

# UT9.2 Programación Shell en Ubuntu

Administración de Sistemas Operativos  
2º ASIR

# PROGRAMACIÓN SHELL EN UBUNTU.

## INTRODUCCIÓN

El shell es más que un interprete de comandos, es un lenguaje de programación completo, con instrucciones condicionales, asignación, ciclos y funciones.

Los programas de shell son interpretados, por lo que no necesitan ser compilados y linkados para ejecutarlos.

Aunque existen diferentes variantes de éste lenguaje:

- ❑ **sh** o **bsh**: Shell limitada, utilizada desde los primeros UNIX.
- ❑ **csh** o **tcsh**: Utilizada por Unix BSD y basada en el lenguaje C.
- ❑ **ksh**: Surje para unir lo mejor de la bourne shell y la c-shell.
- ❑ **bash.**: Es la que viene en la mayoría de las distribuciones de Linux. Se trata de la Bourne Shell mejorada y posee toda la funcionalidad del sh con características avanzadas de C Shell. La shell bash es la que utilizaremos para crear nuestros scripts.

---

# PROGRAMACIÓN SHELL. UBUNTU VARIABLES

# PROGRAMACIÓN SHELL EN UBUNTU.

## VARIABLES

- Las variables permiten guardar valores (numéricos, alfanuméricos, lógicos, arrays o funciones), para que luego puedan ser utilizadas en operaciones, funciones o expresiones condicionales.
- El nombre de una variable puede contener solo letras (de a a z o de A a Z), números (0 a 9) o el guión bajo “ \_ ”.
- El nombre de una variable **solo** puede **comenzar** con una **letra** o un **guión bajo**, pero **no** puede comenzar por un **número**. Tampoco son válidos los **caracteres** **!**, **\*** o **-** (porque ya tienen un significado propio en el shell). Serían válidos los siguientes nombres de variables.

# PROGRAMACIÓN SHELL EN UBUNTU.

## Ejemplos de VARIABLES

`_TOTAL`  
`resultado_final`  
`fichero1`  
`_data`  
`nombre_de_archivo`  
`_2total`

# VARIABLES DEL SISTEMA

VARIABLE	DESCRIPCION
\$0	Nombre del Shell-Script que se está ejecutando.
\$n	Parámetro o argumento pasado al Shell-Script en la posición n, n=1,2,...
\$PS1	Prompt
\$#	Número de argumentos.
\$*	Lista de todos los argumentos.
\$?	Salida del último proceso ejecutado.
\$\$	Número de identificación del proceso (PID)
\$_	Número del último proceso invocado por la shell

# PROGRAMACIÓN SHELL EN UBUNTU.

## Acceso al valor de las VARIABLES

Si queremos acceder al valor que contiene la variable tendremos que utilizar el símbolo \$ delante de dicha variable. Veamos algunos ejemplos:

### Ejemplo 1

```
#!/bin/bash  
operando=2  
echo "operando vale" $operando
```

En la primera línea del anterior ejemplo aparece: **#!/bin/bash** para indicar dónde se ubica el shell.

**operando=2** define la variable **operando** y le asigna el número 2.

**echo "operando vale" \$operando** muestra por pantalla (salida estándar): **operando vale 2**

#### Notas:

**/etc/shells** contiene la lista de shells instaladas en nuestra distribución.

**echo \$SHELL** nos devuelve la shell que estamos utilizando.

# PROGRAMACIÓN SHELL EN UBUNTU.

## Acceso al valor de las VARIABLES

Ejemplo 2. Asignando el valor de una variable a otra.

```
Prompt > X=hola  
Prompt > Y="$X mundo"  
Prompt > echo $Y
```

**Resultado:** Se mostrará, por la salida estándar (pantalla) la frase:  
hola mundo



# VARIABLES DE ENTORNO

Existen dos áreas de memoria en las shells para almacenar variables, el Área local de datos y el Entorno.

Cuando asignamos un valor a una variable, es local, es decir, es conocida por esa shell, pero si se abre otra shell a partir de la que estamos, estas nuevas 'subshells' desconocen el valor de las variables que hemos asignado anteriormente.

Para ver todas las variables de entorno definidas usamos el comando: **printenv**

Éstas son algunas de las variables que más se usan:

**HOME:** ruta de nuestro directorio personal.

**USER:** nombre de usuario asignado.

**SHELL:** ruta al intérprete de órdenes que se ejecuta por defecto.

**HOSTNAME:** nombre asignado al equipo.

**PATH:** rutas en las que el intérprete busca las órdenes a ejecutar cuando no especificamos donde están.

Para hacer que una variable se almacene en el área de Entorno, se utiliza el siguiente comando: **export**

**Ejemplo:** **export fruta=manzana**

**NOTA:** Probablemente sea necesario incluir el comando anterior en el archivo **/etc/profile**. Luego habría que reiniciar sesión o bien: **\$ source /etc/profile**

# PROGRAMACIÓN SHELL EN UBUNTU.

## Comillas simples y dobles

Si lo que entrecomillamos es simplemente texto el resultado es el mismo:

```
mkdir "esto es una prueba" //Se crea un directorio con el nombre esto es una prueba
```

```
mkdir 'esto es una prueba' //Se crea un directorio con el nombre esto es una prueba
```

### Expansión de variables

La diferencia entre el entrecomillado sencillo y doble toma su importancia cuando tratamos con variables en la línea de comando, veamos algunos ejemplos:

```
prueba="Esto es una prueba"
```

```
echo $prueba //El terminal te devolverá: Esto es una prueba
```

```
echo "$prueba" //Devolverá: Esto es una prueba (con las dobles comillas se sustituye la variable por su contenido)
```

```
echo '$prueba' // Devolverá: $prueba (con las comillas simples se muestra el contenido tal cual, no interpreta los caracteres especiales)
```

```
echo `pwd` // Las comillas simples inversas hacen que se ejecute la orden que contienen.
```

```
$ var=5 ; echo "No aparece el valor \ $var" //Si ponemos el carácter especial \ delante de la variable, se mostrará por pantalla el mensaje tal cual: No aparece el valor $var.
```

# PROGRAMACIÓN SHELL EN UBUNTU.

## Comillas simples y dobles. Ejemplos

```
#!/bin/bash
VAR=auto
echo "Me compré un $VAR"      Imprimirá Me compré un auto
echo 'Me compré un $VAR'      Imprimirá Me compré un $VAR
echo "Me compré un \$VAR"      Imprimirá Me compré un $VAR
```

Guión 1

```
#!/bin/bash
echo Hola mundo
```

Cuando se corre este guión se imprimirá a la pantalla Hola mundo

Guión 2 Lo mismo usando una variable

```
#!/bin/bash
VARIABLE=Hola mundo
echo "$VARIABLE"
```

Nótese la variable entre comillas dobles para que imprima todo el texto.

Guión 3 Cuando se usan más de una variable

```
#!/bin/bash
VARIABLE=Hola
SALUDO=mundo
echo "$VARIABLE" "$SALUDO"
```

En los tres casos se imprimirá a la pantalla Hola mundo

# PROGRAMACIÓN SHELL EN UBUNTU. ARRAYS

Es una variable que contiene una serie de elementos del mismo o distinto tipo, y almacenados de forma continua.

## Ejemplo 3

```
#!/bin/bash
# crear arrays sencillos de una dimensión
equipo[0]="España"
equipo[1]="Suecia"
equipo[2]="Italia"
equipo[3]="Alemania"
echo "El Grupo A está formado por: " ${equipo[0]} ${equipo[1]} ${equipo[2]} ${equipo[3]}
```

**Resultado:** El Grupo A está formado por: España Suecia Italia Alemania

## NOTAS:

Otra forma de definir el array anterior sería: `equipo=("España" "Suecia" "Italia" "Alemania" )`

`echo ${equipo[@]}` // mostrará todos los elementos del array

`echo ${#equipo[@]}` // mostrará el número de elementos del array

**Puede que cuando lancemos un script con sh nos de errores, entonces lo ejecutaremos con bash**

# PROGRAMACIÓN SHELL CONSTANTES

# PROGRAMACIÓN SHELL EN UBUNTU. CONSTANTES

Para definir una constante en el Shell del Bash se utiliza el comando `declare` con el parámetro `-r` (read only).

Sintaxis: **`declare -r nombre_constante= valor_de_constante`**

## Ejemplo

```
#!/bin/bash  
# Declarando constante  
declare -r PI=3.141592
```

**NOTA:** Si a una constante, tratamos de asignarle otro valor, recibiremos el siguiente error:

```
alumno@ClienteUbuntu00:~$ declare -r PI=3.141596  
alumno@ClienteUbuntu00:~$ PI=4  
bash: PI: variable de sólo lectura  
alumno@ClienteUbuntu00:~$
```

---

# EJECUCIÓN DE SCRIPTS EN UBUNTU

# EJECUCIÓN DE SCRIPTS (I)

**Script. Definición.-** Es un programa normalmente almacenado en un archivo de texto plano que al ejecutarse permite automatizar una serie de tareas.

Para ejecutar un script en el shell de Ubuntu (Bash) podemos usar varios métodos, aquí se muestran tres:

```
$ bash <nombrescript.sh>
```

```
$ sh <nombrescript.sh>
```

```
$ ./<nombrescript.sh> //Si el script está en el directorio actual
```

```
$ /ruta_script/nombrescript.sh //Indicando la ruta absoluta
```

**NOTA:** En alguna de las opciones anteriores tendremos que dar permisos de ejecución al script previamente. Es decir, teclear:

```
$sudo chmod +x <nombrescript.sh>
```



# EJECUCIÓN DE SCRIPTS (II)

Por ejemplo si abrimos un editor de texto en Ubuntu (**gedit**, **nano**, **vi**, etc) y copiamos el texto íntegro del **Ejemplo 1** visto con anterioridad y lo guardamos con el nombre de **ejemplo1.sh**, para ejecutarlo desde el shell teclearíamos alguna de las siguientes líneas:

```
# bash ejemplo1.sh
```

```
# sh ejemplo1.sh
```

```
# ./ejemplo1.sh //Si el script está en el directorio actual
```

```
# /home/juan/scripts/ejemplo1.sh //Si está en la ruta mostrada
```

**NOTA:** Antes de ejecutar el script deberíamos darle permisos de ejecución:  
**sudo chmod +x ejemplo1.sh**

# PARÁMETROS EN LOS SCRIPTS

Un shell-script soporta **argumentos** o **parámetros** de entrada, que se referencian mediante las variables especiales \$i con i=1,2, n. Es decir, si lanzamos el siguiente script:

```
# myscript.sh 4 2 pepe
```

Los parámetros de dicho script serían:

**\$1** contendrá el valor 4

**\$2** contendrá el valor 2

**\$3** contendrá el valor pepe

Dichas variables (\$1,\$2,\$3) se utilizarán dentro del script cuyo nombre también se almacena en una variable del sistema, en este caso en **\$0**.

---

# ENTRADA Y SALIDA DE DATOS

# ENTRADA Y SALIDA DE DATOS

Hay que decir que la entrada de datos por defecto es el **TECLADO** y la salida de datos es a través de la **PANTALLA** (también por defecto). Aunque, como veremos más adelante, podremos redirigir la entrada y salida de datos.

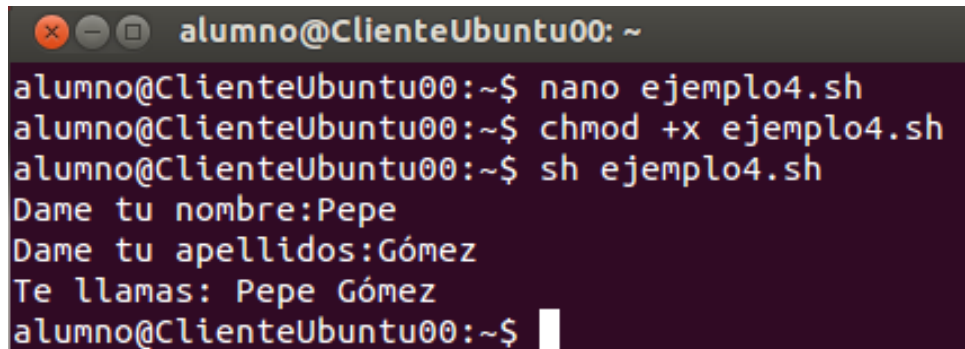
El comando utilizado para sacar por pantalla los datos es **echo** y para la entrada de datos a través del teclado se utiliza el comando **read**.

# ENTRADA Y SALIDA DE DATOS. Ejemplo

**Ejemplo 4.** Crea un script que pida tu nombre y apellidos (por separado) y los muestre por pantalla

```
#!/bin/bash
echo -n Dame tu nombre:
read Nombre
echo -n Dame tus Apellidos:
read Apellidos
echo Te llamas: $Nombre $Apellidos
```

**Ejecución:**

A terminal window titled 'alumno@ClienteUbuntu00: ~' showing the execution of a shell script. The user runs 'nano ejemplo4.sh' to create the script, 'chmod +x ejemplo4.sh' to make it executable, and 'sh ejemplo4.sh' to run it. The script prompts for a name and surnames, and then displays them. The user enters 'Pepe' for the name and 'Gómez' for the surnames. The output is 'Te llamas: Pepe Gómez'.

```
alumno@ClienteUbuntu00:~$ nano ejemplo4.sh
alumno@ClienteUbuntu00:~$ chmod +x ejemplo4.sh
alumno@ClienteUbuntu00:~$ sh ejemplo4.sh
Dame tu nombre:Pepe
Dame tu apellidos:Gómez
Te llamas: Pepe Gómez
alumno@ClienteUbuntu00:~$
```

---

# OPERACIONES ARITMÉTICO- LÓGICAS

# OPERACIONES ARITMÉTICO - LÓGICAS.

Para realizar operaciones aritmético-lógicas con el shell de Ubuntu podremos utilizar el comando **expr** o el **doble** paréntesis: **((expresión))**

## Ejemplo 5

```
total=$((2 * 8))  
echo $total
```

## Ejemplo 6

```
total=$(expr 2 \* 8)  
echo $total
```

**Resultado:** 16 (en ambos ejemplos)

# OPERADORES ARITMÉTICOS

Operador	Descripción	Ejemplos de sencillos scripts
----------	-------------	-------------------------------

## Aritméticos

+	Suma	<pre>Suma=\$((2 + 5)) echo \$Suma</pre>
-	Resta	<pre>resta=\$((expr 5 - 3)) echo \$resta</pre>
\*	Producto	<pre>Producto=\$((expr 5 \* 2)) echo \$Producto</pre> <p>O bien: <pre>Producto=\$((5*2)) echo \$Producto</pre></p>
/	División	<pre>Division=\$((expr 5 / 2)) echo \$Division</pre>
%	Resto de la división entera	<pre>resto=\$((expr 2 % 100)) echo \$resto</pre>
**	Exponenciación	<pre>echo \$((2**3)) Resultado: 8</pre>



# OPERADORES ARITMÉTICOS. Aclaraciones

➤ Bash permite el uso de los operadores aritméticos comunes, pero tiene una limitación importante, **sólo soporta números enteros**, cualquier valor numérico con decimales es truncado a entero.

➤ La forma común de calcular es: **let "expresión"**

**Ejemplo: let res=1+3**

➤ No se puede usar esta notación en las operaciones que aparecen los operadores \*, < o >, puesto que estos caracteres tienen un significado especial para bash. Podemos evitar que shell los tome como caracteres especiales anidando la expresión dentro de dos paréntesis, de esta forma: (( res= 1 \* 3 ))

➤ En la práctica sólo se usa ésta última notación. Dentro de los paréntesis podemos llamar a las variables por su nombre con o sin "\$".

**NOTA:** Si queremos operar con decimales, tenemos la herramienta **bc**, se trata de una calculadora que podemos incorporar a nuestros scripts. Veamos un ejemplo:

```
#!/bin/bash
```

```
echo "scale=10; 10 / 3" | bc
```

```
// scale es una variable interna que determina el número de decimales y tras la operación enviamos el resultado a la herramienta calculadora bc.
```

```
# resultado: 3.3333333333
```

# OPERADORES COMPARATIVOS

Operador	Descripción	Ejemplos de sencillos scripts
		<b>Comparación</b>
<b>=</b>  <b>O bien</b> <b>==</b>	Igualdad	<pre>#!/bin/bash var1=hola var2=HOLA if test [\$var1 = \$var2] then echo Las variables son iguales else echo Las variables son diferentes fi</pre>
<b>!=</b>	Distinto de	<pre>#!/bin/bash var1=1 var2=8 if ((\$var1 != \$var2)) then echo Las variables son diferentes else echo Las variables son iguales fi</pre>
<b>&lt;</b>	Menor que	<pre>echo \$((1 &lt; 2)) Devuelve 1 al cumplirse la condición. 0 en caso contrario</pre>

# OPERADORES COMPARATIVOS

Operador	Descripción	Ejemplos de sencillos scripts
----------	-------------	-------------------------------

## Comparación

>	Mayor que	<code>echo \$((1 &gt; 2))</code> Devuelve 0 al no cumplirse la condición. 1 en caso contrario
---	-----------	--

<=	Menor o igual que	<code>echo \$((1 &lt;= 2))</code> Devuelve 1 caso de no cumplirse la condición. 0 en caso contrario
----	-------------------	--

>=	Mayor o igual que	<code>echo \$((1 &gt;= 2))</code> Devuelve 1 al no cumplirse la condición. 0 en caso contrario
----	-------------------	---

# OPERADORES LÓGICOS

Operador	Descripción	Ejemplos de sencillos scripts
----------	-------------	-------------------------------

## Lógico

	Operación OR
--	--------------

```
#!/bin/bash
var=5
if [ $var -eq 5 ] || [ $var -gt 5 ]
then echo variable es mayor o igual a 5
else echo variable inferior a 5
fi
```

&&	Operación AND
----	---------------

```
#!/bin/bash
var1=5
var2=6
if [ $var1 -eq 5 ] && [ $var2 -gt 5 ]
then echo CIERTO
else echo FALSO
fi
```

# COMANDO TEST

Para realizar operaciones de comparación con el shell de Ubuntu.

Es un comando muy potente, que nos permitirá comparar archivos, cadenas y números. Lo utilizaremos con if, until, while, etc.

## Opciones de test para usar con archivos

Opción	Descripción
-x	El archivo existe con permiso de ejecución
-e	El archivo existe
-s	El archivo existe y no está vacío
-f	El archivo existe y no es un directorio o archivo de dispositivo
-d	El archivo existe y es un directorio
-r	El archivo existe con permiso de lectura
-w	El archivo existe con permiso de escritura

**Nota:** Si se cumple la condición devuelve 0

# COMANDO TEST. Ejemplos con archivos

**Ejemplo 7.** Comprobamos si el directorio datos existe. En caso afirmativo accedemos al directorio (cd datos), sino se creará (mkdir datos) y luego se accederá a él (cd datos).

```
if test -d datos
then
    cd datos
    touch notas
else
    mkdir datos
    cd datos
    touch notas
fi
```

# COMANDO TEST. Ejemplos con archivos

**Ejemplo 8.** comprobamos si el directorio **practica** NO tiene permiso de escritura. En caso afirmativo le asignamos dicho permiso (**chmod +w practica**). En caso contrario nos indicará por pantalla que ya posee el permiso de escritura.

```
if test ! -w practica
then
    chmod +w practica
else
    echo El archivo ya tiene permiso de escritura
fi
```

# COMANDO TEST. Opciones test para números

## Opciones de test para usar con números

Opción	Descripción
-lt	Menor que
-le	Menor o igual que
-gt	Mayor que
-ge	Mayor o igual que
-eq	Igual que
-ne	No es igual que

**Nota:** Si se cumple la condición devuelve 0



# COMANDO TEST. Ejemplos con números

**Ejemplo 9.** Comparando dos variables numéricas.

```
#!/bin/bash
cad1=1
cad2=2
if test $cad1 -eq $cad2  # O bien if test [ $cad1 -eq $cad2 ]
then
    echo Las variables son iguales
else
    echo Las variables son diferentes
fi
```

Otra posible solución

```
#!/bin/bash
cad1=1
cad2=2
if [ $cad1 -eq $cad2 ]
then
    echo Las variables son iguales
else
    echo Las variables son diferentes
fi
```

# COMANDO TEST. Opciones test para cadenas

## Opciones de test para usar con cadenas

Opción	Descripción
-z	Tamaño de la cadena es cero (cadena vacía)
-n	Tamaño de la cadena mayor que 0
$\$c1 = \$c2$ $\$c1 == \$c2$	Las cadenas c1 y c2 son idénticas
$\$c1 != \$c2$	Las cadenas c1 y c2 son diferentes

**Nota:** Si se cumple la condición devuelve 0

# COMANDO TEST. Ejemplos test para cadenas

**Ejemplo 10.** Comparando dos cadenas alfanuméricas.

```
#!/bin/bash
cad1="hola"
cad2="adiós"
if test $cad1 == $cad2
then
    echo Las cadenas son iguales
else
    echo Las cadenas son diferentes
fi
```

**Ejemplo 11.** Comprueba si una cadena está o no vacía

```
#!/bin/bash
cad1="hola"
if test -z $cad1
then
    echo "La cadena está vacía"
else
    echo "La cadena no está vacía"
fi
```

# COMANDO TEST. Opciones test para cadenas

## Opciones de test con operaciones lógicas

Opción	Descripción
-!	Negación
-o	OR
-a	AND

**Nota:** Si se cumple la condición devuelve 0

---

# ESTRUCTURAS DE CONTROL

# COMANDO if

Permite ejecutar una serie de instrucciones en el caso de que la condición marcada sea cierta y otras en el caso de que no lo sea.

## Estructura:

```
if expresión          // Si se cumple la expresión  
then                  // Entonces  
    instrucciones    // ejecutar éstas instrucciones  
else                  // si no se cumple la condición  
    instrucciones    // ejecutar éstas instrucción  
fi                    // fin del if
```

# COMANDO if. Ejemplos

**Ejemplo12.** Dado un número por teclado, indicar si éste es mayor o menor que 10.

```
#!/bin/bash
echo -n "Introduce un número : "
read num
if [ $num -lt 10 ]
then
    echo "El número dado es menor que 10"
else
    echo "El número dado es mayor o igual que 10"
fi
```

## COMANDO if. Encadenación de if: if.. then.. elif.. else...

**Ejemplo13.** Dada una nota, mostrar en pantalla la nota en formato de texto correspondiente.

```
#!/bin/bash
echo -n Introduce una nota:
read nota
if test $nota -lt 5
then
    echo La nota introducida corresponde a un SUSPENSO
elif test $nota -eq 5
then
    echo La nota introducida corresponde a un SUFICIENTE
elif test $nota -eq 6
then
    echo La nota introducida corresponde a un BIEN
elif test $nota -gt 6 -a $nota -le 8
then
    echo La nota introducida corresponde a un NOTABLE
elif test $nota -gt 8 -a $nota -le 10
then
    echo
        La nota introducida corresponde a un SOBRESALIENTE
else
    echo La nota introducida no válida
fi
```



# COMANDO case

Utilizado cuando se aplican varias condiciones a un mismo valor.

## Estructura:

```
case $nombre_variable in      // Caso de que la variable sea igual a
    valor1) instrucciones;;    // valor1 ejecuta éstas instrucciones
    valor2) instrucciones;;    //si es valor2 ejecuta éstas instrucciones
    valor3) instrucciones;;    //si es valor3 ejecuta éstas instrucciones
    ...
    *) instrucciones;;        //si no es ninguno de los anteriores ejecuta éstas instrucciones
esac                          //fin del case.
```

## COMANDO case. Ejemplos

**Ejemplo14.** Mostrar un mensaje indicando el valor numérico comprendido entre 1 y 8. Caso de no estar en dicho rango mostrar mensaje que lo indique.

```
#!/bin/bash
echo -n "Introduce un número del 1 al 8: "
read num
case $num in
    1) echo "Pulsaste el uno ";;
    2) echo "Pulsaste el dos ";;
    3) echo "Pulsaste el tres ";;
    4) echo "Pulsaste el cuatro ";;
    5) echo "Pulsaste el cinco ";;
    6) echo "Pulsaste el seis ";;
    7) echo "Pulsaste el siete ";;
    8) echo "Pulsaste el ocho ";;
    *) echo "Pulsaste el número fuera de rango ";;
esac
```

# COMANDO for

Permite hacer un bucle para repetir una serie de comandos, mientras se cumpla una condición. Existen dos estructuras diferentes:

## Estructura 1 :

```
for variable in arg 1 arg 2 .....arg n  
do  
    comando 1  
    comando 2  
    ...  
done
```

# COMANDO for

Permite hacer un bucle para repetir una serie de comandos, mientras se cumpla una condición. Existen dos estructuras diferentes:

## Estructura 2:

```
for ((inicialización; condición; actualización))  
do  
    comando 1  
    comando 2  
    ...  
done
```

## Ejemplo:

```
#!/bin/bash  
LIMIT=10  
for ((a=1,b=LIMIT;a<=LIMIT;a++,b- -))  
do  
    echo $((a-$b))  
done
```

## COMANDO for. Ejemplos

**Ejemplo15.** Imprimir por pantalla los números del 1 al 10

```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8 9 10
do
    echo $i
done
```

**Ejemplo16.** Imprimir por pantalla las primeras letras del abecedario

```
#!/bin/bash
for LETRA in a b c d e f g h i j k l m n ñ
do
    echo $LETRA
done
```

# COMANDO while

Hace que se ejecute un código **mientras** se cumpla la expresión condicional.

**Estructura :**

```
while [expresión condicional]
do
    instrucciones
    ....
done
```

# COMANDO while. Ejemplos

**Ejemplo 17.** Cuenta atrás. Imprime por pantalla el número introducido por teclado y los anteriores hasta el 1.

```
#!/bin/bash
echo -n "Escribe un número positivo: "
read num
if test $num -lt 0
then
    echo " Tecleaste un número negativo "
else
    while [ $num -gt 0 ]
    do
        echo $num
        num=$(( $num - 1 ))
    done
fi
```

# COMANDO while. Ejemplos

**Ejemplo 18.** Imprime por pantalla los números del 1 al 10

```
#!/bin/bash
NUM=0
while [ $NUM -le 10 ]
do
    echo $NUM
    NUM=$((NUM+1))
done
```

**NOTA:** También se podría haber hecho la línea 3 así:  
`while [ $NUM -le 10 ]; do`



# COMANDO until

El código se ejecuta **hasta** que deja de cumplirse la expresión condicional.

**Estructura:**

```
until [expresión condicional]
do
    instrucciones
done
```

# COMANDO until. Ejemplo

**Ejemplo 19.** Imprime por pantalla los números del 20 al 10.

```
#!/bin/bash
cont=20
until [ $cont -lt 10 ]
do
    echo $cont
    cont=$(( $cont - 1 ))
done
```

---

# FUNCIONES

# COMANDO Funciones.

Es un bloque de instrucciones que devuelve un valor, pudiéndole pasar parámetros. Cuentan con la ventaja de poderse reutilizar dentro de un programa. La estructura básica de una función es la siguiente:

```
nombre_funcion()  
{ instrucciones }
```

Para **invocar** a la función se escribe su nombre seguido de los parámetros que se pasen a la función (caso de existir).

**NOTA:** La función debe definirse antes de su invocación

```
nombre_funcion par1 par2
```

# VARIABLES LOCALES Y GLOBALES

**VARIABLE LOCAL.-** Aquella que sólo es visible dentro del bloque de código en el que aparece. Es decir, si una variable se define dentro de una función, sólo tendrá significado dentro de la misma.

**VARIABLE GLOBAL.-** Aquella que se define en el cuerpo del programa. Será visible desde cualquier bloque del programa.

En el ejemplo:

Las variables: **var1** y **cadena** definidas dentro de la función sólo serán accesibles desde dicha función (**son locales**).

Las variables: **var2** y **nomb** (**variables globales**) serán accesibles desde cualquier punto del script (incluidas las funciones definidas).

## SCRIPT

### FUNCIÓN

```
var1=3  
cadena="Hola"
```

```
# Cuerpo del script  
var2=56.43  
nomb="Juana"
```

# COMANDO Funciones. Ejemplos

## Ejemplo 20. Función sencilla

```
#!/bin/bash
saludo ()
{
    echo " Hola amigos "
}
# Ahora invocamos a la función
saludo
```

# COMANDO Funciones.

Las funciones en bash shell pueden devolver valores enteros mediante el comando **return**.

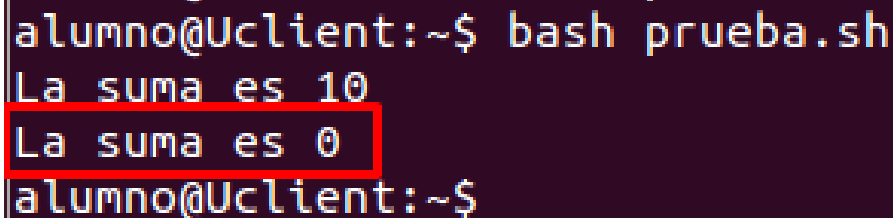
Podemos acceder al valor de retorno de cualquier función usando la variable especial \$?.

```
#!/bin/bash
suma ()
{
    (( result = $1 + $2 ))
    return $result
}
# Programa principal
suma 4 6
echo "La suma es $?"
```

# COMANDO Funciones.

`$?` realmente devuelve el código de salida del último comando ejecutado, y una llamada a función no es más que un caso particular de ejecución de un comando. Esto significa que si deseamos utilizar en varias ocasiones el valor de retorno de una función, necesariamente **debemos guardarlo en una variable auxiliar**.

```
#!/bin/bash
suma ()
{
    (( result = $1 + $2 ))
    return $result
}
# Programa principal
suma 4 6
echo "La suma es $?"
echo "La suma es $?"
```



```
alumno@Uclient:~$ bash prueba.sh
La suma es 10
La suma es 0
alumno@Uclient:~$
```

**Nota:** Como se puede observar, el resultado del segundo echo, la variable `$?` nos devuelve 0, indicando que el comando anterior se ejecutó correctamente.



# COMANDO Funciones. Ejemplos

**Ejemplo 21.** Función suma con paso de parámetros.

```
#!/bin/bash
suma ()
{
    (( result = $1 + $2 ))
    return $result
}
# Programa principal
suma 4 6

echo "La suma es" $result
```

**Nota:** La variable de retorno **result**, se considera global, por eso podemos acceder a ella desde el programa principal.

# COMANDO Funciones. Ejemplos

**Ejemplo 22.** Función suma con paso de parámetros. Variables locales y globales

```
#!/bin/bash
```

```
locales ()
```

```
{
```

```
    local suma=0
```

```
    suma=$(( $1 + $2 ))
```

```
    echo "variable local suma = " $suma
```

```
    echo "usuario es una variable global = " $usuario
```

```
}
```

```
# Programa ppal
```

```
usuario=`echo $USER`
```

```
locales 1 2
```

```
echo "El valor de la variable suma desde el programa ppal es: " $suma
```

```
alumno@ClienteUbu16:~$ bash localesglobales.sh
variable local suma = 3
usuario es una variable global = alumno
El valor de la variable suma desde programa ppal es:
alumno@ClienteUbu16:~$
```

# COMANDO Funciones. Ejemplos

## Ejemplo 22. Función resta con paso de parámetros y retorno del resultado

```
#!/bin/bash
```

```
resta ()
```

```
{
```

```
    #valor=$(expr $1 - $2)
```

```
    valor=`echo "scale=2;$1-$2" | bc`
```

```
    return $valor
```

```
}
```

# Ahora invocamos a la función pasándole los parámetros a y b

```
echo -n "Dame el minuendo: "
```

```
read min
```

```
echo -n "Dame el sustraendo: "
```

```
read sust
```

```
resta $min $sust
```

```
resultado=$valor
```

```
echo "$min - $sust = " $resultado
```

```
alumno@ClienteUbuntu00:~/scripts$ nano resta.sh
alumno@ClienteUbuntu00:~/scripts$ bash resta.sh
Dame el minuendo: 3.4
Dame el sustraendo: 3.1
resta.sh: línea 6: return: .3: se requiere un argumento numérico
3.4 - 3.1 = .3
```

# COMANDO Funciones. Ejemplos

## Ejemplo 23. Uso de varias funciones

```
#!/bin/bash
saludo ()
{
    echo "Le saludamos con un: "$1
}
despedida ()
{
    echo "Le despedimos con un: "$1
}
# Ahora invocamos a las funciones creadas
saludo hola
despedida adiós
```

# COMANDO Funciones. Ejemplos

## Ejemplo 24. Convertidor decimal a binario

```
#!/bin/bash
function convertidor () {
case $num in
    0) num="0000";;
    1) num="0001";;
    2) num="0010";;
    3) num="0011";;
    4) num="0100";;
    5) num="0101";;
    6) num="0110";;
    7) num="0111";;
    8) num="1000";;
    9) num="1001";;
    10) num="1010";;
esac
}
echo -n "Dame el número decimal entre el 0 y el 10 a convertir: "
read num
convertidor $num
echo "El resultado es: "$num
```

---

# REDIRECCIÓN DE ENTRADA/SALIDA

# Redirección de Entrada-Salida

En Linux la salida estándar y el error estándar están conectados a la pantalla, mientras que la entrada estándar lo está al teclado. Sin embargo, esto puede modificarse.

*a) Podemos enviar la salida del resultado de un comando a un fichero:*

**Ejemplo 26.** Crea el fichero `etc.txt` con el contenido del directorio `/etc`  
`ls /etc >ficheroetc.txt`

**Ejemplo 27.** Crea `ficherosd.txt` con los ficheros con extensión `.d` de `/etc`  
`ls /etc/* | grep .d >ficherosd.txt`

*b) Si queremos que un determinado fichero muestre su contenido o parte de él por pantalla.*

**Ejemplo 28.** Mostrar en pantalla aquellos archivos de `ficheroetc.txt` que contengan la palabra `cron`  
`grep cron <ficheroetc.txt`

# MOSTRAR EL CONTENIDO DE UN FICHERO

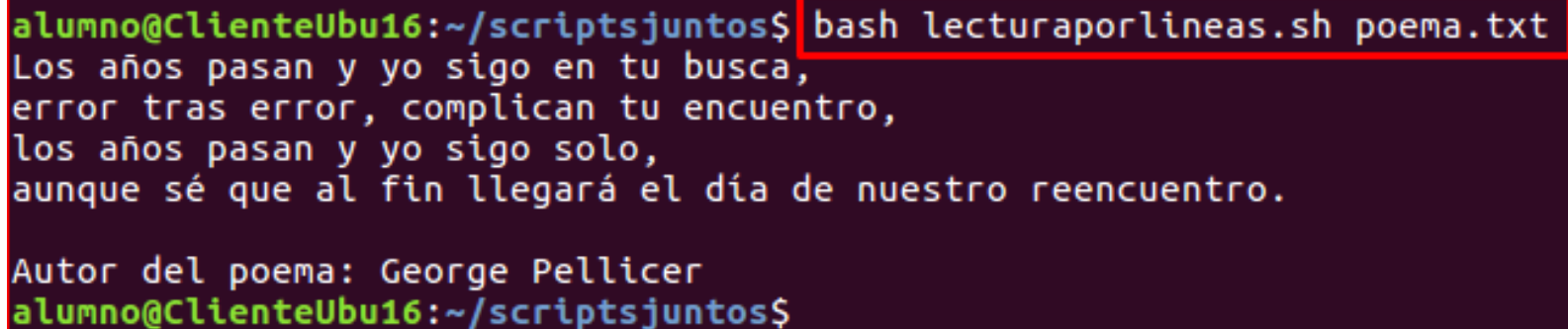
Comando	Descripción	Ejemplos
<b>less &lt;archivo&gt;</b>	Muestra el contenido de <b>archivo</b>	<code>less /etc/passwd</code>
<b>more &lt;archivo&gt;</b>	Muestra el contenido paginado de <b>archivo</b> .	<code>more /etc/passwd</code>
<b>grep &lt;expresión&gt;</b>	Filtra las líneas que cumplan la <b>expresión</b> .	<code>less /etc/passwd   grep root</code>
<b>head -n n°_lineas</b>	Muestra las primeras n líneas que indique el parámetro n°_lineas	<code>less /etc/group   head -n 7</code>
<b>tail -n n°_lineas</b>	Muestra las últimas n líneas que indique el parámetro n°_lineas	<code>less /etc/group   tail -n 7</code>
<b>cut -d "delimitador" -f filas</b>	Muestra en filas el campo que se encuentra tras el delimitador. Podremos indicar cuál de los elementos de cada fila del archivo queremos mostrar (si filas es 2 entonces se mostrará el segundo campo de cada fila, tras el delimitador).	<code>less /etc/passwd   cut -d ":" -f6</code>
<b>sort</b>	Permite ordenar una salida de datos	<code>less /etc/passwd   cut -d ":" -f6   sort</code>



# TRABAJANDO CON FICHEROS

## Ejemplo 29. Lectura línea a línea de un fichero.

```
#!/bin/bash
while read linea
do
    echo $linea
done < $1
```

A terminal window with a dark background and light-colored text. The prompt is 'alumno@ClienteUbu16:~/scriptsjuntos\$'. The command 'bash lecturaporlineas.sh poema.txt' is entered and executed. The output consists of four lines of a poem. Below the poem, the text 'Autor del poema: George Pellicer' is displayed. The prompt 'alumno@ClienteUbu16:~/scriptsjuntos\$' is shown again at the bottom.

```
alumno@ClienteUbu16:~/scriptsjuntos$ bash lecturaporlineas.sh poema.txt
Los años pasan y yo sigo en tu busca,
error tras error, complican tu encuentro,
los años pasan y yo sigo solo,
aunque sé que al fin llegará el día de nuestro reencuentro.

Autor del poema: George Pellicer
alumno@ClienteUbu16:~/scriptsjuntos$
```

# TRABAJANDO CON FICHEROS

**Ejemplo 30. Lectura palabra a palabra de un fichero pasado por parámetro a dicho script.**

```
#!/bin/bash  
for palabra in $(cat $1)  
do  
    echo $palabra  
done
```

# TRABAJANDO CON FICHEROS

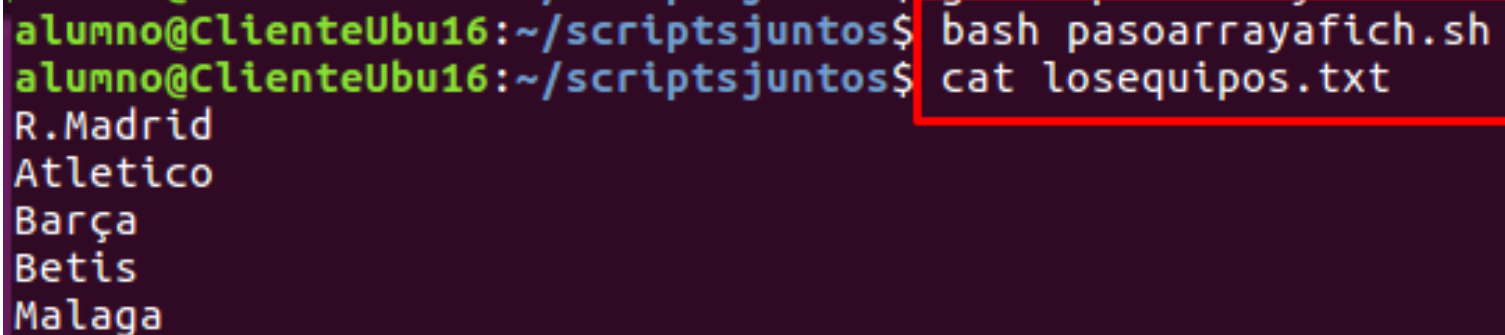
## Ejemplo 30. RESULTADO DE EJECUCIÓN:

```
alumno@ClienteUbu16:~/scriptsjuntos$ bash lecturaporpalabras.sh poema.txt
Los
años
pasan
y
yo
sigo
en
tu
busca,
error
tras
error,
complican
tu
encuentro,
los
años
pasan
y
yo
sigo
solo,
aunque
sé
que
al
fin
llegará
el
día
de
nuestro
reencuentro.
Autor
del
poema:
George
Pellicer
alumno@ClienteUbu16:~/scriptsjuntos$
```

# TRABAJANDO CON FICHEROS

## Ejemplo 31. Lectura de un array y escritura en un fichero.

```
#!/bin/bash
equipos=("R.Madrid" "Atletico" "Barça" "Betis" "Malaga")
for equip in ${equipos[@]}
do
    echo $equip
done >> /home/alumno/scriptsjuntos/losequipos.txt
```

A terminal window with a dark background. The prompt is 'alumno@ClienteUbu16:~/scriptsjuntos\$'. The user enters 'bash pasoarrayafich.sh'. The prompt changes to 'alumno@ClienteUbu16:~/scriptsjuntos\$'. The user enters 'cat losequipos.txt'. The output of the command is displayed on the following lines: 'R.Madrid', 'Atletico', 'Barça', 'Betis', and 'Malaga'.

```
alumno@ClienteUbu16:~/scriptsjuntos$ bash pasoarrayafich.sh
alumno@ClienteUbu16:~/scriptsjuntos$ cat losequipos.txt
R.Madrid
Atletico
Barça
Betis
Malaga
```