PRÁCTICA 2

man:

Ctrl F: avanzar página
Ctrl B: retrasar página
Ctrl P: moverse una línea hacia arriba
Ctrl N: moverse una línea hacia abajo
/texto: busca el texto que se escribe a continuación (incluyendo los huecos en blanco) desde la primera línea mostrada en la pantalla en adelante.
?texto: busca el texto que se escribe a continuación (incluyendo los huecos en blanco) desde la primera línea mostrada en la pantalla hacia atrás.
n: siguiente elemento en la búsqueda
N: elemento previo en la búsqueda
v: lanza (si es posible) el editor por defecto para editar el fichero que estamos viendo
q Q :q :Q zz: sale del man

info

help

who: devuelve usuario y hora

rm: borra archivos y directorios con contenido

more: visualiza un archivo fraccionándolo una pantalla cada vez.

touch: crea/cambia fecha de un archivo de texto

file: muestra el tipo de archivo.

file <archivo>

mkdir/rmdir: crea/borra un directorio

head/tail: te muestra las primeras/últimas 10 líneas del contenido de un archivo.

ls: lista los contenidos de un directorio

ls <directorio>

-a lista de los archivos del directorio actual, incluidos aquellos cuyo nombre comienza con .
-c lista en formato multicolumna
-l formato largo

$ ls -l
-rw-r-—r-- 1 x00000 alumnos 23410 Mar 15 2009 programa.c

-r lista en orden inverso
-R lista subdirectorios recursivamente, además del directorio actual
-t lista de acuerdo con la fecha de modificación de los archivos

cp: copia el archivo1 en el archivo2. Si archivo2 no existe, se crea.

cp <archivo1> <archivo2>

cd: cambia de directorio.

Se usan las abreviaciones **.** y **..** para hacer referencia al actual y al padre, respectivamente.
El símbolo **~** es el directorio HOME.

cd <directorio>

sort: ordena, según un criterio elegido, el contenido de los archivos.

sort <archivo>

pwd: devuelve en qué directorio estás

mv: renombra archivos/directorios.

Puede mover de lugar un archivo/directorio.
mv fuente destino

cat:

- Muestra el contenido de un archivo o varios.
- Concatena archivos.
- Copia un archivo.
- Crea un archivo de texto.
- Muestra los caracteres invisibles de control

cat <archivo>

Si el archivo contiene caracteres no imprimibles, por ejemplo, un archivo ejecutable, la visualización del mismo suele dejar el terminal con caracteres extraños. Podemos restablecer el estado normal del terminal con: *$ setterm -r*

clear: limpia la terminal

PERMISOS

r:
archivos: read
directorios: se puede listar su contenidos

w:
archivos: write
directorios: podemos modificarlo

x:
archivos: execute
directorios: podemos acceder a él

-:
No hay permiso

Grupos:
u: user
g: group
o: others
a: todos los grupos de usuarios

METACARACTERES DE ARCHIVO

?: cualquier carácter simple en la posición en la que se indique.

***:** cualquier secuencia de cero o más caracteres

[]: designan un carácter o rango de caracteres que representan un carácter simple a través de una lista de caracteres o mediante un rango, en cuyo caso, mostramos el primer y último carácter del rango separados por un guión "-".

{}: sustituyen conjuntos de palabras separadas por comas que comparten partes comunes.

~: para abreviar camino absoluto (path) del directorio HOME.

PRÁCTICA 3

chmod: modificar permisos de archivos/directorios.

$ chmod <grupo>+/-<permiso> <archivo/directorio>

+ : permitir

- : denegar

$ chmod ug+rw archivo1

wc: número de líneas, palabras y bytes de un archivo.

echo: muestra mensaje inactivo por pantalla.

date: proporciona fecha y hora.

METACARACTERES DE REDIRECCIÓN

Dispositivo de entrada: 0

Dispositivo de salida: 1

Dispositivo estándar para la salida de errores u otra información: 2

Los metaracteres de redirección permiten alterar ese flujo y redireccionar la entrada estándar desde un archivo y redirigir la salida estándar y el error estándar hacia archivos, además de poder enlazar la salida de una orden con la entrada de otra permitiendo crear un cauce (*pipeline*) entre varias órdenes.

< <archivo> redirecciona la entrada de una orden para que la obtenga el archivo

$ cat archivo = $ car < archivo

> <archivo> redirige la salida de una orden para que la escriba en el archivo. Si ya existe, lo sobreescribe.

$ pwd
/home/users/quasimodo
$ ls
listado notas
$ ls > temporal
$ ls
listado notas temporal
$ cat temporal
listado
notas
$ pwd > temporal
$ cat temporal
/home/users/quasimodo
$ ls >> temporal
$ cat temporal
/home/users/quasimodo
listado
notas
temporal

&> <archivo> la salida estándar se combina con la salida de error estándar y ambas se escriben en el archivo.

$ ls
listado notas
$ cat practica
cat: practica: No existe el archivo o el directorio
$ cat practica > temporal
cat: practica: No existe el archivo o el directorio
$ ls
listado notas temporal
$ cat temporal
$ cat practica 2> temporal # también se puede poner cat practica &> temporal
$ cat temporal
cat: practica: No existe el archivo o el directorio

>> <archivo> funciona como **>** pero añade la salida estándar al final del contenido del archivo.

&>> <archivo> funciona como &> pero añade las dos salidas combinadas al final del archivo.

2> <archivo> redirige la salida de error estándar a un archivo.

| crea un cauce entre dos órdenes. La salida de una de ellas se utiliza como entrada de la otra.

*$ ls e**
ej1 ej31 ej32 ej4
$ ls -l e | head -2*
-rw-r-—r-- 1 quasimodo alumnos 23410 Mar 15 2010 ej1
-rw-r-—r-- 1 quasimodo alumnos 3410 May 18 2010 ej31

|& crea un cauce entre dos órdenes utilizando las dos salidas (estándar y error) de una de ellas como entrada de la otra.

METACARACTERES SINTÁCTICOS

Sirven para combinar varias órdenes y construir una única orden lógica.

; separador entre órdenes que se ejecutan secuencialmente.

$ cd dir1 ; ls
programa1
programa2
$ pwd
/home/users/quasimodo/dir1

( ) se usan para aislar órdenes separadas por ";" o por "|". Las órdenes dentro de los paréntesis son tratadas como una única orden.

$ date
Wed oct 6 10:12:04 WET 2010
$ pwd
/home/users/quasimodo
$ pwd ; date | wc
/home/users/quasimodo
1 6 27
$ (pwd ; date) | wc
2 7 48

&& separador entre órdenes, en la que la orden que sigue al metacarácter "&&" se ejecuta solo si la orden precedente ha tenido éxito (no ha habido errores).

$ pwd
/home/users/quasimodo
$ ls
listado notas
$ ls -l notas && pwd
-rw-r-—r-- 1 quasimodo alumnos 3418 Mar 15 201 notas
/home/users/quasimodo

$ ls -l notas && pwd
ls: notas: No existe el archivo o el directorio

|| separador entre órdenes, en la que la orden que sigue al metacarácter "||" se ejecuta solo si la orden precedente falta.

$ ls -l notas || pwd
-rw-r-—r-- 1 quasimodo alumnos 3418 Mar 15 2010 notas

$ ls -l notas || pwd
ls: notas: No existe el archivo o el directorio
/home/users/quasimodo

PRÁCTICA 4: Variables, alias, órdenes de búsqueda y guiones

VARIABLES

Tipos:

- de entorno o globales: son comunes a todos los shells. Para visualizarlas se hace con ***env*** o ***printenv***. Se usan las mayúsculas.
- locales: solo son visibles en el shell donde se definen y se les da valor. Para verlas se usa ***set***.

Contenidos:

- Cadenas: secuencia de caracteres.
- Números: se podrán usar en operaciones aritméticas.
- Constantes: su valor no puede ser alterado.
- Vectores o arrays: conjunto de elementos a los cuales se puede acceder mediante un índice. Normalmente el índice es un número entero y el primer elemento es el 0.

Creación y visualización:

$ variable=numero
$ echo $variable
numero

¡A cada lado del signo igual no debe haber ningún espacio!

Variable de tipo vector: los elementos se ponen entre paréntesis.

$ colores=(rojo azul verde)

Para acceder a uno de sus elementos:

$ echo ${colores[0]}
rojo

Variables <u>especiales o creadas al entrar el usuario:</u>
$BASH contiene la ruta de acceso completa usada para ejecutar la instancia actual de bash.
$HOME almacena el directorio raíz del usuario.
$PATH guarda el camino de búsqueda de las órdenes, este camino está formado por una lista de todos los directorios en los que queremos buscar una orden.
$? contiene el código de retorno de la última orden ejecutada (instrucción o guión).

Para **<u>borrar</u>** una variable se usa **unset** junto con el nombre de la variable.

Para **<u>crear una variable con ciertos atributos</u>** usamos **declare**.

-i para indicar que es numérica.
-p para ver los atributos.
-r indica que es de solo lectura.
-a indica que es una matriz (vector o lista).
-x indica que es exportable

$ declare -i IVA=18
$ declare -p IVA
declare -i IVA="18"
$ declare -i IVA=hola
declare -p IVA
declare -i IVA="0"

De esta forma cualquier intento de asignar otra cosa diferente de un número a la variable **no dará error.**

<u>Exportar variables:</u>
$ export *variable*
$ export *variable=valor*

<u>Significado de las diferentes comillas en las órdenes:</u>
Sustitución de órdenes: permite la ejecución de una orden, con o sin argumentos, de forma que su salida se trata como si fuese el valor de una variable.

Se puede hacer poniendo *$(orden argumentos)* o *`orden argumentos`*. Son equivalentes:

$ echo "Los archivos que hay en el directorio son: $(ls -l)"
$ echo "Los archivos que hay en el directorio son: `ls -l`"

Las **comillas dobles** se utilizan como mecanismo de acotación débil, para proteger cadenas desactivando el significado de los caracteres especiales que haya entre ellas, salvo los caracteres *!*, *$*, \ y `, que quedan desprotegidos.

También se pueden proteger cadenas usando **comillas simples** como mecanismo de acotación fuerte, aunque no se protege *!*.

$ echo 'En el libro de inglés aparece Peter's cat'
> #NO

$ echo 'En el libro de inglés aparece Peter'\''s cat'
En el libro de inglés aparece Peter's cat #SI

Asignación de resultados de órdenes a variables:

Con `.

variable=`orden`

$ listadearchivos=`ls .` #Ahora la variable contendrá la lista de todos los archivos existentes en el directorio actual.

Cuando se ejecuta la orden *cat <archivo>* y el archivo dado como argumento no existe, se produce error. Para depurar el correcto funcionamiento, hay que conocer el estado de la ejecución de la última orden, que valdrá 0 si se ejecutó correctamente, o 1 si hubo algún error.

$ cat archivomio
cat: archivomio: No existe el fichero o el directorio
$ echo $?
1

$ numero=1
$ numero=$numero+1
$ echo $numero
1+1

Como vemos, no se ha realizado la operación, todo se ha convertido en carácter. Para resolverlo, usamos **expr** con **apóstrofes inversos:**

$ numero=1
$ echo $numero
1
$ numero=`expr $numero + 1` #Con espacios en blanco
$ echo $numero
2

La orden empotrada printf:

Es mejor usar **printf** en vez de **echo**.

Printf imprime un mensaje en la pantalla utilizando el formato que se le especifica:

printf formato [argumentos]

Donde formato es una cadena que describe cómo se deben imprimir los elementos del mensaje. Tiene tres tipos de objetos:

- **Texto plano:** se copia en la salida estándar.
- **Secuencias de caracteres de escape:** son convertidos y copiados en la salida.
  - **\b** espacio atrás
  - **\n** nueva línea
  - **\t** tabulador
  - **\’** carácter comilla simple
  - \\ barra invertida
  - **\0n** n=número en octal que representa un carácter ASCII de 8 bits
- **Especificaciones de formato**: se aplican cada una a uno de los argumentos.
  - **%d** número con signo
  - **%f** número en coma flotante (decimal) sin notación exponencial
  - **%q** entrecomilla una cadena
  - **%s** muestra una cadena sin entrecomillar
  - **%x** muestra un número en hexadecimal
  - **%o** muestra un número en octal

EJEMPLOS:

$ printf "%10d\n" 25
25 #Imprime un número en una columna de 10 caracteres de ancho

$ printf "%-10d %-10d\n" 11 12
11 12 # Justificar a la izquierda si usamos un número negativo

$ printf "%10.3f\n" 15,4
15,400 #Si el número es decimal, la parte entera se interpreta como la anchura de la columna y el decimal como el número mínimo de dígitos.

$ printf "%d %d\n" 010 0xF
8 15 #Convertimos un número de octal o hexadecimal a decimal

$ printf "0%o 0x%x\n" 8 15
00 0xf #De decimal a octal/hexadecimal

$ printf "El valor actual del IVA es del %d\n" $IVA
El valor actual del IVA es del 18

#Podemos usar variables como argumentos de la orden printf. Por ejemplo, si queremos mostrar la *variable IVA, antes declarada, junto con un mensaje explicativo.*

Alias:

Se crean con la orden empotrada **alias** y se borran con la orden **unalias**.
Son útiles par definir un comportamiento por defecto de una orden o cambiar el nombre de una orden por estar acostumbrado a usar otro sistema.

Dentro de un alias y entre comillas podemos poner varias órdenes separadas por ";". Se ejecutarán secuencialmente.

Para ignorar un alias y ejecutar la orden original se antepone una barra invertida (\) al nombre del alias:

$ \ls -l $HOME

Órdenes de búsqueda: find y grep, egrep, fgrep:

find se utiliza para buscar por la estructura de directorios los archivos que satisfagan los criterios especificados.

f*ind <lista-de-directorios> [expresiones]*

donde *lista-de-directorios* es la lista de directorios a buscar, y las *expresiones* son los operadores que describen los criterios de selección para los archivos que se desea localizar y la acción que se quiere realizar cuando *find* encuentre dichos archivos.

En las expresiones se pueden usar los metacaracteres de archivo.

Los criterios se especifican mediante una palabra precedida por un guion, seguida de un espacio y por una palabra o número entero precedido o no por un + o un -. Criterios comunes:

1. Por el nombre del archivo: -name archivo.

$find / -name "-.c"*

2. Por el último acceso: -atime númerodías o númerosigno.

-atime 7 busca los archivos a los que se accedió hace 7 días.
-atime -2 busca los archivos a los que se accedió hace menos de 2 días.
-atime +5 busca los archivos a los que se accedió hace más de 5 días.

3. Por ser de un determinado tipo: -type carácter. Se usa la opción *f* para referirse a archivos regulares y la opción *d* para directorios.

$ find . -type f

4. Por su tamaño en bloques: -size número con o sin signo.

Si el número va seguido de la letra *c* el tamaño es dado en bytes.
-size 100 #Busca los archivos cuyo tamaño es de 100 bloques

Para negar cualquier operador de selección o acción se usa ***!*** que debe ir entre espacios en blanco y antes del operador a negar.

$ find / ! -user pat #Busca los archivos del directorio raíz que no pertenezcan al usuario llamado pat

También se puede especificar un operador u otro utilizando el operador **-o.** Este operador conecta dos expresiones y se seleccionarán aquellos archivos que cumplan una de las dos expresiones.

$ find . -size 10 -o -atime +2 #Busca los archivos de tamaño igual a 10 bloques o cuyo último acceso (modificación) se haya efectuado hace más de dos días

5. -print: visualiza los nombres de camino de cada archivo que se adapta al criterio de búsqueda.

$ find . -print #Visualizar los nombres de todos los archivos y directorios del directorio actual

6. -exec: permite añadir una orden que se aplicará a los archivos localizados.

La orden se situará a continuación de la opción y debe terminarse con un espacio, un carácter \ y a continuación un ;. Se utiliza {} para representar el nombre de archivos localizados.

$ find . -atime +100 -exec rm {} \; #Elimina todos los archivos del directorio actual (y sus descendientes) que no han sido utilizados en los últimos 100 días

7. -ok: es similar a -exec, con la excepción de que solicita confirmación en cada archivo localizado antes de ejecutar la orden.

grep permite buscar cadenas en archivos utilizando patrones par especificar dicha cadena.
Lee de la entrada estándar o de una lista de archivos especificados como argumentos y escribe en la salida estándar aquellas líneas que contengan la cadena.

grep opciones patrón archivos

El patrón puede ser una cadena de caracteres o también pueden usarse expresiones regulares.

*$ grep mundo ** #Buscar la palabra mundo en todos los archivos del directorio actual

Opciones:
-x localiza líneas que coincidan totalmente, desde el principio hasta el final de línea, con el patrón especificado.
-v selecciona todas las líneas que no contengan el patrón especificado.
-c produce solamente un recuento de las líneas coincidentes.
-i ignora las distinciones entre mayúsculas y minúsculas.
-n añade el número de línea en el archivo fuente a la salida de las coincidencias.
-l selecciona sólo los nombres de aquellos archivos que coincidan con el patrón de búsqueda.
-e especial para el uso de múltiples patrones e incluso si el patrón comienza por el carácter (-).

fgrep acepta solo una cadena simple de búsqueda en vez de una expresión regular.

egrep permite un conjunto más complejo de operadores en expresiones regulares.

GUIONES / SCRIPT / PROGRAMA SHELL

Es un archivo de texto que contiene órdenes del shell y del sistema operativo.

Diferentes invocaciones de variables con comillas simples, dobles y barra invertida:

GUION: (llamado imprimevar)
variable=ordenador
printf "Me acabo de comprar un $variable\n"
printf 'Me acabo de comprar un $variable\n'
printf "Me acabo de comprar un \$variable\n"

TERMINAL:
$ bash imprimevar
Me acabo de comprar un ordenador
Me acabo de comprar un $variable
Me acabo de comprar un $variable

Para no tener que poner $ bash , ponemos en el guion *#!/bin/bash.*

GUION: (prueba)
#!/bin/bash
...

TERMINAL:
$ ./prueba

Aclaración: Hemos antepuesto ./ al nombre de la orden por la siguiente razón: tal y como podemos observar, la variable $PATH (que contiene la lista de directorios donde el sistema busca las órdenes que tecleamos) no contiene nuestro directorio de trabajo, por lo que debemos indicarle al shell que la orden a ejecutar está en el directorio actual (.).

Variables de entorno definidas para los argumentos de un guion:

$0	nombre del guion que se ha llamado. Solo se emplea dentro del guion.
$1 .. $9	Son los distintos argumentos que se pueden facilitar al llamar a un guion. Los nueve primeros se referencian con $1,$2,...,$9, y a partir de ahí es necesario encerrar el número entre llaves, esdecir, ${n}, para n>9.
${n}, n>9	
$*	Contiene el nombre del guion y todos los argumentos que se le han dado. Cuando va entre comillas dobles es equivalente a "$1 $2 … $n".
$@	Contiene el nombre del guion y todos los argumentos que se le han dado. Cuando va entre comillas dobles es equivalente a "$1" "$2" … "$n".
$#	Contiene el número de argumentos que se han pasado al llamar al guion.
${arg:-val}	Si el argumento tiene valor y es no nulo, continua con su valor, en caso contrario se le asigna el valor indicado por val .
${arg:?val}	Si el argumento tiene valor y es no nulo, sustituye a su valor; en caso contrario, imprime el valor de val y sale del guion. Si val es omitida, imprime un mensaje indicando que el argumento es nulo o no está asignado.

Los argumentos por encima del 9 se suelen ponen entre llaves, por ejemplo, ${12}.

Normas de estilo:

Es buena costumbre comentarlo para conocer siempre quién lo ha escrito, en qué fecha, qué hace... Para ello se usa **#** al inicio de línea o tras una orden.

PRÁCTICA 5: Expresiones con variables y expresiones regulares

EXPRESIONES CON VARIABLES

Se evalúa una expresión aritmética y sustituye el resultado de la expresión en el lugar donde se utiliza. Ambas posibilidades son:

$((…))
$[…]

Lo que se ponga en … se interpretará como una expresión aritmética.

No es necesario dejar huecos en blanco entre los paréntesis/corchetes más internos y la expresión contenida en ellos.

Las expresiones aritméticas se pueden anidar.

Por ejemplo, la orden **date**, que permite consultar o establecer la fecha y la hora del sistema, y que admite como argumento +%j para conocer el número del día actual del año en curso, puede utilizarse para saber cuántas semanas faltan para el fin de año:

$ echo "Faltan $(((365 - $(date +%j)) / 7)) semanas hasta el fin de año"

Operadores aritméticos:

+ -	Suma y resta, o más unario y menos unario.
*** / %**	Multiplicación, división (truncando decimales), y resto de la división.
******	Potencia.
++	Incremento en una unidad. Puede ir como prefijo (primero se incrementa y luego se hace lo que se desee con ella) o sufijo (primero se hace lo que se desee y luego se incrementa) de una variable.
--	Decremento en una unidad. Puede ir como prefijo o sufijo de una variable.

() Agrupación para evaluar conjuntamente, permite indicar el orden en el que se evaluarán las subexpresiones o partes de una expresión.
, Separador entre expresiones con evaluación secuencial.
= x=expresion, asigna a x el resultado de evaluar la expresión.
+= x+=y → x=x+y
-= x-=y → x=x-y
= x=y → x=x*y
/= x/=y → x=x/y
%= x%=y → x=x%y

Bash solo trabaja con números enteros, por lo que si se necesitase calcular un resultado con **decimales**, habría que usar la orden **bc -l**

$ echo 6/5 | bc -l

Asignación y variables aritméticas:

Otra forma de asignar valor a una variable entera es con **let**. Aunque se usa para evaluar expresiones aritméticas.

l*et variableEntera=expresión*

Expresión debe ser una expresión aritmética.

Operadores relacionales:

La evaluación de una relación entre expresiones tomará finalmente un valor numérico:
1: true
0: false.

A = B → A == B → A -eq B

A != B → A -ne B

A < B → A -lt B

A > B → A -gt B

A <= B → A -le B

A>= B → A -ge B

! A

A && B

A || B

Operadores de consulta de archivos:

Usaremos **test** e **if.**

test expresión

Esta orden evalúa una expresión condicional y da como salida el estado 0 (true), o el estado 1 (false) o se le dio algún argumento no válido.

-a archivo	archivo existe.
-b archivo	archivo existe y es un dispositivo de bloques.
-c archivo	archivo existe y es un dispositivo de caracteres.
-d archivo	archivo existe y es un directorio.
-e archivo	archivo existe. Es igual que -a.
-f archivo	archivo existe y es un archivo plano o regular.
-G archivo	archivo existe y es propiedad del mismo grupo del usuario.
-h archivo	archivo existe y es un enlace simbólico.
-L archivo	archivo existe y es un enlace simbólico. Es igual que -h.
-O archivo	archivo existe y es propiedad del usuario.
-r archivo	archivo existe y el usuario tiene permiso de lectura sobre él.
-s archivo	archivo existe y es no vacío.
-w archivo	archivo existe y el usuario tiene permiso de escritura sobre él.
-x archivo	archivo existe y el usuario tiene permiso de ejecución sobre él, o es un directorio y el usuario tiene permiso de búsqueda en él.

archivo1 **-nt** archivo2	archivo1 es más reciente que archivo2, según la fecha de modificación, o si archivo1 existe y archivo2 no.
archivo1 **-ot** archivo2	archivo1 es más antiguo que archivo2, según la fecha de modificación, o si archivo2 existe y archivo1 no.
archivo1 **-ef** archivo2	archivo1 es un enlace duro al archivo2, es decir, si ambos se refieren a los mismos números de dispositivo e inode .

La orden **test expresión** es equivalente a la orden [expresión], donde los huecos en blanco entre los corchetes y expresión son necesarios.

```
$ test -d /bin          # comprueba si /bin es un directorio
$ echo $?               # nos muestra el estado de la última orden ejecutada, aunque
0                       # usado después de test o [ ] da 0 si la evaluación era verdadera

$ [ -w /bin ]           # comprueba si tenemos permiso de escritura en /bin
$ echo $?               # usado después de test o [ ] da 1 si la evaluación era falsa
1

$ test -f /bin/cat      # comprueba si el archivo /bin/cat existe y es plano
$ echo $?
0

$ [ /bin/cat -nt /bin/zz ]      # comprueba si /bin/cat es más reciente que /bin/zz
$ echo $?
0                               # la evaluación devuelve 0 porque /bin/zz no existe
```

Ejemplo: Vemos algunas características de archivos existentes en el directorio /bin y asignamos el resultado a una variable:

```
$ cd /bin
$ ls -l cat
-rwxr-xrx 1 root root 38524 2010-06-11 09:10 cat

$ xacceso=`test -x cat && echo “true” || echo “false”`      # se pueden omitir las “”
```

```
$ echo $xacceso
true                                                    # indica que sí tenemos permiso de
                                                        ejecución sobre cat

$ wacceso=`test -w cat && echo “true” || echo “false”`  # se pueden omitir las “”
$ echo $wacceso
false                                                   # indica que no tenemos permiso de
                                                        escritura en cat
```

Orden if / else:

La sintaxis de la orden condicional if es:

if condición;
then
declaraciones;
[elif condición;
then declaraciones;]...
[else
declaraciones;]
fi

Cada condición representa una lista de declaraciones, con órdenes, y no una simple expresión booleana. Como las órdenes terminan con un estado de finalización, condición se considera true si su estado de finalización (status) es 0, y false en caso contrario (estado de finalización igual a 1).

Puede darse un anidamiento.

El funcionamiento de la orden if es el siguiente:

se comienza haciendo la ejecución de la lista de órdenes contenidas en la primera condición;
- si su estado de salida es 0, entonces se ejecuta la lista de declaraciones que sigue a la palabra then y se termina la ejecución del if;
- si el estado de salida fuese 1, se comprueba si hay un bloque que comience por elif.

En caso de haber varios bloques elif, se evalúa la condición del primero de ellos de forma que si su estado de salida es 0, se hace la parte then correspondiente y termina el if,
pero si su estado de salida es 1, se continúa comprobando de manera análoga el siguiente bloque elif, si es que existe.

Si el estado de salida de todas las condiciones existentes es 1, se comprueba si hay un bloque else, en cuyo caso se ejecutarían las declaraciones asociadas a él, y termina el if.

La condición de la orden if puede expresarse utilizando la orden test para hacer una comprobación. De forma análoga se puede utilizar [...]; si se usa “if [expresión];”, expresión puede ser una expresión booleana y puede contener órdenes, siendo necesarios los huecos en blanco entre los corchetes y expresión.

Comparaciones aritméticas:

Con corchetes o doble paréntesis.
Los **espacios en blanco** a los lados son necesarios.
Con paréntesis no se pueden usar los operadores -eq, -ne, -le…

```
$ valor=34
$if [ $valor = 34 ]; then echo si; else echo no; fi
si
```

Comparaciones entre cadenas de caracteres

Los operadores < y > permiten comparar si una cadena de caracteres se clasifica antes o después de otra cadena siguiendo el orden lexicográfico.
Si hay espacios en blanco en la cadena, la variable se debe poner entre comillas.

$ valor=”hola amigos”
$ if [$valor == “hola amigos”]; then echo si; else echo no; fi
-bash: [: demasiados argumentos
no

$ valor=”hola amigos”
$ if [“$valor” == “hola amigos”]; then echo si; else echo no; fi
si

Comparaciones usando órdenes

Con if.
Podemos usar *test* dentro de la comparativa de una orden if.

EXPRESIONES REGULARES

Es un patrón que describe un conjunto de cadenas y que se puede utilizar para búsquedas dentro de una cadena o un archivo.

Para comparar con el carácter asterisco (*) este debe ir entre comillas simples (‘*’).

En las expresiones regulares se puede utilizar una barra inclinada invertida (\), denominada a veces como barra de escape, para modificar la forma en la que se interpretará el carácter que le siga. Cuando los metacaracteres “?”, “+”, “{”, “}”, “|”, “(” y “)” aparecen en una expresión regular no tienen un significado especial, salvo que vayan precedidos de una barra de escape; si se utilizan anteponiendo esa barra, es decir “\?”, “\+”, “\{”, “\}”, “\|”, “\(”o “\)”, su significado será el que corresponda.

Las reglas de precedencia indican que primero se trata la repetición, luego la concatenación y después la alternación, aunque el uso de paréntesis permite considerar subexpresiones y cambiar esas reglas.

Expresiones regulares con órdenes de búsqueda

Las órdenes de búsqueda son: find, grep y egrep.

find sirve para buscar a nivel de características generales de los archivos (nombres o condiciones de acceso, pero sin entrar en su contenido.

grep y **egrep** examinan la cadena que se le dé mediante la entrada estándar o el contenido de los archivos que se le pongan como argumento.

Ejemplo: buscar en el directorio /bin/usr los archivos cuyo nombre comience con las letras a o z y acabe con la letra m:

*$ find /usr/bin -name "[az]*m"*

Patrón	Representa
\	la barra de escape; si en un patrón se quiere hacer referencia a este mismo carácter, debe ir precedido por él mismo y ambos entre comillas simples.
.	cualquier carácter en la posición en la que se encuentre el punto cuando se usa en un patrón con otras cosas; si se usa solo, representa a cualquier cadena; si se quiere buscar un punto como parte de un patrón, debe utilizarse \. entre comillas simples o dobles.
( )	un grupo; los caracteres que se pongan entre los paréntesis serán considerados conjuntamente como si fuesen un único carácter. (Hay que usar \)
?	que el carácter o grupo al que sigue puede aparecer una vez o no aparecer ninguna vez. (Hay que usar \)
*	que el carácter o grupo al que sigue puede no aparecer o aparecer varias veces seguidas. (No hay que usar \)
+	que el carácter o grupo previo debe aparecer una o más veces seguidas.
{n}	que el carácter o grupo previo debe aparecer exactamente *n* veces. (Hay que usar \)
{n,}	que el carácter o grupo previo debe aparecer *n* veces o más seguidas. (Hay que usar \)
{n,m}	que el carácter o grupo previo debe aparecer de *n* a *m* veces seguidas; al menos *n* veces, pero no más de *m* veces. (Hay que usar \)
[ ]	una lista de caracteres que se tratan uno a uno como caracteres simples; si el primer carácter de la lista es "^", entonces representa a cualquier carácter que no esté en esa lista.
-	un rango de caracteres cuando el guion no es el primero o el último en una lista; si el guion aparece el primero o el último de la lista, entonces se trata como él mismo, no como rango; en los rangos de caracteres, el orden es el alfabético, pero intercalando minúsculas y mayúsculas – es decir: aAbB...-; en los rangos de dígitos el orden es 012... También es posible describir rangos parciales omitiendo el inicio o el final del rango (por ejemplo [m-] representa el rango que va desde la "m" hasta la "z").
^	indica el inicio de una línea; como se ha dicho anteriormente, cuando se usa al comienzo de una lista entre corchetes, representa a los caracteres que no están en esa lista. Situando a continuación de ^ un carácter, filtrará todas aquellas líneas que comiencen por ese carácter.
$	indica el final de una línea. Situando un carácter antes del $, filtrará todas aquellas líneas que terminen por ese carácter.
\b	el final de una palabra. (Debe utilizarse entre comillas simples o dobles)
\B	que no está al final de una palabra. (Debe utilizarse entre comillas simples o dobles)
\<	el comienzo de una palabra. (Debe utilizarse entre comillas simples o dobles)
\>	el final de una palabra. (Debe utilizarse entre comillas simples o dobles)
|	el operador OR para unir dos expresiones regulares, de forma que la expresión regular resultante representa a cualquier cadena que coincida con al menos una de las dos subexpresiones. (La expresión global debe ir entre comillas simples o dobles; además, cuando se usa con grep, esta orden debe ir acompañada de la opción -E)

Ejemplo: buscar en el directorio /etc los archivos de configuracion que contengan la palabra “dev”:

*$ find /etc -name “*dev*.conf*

Seguidamente se utiliza la **orden find con la opción -regex.** Si no se usa esta opción se interpretarán los símbolos como metacaracteres y no como parte de una expresión regular.

El ejemplo siguiente se ha resuelto de dos formas equivalentes y sirve para localizar en el directorio /usr/bin los nombres de archivos que contengan las letras cd o zip:

$ find /usr/bin -regex '.\(cd\|zip\).*'*

*$ find /usr/bin -regex '.*cd.*' -o -regex '.*zip.*'*

Ejemplo: Buscar en el archivo /etc/group las líneas que contengan un carácter k, o y o z:

$ grep [kyz] /etc/group

o

$ cat /etc/group | grep [kyz]

PRÁCTICA 6

for ejecuta una lista de declaraciones un número fijo de veces.
while ejecuta una lista de declaraciones cierto número de veces mientras cierta condición se cumple.
until ejecuta una lista de declaraciones repetidamente hasta que se cumpla cierta condición.
case ejecuta una de varias listas de declaraciones dependiendo del valor de una variable.

Lectura del teclado

read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-p prompt] [-t timeout] [-ufd] [name …]

Orden for

for nombre [in lista]
do
    declaraciones que pueden usar $nombre
done

Ejemplo: Cómo ver de qué tipo son todos los archivos de nuestro directorio de trabajo:

```
#!/bin/bash
for archivo in $(ls)
do
    file $archivo
done
```

Ejemplo: 5 primeros números impares mediante **seq**, que genera una secuencia de números desde el valor indicado como primer argumento (éste será el valor inicial), hasta el valor final dado como tercer argumento, tomando como incremento el segundo.

```
#!/bin/bash
for NUM in `seq 0 1 4`;
do
    let "NUM=$NUM * 2 + 1"
    printf "Número impar %d\n" $NUM
done
```

Ejemplo: Contar desde 1 a 9

```
#!/bin/bash
for (( CONTADOR=1; CONTADOR<10; CONTADOR++ )) ; do
    printf "Contador vale ahora %d\n" $CONTADOR
done
```

Orden case

```
case expresión in
    patron1 )
        declaraciones;;
    patron2 )
        declaraciones;;
    ...
esac
```

Ejemplo:

```
case "$REPLY" in
        1) rm "$TEMPFILE" ;;
        2) mv "$TEMPFILE" "$TEMPFILE.old" ;;
        *) printf "%s\n" "$REPLY no es una de las opciones" ;;
esac
```

Órdenes while y until

La orden **while** ejecuta las declaraciones comprendidas entre do y done mientras que la expresión sea true. Si falla en el primer intento, nunca se ejecutan.

```
        while expresión;
        do
                declaraciones ...
        done
```

Ejemplo:

```
while read -p "Archivo ?" ARCHIVO ;
do
        if test -f "$ARCHIVO" ;
        then
                printf "%s\n" "El archivo existe"
        else
                printf "%s\n" "El archivo NO existe"
        fi
done
```

Cuando el shell encuentra la orden **break**, sale del blucle (orden while) y continúa ejecutando la orden siguiente. La orden break puede venir seguida de un número que indica cuántos bucles anidados romper, por ejemplo, break 2.

```
while true ;
do
        read -p "Archivo ?" ARCHIVO
        if [ "$ARCHIVO" = "FIN" ] ; then
                break
        elif test -f "$ARCHIVO" ; then
                printf "%s\n" "El archivo existe"
        else
                printf "%s\n" "El archivo NO existe"
        fi
done
```

La orden **continue** permite iniciar una nueva iteración del bucle saltando las declaraciones que haya entre ella y el final del mismo.

```
for n in {1..9}
do
        x=$RANDOM# Devuelve un entero diferente entre 0 y 32676.
        [ $x -le 20000 ] && continue
        echo "n=$n x=$x"
done
```

La orden **until** es similar a while, excepto que se repite el cuerpo hasta que la condición sea cierta.

```
until expresión;
do
        declaraciones ...
done
```

Ejemplo: generar los 10 primeros números:

```
n=1
until [ $n -gt 10 ]
do
        echo "$n"
        n=$(( $n + 1 ))
done
```

Construcción de menús:

En la expresión del bucle while aparece la orden : que es equivalente a “no operación”.

Una vez elegida la opción dentro de la orden case, hacemos que, mediante la orden sleep 2, espere durante 2 segundos hasta que vuelva a solicitarnos una de las opciones del menú.

```
#!/bin/bash
while :
do
        printf "\n\nMenu de configuración:\n"
        printf "\tInstalar ... [Y, y]\n"
        printf "\tNo instalar [N, n]\n"
        printf "\tPara finalizar pulse 'q'\n"

        read -n1 -p "Opción: " OPCION
        case $OPCION in
                Y | y ) printf "\nHas seleccionado %s \n" $OPCION;;
                N | n ) printf "\nHas seleccionado %s \n" $OPCION;;
                q ) printf "\n"
                        break;;
                * ) printf "\n Selección inválida: %s \n";;
        esac
        sleep 2 # duerme durante 2 segundos
done
```

FUNCIONES

Se ejecutan dentro de la memoria del proceso bash que las utiliza, por lo que su invocación es más rápida que invocar a un guion u orden de Unix.

```
function nombre_fn {
        declaraciones
}
```

```
nombre_fn() {
        declaraciones
}
```

Para borrar: **unset -f nombre_fn**

Para ver qué funciones tenemos definidas:

- junto con su definición: **declare -f**
- junto con su nombre: **declare -F**

Orden de **preferencia** a la hora de resolver un símbolo:
1. Alias
2. Palabra clave (if, function, etc.)
3. Funciones
4. Órdenes empotradas
5. Guiones y programas ejecutables

Las funciones se nombran con un signo "_" delante del nombre (_función).

Una función con el mismo nombre que una orden empotrada se ejecutaría antes que el guion.

Variables locales en funciones. Parámetros

Para invocar una función dentro de un guion solamente debemos escribir su nombre seguido de los argumentos correspondientes, si los hubiera.

Una función puede tener variables locales. Para ello, debemos declararlas dentro de la función con el modificador local.

```
# definimos la función _uso()
_uso()
{
        echo "Uso: $0 nombre_archivo"
        exit 1
}
# definimos la función _si_existe_file
_si_existe_file()
{
        local f="$1"
        [ -f "$f" ] && return 0 || return 1      # $f almacena los argumentos pasados al guion
}
# llamamos a _uso() si no se da el nombre de archivo
[ $# -eq 0 ] && _uso
# invocamos a _si_existe_file()
if ( _si_existe_file "$1" )
then
        echo "El archivo existe"
else
        echo "El archivo NO existe"
fi
```

Ejemplo: En el ejemplo siguiente, la función devuelve el doble de un número que se pide por teclado; en la línea 10 recogemos el valor devuelto por _dbl en la variable $resultado:

```
function _dbl
{
        read -p "Introduzca un valor: " valor
        echo $[ $valor * 2 ]
}
resultado=`_dbl`        #Observad las ` `
echo "El nuevo valor es $resultado"
```

Archivos de configuración

Existen diferentes archivos de configuración que son leídos por el shell cuando se lanza, son los archivos de arranque. Estos guiones tienen como objetivo establecer la configuración del shell definiendo variables, funciones, alias, etc.

Los archivos de arranque ejecutados van a depender del tipo de shell. El shell que aparece cuando arrancamos la máquina se denomina login shell (desde el que ejecutamos startx). Este shell utiliza los archivos:

- /etc/profile
- $HOME/.bash_profile
- $HOME/.bash_login
- $HOME/.profile
- $HOME/.bashrc
- $HOME/.bash_logout

El primero de ellos es un archivo del sistema y, por el directorio donde se encuentra y los permisos asignados, sólo puede ser modificado por el administrador. El resto de archivos de configuración pueden ser modificados por el usuario para adaptar la shell a sus necesidades.

Orden tar

Permite almacenar/extraer varios archivos de otro archivo.

tar -cvf <archivo.tar> <directorio>

La opción -x extrae los archivos de un archivo.tar

Orden gzip

Permite comprimir el contenido de un archivo para que ocupe menos espacio.

g*zip <archivo>*

El archivo comprimido tendría el mismo nombre pero con la extensión .gz

Para descomprimir un archivo .gz o .zip: **gunzip.**