# SHELLS Y COMANDOS

# OBJETIVOS

# ÍNDICE DE CONTENIDOS



# INTRODUCCIÓN

# INTRODUCCIÓN

Existen múltiples y variadas formas de **obtener ayuda** en un sistema Linux. A continuación se describen algunas de ellas:

Muchos comandos poseen una opción para mostrar una ayuda breve acerca de su utilización.

- Esta opción usualmente es **-h**, **--help** ó **-?**.

El comando ***man*** formatea y despliega un manual bastante amplio acerca de comandos, formatos de ficheros de configuración, llamados al sistema, etc. Los manuales están disponibles y pueden instalarse en múltiples idiomas. Estos se dividen internamente en secciones. Un mismo objetivo puede estar representado en varias secciones. De no especificarse ninguna sección a través del primer argumento del comando se tomará la primera donde aparezca.

- Ejemplo: ***$ man chmod***

# EL SISTEMA DE ARCHIVOS

Existe un conjunto de comandos integrados al **bash** (interprete de comandos) que no poseen un manual propio. Para ellos se puede emplear el comando help. Si se usa man sólo se mostrará la lista de los comandos integrados. La ayuda que muestra help es un fragmento del manual de bash ($ man bash), el cual es muy amplio y resulta incómoda la búsqueda. Algunos de los comandos integrados al bash y que veremos más adelante son: cd, fg, bg, logout, exit, umask, set, help, source, alias, echo, kill, jobs y export.

- Ejemplo: ***$ help logout***

El programa info despliega información acerca de comandos, en ocasiones más amplia que la que brinda man. Las páginas de info poseen una estructura arbórea (nodos), a través de las cuales se puede navegar con ayuda de comandos especiales.

- Ejemplo: ***$ info ln***

# EL SISTEMA DE ARCHIVOS

El comando ***whatis*** realiza una búsqueda en las secciones del man y muestra la descripción abreviada del comando en cada una de las secciones que aparezca. Este comando trabaja con una base de datos que se actualiza periódicamente y se crea con el comando makewhatis.

- Ejemplo: ***$ whatis shadow***

El comando ***apropos***, dada una palabra busca en toda la base de datos del whatis desplegando todo lo encontrado.

- Ejemplo: ***$ apropos passwd***

# EL SISTEMA DE ARCHIVOS

- Muchos programas se empaquetan con su documentación en diversos formatos. Normalmente estas ayudas se agrupan en el directorio /usr/share/doc. También existen los HOWTOs (Como lograr...) escritos en muchos idiomas y formatos para varios temas disponibles. Algunos se empaquetan como cualquier otro programa o se localizan en Internet.

- En **Internet** de forma general existen una gran cantidad de grupos de noticias, listas de discusión, sitios Web y FTP sobre Linux. En muchos de ellos incluso se pueden encontrar paginas de ayuda sobre los comandos en castellano, mientras que en nuestra versión de Linux es posible que nos aparezcan en Ingles.

# ALGUNAS ORDENES BÁSICAS

# ALGUNAS ORDENES BÁSICAS

- ***CLEAR:*** Limpia la pantalla en modo comando.
- ***DATE***: Presenta en pantalla el día y la hora que tiene el equipo en ese momento. Muestra, de izquierda a derecha, el día de la semana, mes, día de mes, hora, meridiano y año.
- ***CAL:*** Calendario perpetuo que incorpora el sistema. Nos permite ver calendarios completos de otros años o calendarios de un mes determinado.
  - Si se pone solo ***cal***, muestra el mes actual.
  - Si escribimos ***cal*** seguido del mes y del año que queremos que nos muestre.

# ALGUNAS ORDENES BÁSICAS

- ***WHO:*** Muestra por pantalla una línea por cada usuario que es ese momento está conectado al sistema. Muestra, de izquierda a derecha, el nombre de presentación del usuario, el número de terminal y la fecha y hora de presentación.
- ***PASSWD***: Permite al usuario cambiar o asignar una contraseña, siempre que el administrador lo permita.
- ***UNAME***: muestra información como el tipo de SO, versión, tipo de ordenador desde el que nos conectamos…

# COMANDOS PARA PAGINAR, VISUALIZAR Y EDITAR FICHEROS

# COMANDOS PARA PAGINAR, VISUALIZAR Y EDITAR FICHEROS

- En Linux también existen editores de texto para poder trabajar con archivos de texto plano que no generan ningún tipo de caracteres de control especiales.
- Para poder editar archivos de texto en Linux utilizaremos alguno de los siguientes editores de texto:
  - **vi.** Editor de textos heredado de las primeras versiones de Unix, que no permite el manejo de las teclas de edición ni el manejo del ratón. Su uso es mínimo hoy por hoy, pero es conveniente conocer su existencia, al menos por cuestiones históricas. Lo ejecutaremos desde la línea de comandos de Linux.
  - **nano.** Editor similar a vi, pero con mejor funcionalidad a la hora de editar textos planos. Este editor de texto lo podremos ejecutar desde la línea de comandos de Linux.
  - **gedit.** Podemos ejecutarlo directamente desde la línea de comandos, o en entorno gráfico desde Aplicaciones > Accesorios > Editor de textos.

# COMANDOS PARA PAGINAR, VISUALIZAR Y EDITAR FICHEROS

## EDITOR VI

- El editor ***vi*** es el editor estándar de Unix.
- Resulta incomodo ***vi*** ya que para utilizar todas sus potencialidades es necesario conocer muchas combinaciones de teclas, pero si se llega a dominar resulta muy funcional.
- Su principal virtud es que encontraremos ***vi*** en prácticamente cualquier versión de Unix que usemos, cosa que no se puede decir de otros editores (joe, pico, edit, gedit, nano, emacs, etc.).
- Básicamente ***vi*** posee dos modos de interacción: el de inserción (edición) y el de comandos.
- Para pasar al modo comando se pulsa ***Esc*** y para pasar al de inserción se pulsa ***i***.

# COMANDOS PARA PAGINAR, VISUALIZAR Y EDITAR FICHEROS

**EDITOR VI**

- Algunos comandos útiles en ***vi*** (pulsando ESC para pasar al modo de comandos, que veremos en la ultima línea de la pantalla).
  - dd - borra la línea actual.
  - D - borra desde la posición actual hasta el final de la línea.
  - dG - borra hasta el final del fichero.
  - u - deshace el último comando.
  - :q - sale del editor (si se han hecho modificaciones y no se ha salvado se genera un error).
  - :q! - sale sin salvar.
  - :w - salva. □ :wq - salva y sale.
  - :x - salva y sale.
  - <n><comando> - ejecuta el comando c n veces.

# COMANDOS PARA PAGINAR, VISUALIZAR Y EDITAR FICHEROS

## COMANDO MORE Y LESS

- Los comandos ***more*** y ***less*** paginan (dividen en páginas) uno o varios ficheros y los muestran en la terminal (pantalla).
- De no indicárseles un fichero, paginan la entrada estándar (que se manda mediante una tubería).
- Se diferencian en las facilidades que brindan.
- Por ejemplo ***more*** es más restrictivo en cuanto al movimiento dentro del texto, mientras que ***less*** no limita este aspecto pues acepta el empleo de todas las teclas de movimiento tradicionales.
- Cuando se alcanza el final del último fichero a paginar, more termina automáticamente, no así ***less***.
- También ***more*** muestra sucesivamente el porcentaje del fichero visto hasta el momento.
- Tanto ***less*** como more proveen una serie de comandos para moverse con facilidad dentro del texto paginado.

# COMANDOS PARA PAGINAR, VISUALIZAR Y EDITAR FICHEROS

## COMANDO MORE Y LESS

- Ejemplos:

  $ less /etc/passwd

  $ more /etc/passwd

  $ cat fichero | less

  - Algunas teclas que podemos usar mientras usamos estos programas son:
    - q - permite interrumpir el proceso y salir.
    - /p - realiza búsquedas del patrón p dentro del texto. Para repetir la búsqueda del mismo patrón sólo es necesario escribir /.
    - [n]b - en more permite regresar n páginas (por defecto n es 1).
    - [n]f - en more se adelantan n páginas y en less, n líneas.
  - El ***man***, para dar formato a su salida, utiliza por defecto el paginador less.

# PERMISOS

# PERMISOS

- La información sobre grupos, usuarios y permisos se puede obtener mediante el comando ***ls*** junto con la opción **-l**.
- Vamos a ver los permisos que tiene establecidos el fichero *whatis* que se encuentra en el directorio */usr/bin*.

**$ ls -l /usr/bin/whatis**

-rwxr-xr-x 1 root root 87792 2008-03-12 14:24 /usr/bin/whatis

- En la primera columna aparecen los permisos, en la tercera se indica el usuario (en este caso es el administrador del sistema) y en la cuarta columna aparece el nombre del grupo (que en este caso coincide con el de usuario).

# PERMISOS

- Vamos a ver qué significan exactamente los caracteres de la primera columna:

| - | r | w | x | r | - | x | r | - | x |
|---|---|---|---|---|---|---|---|---|---|
| Tipo de fichero. | Permisos para el dueño del fichero. | | | Permisos para el grupo al que pertenece el fichero. | | | Permisos para el resto de usuarios | | |

| *Tipo de fichero* | |
|---|---|
| l | Enlace simbólico. |
| c | Dispositivo especial de caracteres. |
| b | Dispositivo especial de bloques. |
| p | FIFO (estructura de datos). |
| s | Socket (comunicaciones). |
| - | Ninguno de los anteriores. Puede ser un fichero de texto, un binario, etc. |

| | |
|---|---|
| r | Permiso de **lectura**. |
| w | Permiso de **escritura**. |
| x | Permiso de **ejecución**. |

# PERMISOS

- Vamos a ver qué significan exactamente los caracteres de la primera columna:

| - | r | w | x | r | - | x | r | - | x |
|---|---|---|---|---|---|---|---|---|---|
| Tipo de fichero. | Permisos para el dueño del fichero. | | | Permisos para el grupo al que pertenece el fichero. | | | Permisos para el resto de usuarios | | |

- En el caso que nos ocupa tenemos un carácter "-" como tipo de fichero, porque se trata de un binario (un programa).
- El dueño del fichero tiene los permisos rwx, lo que quiere decir que puede leer, escribir y ejecutar el fichero. Que tiene permiso para escribir significa que puede borrarlo, cambiarle el nombre o editarlo.
- Tanto el grupo como el resto de usuarios tienen los permisos r-x, lo que significa que pueden utilizarlo (pueden leerlo y ejecutarlo) pero no lo pueden modificar.

# PERMISOS

## COMANDO CHMOD

- El comando ***chmod*** sirve para cambiar los permisos de uno o varios ficheros.
- Esos mismos permisos que se pueden ver con ***ls -l***.

**$ ls -l**

**-rw-r--r-- 1 usuario grupo  0 2018-04-19 15:38 ejemplo.rb**

**$ chmod +x ejemplo.rb**

**$ ls –l**

**-rwxr-xr-x 1 usuario grupo 0 2018-04-19 15:38 ejemplo.rb**

- Hemos añadido el permiso de ejecución al fichero *ejemplo.rb*. Vemos que ahora hay tres **x**, la que corresponde al dueño del fichero, la de todos los usuarios que pertenecen al grupo y la del resto de usuarios.
- Cuando no se especifica ninguna de estas tres letras correspondientes a los usuarios (**u, g, o**) como en el ejemplo anterior, se sobreentiende que nos referimos a todos ellos. Se puede indicar de forma explícita con el carácter ***a*** (all).

# PERMISOS

## COMANDO CHMOD

- Para entenderlo mejor, en la siguiente tabla, se muestran de forma esquemática, los parámetros del comando **chmod**:

| u | g | o | + − | r | w | x |
|---|---|---|---|---|---|---|
| (user) dueño del fichero | (group) usuarios que pertenecen al mismo grupo | (others) el resto de usuarios | dar permiso quitar permiso | (read) lectura | (write) escritura | (execution) ejecución |

- Quitaremos ahora el permiso de ejecución para el resto de usuarios (others) y daremos permiso de escritura (write) a los usuarios del mismo grupo (group).

```
$ ls -l
-rwxr-xr-x 1 usuario grupo  0 2018-04-19 15:38 ejemplo.rb
$ chmod o-x ejemplo.rb
$ chmod g+w ejemplo.rb
$ ls -l
-rwxrwxr-- 1 usuario grupo  0 2018-04-19 15:38 ejemplo.rb
```

# SCRIPTS

# SCRIPTS

- Los scripts no son más que ficheros de texto ASCII puro, que pueden ser creados con cualquier editor del que dispongamos (vi, nano, gedit, emacs, etc.).
- Cread un fichero de texto de nombre primero.sh, con el siguiente contenido:

  ***#!/bin/bash***

  ***echo "Hola Mundo"***

- La primera línea sirve para indicar que shell utilizamos (en nuestro caso bash) y donde puede ser encontrado en nuestro sistema (para saberlo, podemos hacer locate bash).
- Esta línea debe ser la primera de todos los scripts que realicemos.
- La segunda línea de nuestro script, simplemente utiliza el comando para escribir en pantalla (echo) y escribe la línea Hola Mundo.

# SCRIPTS

- Una vez creado el fichero, debemos darle permisos de ejecución, mediante el comando

  **chmod +x primero.sh**

- Posteriormente para ejecutarlo debemos llamarlo como **./primero.sh.**
- El punto barra es para indicarle que lo busque en el directorio actual, ya que dicho directorio no estará seguramente incluido en el PATH del sistema.
- Si queremos ejecutar un script para comprobar cómo funciona sin hacerlo ejecutable, podemos hacerlo mediante el comando **source primero.sh** que permite lanzar un script no ejecutable.

# SCRIPTS

- EJEMPLO:
  - *echo esto es un asterisco * sin comillas*
  - *echo "esto es un asterisco * entre comillas dobles"*
  - *echo 'esto es un asterisco * entre comillas simples'*
  - *echo esto es un dólar y tres letras $ABC sin comillas*
  - *echo "esto es un dólar y tres letras $ABC entre comillas dobles"*
  - *echo 'esto es un dólar y tres letras $ABC entre comillas simples'*

# SCRIPTS

- Si tenemos que ejecutar varias líneas y queremos escribirlas en una sola, podemos hacerlo usando el símbolo punto y coma para indicar que lo siguiente es otra línea, aunque este en la misma:

  ***echo Hola ; pwd ; echo Adios*** # esto son tres líneas en una sola

- *También podemos hacer lo contrario, escribir una sola línea en varias. Para ello usamos el carácter contrabarra cuando queramos que nuestra línea se "rompa" y continué en la línea de abajo.*

  ***echo esto ***

  ***es una sola línea ***

  ***aunque ocupe 3 en pantalla.*** # esto es una línea escrita en tres.

- Para poner comentario con usar el símbolo almohadilla **(#)** donde queramos, todo lo que quede a la derecha de dicho símbolo es un comentario. Si usamos # como primer carácter de una línea, toda la línea es de comentario.

# USO DE VARIABLES EN LOS SCRIPTS

# USO DE VARIABLES EN LOS SCRIPTS

- Las variables de los shell scripts son muy simples, ya que no tienen tipo definido ni necesitan ser declaradas antes de poder ser usadas. Para introducir valor en una variable simplemente se usa su nombre, y para obtener el valor de una variable se le antepone un símbolo dólar.

```
#!/bin/bash
DECIR="Hola Mundo"
echo $DECIR
```

- Este script realiza exactamente la misma función que el anterior, pero usando una variable.
- Cualquier valor introducido en una variable se considera alfanumérico, así que si realizamos lo siguiente:

```
NUMERO=4   # No se debe dejar ningún espacio en la asignación.
echo $NUMERO+3
```

- Obtendremos por pantalla la cadena de caracteres 4+3.

# USO DE VARIABLES EN LOS SCRIPTS

## $ (ORDEN)

- En Linux podemos usar varias expansiones en las líneas de comandos, que son especialmente útiles en los scripts. La primera expansión consiste en usar $( ). Esta expansión permite ejecutar lo que se encuentre entre los paréntesis, y devuelve su salida.

***echo pwd*** # escribe por pantalla la palabra pwd

***echo $(pwd)*** # ejecuta la orden pwd, y escribe por pantalla su resultado.

# USO DE VARIABLES EN LOS SCRIPTS

**$ ((OPERACIÓN ARITMÉTICA))**

- Otra expansión que podemos usar es $(()). Los dobles paréntesis podemos sustituirlos si queremos por corchetes. $[ ].
- Esta expansión va a tratar como una expresión aritmética lo que esté incluido entre los paréntesis, va a evaluarla y devolvernos su valor.

***NUMERO=4***

***echo $(($NUMERO+3))*** # sería lo mismo poner echo $[$NUMERO+3]

- Obtenemos en pantalla el valor 7.

# USO DE VARIABLES EN LOS SCRIPTS

**LET**

- Nos permite realizar operaciones aritméticas como la anterior, pero sin tener que usar expansiones ni dólares para las variables.

```
NUMERO=4
let SUMA=NUMERO+3
echo $SUMA
```

- Obtenemos el mismo valor 7, y como vemos no hemos usado ni dólar, ni paréntesis.
- Los operadores aritméticos que podemos usar para realizar operaciones son: Resta (-), Suma (+), División (/), Multiplicación (*) y Modulo o Resto (%).

# USO DE VARIABLES EN LOS SCRIPTS

## LET. BC

- ***bash*** no trabaja con decimales, por lo que si hacemos una división de 10/3 no da 3,333333 sino daría como resultado 3.
- Podemos obligar a que ***bash*** trabaje con decimales utilizando un comando que sirve como calculadora en Linux, este comando es ***bc***.
- Utilizando el Ejer1.sh modificamos el archivo para poner:

  ***Media=$[total/3]***

  ***echo $Media*** #El resultado sería 5.
- Para indicar que queremos obtener decimales y cuantos hay que hacer los siguiente:

  ***Media=$(echo "scale=2; $total/3" \| bc -l)***

  ***echo $Media*** #El resultado sería 5,33.

# USO DE VARIABLES EN LOS SCRIPTS

**LET. BC**

***Media=$(echo "scale=2; $total/3" |bc -l)***

- Vemos cómo debemos generar una salida con echo:
  - El primer campo ***scale*** indica cuantos decimales queremos obtener.
  - Luego y separado por un punto y coma ponemos la operación aritmética que deseamos realizar, sin necesidad de poner corchetes, dobles paréntesis o usar ***let***.
  - El resultado de este echo lo enviamos al comando ***bc –l*** mediante una tubería.
- EJEMPLOS:

```
echo $[20/6]                      #El resultado es 3.
echo "scale=2; 20/6" | bc -l      #El resultado es 3.33.
echo "scale=4; 20/6" | bc -l      #El resultado es 3.3333.
```

# USO DE VARIABLES EN LOS SCRIPTS

**READ**

- El comando READ nos permite solicitar un valor de entrada por teclado para almacenarlo en una variable.
- Tiene muchas opciones de entrada pero las más utilizadas son las siguientes:
  - ***-p*** muestra un mensaje.
  - ***-nN*** acepta sólo N caracteres de entrada por teclado.

# USO DE VARIABLES EN LOS SCRIPTS

**READ**

- Ejemplo1:

  ***echo "Por favor introduce tu nombre:"***

  ***read nombre***

  ***echo "Bienvenid@ a nuestro script $nombre"***

- Ejemplo2:

  ***read -p "Por favor introduce tu nombre: " nombre***

  ***echo "Bienvenid@ a nuestro script $nombre"***

- Realizan lo mismo el Ejemplo1 y el Ejemplo2.

# USO DE VARIABLES EN LOS SCRIPTS

**READ**

- Ejemplo3: Igual que el ejemplo1, pero solo deja introducir 4 valores.

  ***echo "Por favor introduce tu nombre:"***

  ***read –n4 nombre***

  ***echo "Bienvenid@ a nuestro script $nombre"***

- Ejemplo4: Igual que el ejemplo1, pero solo deja introducir 4 valores.

  ***read –n4 -p "Por favor introduce tu nombre: " nombre***

  ***echo "Bienvenid@ a nuestro script $nombre"***

- Realizan lo mismo el Ejemplo3 y el Ejemplo4.

# USO DE VARIABLES EN LOS SCRIPTS

**EJERCICIO1:**

- Realiza en ejercicio donde se pidan dos variables y realices la suma de esas dos variables, la resta, la multiplicación y la división.

**EJERCICIO2:**

- Realiza en ejercicio donde se pida el nombre y los apellidos del alumno, y también las notas de las 6 asignaturas del modulo y haz la media final del alumno.

**EJERCICIO3:**

- Realiza un cuestionario al usuario de 5 preguntas donde solo pueda responder las preguntas con si o no. Pon limitación de caracteres. Al finalizar las 5 preguntas tiene que salir por pantalla todas las preguntas con sus respuestas.

# FUNCIONES

# FUNCIONES

- Usar funciones en los scripts es muy simple. Basta con usar la siguiente estructura al principio del script:

***function nombre_función*** **{**

***líneas de la función***

**}**

- Estas líneas de la función no se ejecutarán al procesar el script, sino que solo se ejecutarán cuando en el cuerpo del script usemos ***nombre_funcion***.

# FUNCIONES

- Ejemplo:

```
#!/bin/bash
function doble {
        echo "voy a doblar el valor de numero"
        let NUMERO=NUMERO*2
}
NUMERO=3
echo '$NUMERO vale : ' $NUMERO
doble                                   # llamamos a la función
echo '$NUMERO vale : ' $NUMERO
```

- Podría parecer en una lectura rápida del script anterior, que estamos pasando por ***let NUMERO=NUMERO*2*** antes de asignarle el valor 3. No es cierto, ya que aunque veamos esas líneas físicamente anteriores a la asignación, solo serán procesadas cuando en el script escribimos ***doble***.

# FUNCIONES

- Por defectos, todas las variables que usemos son **globales**, es decir, que las funciones y el script las comparten, pueden modificar sus valores, leer las modificaciones realizadas, etc.
- Sin embargo, en determinadas ocasiones nos puede interesar que las variables sean locales a la función, es decir, que si la función modifica su valor el script no se entera…
- Ejemplo1:

```
#!/bin/bash
function saludo {
        NOMBRE=”Jose Antonio”
        echo “Hola señor $NOMBRE encantado de conocerle”
}
NOMBRE=”Juana”
saludo
echo “En el script principal, mi nombre es $NOMBRE”
```

# FUNCIONES

- Esto es así porque al inicializar la variable NOMBRE como NOMBRE=”Jose Antonio” estamos creando una variable global, visible tanto en la función como en el script. Sin embargo…
- Ejemplo2:

```
#!/bin/bash
function saludo {
        local NOMBRE=”Jose Antonio”
        echo “Hola señor $NOMBRE encantado de conocerle”
}
NOMBRE=”Juana”
saludo
echo “En el script principal, mi nombre es $NOMBRE”
```

- La diferencia entre el Ejemplo1 y el Ejemplo2 es que al anteponer local a la variable de NOMBRE en el Ejemplo2, las modificaciones que se realicen sólo afectan a la propia función, por lo que en pantalla vemos como aparece ***En el script principal, mi nombre es Juana***.

# ESTRUCTURAS CONDICIONALES

# ESTRUCTURAS CONDICIONALES

- Esto es así porque al inicializar la variable NOMBRE como NOMBRE=”Jose Antonio” estamos creando una variable global, visible tanto en la función como en el script. Sin embargo…
- Ejemplo2:

```
#!/bin/bash
function saludo {
        local NOMBRE=”Jose Antonio”
        echo “Hola señor $NOMBRE encantado de conocerle”
}
NOMBRE=”Juana”
saludo
echo “En el script principal, mi nombre es $NOMBRE”
```

- La diferencia entre el Ejemplo1 y el Ejemplo2 es que al anteponer local a la variable de NOMBRE en el Ejemplo2, las modificaciones que se realicen sólo afectan a la propia función, por lo que en pantalla vemos como aparece ***En el script principal, mi nombre es Juana***.

# ESTRUCTURAS CONDICIONALES

**IF**

- La principal estructura condicional de los scripts en shell es el if (Sí en inglés):

***if [ expresión ]; then***

***Realizar si expresión es verdadera***

***fi***

- La expresión es cualquier expresión lógica que produzca un resultado verdadero o falso. Si estamos operando con cadenas **alfanuméricas**, los operadores que podemos utilizar son los siguientes:

# ESTRUCTURAS CONDICIONALES

| | |
|---|---|
| **Cadena1 = Cadena2** | **Verdadero si Cadena1 es IGUAL a Cadena2** |
| **Cadena1 != Cadena2** | **Verdadero si Cadena1 NO es IGUAL a Cadena2** |
| **Cadena1 < Cadena2** | **Verdadero si Cadena1 es MENOR a Cadena2** |
| **Cadena1 > Cadena2** | **Verdadero si Cadena1 es MAYOR a Cadena2** |
| **-n Variable1** | **Verdadero si C1 NO ES NULO (tiene algún valor)** |
| **-z Variable1** | **Verdadero si C1 ES NULO (está vacía o no está definida** |

# ESTRUCTURAS CONDICIONALES

**IF**

EJEMPLO1:

- Vamos a comparar dos palabras.

```
if [ "jose" = "juan" ]; then
        echo "iguales";
fi
```

- En este primer ejemplo no hay respuesta porque no son iguales.

EJEMPLO2:

- Vamos a comparar dos palabras.

```
if [ "jose" = "jose" ]; then
        echo "iguales";
fi
```

- En este segundo ejemplo saldrá por pantalla la palabra "iguales".

# ESTRUCTURAS CONDICIONALES

**IF**

EJEMPLO3:

- Vamos a comparar si hoy es viernes.

```
cat > dia.sh
#!/bin/bash
DIA=$( date +%A)
if [ $DIA= “viernes” ]; then
        echo “Bravo, por fin es viernes”
fi
```

- En este ejemplo saldrá la frase “Bravo, por fin es viernes” cuando ejecutemos el programa un viernes.

# ESTRUCTURAS CONDICIONALES

## IF

- Los anteriores operadores sólo son válidos para comparar cadenas, si queremos comparar **valores numéricos**, hemos de usar los siguientes operadores:

| | |
|---|---|
| **N1 –eq N2** | **Verdadero si N1 es IGUAL a N2. (equal)** |
| **N1 –ne N2** | **Verdadero si N1 NO es IGUAL a N2. (not equal)** |
| **N1 –lt N2** | **Verdadero si N1 es MENOR a N2. (less that)** |
| **N1 –gt N2** | **Verdadero si N1 es MAYOR que N2. (greater that)** |
| **N1 –le N2** | **Verdadero si N1 es MENOR O IGUAL que N2. (less or equal)** |
| **N1 –ge N2** | **Verdadero si N1 es MAYOR O IGUAL que N. (greater or equal)** |

# ESTRUCTURAS CONDICIONALES

**IF..ELSE**

- La estructura ***if*** podemos ampliarla usando la construcción ***else*** (en caso contrario) y ***elif*** (en caso contrario si…).
- La estructura simple de **else** es la siguiente:

***if [ expresión 1 ]; then***

***Realizar si expresión 1 es verdadera***

***else***

***Realizar si expresión 1 es falsa***

***fi***

# ESTRUCTURAS CONDICIONALES

**IF..ELSE**

**EJEMPLO**

- Vamos a realizar lo mismo que el ejemplo4. Si un número es mayor a 5.

```
cat > numeros2.sh
!#/bin/bash
Numero=7
if [ $Numero –ge 5]; then
        echo “El número es mayor o igual que 5”
else
        echo “El número es menor que 5”
fi
```

# ESTRUCTURAS CONDICIONALES

**IF..ELIF..ELSE**

- La estructura con **elif** (else if) es la siguiente:

```
if [ expresión 1 ]; then
        Realizar si expresión 1 es verdadera
elif [ expresión 2 ]; then
        Realizar si exp1 es falsa, pero es exp2 es verdadera
elif [ expresión 1 ]; then
        Realizar si exp1 y exp2 son falsas, pero es ex3 es verdadera
else
        Realizar si todas las expresiones anteriores son falsas
fi
```

# ESTRUCTURAS CONDICIONALES

- Hay que tener muchísimo cuidado con los **espacios en blanco**, y seguramente durante nuestros primeros scripts casi todos los errores vendrán por haberlos usado mal en las estructuras **if.**
- Hay que recordar que **los corchetes llevan espacios en blanco** tanto a izquierda como derecha, que el punto y coma sin embargo va pegado al corchete cerrado, y que **SIEMPRE** hay que poner espacios en blanco en las expresiones.

# ESTRUCTURAS CONDICIONALES

**EJERCICIO 4:**

- En el ejercicio 2 añade un **if** para ver si ha sacado igual o más de un 5 entonces habrá aprobado, sino habrá suspendido el curso. Y que lo muestre por pantalla.

**EJERCICIO 5:**

- En el ejercicio 2 añade un **if..else** para ver si ha sacado igual o más de un 5 entonces habrá aprobado, sino habrá suspendido el curso. Y que lo muestre por pantalla.

**EJERCICIO 6:**

- En el ejercicio 2 añade un **if..elif..else** para indicar la nota que tiene al final en letra. (Si ha sacado entre 0 y 4,99 es un suspenso, entre 5 y 5,99 un aprobado, entre 6 y 6,99 un bien, entre 7 y 8,99 un notable, entre 9 y 9,99 un sobresaliente y un 10 es matricula.

# ESTRUCTURAS CONDICIONALES

**CASE**

- Hemos visto la principal estructura condicional que es el if, pero tenemos alguna otra a nuestra disposición, como el case. Esta estructura nos permite ejecutar varias acciones, dependiendo del valor de una variable o expresión.

```
case VARIABLE in
        valor1)
                se ejecuta si VARIABLE tiene el valor1
                ;;
        valor2)
                se ejecuta si VARIABLE tiene el valor2
                ;;
        *)
                Se ejecuta si VARIABLE tiene otro valor
distinto
                ;;
esac
```

# ESTRUCTURAS CONDICIONALES

**CASE**

- EJEMPLO: se van a introducir por teclado los dos primeros números del código postal y se va a reconocer de que provincia proviene.

```
#!/bin/bash
read –n2 –p “Introduzca los 2 primeros dígitos del código postal:” CP
case $CP in
        46)
                echo -e “\n Provincia de Valencia”      ;;
        03)
                echo –e “\n Provincia de Alicante”      ;;
        12)
                echo –e “\n Provincia de Castellón”     ;;
        *)
                echo –e “\n No es provincia de la C. Valenciana”    ;;
esac
```

# ESTRUCTURAS ITERATIVAS

# ESTRUCTURAS ITERATIVAS

**FOR**

- La estructura básica de ***for*** es la siguiente:

***for VARIABLE in conjunto; do***

***Estas líneas se repiten una vez por cada elemento del conjunto, Y variable va tomando los valores del conjunto uno por uno.***

***done***

- Ese ***conjunto*** que aparece en la estructura del ***for***, es normalmente un conjunto de valores cualesquiera, separados por espacios en blanco o retornos de línea.

# ESTRUCTURAS ITERATIVAS

**FOR**

- Vamos a mostrar los días de la semana por pantalla:

```
cat > dias.sh
#!/bin/bash
for DIA in lunes martes miércoles jueves sábado domingo; do
        echo Día de la semana : $DIA
done
```

- Resultado:

**Día de la semana : lunes**
**Día de la semana : martes**
**Día de la semana : miércoles**
**Día de la semana : jueves**
**Día de la semana : viernes**
**Día de la semana : sabado**
**Día de la semana : domingo**

# ESTRUCTURAS ITERATIVAS

**FOR**

- Vamos a mostrar los número del 1 al 10:

```
cat > numeros.sh
#!/bin/bash
for NUM in 1 2 3 4 5 6 7 8 9 10; do
        echo "NUM vale $NUM en este paso."
done
```

- Resultado:

**NUM vale 1 en este paso.**
**NUM vale 2 en este paso.**
**…**
**NUM vale 10 en este paso.**

# ESTRUCTURAS ITERATIVAS

**SEQ**

- Existe una orden en GNU/Linux que nos permite obtener una secuencia de números como salida de la orden, esta orden es ***seq***.

  ***seq último-número***

  ***seq primer-número último-número***

  ***seq primer-número incremento último-número***

- Modifiquemos el ejercicio de mostrar los números del 1 al 10 que hicimos anteriormente, usando esta vez la orden ***seq***.

```
for NUM in $( seq 10); do
        echo "NUM vale $NUM en este paso."
done
```

# ESTRUCTURAS ITERATIVAS

**FOR - SEQ**

- Vamos a realizar un ejemplo algo más complejo utilizando ***for*** y ***seq***.
- Vamos a crear un script llamado suma100.sh que nos va a decir por pantalla cuanto suman todos los números del 1 al 100, es decir, 1+2+3+4+5…+100. El total de la suma, no paso a paso.

```
cat > suma100.sh
        #!/bin/bash
        SUMA=0
        for NUM in $( seq 1 100); do
                let SUMA=SUMA+NUM
        done
        echo "Los números del 1 al 100 suman: " $SUMA
```

# ESTRUCTURAS ITERATIVAS

**FOR**

- Permite utilizar directamente rangos sin tener que usar la orden ***seq***.

```
for NUM in {1..5}; do
        echo NUM vale $NUM
done
```

- Otro cambio, que permite utilizar también incrementos en los rangos, de la siguiente manera:

```
for NUM in {1..50..5}; do     # de 1 a 50 con incremento de 5
        echo NUM vale $NUM
done
```

- Como vemos, podemos utilizar los rangos de la siguiente forma:

***{INICIO..FINAL}***

***{INICIO..FINAL..INCREMENTO}***

# ESTRUCTURAS ITERATIVAS

**FOR**

- El ***for*** de ***bash*** también permite utilizar el formato basado en trio de expresiones común al lenguaje C.

***for (( VARIABLE=inicio; condición-para-seguir; incrementamos ))***

- **EJEMPLO**: Un script que como salida nos muestre los números pares entre 2 y 40.

```
cat > pares.sh
#!/bin/bash
for (( NUM=2; NUM<=40; NUM=NUM+2 )); do
        echo $NUM
done
```

# ESTRUCTURAS ITERATIVAS

**FOR**

**EJERCICIO 8**:

- Hacer un script que nos pida el número de alumnos de una clase.
- Posteriormente irá pidiendo la nota de cada una de ellos para la asignatura de SOM.
- Al final indicará el número de aprobados, el número de suspensos y la nota media.
- Recuerda que se pueden utilizar dos estructuras, una dentro de otra.