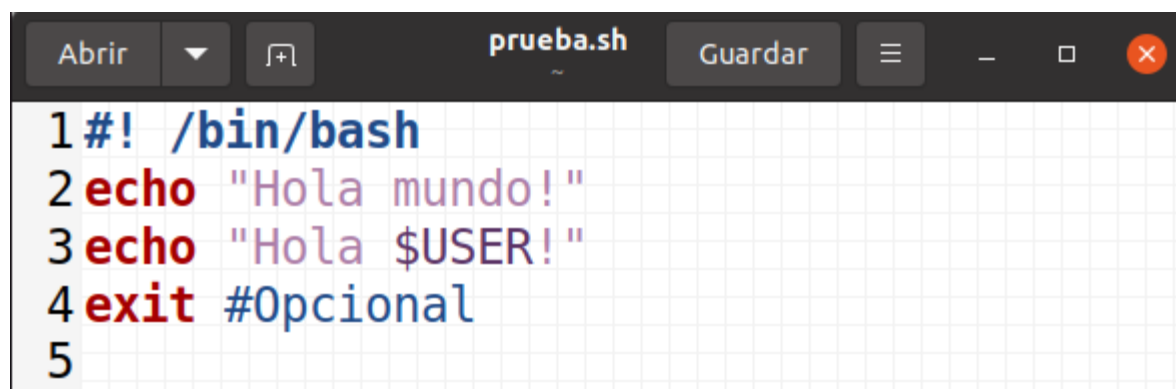
- En `/etc/shells` podemos ver todos los terminales o shells conocidos por nuestro sistema:

```
rgonzalez@rgonzalez-ubuntu:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
/bin/zsh
/usr/bin/zsh
/bin/csh
/usr/bin/csh
/usr/bin/fish
```

- Cuando se ejecuta un shell script en Bash, **este crea un proceso hijo que ejecuta otro Bash**, que será el que lea las líneas del script una a una, interpretándolas y ejecutándolas **como si provinieran del teclado**.
- El proceso padre Bash espera mientras el proceso Bash hijo ejecuta el script hasta el final, momento en el que el control vuelve al proceso padre y este activa el *prompt* nuevamente.

- Con el comando `touch` o directamente con cualquier editor de texto (`nano`, `emacs`, `gedit`, `pluma`...), crearemos un nuevo fichero de texto plano que, opcionalmente, tendrá extensión `.sh`
- Se recomienda que la primera línea del fichero sea **`#!/bin/bash`**
 - **`#!`** Es el llamado **shebang** o **shabang**, que indica que el fichero contiene comandos
 - **`/bin/bash`** sirve para indicar el shell que debe ejecutar estos comandos

- A continuació se mostra un primer script. Como podemos observar, **los comentarios van precedidos por #**



```
1#!/bin/bash
2# Hola mundo!
3echo $USER!
4#Opcional
5
```

- **Cualquier comando que utilicemos en el terminal** es susceptible de ser utilizado en un shell script

- Para ejecutarlo, tenemos dos formas principales:

1. Ejecutándolo con el comando `bash` o con el comando `sh`

```
rgonzalez@rgonzalez-ubuntu:~$ bash prueba.sh  
Hola mundo!  
Hola rgonzalez!
```

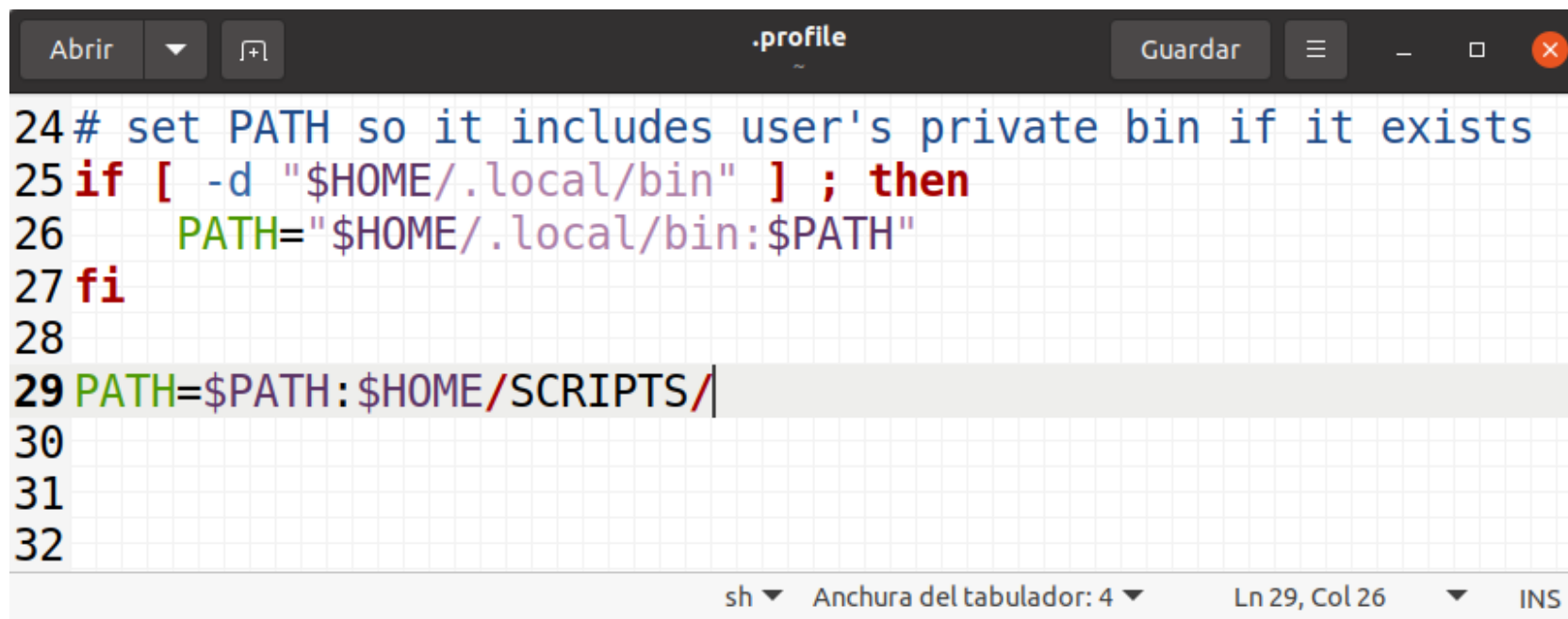
2. Dándole permisos de ejecución e invocándolo con su ruta absoluta o relativa (*forma recomendada*)

```
rgonzalez@rgonzalez-ubuntu:~$ chmod u+x prueba.sh  
rgonzalez@rgonzalez-ubuntu:~$ ./prueba.sh  
Hola mundo!  
Hola rgonzalez!
```

- Podemos hacer que no sea necesario especificar la ruta absoluta o relativa en la que se encuentra nuestro script e invocarlo simplemente por su nombre.
- Para ello, o bien ponemos nuestro script en un directorio que ya se encuentre en la **variable de entorno \$PATH** (p. ej. `/usr/bin` o `$HOME/bin`), o bien modificamos `$PATH` para que incluya el directorio en el que se encuentra nuestro script.

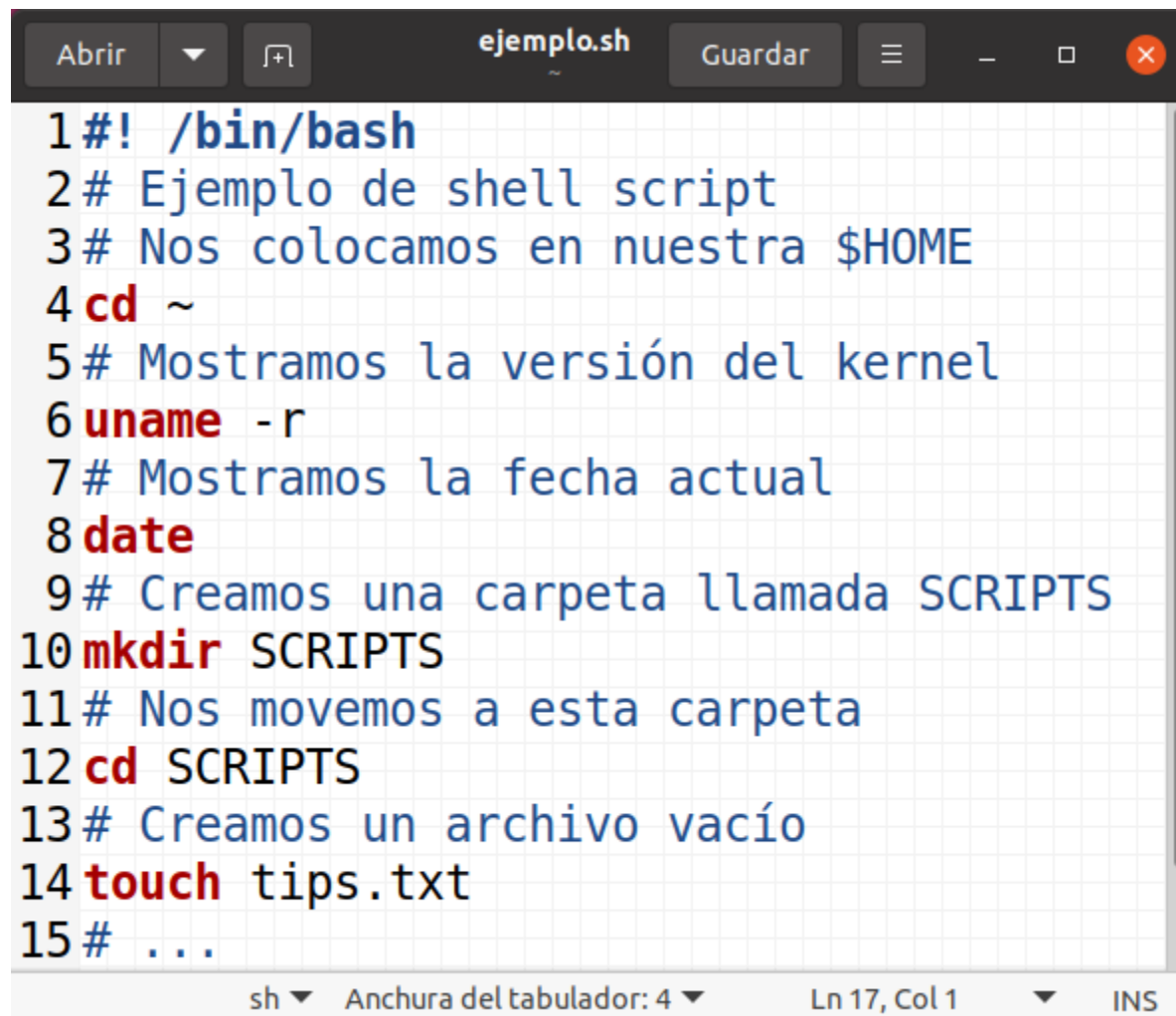
- Para añadir un directorio a `$PATH` de forma que el cambio sea persistente, **modificamos alguno de los shell scripts que se ejecutan al iniciar sesión y configuran el entorno de usuario**. Por ejemplo:
 - `/etc/profile`
 - `$HOME/.bashrc`
 - `$HOME/.profile`

- En uno de estos scripts, modificamos el valor de `$PATH` y recargamos el script con el comando `source` (o bien reiniciamos la sesión o el sistema).



```
24 # set PATH so it includes user's private bin if it exists
25 if [ -d "$HOME/.local/bin" ] ; then
26     PATH="$HOME/.local/bin:$PATH"
27 fi
28
29 PATH=$PATH:$HOME/SCRIPTS/
30
31
32
```

sh Anchura del tabulador: 4 Ln 29, Col 26 INS



```
1#!/bin/bash
2# Ejemplo de shell script
3# Nos colocamos en nuestra $HOME
4cd ~
5# Mostramos la versión del kernel
6uname -r
7# Mostramos la fecha actual
8date
9# Creamos una carpeta llamada SCRIPTS
10mkdir SCRIPTS
11# Nos movemos a esta carpeta
12cd SCRIPTS
13# Creamos un archivo vacío
14touch tips.txt
15# ...
```

sh Anchura del tabulador: 4 Ln 17, Col 1 INS

- Los shell scripts **no contienen solamente** secuencias de comandos, sino que disponen de las **herramientas típicas de los lenguajes de programación** como:
 - Variables
 - Operadores aritméticos y de cadenas de texto
 - Estructuras condicionales
 - Estructuras de repetición
 - Etc.

- Las **variables** nos permiten guardar valores numéricos o cadenas de caracteres en memoria, pudiendo acceder a esta mediante el nombre que le asignemos
- **Declaración y asignación:** la declaración de la variable se efectúa en el momento en el que le asignamos un valor. Esta asignación se efectúa mediante el signo "=":

```
var1=5
```

- **Acceso al valor de una variable:** El acceso a una variable es tan sencillo como poner el signo \$ antes del nombre.

```
echo $var1
```

- **Borrado una variable**

```
unset var1
```

- Para operar con números y hacer operaciones con ellos, disponemos del comando `typeset`, al que le pasaremos el parámetro `-i` (integer) y la variable que queremos convertir a tipo numérico.

```
1 #! /bin/bash
2 typeset -i suma
3 suma=4+6
4 echo $suma
```

- Más adelante se presentan otras maneras de realizar operaciones numéricas.

- El comando `read` nos permite **crear una nueva variable con el valor introducido por la entrada estándar** (teclado)

```
1#!/bin/bash
2echo "Introduce tu ciudad:"
3read ciudad
4echo "Tu ciudad es: "$ciudad
```

- Podemos **solicitar un valor, leerlo y almacenarlo en una variable en un solo comando**, mediante `read -p`

```
1#!/bin/bash
2read -p "Introduce tu ciudad: " ciudad
3echo "Tu ciudad es: "$ciudad
```

- En lugar de pedirle valores al usuario dentro de nuestro script, podemos hacer que el script reciba información en el momento de su ejecución, es decir, **que se le pasen parámetros como si de un comando más se tratase.**
- Estos parámetros, llamados **parámetros de posición**, son argumentos numerados por la posición que ocupan, comenzando por la número 1, cuyo valor estará disponible en \$1

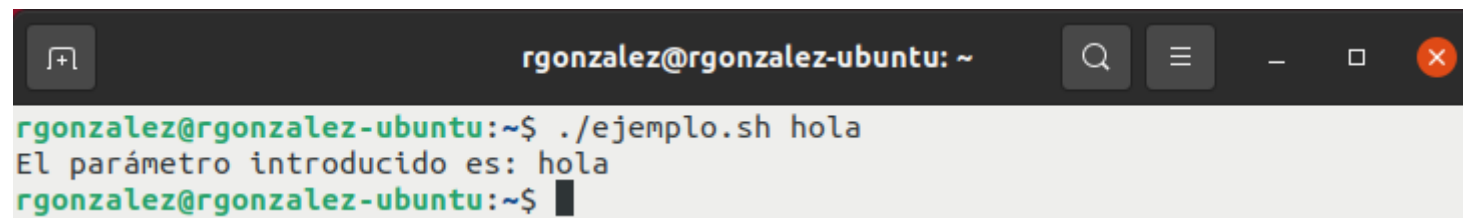
- Por ejemplo, podríamos ejecutar nuestro script en el terminal como:

```
./ejemplo.sh hola
```

- Y nuestro script recibirá automáticamente el valor “hola” en \$1



```
ejemplo.sh
Abrir  Guardar  -  □  ×
1 #! /bin/bash
2 echo "El parámetro introducido es: "$1
3
```



```
rgonzalez@rgonzalez-ubuntu: ~
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh hola
El parámetro introducido es: hola
rgonzalez@rgonzalez-ubuntu:~$
```

- Para interactuar con los parámetros de posición recibidos, en un Shell script tenemos a nuestra disposición las siguientes variables:
- **\$1 ... \$9** : Hasta 9 parámetros o argumentos pasados al script
- **NOTA:** Si usamos las llaves “{}” para nombrar las variables, el límite de parámetros se podría extender: `${10}`, `${11}`, `${12}`... pero rara vez necesitaremos tantos argumentos

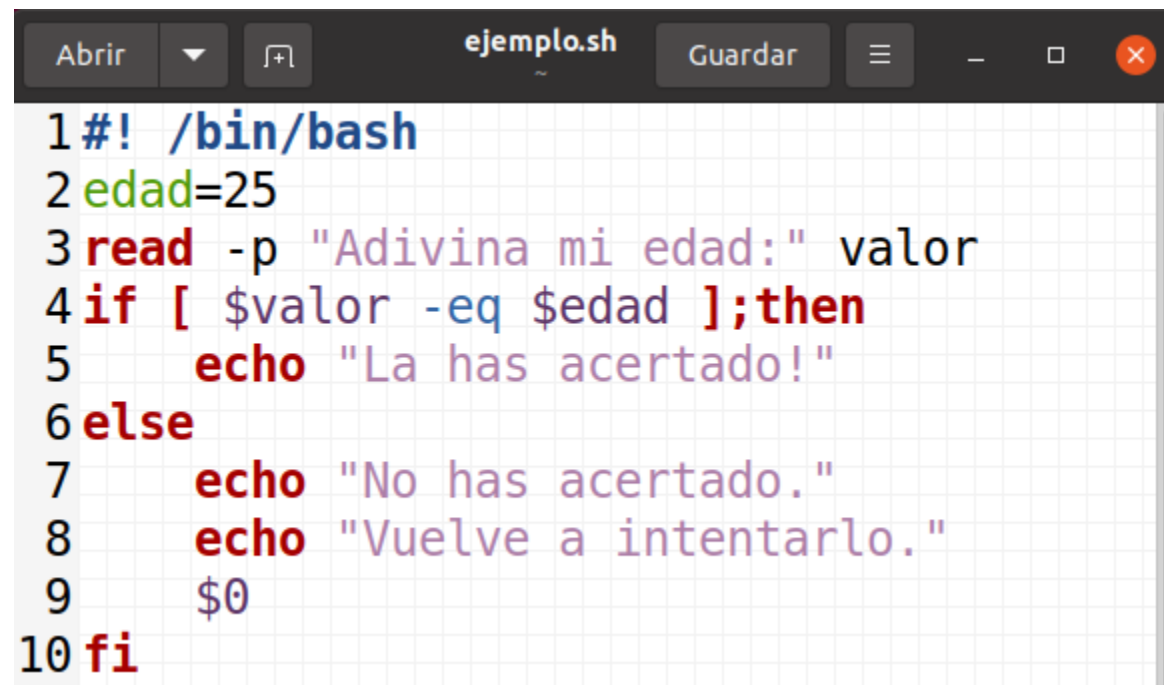
- Además, relacionado con estos parámetros de posición, disponemos de otras variables disponibles automáticamente:
- **\$0** : Nombre del script que se está ejecutando
- **\$#** : Número de argumentos que se han pasado al script
- **\$*** : Todos los argumentos pasados al script


```
ejemplo.sh
1#!/bin/bash
2echo "El script se llama: $0"
3echo "Has introducido $# parámetros"
4echo "Los parámetros introducidos son:"
5echo $*
```

```
rgonzalez@rgonzalez-ubuntu: ~
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh hola qué tal
El script se llama: ./ejemplo.sh
Has introducido 3 parámetros
Los parámetros introducidos son:
hola qué tal
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh "hola qué tal" "buenos días"
El script se llama: ./ejemplo.sh
Has introducido 2 parámetros
Los parámetros introducidos son:
hola qué tal buenos días
```

- Funciona como en otros lenguajes de programación, pero hay que tener ojo con la sintaxis (nótese el espacio después de [y antes de])

```
if [ CONDICION ];then
    # Comandos a ejecutar
    # si se cumple
    # la condición
else
    # Comandos a ejecutar
    # en caso contrario
fi
```



```
1#!/bin/bash
2edad=25
3read -p "Adivina mi edad:" valor
4if [ $valor -eq $edad ];then
5    echo "La has acertado!"
6else
7    echo "No has acertado."
8    echo "Vuelve a intentarlo."
9    $0
10fi
```

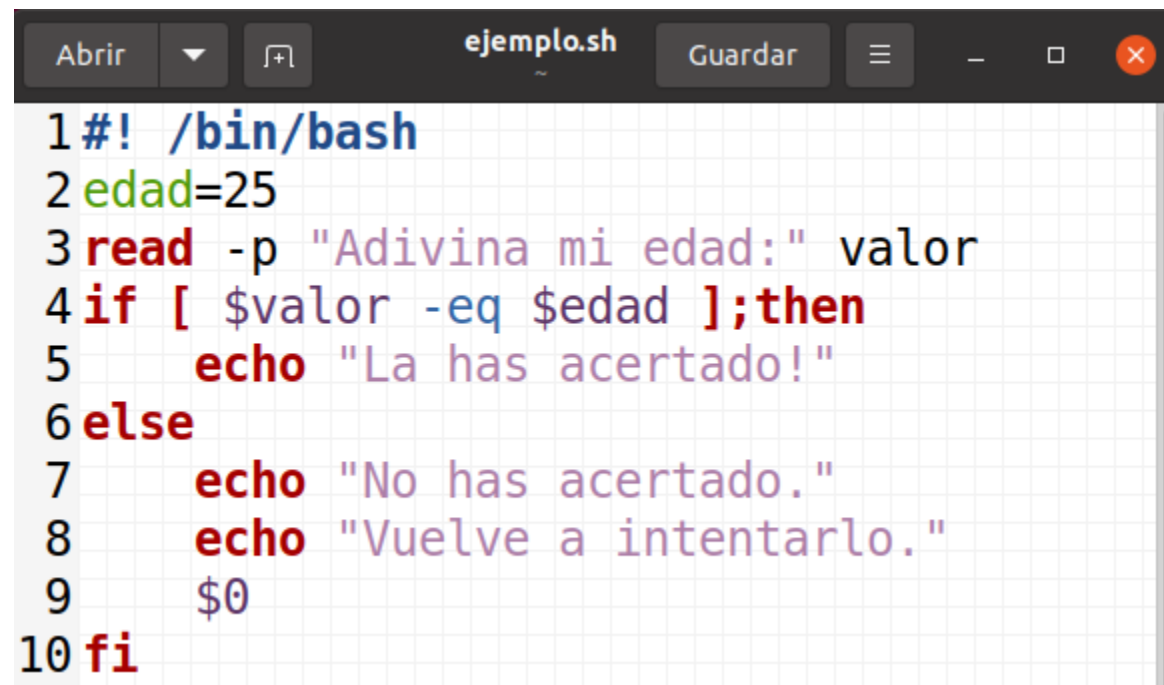
- Otro ejemplo:

```
ejemplo.sh
1#!/bin/bash
2if [ $# -eq 2 ];then
3    if [ $1 = $2 ]; then
4        echo "Son iguales."
5    else
6        echo "Son diferentes."
7    fi
8    exit
9fi
10echo "No has introducido 2 parámetros"
```

```
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh
No has introducido 2 parámetros
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh hola
No has introducido 2 parámetros
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh hola hola
Son iguales.
rgonzalez@rgonzalez-ubuntu:~$ ./ejemplo.sh hola adiós
Son diferentes.
```

- La estructura **if-then-else** también admite **elif** para alternar entre condiciones de manera compacta

```
if [ CONDICION ];then
    # Comandos a ejecutar
    # si se cumple
    # la condición
else
    # Comandos a ejecutar
    # en caso contrario
fi
```



```
1#!/bin/bash
2edad=25
3read -p "Adivina mi edad:" valor
4if [ $valor -eq $edad ];then
5    echo "La has acertado!"
6else
7    echo "No has acertado."
8    echo "Vuelve a intentarlo."
9    $0
10fi
```

- Al comparar dos **valores enteros** en una [**CONDICIÓN**], utilizaremos los flags:

-**eq** (is equal to), -**gt** (is greater than), -**ge** (is greater than or equal to), -**lt** (is less than), -**le** (is less than or equal to), -**ne** (is not equal to)

```
10 if [ $primer -gt $segundo ]; then
11     echo $primer " es mayor que " $segundo
12 else
13     if [ $primer -lt $segundo ]; then
14         echo $primer " es menor que " $segundo
15     else
16         echo "Los valores son iguales"
17     fi
18 fi
```

- Podemos comparar **cadenas de caracteres (strings)** mediante los operadores **=** y **!=**

```
1#!/bin/bash
2cadena1=$1
3cadena2=$2
4
5if [ $cadena1 = $cadena2 ]; then
6echo "Las cadenas de texto son iguales."
7else
8echo "Las cadenas de texto son distintas."
9fi
```

- Para **ejecutar un COMANDO** dentro de nuestro Shell script, no hay que hacer nada especial, tan solo ponerlo como si estuviéramos en el terminal.
- En cambio, si queremos almacenar su resultado, deberemos utilizar la nomenclatura **\$ (COMANDO)** o bien **`COMANDO`** (nótese la tilde invertida)

```
1#!/bin/bash
2# Ambas formas son equivalentes
3num_lineas=$(cat archivo.txt | wc -l)
4num_lineas2=`cat archivo.txt | wc -l`
5# Y por tanto los resultados deberían coincidir
6if [ $num_lineas -eq $num_lineas2 ];then
7    echo "El archivo contiene $num_lineas líneas"
8fi
```

- Las siguientes formas a la hora de realizar **operaciones aritméticas básicas con valores enteros** son equivalentes:

```
1#!/bin/bash
2a=50
3b=35
4
5# INCORRECTO!
6# c=a+b
7# Correcto, con let
8let c1=$a+$b
9# Correcto, con $() y expr
10c2=$(expr $a + $b)
11# Correcto, con ``y expr
12c3=`expr $a + $b`
13# Correcto, con $(( ))
14c4=$(( $a+$b ))
15
16echo $c1 $c2 $c3 $c4
```


- Gracias al comando **bc** podemos hacer **cualquier tipo de cálculo, ya sea con enteros o con decimales**, incluso especificando el número de decimales deseado mediante la variable **scale**:

```
1#!/bin/bash
2read -p "Introduce un número entero o decimal: " num1
3read -p "Introduce otro número entero o decimal: " num2
4
5echo "La suma es:" $(echo "$num1+$num2" | bc)
6echo "La resta es:" $(echo "$num1-$num2" | bc)
7echo "La multiplicación es:" $(echo "$num1*$num2" | bc)
8echo "La división es:" $(echo "scale=1;$num1/$num2" | bc)
```

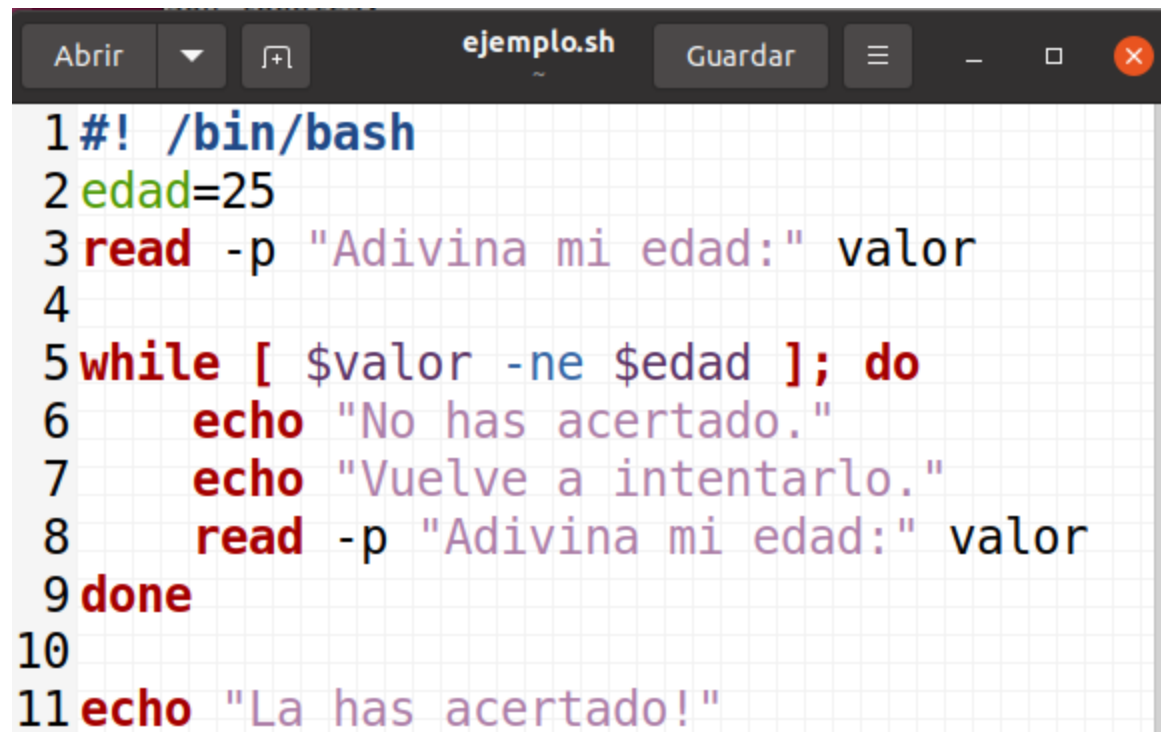


- Mediante un bloque **case-esac** podemos seleccionar entre varias alternativas de manera sencilla. Es equivalente al **switch** de otros lenguajes de programación.

```
1#!/bin/bash
2clear
3echo "Bienvenido a mi programa"
4echo "====="
5echo "Elige una opción:"
6echo "1) Ejecutar firefox"
7echo "2) Listar los ficheros del directorio actual"
8echo "3) Ver el manual de gcc"
9echo "4) Salir"
10read respuesta
11echo "La opción elegida es: " $respuesta
12case $respuesta in
131) firefox & ;;
142) ls -l ;;
153) man gcc ;;
164) exit ;;
17*) echo "No has seleccionado una opción correcta";;
18esac
```

- Típico bucle condicional en el que ejecutaremos una serie de instrucciones mientras se cumpla la condición

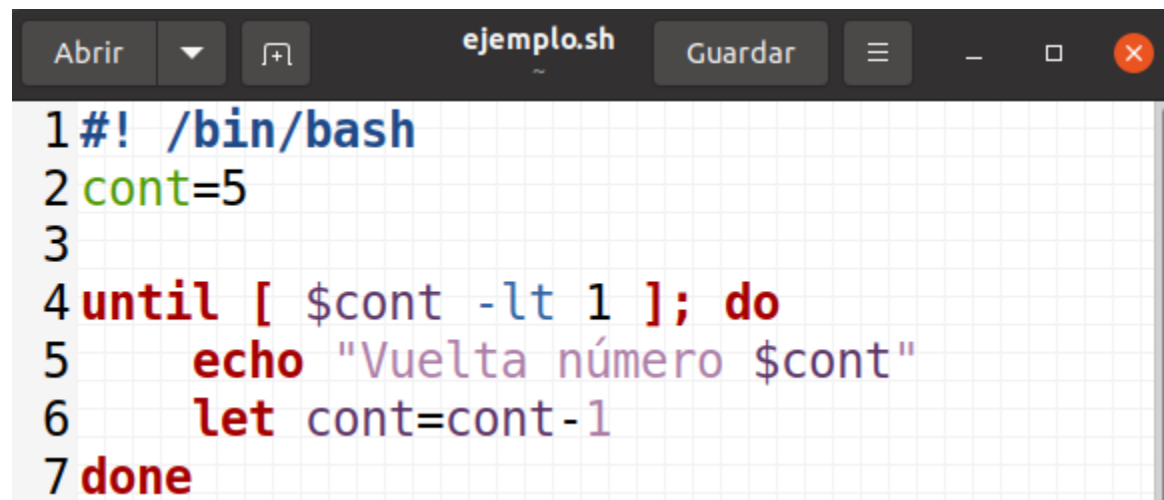
```
while [ CONDICION ];do
    # Comandos a ejecutar
    # MIENTRAS se cumpla
    # la condición
done
```



```
Abrir  ejemplo.sh  Guardar  -  □  ×
1 #! /bin/bash
2 edad=25
3 read -p "Adivina mi edad:" valor
4
5 while [ $valor -ne $edad ]; do
6     echo "No has acertado."
7     echo "Vuelve a intentarlo."
8     read -p "Adivina mi edad:" valor
9 done
10
11 echo "La has acertado!"
```

- Bucle condicional en el que **ejecutaremos una serie de instrucciones hasta que se cumpla la condición**

```
until [ CONDICION ];do
    # Comandos a ejecutar
    # HASTA QUE se cumpla
    # la condición
done
```

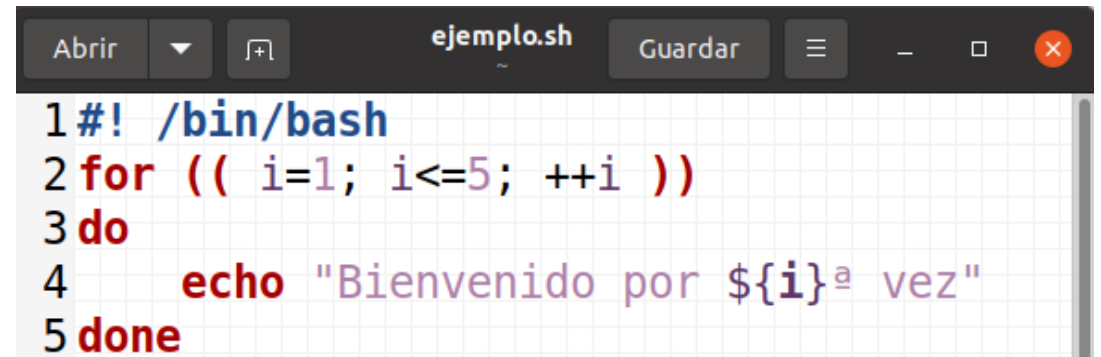


```
1#!/bin/bash
2cont=5
3
4until [ $cont -lt 1 ]; do
5    echo "Vuelta número $cont"
6    let cont=cont-1
7done
```



- Bucle condicional que, además de la condición, incluye la inicialización y el paso (normalmente un incremento o decremento)
- Su sintaxis es algo diferente a la de los otros bucles

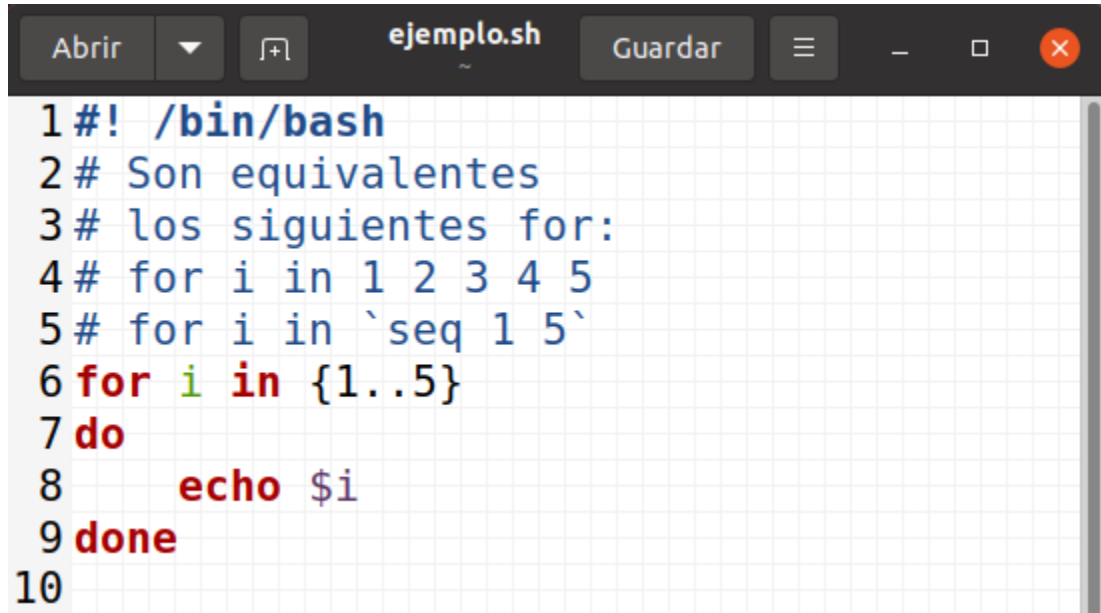
```
for (( INICIALIZACION; CONDICION; PASO ))  
do  
    # INICIALIZACION en 1ª ejecución  
    # Comandos a ejecutar  
    # MIENTRAS se cumpla  
    # la CONDICION  
    # Tras ellos, se ejecuta el PASO  
done
```



```
ejemplo.sh  
1#!/bin/bash  
2for (( i=1; i<=5; ++i ))  
3do  
4    echo "Bienvenido por ${i}ª vez"  
5done
```

- Bucle incondicional, equivalente al `foreach` de otros lenguajes de programación, que sirve para recorrer una serie de valores

```
for VARIABLE in SECUENCIAL
do
    # Hacer algo con VARIABLE
    # en cada posible valor
    # del SECUENCIAL
done
```



```
1#!/bin/bash
2# Son equivalentes
3# los siguientes for:
4# for i in 1 2 3 4 5
5# for i in `seq 1 5`
6for i in {1..5}
7do
8    echo $i
9done
10
```



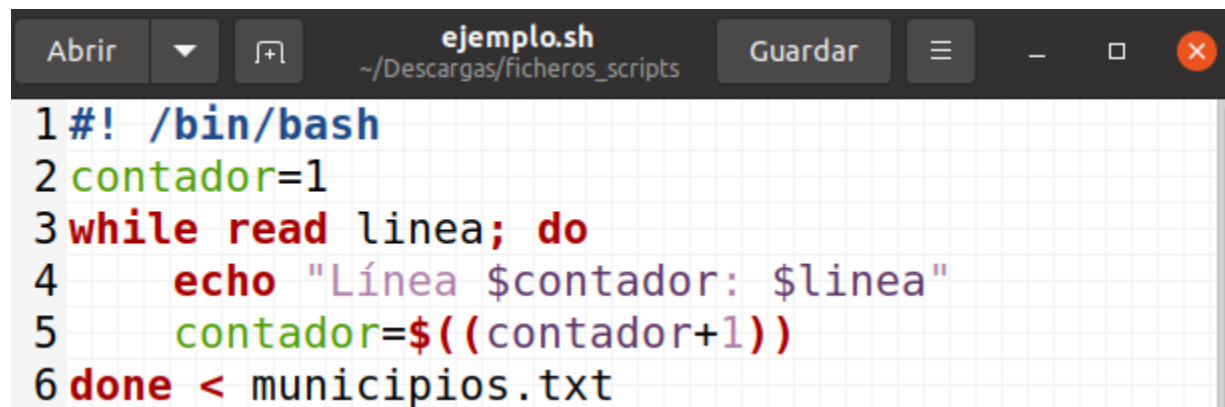
- Es posible recorrer el secuencial en pasos diferentes de 1, que es el paso por defecto, mediante ``seq INICIO PASO FIN`` o `{ INICIO..FIN..PASO }`

```
Abrir  ejemplo.sh  Guardar  -  x
1#! /bin/bash
2# Múltiplos de 3 hasta el 60
3for i in `seq 3 3 60`
4do
5    echo $i
6done
```

```
Abrir  ejemplo.sh  Guardar  -  x
1#! /bin/bash
2# Múltiplos de 3 hasta el 60
3for i in {3..60..3}
4do
5    echo $i
6done
```

- Hay diversas formas de **procesar (recorrer) un fichero** en shell script
- Podemos **leerlo línea a línea** proporcionando el fichero como entrada del bucle `while` y utilizando `read` en su condición

```
while read LINEA; do
    # hacer algo con LINEA
done < FICHERO
```



```
ejemplo.sh
~/Descargas/ficheros_scripts

1#!/bin/bash
2contador=1
3while read linea; do
4    echo "Línea $contador: $linea"
5    contador=$((contador+1))
6done < municipios.txt
```


- Al ejecutar nuestro script, **el terminal nos mostrará los errores de sintaxis y de ejecución**, si los hubiere, de forma descriptiva
- También es habitual **depurar nuestro script insertando instrucciones `echo` o `printf`** con la información y el valor de las variables que queremos comprobar.

- Adicionalmente, podemos **ejecutar el script en modo depuración** con el comando `bash -x script.sh` o poniendo este parámetro en la primera línea: `#! /bin/bash -x`
- También podemos **depurar una sección de código** incluyendo:

```
set -x # activa debugging desde aquí  
código a depurar  
set +x # para el debugging
```

- Las expresiones de condición mediante corchetes [CONDICIÓN], en estructuras **if-then-else**, **while**, **until**, etc. se comprueban en realidad mediante el comando **test** que, por tanto, podemos usar de manera totalmente equivalente:

```
10 if [ $primer -gt $segundo ]; then
11     echo $primer " es mayor que " $segundo
12 else
13     if [ $primer -lt $segundo ]; then
14         echo $primer " es menor que " $segundo
15     else
16         echo "Los valores son iguales"
17     fi
18 fi
```

```
10 if test $primer -gt $segundo; then
11     echo $primer " es mayor que " $segundo
12 else
13     if test $primer -lt $segundo; then
14         echo $primer " es menor que " $segundo
15     else
16         echo "Los valores son iguales"
17     fi
18 fi
```

- El comando **test** no solo permite comparar valores enteros y cadenas, sino también comprobar la existencia de ficheros, comprobar su tipo, verificar el propietario, etc. Ver el manual de shell para más información: `$ man test`

```
-d FICHERO
    El FICHERO existe y es un directorio

-e FICHERO
    El FICHERO existe

-f FICHERO
    El FICHERO existe y es un fichero regular

-g FICHERO
    El FICHERO existe y tiene cambio-de-ID-de-grupo

-G FICHERO
    El FICHERO existe y su propietario es el ID
    efectivo de grupo
```

```
Manual page test(1) line 87 (press h for help or q to quit)
```

- Podemos indicar `if [CONDICIÓN]; then` en la misma línea o bien omitir el `;` y poner `then` en la línea siguiente. Esto también se aplica a `while-do`, `until-do`, etc.

```
1#!/bin/bash
2
3read -p "Introduce edad: " edad
4
5if [ $edad -ge 18 ];then
6    echo "Eres mayor de edad"
7else
8    echo "Eres menor de edad"
9fi
```

```
1#!/bin/bash
2
3read -p "Introduce edad: " edad
4
5if [ $edad -ge 18 ]
6then
7    echo "Eres mayor de edad"
8else
9    echo "Eres menor de edad"
10fi
```

- Podemos emular un bloque **if-then-else** mediante los operadores **&&** (and) y **||** (or), de la siguiente manera:

```
1#!/bin/bash
2
3read -p "Introduce edad: " edad
4
5test $edad -ge 18 && echo "Eres mayor de edad" || echo "Eres menor de edad"
6
```