ORDENES DE LINUX (I)

1.1. - Sintaxis de las ordenes

Dentro de cada orden tenemos que distinguir entre comandos, indicadores y parámetros. La sintaxis general de toda orden es:

comando [-indicadores] [parámetro(s)]

Para ejecutar un comando sólo se tiene que escribir el nombre de dicho comando que es en realidad un archivo que se encuentra bajo el directorio /bin (esto lo veremos más adelante). Algunos comandos no son archivos independientes. Estos comando están integrados en los propios shells (command.com del MS-DOS). Por ejemplo, el comando CD (cambiar de directorio) está integrado en el shell y es ejecutado directamente por el intérprete de comandos sin tener que buscar un archivo.

Por ejemplo: `$ls`

Este comando lista el nombre de los archivos y directorios en orden alfabético del directorio actual.

Los **indicadores** también se les llama **opciones** son letras independientes precedidas de un guión (-) que modifican el comportamiento de un comando. Por ejemplo la orden "ls" si le añadimos un indicador como (-l) se puede listar el contenido del directorio actual de forma diferente a la vista en el caso anterior. En este caso estamos visualizando el nombre de los archivos con todos sus atributos. Se trata de un listado largo.

Cuando se quiere utilizar más de un indicador sólo hay que añadirlos uno a continuación del otro. Solo se pone el signo (-) al principio, aunque se puede poner en cada uno de los indicadores.

Por ejemplo: `$ls -l -a` que también es equivalente a: `$ls -la`

No puede haber espacio entre el guión y la opción. Y en caso de que existan subopciones siempre deben acompañar a su opción aunque estén separadas por un guión.

Los **parámetros** normalmente son nombres de archivos o cadenas que dan instrucciones al comando para que realice una función. Si un parámetro contiene un espacio en intercalado debe situarse la cadena entre comillas a fin de evitar que el shell la amplíe.

Por ejemplo: `$grep New York`

esta orden trata de buscar la palabra `New` en el fichero `York`

Si el objetivo consiste en encontrar la cadena `"New York"` en la entrada estándar, el comando deberá escribirse así:

```
$grep "New York"
```

y en ese caso la cadena `New York` se pasa al comando `grep` como un parámetro.

1.2. - Expansión de la línea de órdenes

Existe una técnica por la cual podemos hacer referencia a un grupo de archivos simultáneamente, utilizando los llamados caracteres comodines o referencias ambiguas.

Los comodines son caracteres mediante los que podemos simbolizar parte del nombre de un archivo haciendo referencia a más de uno al mismo tiempo. Los comodines * y ? del DOS funcionan de modo idéntico en LINUX.

- Con * podemos simbolizar parte de un nombre, o un nombre completo.
- Con ? hacemos referencia a un sólo carácter, pudiendo repetirse tantas veces como caracteres queramos sustituir.

Ejemplos:

***arch** Identifica cualquier archivo que acabe con arch, incluido arch.

arch* Identifica cualquier archivo que comience por arch, incluido arch.

arch Identifica cualquier archivo que contenga la cadena arch.

arch? Identifica cualquier archivo que comience por arch seguido de un carácter

Ejemplos utilizando el comando **cat**:

`$cat *` Visualiza todos los archivos del directorio

`$cat LE*` Idem que empiecen por LE. Por ejemplo LE, LEI, LEEME

`$cat ?ota` Idem que contengan una letra seguida de ota. Por ejemplo nota, sota, rota. No visualizará el fichero ota.

Además tenemos la posibilidad de especificar rangos, para lo que dispondremos, entre corchetes, el rango de caracteres que deseemos utilizar.

`$ls fich[1-9]` lista los ficheros de nombre fichl, fich2, ...fich9

`$ls fich[19]` Idem para fich1 y fich9

También sería válida una orden como:

```
$ls [LMN][Ee][Ee][Mm][AE]
```

Que daría una lista de aquellos ficheros cuyo nombre tenga 5 letras, siendo la primera cualquiera de L, M o N; la segunda y tercera, E o e; después Mm; y por último, una A o una E.

En una misma orden se puede incluir cualquier número y tipo de comodines, siendo afectados por la orden sólo aquellos archivos que cumplan con la plantilla.

Una orden ideal para ver en pantalla los archivos a los que afectaría una referencia compuesta por caracteres comodines es **echo**. Esta orden se utiliza para imprimir cualquier parámetro que se dé a continuación, pero en el caso de que contengan caracteres comodines se intentará buscar nombres de archivos existentes que los cumplan. Es aconsejable utilizar esta orden para ver los archivos que se verían afectados antes de ciertas operaciones, como la de borrado.

Veamos un ejemplo:

```
$ls
     archivo archivo1 archivo90 archivoa archivo3
$echo archivo*
     archivo archivo1 archivo90 archivoa archivo3
$echo archivo?
     archivo1 archivoa archivo3
$echo archivo[1a]
     archivo1 archivoa
$echo archivo[1-9]
     archivo1 archivo3
```

Para evitar la interpretación de los comodines:

- El carácter de escape \ precediendo al comodín:

  ```
  $echo  \  *
  ```

- Encerrando la cadena entrecomillas simples:

  ```
  $echo 'No puedo ver $HOME ni * '
  ```

- Encerrando la cadena entre dobles comillas: se evita la interpretación de lo que aparece en su interior, menos el $:

  ```
  $echo "puedo ver $HOME, pero no * "
  ```

Ejemplos:

```
$echo ABC'\*'
$echo "El símbolo del prompt es \$PS1"
```

1.3. - Algunas ordenes sencillas

ls

Nos permite ver el contenido del directorio en el que nos encontramos, se utiliza la orden ls (list). La salida podría ser algo así:
Sintaxis:

```
$ls
      nota
      LEEME
      fic.hero
```

que indica que en el directorio en el que nos encontramos hay 3 ficheros llamados nota, LEEME, y fic.hero. Obsérvese que no existen las extensiones en el sentido del DOS. El carácter "." es un carácter como otro cualquiera en el nombre del fichero.

Si no hay ficheros en el directorio, LINUX no dice nada. Esta es una característica general del sistema.

```
$ls
$
```

la opción -l (largo) de la orden ls proporciona más información que el nombre de los ficheros.

```
$ls -l
      total 4
       -rw-r--r--  1   User  Group   123 Nov 16  10:29  nota
       -rw-r--r--  1   User  Group    85 Jul 12  17:25  LEEME
       -rw-r--r--  1   User  Group   780 Ago 26  11:35  fic.hero
```

1er campo	2º	3º	4º	5º	6º	7º	8º

Veamos el significado que tiene esta salida:

1er campo:	atributos de permisos
2º campo:	número de enlaces o entradas de directorio
3º campo:	propietario del fichero
4º campo:	grupo de usuarios al que pertenece
5º campo:	tamaño en bytes
6º campo:	fecha de la última modificación
7º campo:	hora de la última modificación
8 campo:	nombre del fichero

Con la opción `-a` se visualizan los ficheros ocultos. Un fichero oculto se caracteriza por tener como primer carácter del nombre del fichero el carácter “.”

Estudiaremos más adelante el significado de alguno de los campos (atributos enlaces, etc.). La primera línea (total 4) nos muestra la cantidad de bloques o porciones de disco que utilizan esos ficheros. El bloque es la unidad de almacenamiento en disco.

Otras opciones de la orden ls

- **-d** Trata cada entrada como un directorio
- **-t** Clasifica por fecha y hora de la última modificación.
- **-u** Clasifica por fecha y hora del último acceso.
- **-r** Invierte el orden de clasificación
- **-i** Muestra el número de nodo-i (también llamado inode) del archivo
- **-A** Evita que en el listado se impriman los . y .. de directorio
- **-n** Relaciona los números de identificación de grupo y usuario, en lugar de nombres, asociados con cada archivo y directorio.
- **-s** Proporciona el tamaño de cada archivo en bloques.
- **-x** Lista los archivos en varias columnas ordenadas alfabéticamente de izquierda a derecha
- **-m** Genera la lista con los nombres separados por comas.
- **-R** Lista recursivamente. Si en el directorio existen subdirectorios, también los listará junto con los archivos que contiene.

help

Muestra información breve sobre la orden especificada incluyendo descripción, sintaxis y opciones.
Sintaxis:

```
$help  orden
```

man

Muestra información detallada sobre la orden especificada con observaciones, referencias a otras órdenes, etc.
Sintaxis:

```
$man orden
```

who

Nos permite saber los usuarios que hay conectados al sistema. Esto es conveniente para poder comunicarse, compartir ficheros, etc.
Sintaxis:

```
$who
      rosa          tty1    Mar 23   9:50
      angel         tty2    Mar 23   10:05
```

Los cuatro campos de salida de who significan:

1º nombre de usuario
2º número de terminal por el que está conectado
3º fecha de conexión
4º hora de conexión

Sintaxis:

```
$who am i
```

informa de la cuenta con la que hemos entrado y el terminal por el que lo hemos hecho.

passwd

Cambio de contraseña.

El identificador de usuario y la palabra clave son la base de la seguridad y privacidad del usuario. Mientras que si interesa que los demás conozcan nuestro identificador para poder recibir correo, mensajes, etc. es conveniente mantener en secreto la clave y cambiarla a menudo.

Linux guardará esta clave encriptada, de tal modo que si se nos olvida, ninguno, ni el superusuario, podrá adivinarla. No obstante, el superusuario puede asignar una nueva clave o eliminarla. Además, también puede entrar en cualquier cuenta de usuario sin necesidad de contraseña.

La contraseña consta de al menos 6 caracteres. Es conveniente incluir un dígito numérico o un carácter no alfabético. En caso contrario, Linux podría no aceptarla.

La orden que permite asignar o cambiar la contraseña a un usuario es:

`$ passwd nombre` si eres superusuario.
`$ passwd` para cambiarse cada uno su clave.

La acción que realiza por defecto es cambiar la del propio usuario.

Si el usuario no tenía palabra de paso asignada, Linux pedirá introducir la contraseña actual y la nueva palabra de paso.

Si en la confirmación no coincide con el password tecleado en "New password" Linux manda el mensaje de login incorrecto.

Consejos: la contraseña debe elegirse de tal modo que sean fáciles de recordar pero difíciles de averiguar. Nunca se debe escoger una palabra que exista en el diccionario de cualquier idioma, ni tampoco un nombre de persona. Asimismo, evite la utilización de un número de identificación asociado a nosotros, por ejemplo, matrícula de coche, DNI, etc.

Un excelente truco para elegir contraseñas es coger dos palabras no relacionadas y unirlas con un carácter especial o elegir la primera letra de una frase, por ejemplo:

Cuatro gatos y sólo quedan tres: `4g&sq3` ó `libro%coche`

Por último recordar que Unix también diferencia mayúsculas de minúsculas en una contraseña.

Todas las contraseñas se guardan en el archivo `/etc/passwd`

cal

Muestra un calendario en la salida standard.
Sintaxis:

$cal [-m][-j][-3][-y] [[mes] año]

-m Muestra el lunes como primer día de la semana.
-j Muestra fechas julianas. Del 1 (1 de enero) al 365 o 366 si es bisiesto (31 de diciembre).
-3 Muestra el mes anterior, el actual y el siguiente
-y Muestra una agenda del año en curso.
mes Puede ser un número del 1 al 12 o un conjunto de letras suficiente para determinar el mes. El valor predeterminado es el mes en curso.
Año Puede ser cualquier número entre 1 y 9999. El valor predeterminado es el año en curso. Hay que especificar los cuatro números correspondientes al año.

Ejemplo:

`$cal -j 96` no visualiza nada del año 96.
Debería poner `$cal jun 1996` ó `$cal 6 1996`

date

Muestra la hora y fecha actuales
Sintaxis:

```
$date
      Sun Jun 1 11
      26:53 MDT 1997
```

Además date puede formatear la salida a diferentes formas. Para ello debe llevar un argumento que comience con el signo + y, a continuación, una cadena de formato del estilo de la que se utiliza con printf() en Lenguaje C, con caracteres de formato que van precedidos por el símbolo %. Algunos son:

%d	día del mes (1-31)	%S	segundos
%y	2 últimos dígitos del año	%B	mes(Enero-Diciembre)
%Y	4 dígitos del año	%r	hora en notación AM/PM
%T	hh:mn:ss	%n	nueva línea
%H	hora	%a	día de la semana abreviado
%M	minutos	%A	día de la semana completo

Ejemplo:

```
$date "+Hoy es %d de %B - Hora: %r"
      Hoy es 1 de junio - Hora: 12:07:24 PM
```

El superusuario puede cambiar la hora del sistema mediante :

```
$date MMDDhhmmAA
```

siendo:

MM	mes
DD	dia
hh	hora
mm	minuto
AA	año

Ejemplo :

```
#date 0601121197
      Wed Jun 1 12:11:53 MDT 1997
```

time

Determina el tiempo que tarda un comando en ejecutarse.
Sintaxis:

```
$time [-opciones]  comando
```

time informa sobre tres tiempos:

- t real: El tiempo total transcurrido desde que ha invocado el comando. Se le denomina a veces como tiempo de reloj, porque es tiempo que ha transcurrido en nuestro reloj.
- t user: La cantidad de tiempo actualmente consumido en la CPU fuera del tiempo sys.
- t sys: La cantidad de tiempo consumido en el kernel, que es el tiempo empleado en contestar peticiones del sistema.

Opciones:

- '-a' Valor por omisión. Muestra la información en una línea.
- '-p' Muestra la información en tres líneas. Una para *real* otra para *user* y otra para *sys*.
- '-v' Saca información muy detallada del comando que se ejecuta, los tiempos, etc.
- '-V' Muestra la versión del comando time.

uname

Imprime el nombre del sistema.
Sintaxis:

```
$uname     [-opciones]
```

Imprime el nombre del sistema donde está trabajando por la salida estándar. Esta es información necesaria cuando se trabaje con comunicaciones linux-linux.

Opciones:

- '-s' Imprime nombre del sistema operativo. (Por omisión)
- '-n' Imprime nombre nodo dentro de una red de comunicaciones. (El hostname)
- '-r' Imprime la revisión del sistema operativo.
- '-v' Versión del sistema operativo. (Fecha y hora)
- '-m' Presenta el nombre del microprocesador sobre el que se ejecuta.
- ' -a' Imprime toda la información anterior.

logname

Muestra el nombre de usuario o "Identificador login" con el que se ha accedido al sistema.
Sintaxis:

```
$logname
```

clear

Borra la pantalla y deja el indicador en la parte superior izquierda de la misma.
Sintaxis:

```
$clear
```

Truco: Una forma de escribir menos es utilizando la tecla de tabulación.
Por ejemplo: Si queremos teclear *more directorio/larguisimo.soy/joselito* una forma de hacerlo más corto sería: *mor<tab>direc<tab>larg<tab>jos<tab>* . Tal y como va encontrado ocurrencias que sean identificables de forma única, irá escribiendo en pantalla. Si encuentra más de una ocurrencia que puede darse no escribe y al volver a pulsar <tab> saca la lista de posibilidades.

tty

Muestra el controlador de terminal asignado y su ruta de acceso.
Sintaxis:

```
$tty
```

cat (Concatenar)

Visualiza por la salida estándar el contenido de uno o varios archivos.
Sintaxis:

```
$cat  [-opciones]   fichero1 fichero2 .......
```

El funcionamiento de esta orden es el siguiente: Lee secuencialmente cada archivo de entrada y lo escribe en la salida estándar, la referencia de la orden concatenar establece un grupo de archivos en una sola salida. Otros usos de esta orden permiten: crear, copiar, concatenar o encolar los datos contenidos en los ficheros.

Opciones :

- '-b' Numera todas las líneas de salida, menos las que están en blanco, comenzando por 1.
- '-n' Numera todas las líneas de salida.
- '-s' Reemplaza múltiples líneas en blanco adyacentes en una sola línea.
- '-v' Muestra los caracteres de control, y caracteres de 8 bits como 'M-'.
- '-T' Muestra las tabulaciones con ^I.
- '-E' Visualiza '$' al encontrar un retorno de carro.
- '-e' Equivalente a -vE
- '-t' Equivalente a -vT
- '-A' Equivalente a -vET

Si editamos un fichero binario, la configuración del teclado puede perderse, para solucionar este problema pulsaremos *ctrl+v* y *ctrl+o*.

Variantes en el uso de cat:

cat	Si se utiliza sin argumentos, cat espera a que el usuario introduzca caracteres por el teclado hasta que se pulsa CTRL-D. Las líneas introducidas se reproducen en la salida estándar, (produce el efecto eco).
cat fichero	Es el uso normal de esta orden. Muestra el contenido del fichero en la salida estándar.
cat fichero1 fichero2	Visualiza varios ficheros.
cat > fichero	Crea un fichero con lo que se escriba desde el terminal. Se termina con ^D. Si el fichero existe, borra su contenido y lo graba de nuevo.
cat fichero1 > fichero2	Copia fichero1 en fichero2. Si había algo en fichero2 desaparecerá.
cat fichero1 fichero2 > fichero3	Concatena fichero1 y fichero2 y los almacena en fichero3, permaneciendo invariables los dos primeros.
cat fichero1 >> fichero2	Encola o añade fichero1 en fichero2, el primero permanece intacto.
cat >> fichero1	Añade al final del fichero
cat fichero | more	Si el archivo a visualizar es muy grande, podemos paginar la salida.

1.4. – Redirección de la Entrada / Salida

Hasta ahora hemos introducido ordenes tecleándolas en el terminal y la salida de las ordenes se ha visualizado también en el terminal.
En realidad, la mayoría de las ordenes toman la entrada desde cualquier canal de entrada y escriben su salida sobre un canal de salida. Es decir, las ordenes se implementan de tal modo que leen una secuencia de caracteres como entrada y producen una secuencia de caracteres como salida. Estas secuencias de caracteres de entrada y salida se denominan flujos, ya que no hay realmente una estructura interna en las cadenas de caracteres. La orden tan solo ve una larga secuencia de caracteres como entrada.

El canal de entrada para una orden se denomina entrada estándar y el canal de salida se denomina salida estándar.

La entrada de una orden por defecto proviene del teclado del terminal y la salida va a ala pantalla del terminal. Sin embargo se puede redirigir la entrada y la salida estándar a un fichero.

En Linux existen 3 archivos estándar que se refieren a la E/S por defecto que utilizan los programas.

Descriptor	Nombre	Asignación por defecto
0	Entrada estándar (stdin)	Teclado
1	Salida estándar (stdout)	Pantalla
2	Error estándar (stderror)	Pantalla

Linux permite cambiar las asignaciones por defecto de estos archivos stdin, stdout y stderr mediante las redirecciones de E/S.

Los operadores de redirección son:

- < redirección de entrada.
- > redirección de salida.
- >> redirección y adición de salida.

Para redirigir la entrada estándar, haciendo que provenga de un fichero en lugar del terminal, se utiliza el símbolo <.

Ejemplo:

```
$cat < arch1          visualiza el contenido de arch1
```

El carácter > indica al shell que redirija la salida estándar a un fichero. En este caso si el fichero no existe se creará y si existe su contenido anterior se vaciará.

Ejemplo:

```
$ls -l  > arch2       el resultado del listado largo se guardará en arch2
$ls  > tmp            el resultado del listado se guardará en tmp
```

Utilizaremos el carácter >> para redirigir añadiendo a la salida.

Ejemplo:

```
$cat arch1 >> arch2
```
redirecciona el contenido de arch1 añadiéndose al contenido de arch2

```
$ls >> tmp
```
el resultado del listado se añade al final de tmp.

Redirección de la entrada y la salida:

```
$cat <nota >n.copia
```
Este ejemplo hace que el shell ejecute la orden cat con el fichero nota como entrada y que envíe la salida al fichero n.copia.

```
$cat tmp - carta < texto > carta1
```

La E/S estándar y la redirección de canales están entre las características más generales y poderosas del sistema Linux y tienen algunas implicaciones importantes.

La mayoría de las ordenes están diseñadas para trabajar como simples flujos de bytes como entrada y salida, para hacer la redirección tan potente como sea posible. Casi todos los ficheros son sencillos flujos de bytes, sin la estructura interna que puede verse en sistemas de ficheros orientados a registros. El significado de un fichero viene determinado únicamente por la aplicación o por el usuario que utiliza el fichero, y no está forzado por ninguna propiedad del sistema. Puesto que la mayoría de las ordenes utilizan entrada y salida estándar, hay modos consistentes y fiables de utilizar la E/S. En la mayoría de los casos, generalmente se pueden intercambiar los dispositivos de teclado, los ficheros y los dispositivos hardware en las ordenes.

Hay ordenes que permiten mezclar ficheros y entrada estándar en la misma orden.

Ejemplo:

```
$cat nota LEEME n.copia
$cat nota  -  n.copia < LEEME
```
cat procesa primero el fichero nota, toma luego la entrada estándar que fue redirigida desde el fichero LEEME y procesa por último el fichero n.copia.

El – (menos) sólo como argumento, sin letra de opción asociada, se suele utilizar en las ordenes para significar “tomar la entrada estándar en este punto”.

Redirección de la salida de error:

La mayoría de las ordenes utilizan el canal de error estándar para mostrar los mensajes de error o la salida inusual que no debería aparecer en el canal de salida estándar.

Se puede redirigir el error estándar utilizando el operador 2> o 2>> dependiendo si se desea crear un nuevo fichero o añadir datos a un fichero existente.

Esta notación se debe a que los canales de E/S estándar son números asignados. Estos números son utilizados principalmente por programadores al desarrollar software, pero en este caso el número 2 ha surgido en el acceso al shell a nivel de usuario.

Ejemplo:

```
$cat arch1  2>  error
$cat arch1  arch2  > salida  2> error
$cat arch1  arch2  > salida  2> salida
```
¿qué ocurre aquí?

```
$cat arch1  arch2  > salida  2> &1
```
se redirige la salida de error a la salida estándar

```
$cat arch1  arch2 2> salida  1> &2
```

1.5. - Filtros

Las tuberías " | " no están limitadas a dos ordenes. Se pueden construir tuberías de cualquier longitud, sin más que asociar la salida standard de una orden a la entrada standard de la siguiente.
El sistema LINUX incluye muchas ordenes destinadas a ser utilizadas de este modo. Estas ordenes se denominan con frecuencia filtros, ya que transfieren su entrada a su salida, modificándola durante su paso a través del programa (orden). Los cambios particulares efectuados por una orden de filtrado dependen del tipo de orden.

Los filtros leen el fichero de entrada y escriben el resultado de su operación sobre la salida estándar.
Cada orden de gestión de información o filtro realiza unos cambios propios sobre la información del fichero de entrada, que dependen del programa específico utilizado.
En general, los filtros nunca modifican el fichero de entrada, simplemente escriben el resultado en la salida estándar. Si se desea que el resultado se almacene en otro fichero deberá redireccionarse esta salida.
Existe gran variedad de filtros, desde los más sencillos que operan ligeras modificaciones, hasta los más potentes que llevan a cabo operaciones complejas de manipulación de información, próximas a las manipulaciones de los editores de texto o de las bases de datos.

Resumiendo: Los filtros son ordenes destinadas a filtrar y/o modificar parte de la entrada en la salida standard.

Tuberías o pipes

Los tubos permiten conectar la salida de una orden con la entrada de otra.
Se utiliza el símbolo |
Le estamos diciendo al shell que tome la salida estándar de un proceso y la utilice como entrada estándar de otro proceso.

```
$who   |   wc  -l
```

El tubo es equivalente a crear un archivo temporal intermedio:

```
$who  >  temporal
$wc  -l  <  temporal
$rm  temporal
```

La ejecución de las ordenes que componen la tubería es concurrente.
Pueden combinarse con las redirecciones. Hay que ir con cuidado.

```
$cat < arch1   |   grep  'abc'   |   wc  -l  >  arch2
```

La orden tee permite bifurca la salida estándar en dos, de forma que una la entubamos a otra orden y la otra la guardamos en un archivo.

```
$who   |   tee  archivo   |   wc  -l
```

Notas:
¿Existe diferencia entre > y | ? El operador > debe ir seguido siempre de un archivo, y el operador | debe ir seguido de un proceso.

Operadores ; & ` ()

- Operador **;** Para utilizar varias ordenes en una misma línea, separándolas con el carácter "**;**". La ejecución es secuencial. Hasta que no termine la ejecución de la orden a la izquierda de **;** no se lanzará la ejecución de la orden a la derecha del operador **;**

  Ejemplo:
  ```
  $ls -la ;  echo "Hola" ;   clear
  $cat archivo  ;  who   ;   ls
  ```

  En este caso las órdenes no están interconectadas como sucede con los filtros, por lo que se trata de órdenes totalmente independientes.

- Operador **&** Indica ejecución en modo subordinado. La ejecución es simultánea.
  ```
  $ls   &   sleep 5
  $sleep 5   &   ls
  $sleep 5   &   ls   &   who  &
  ```

- Operador grave **`** Sustituye el comando encerrado entre las comillas graves por la salida generada por él.
  ```
  $mail `cat lista`  <  carta
  ```

- Operador **( )** Si encerramos parte de la línea de comandos entre paréntesis el shell crea un subshell que interpretará esa parte de línea y lanzará los procesos implicados. Comparar la ejecución de:
  ```
  $sleep 5  ;  pwd  &
  $(sleep 5  ;  pwd)  &
  ```

Los paréntesis también son útiles cuando hacemos redirecciones. Ejemplos:
```
$sort  |  grep  abc  |  wc  -l  <  archivo
$sort  <  archivo  |  grep abc  |  wc  -l
$(sort  |  grep  abc  |  wc  -l)  <  archivo
```

La 1ª es incorrecta ya que wc tiene redirigida su entrada dos veces. En este caso la tubería se pierde y solo contaría las líneas de archivo.
La 2ª y 3ª hacen lo mismo: ordenan archivo, buscan líneas que contengan el patrón abc y las cuenta.

Ejercicio: Explicar qué hacen las siguientes líneas de comandos y cuál será su salida.
```
$cat  x  y  >  y
$cat  x  >>  x
$echo  *  ¿qué pasa si el directorio está vacío?   Devuelve * ya
          que no puede sustituir
$echo  '*'
$echo  `*`
$ls  *
$echo  *  >  xxx
$(ls  -l  |  wc  -l  <  datos.dat)  >  vacio
$cd  /usr/bin;  (cd  /etc);  pwd
$mkdir  '$esto es un directorio'
```

Grupo de comandos.

Si se quiere redirigir la entrada (<) o salida (>) a todos los comandos como un grupo, se puede convertir la línea de comandos como un grupo de comandos.
Un grupo de comandos es cualquier número de ordenes incluidas entre llaves "{}", dejando un espacio en blanco después de la llave de apertura y otro antes de cerrar la llave.

Ejemplo:

```
${ clear;  ls -la  |  more } > archivo
```

more

Es similar al filtro del mismo nombre en DOS. Es un paginador de uso general. Se utiliza para ver el texto de un fichero largo deteniéndose por pantallas. Al llegar al final del documento abandona el more.

Una vez dentro del entorno more se puede avanzar pantalla pulsando la <barra espaciadora>, para avanzar línea <intro>, y para salir <q>.

Sintaxis:

```
more   [opciones]   lista_de_archivos
```

Opciones :

-'-n'	Cambia el tamaño de la ventana de visualización a n_líneas. Donde n es un número
-'-l'	No trata el carácter ^L(salto de página) de modo especial. Normalmente more trata este carácter igual que cuando la ventana se llena de pausas
-'-s'	Varias líneas en blanco se suprimen y se tratan como una sola.
-'-p'	No se desplaza. Limpia la pantalla y visualiza el texto que falta cuando avanza una pantalla.
-'-u'	Suprime el subrayado.
-'-d'	Aparece un mensaje al final de la pantalla [Press space to continue, 'q' to quit]. Si pulsamos una letra equivocada saca el siguiente mensaje [Press 'h' for instructions]. Si nos equivocamos suena una campana.

Cuando more está en ejecución podremos pulsar las siguientes combinaciones de teclas. Pero tendremos presente que algunas de ellas sólo funcionan cuando estamos visualizando un fichero y no cuando utilicemos el more para paginar una salida estándar.

h	Muestra una lista de otras acciones posibles.
q	Sale de more.
Espaciador	Muestra página a página.
nd, Ctrl + D	Muestra n líneas hacia adelante
nb, Ctrl + B	Retrocede n pantallas
return	Muestra línea a línea.
v	Llama al editor vi y permite modificar el fichero.
/expresión regular	Permite buscar una sección del texto en la que se encuentre la expresión regular especificada.
Ctrl + L	Redibuja la pantalla

Resumen de los Metacaracteres:

Los metacaracteres son caracteres o cadenas que tienen un significado especial para el shell, son elementos del propio shell.

Metacarácter	Significado	Ejemplo
>	Redireccionamiento de la salida estándar	ls > fichero
>>	Redirección de la salida añadiendo	ls >> fichero
<	Redirección de la entrada estándar	mail pepe < fichero
<<carácter	Asume entrada por teclado hasta que se introduzca el carácter especificado.	cat << f > fichero
|	Interconexión, tubo o pipeline. La salida de la primera orden, es la entrada de la segunda.	cat fichero |more
2>	Redirección de la salida de error.	ls /noexiste 2> fichero
*	Sustituye cadenas	ls a*
?	Sustituye un carácter	ls ejemplo?
[cadena]	Sustituye caracteres de cadena. Bien en intervalo, bien por un conjunto de caracteres concreto.	rm ejemplo[1-4] rm ejemplo[1234] rm ejem[1-4]plo
;	Ejecución secuencial de órdenes	who;ls
&	Ejecución paralela o desatendida (background)	ls -R / >fichero & ls -R / fic 2>ferror &
\	Anula interpretación de un carácter especial	ls \> fichero
`---`	Ejecuta órdenes incluidas de shell. Acentos	echo "Directorio:`ls` "
' --- '	Toma literalmente lo que hay entre las comillas simples	echo 'Directorio:`ls` '
"--- "	Igual que el anterior pero con la diferencia de que los siguientes caracteres son interpretados: $ \ ` `	echo -e "Directorio de `pwd` \n `ls`"
$0 $1 ….	Valor de los argumentos de orden	ls $1
var=valor	Asignación de valor a variable *var* .	var=hola
$cadena	Valor de la variable cadena	echo $var
#cadena	Línea de comentario	#comentario
a && b	Si orden *a* entonces orden *b*	ls /noexiste && ls /
a | | b	Si **no** orden *a* entonces orden *b*	ls /noexiste || ls /

SISTEMA DE FICHEROS

2.1 Conceptos básicos sobre ficheros. Manipulación.

Un fichero es una secuencia de bytes, y que forma parte de una estructura llamada Sistema de Ficheros.

Atendiendo a su contenido pueden ser de dos tipos :

- ☒ Fichero Ascii o de texto. Pueden ser leídos y exportados fácilmente a otros sistemas operativos.
- ☒ Ficheros binarios o no Ascii. Normalmente son los ejecutables.

Atendiendo a su utilización, pueden ser de tres tipos :

- ☒ Ordinarios. Se utilizan para almacenar la información (texto, Datos o programas) de los usuarios o del propio sistema.
- ☒ Directorios. Se utilizan para referenciar un conjunto de ficheros y/o directorios.
- ☒ Drivers o Controladores. Se utilizan para simular o referenciar a dispositivos físicos.

En Unix, las normas para el nombrado de ficheros son las siguientes :

- ☒ Un nombre de Fichero puede tener de 1 a 255 caracteres.
- ☒ No se permiten los caracteres : / ? * [ ] ‘ “ ` $ # | \ ; & { } .....etc ni caracteres de control.
- ☒ Unix distingue entre mayúsculas y minúsculas. Se recomienda utilizar siempre minúsculas.
- ☒ No existe extensión. Pueden haber varios puntos. Se recomienda utilizar nombres con un punto y a continuación algo que indique el tipo de fichero de que se trata.

2.2 Soporte de los Ficheros.

Un fichero posee varios componentes : Nombre, Contenido e Información propia (fecha de creación, fecha de último acceso, permisos, dirección de almacenamiento, etc...).

En Unix la información relativa a los ficheros está en las Tablas de Nodos de Indices. También llamadas INODES o NODOS-I.

Las tablas de inodes soportan la información administrativa de los ficheros, son gestionadas directamente por el sistema y son creadas al crear un Sistema de Ficheros, bien sea en la instalación o al crear otro.

Un inode es un registro de esta tabla y contiene información administrativa de cada fichero :

- ☒ Identificador de Grupo.
- ☒ Identificador de Usuario.
- ☒ Permisos.
- ☒ Fecha y hora de la última modificación.
- ☒ Fecha y hora del último acceso.
- ☒ Número de enlaces.
- ☒ Tipo de fichero (ordinario, directorio, driver de caracteres, driver de bloques).
- ☒ Puntero hacia los bloques de disco que contienen la información contenida en el fichero.

El tamaño de los inodos se asigna en la instalación. Normalmente los inodos van cada 4096 bytes, 2048, 1024. Su elección dependerá del espacio en disco disponible. A menor espacio en disco se debe elegir, menor tamaño de inodo, ya que cada vez que se crea un fichero, por pequeño que sea, se usa uno.

Por defecto 1 inodo = 4096 bytes. Con este tamaño de inodo los bloques tienen 102424 bytes.

2.3 Ordenes básicas de manipulación y Gestión de Ficheros.

cat. Visualiza el contenido de los ficheros.

Sintaxis : cat [-opciones] fichero1 fichero2 .....

El funcionamiento de esta orden ya se ha visto anteriormente.

cp Copia ficheros. El nombre del fichero puede contener el camino absoluto desde el raiz para copiar entre subdirectorios.

Sintaxis : cp [-opciones] origen destino
cp [-opciones] origen1 origen2 directorio_destino

La segunda sintaxis de 'cp', copia varios archivos en un directorio dado.

Los permisos asignados del archivo también se copian.

Si el destino es un directorio, entonces son copiados sin cambiar el nombre. Si se quiere copiar en el propio directorio es necesario especificar otro nombre, y si este fichero existe su contenido será reemplazado.

Opciones:

-a Mantiene la misma estructura y atributos del archivo original.
-b Crea copias de seguridad de archivos que van a ser sobreescritos o borrados.
-r Copia directorios recursivamente. c*p -r directorio1 directorio2*
-u No copia archivos si ya existen.
-v Imprime el nombre de cada archivo antes de ser copiado.
-i Pide confirmación antes de sobreescribir.

mv Cambia el nombre a un fichero o directorio.

Sintaxis : mv [-i] fichero1 fichero2
mv [-i] fichero1 fichero2 ..... directorio

En la primera sintaxis fichero1 es el nombre original de un archivo o directorio y fichero2 es el nuevo nombre del fichero o directorio.

La segunda sintaxis de 'mv' permite mover ficheros entre directorios.

Tambien permite mover un directorio y todo su contenido a otro directorio.

Opciones:

-i Pide confirmación antes de sobreescribir.

rm Elimina archivos o directorios.

Sintaxis : rm [-opciones] fichero1 fichero2 .....

Si el nombre del fichero es un directorio obligatoriamente debe utilizarse la opción '-r'.

Opciones:

'-f¨' Borra archivos sin formular preguntas.

'-R' '-r' Permite aceptar nombres de directorios como argumentos, y realiza un borrado recursivo del mismo. Atención a esta opción que puede ser peligrosa.

'-i' Borra archivos solicitando confirmación para cada uno antes de borrarlo.

'-v' Visualiza el nombre del fichero o directorio antes de borrarlo.

2.4 Organización de Directorios.

Los directorios están organizados en grafos (árbol).

Los ficheros almacenados en un directorio pueden ser de cualquier tipo : ficheros ordinarios, directorios o ficheros drivers.

Hay un directorio que no posee antecedente o padre. Es el directorio raíz. Su símbolo es el carácter /.

Cada fichero se puede referenciar unívocamente expresando un camino hacia él, desde el directorio raiz (camino absoluto).

Un fichero se puede referenciar desde el subárbol actual (camino relativo).

El carácter '/' está reservado para la construcción de caminos y por tanto, no podrá formar parte de un nombre de fichero.

Cualquier directorio contiene dos ficheros por defecto que a su vez son directorios :

☒ **.** el propio directorio.

☒ **..** el directorio padre o antecedente al actual.

Un fichero puede hacerse accesible desde más de un directorio mediante enlaces o cambiando la ruta de búsqueda (Path)

Unix se organiza en Sistema de ficheros, que es el conjunto de directorios y su información necesaria, desde su instalación. Es importante no modificar esa organización par el buen funcionamiento del sistema, ya que, el sistema reconoce internamente determinados caminos por defecto a ficheros internos del sistema.

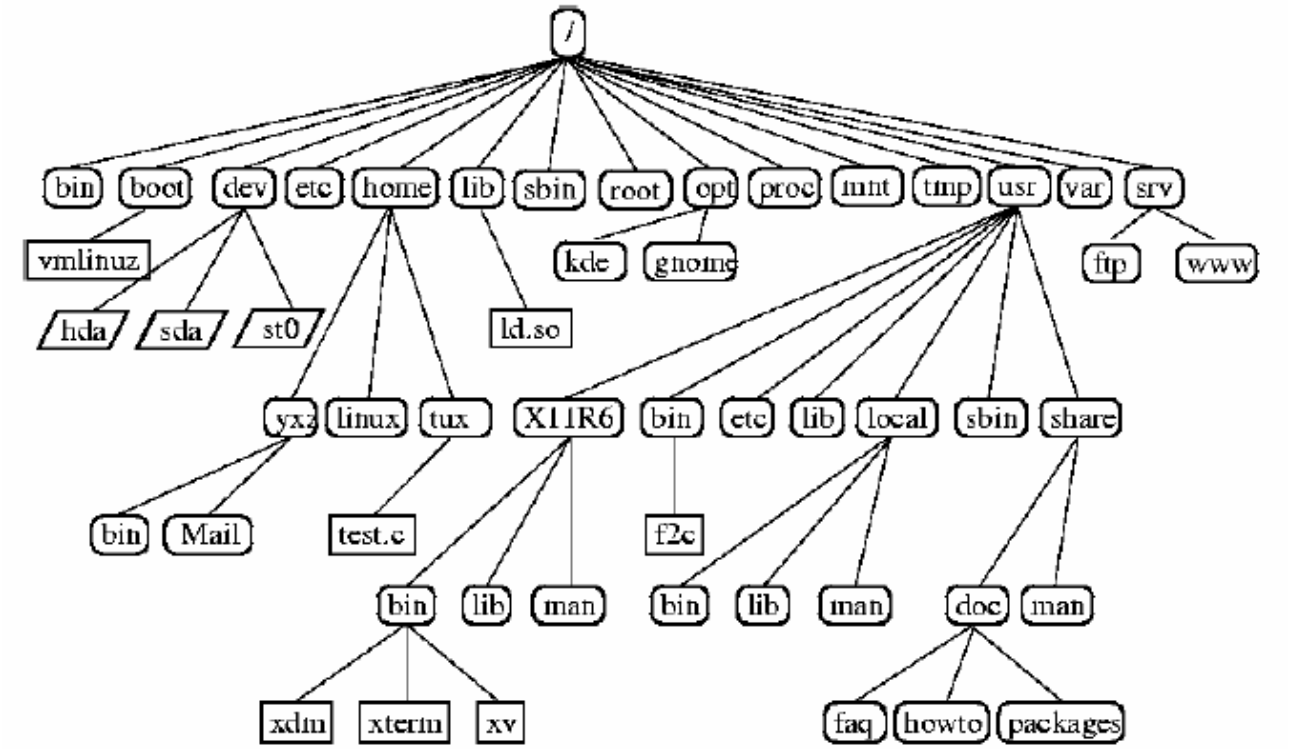
Cada una de las fracciones del árbol de directorios puede ser a su vez un Sistema de Ficheros (file system), ello posibilita la integración de la información entre sistemas o dispositivos, haciendo coincidir esa unidad con un nodo del sistema de ficheros, árbol principal.

En la práctica, un sistema de ficheros se maneja como una unidad lógico-física, y por tanto, se suele asignar un sistema de ficheros a cada unidad, pero no necesariamente debe ser así.

2.5 Organización de Directorios 2. Gráficos comparativos.

Para trabajar de forma eficiente con la shell, debe conocer la estructura de archivos y de directorios en Linux. Los directorios son carpetas electrónicas en las cuales se pueden depositar archivos o programas así como subdirectorios. El directorio de mayor jerarquía es el directorio raíz, que se representa por /. Desde este directorio podremos acceder a todos los demás.

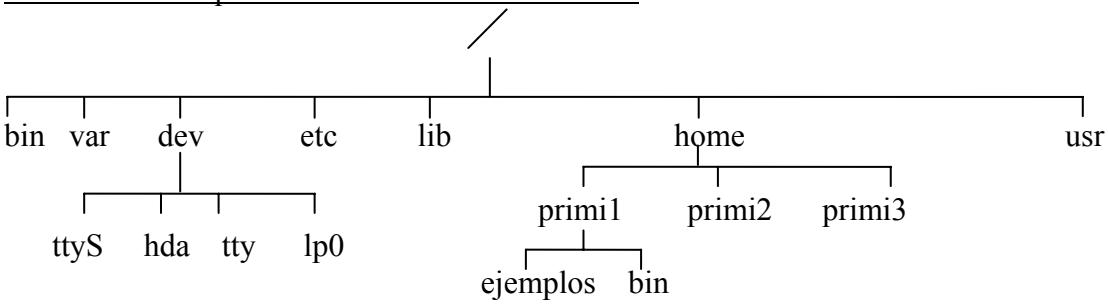
El directorio /home contiene los directorios personales en los que todos los usuarios dejan sus archivos. La figura siguiente muestra el árbol de directorios estándar de Linux con los directorios locales de los usuarios xyz, linux y tux. El árbol de directorios está ordenado por grupos funcionales según el estándar de jerarquía de sistemas de archivos.



La siguiente tabla resume los directorios más típicos de Linux.

/ el directorio raíz root directory, que es el punto de partida del árbol de directorios.
/home los directorios (locales) de usuario.
/dev archivos de dispositivos (device files) que representan componentes del hardware.
/etc archivos importantes para la configuración del sistema.
/etc/init.d scripts de arranque.
/usr/bin comandos de acceso general.
/bin comandos necesarios durante el arranque del sistema.
/usr/sbin comandos reservados para el administrador del sistema.
/sbin comandos reservados para el administrador del sistema y necesarios durante el arranque del sistema.
/usr/include archivos de encabezamiento para el compilador de C.
/usr/include/g++ archivos de encabezamiento para el compilador de C++.
/usr/share/doc diferentes archivos de documentación.
/usr/share/man la ayuda en línea (manual pages).
/usr/src fuentes del software del sistema.
/usr/src/linux fuentes del kernel.
/tmp, /var/tmp para archivos temporales.
/usr contiene todas las aplicaciones.
/var archivos de configuración (por ejemplo enlazados desde /usr).
/var/log archivos de registro.
/var/adm archivos para la administración del sistema.
/lib librerías compartidas (para programas enlazados dinámicamente).
/proc el sistema de archivos de los procesos.
/usr/local extensiones locales, independientes de la distribución.
/opt software opcional, paquetes grandes (por ejemplo KDE, GNOME, Netscape).

Estructura Jerárquica. Punto de vista del usuario :



La estructura Jerárquica viene determinada por el contenido de los directorios.

La lista de directorios que hay que recorrer desde el directorio raíz se denomina vía o ruta. En el gráfico anterior, la vía de acceso al directorio 'ejemplos' , sería : /home/primi1/ejemplos

Estructura Jerárquica. Punto de vista de Unix.

Directorio raíz

1	.
1	..
2	dev
3	home
4	etc

nº inode nombre directorio

Directorio dev

2	.
1	..
34	ttyp1
108	hda1

Directorio home

3	.
1	..
278	primi1
987	os2

Directorio prim1

278	.
3	..
39	.profile
583	ejemplos

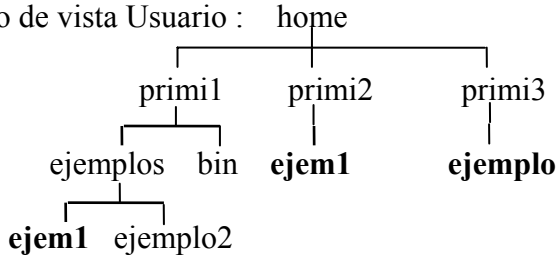
Directorio ejemplos

583	.
278	..
408	Ejem1
922	Ejemplo2

Directorio os2

987	.
3	..

Estructura Jerárquica. **Enlaces.** Punto de vista Usuario : home



Punto de vista Unix :

Direc. prim1

278	.
3	..
583	ejemplos
400	bin

Directorio primi2

290	.
3	..
39	.profile
922	ejem1

Directorio primi3

987	.
3	..
922	ejemplo

Directorio ejemplos

583	.
278	..
408	ejemplo2
922	ejem1

En la figura 'ejem1' y 'ejemplo' son ficheros enlazados. Hacen referencia al mismo fichero. Esto es, si el usuario altera los contenidos de 'ejemplo', los dos ficheros llamados 'ejem1' observarán estas modificaciones.

2.6 Ordenes básicas en la gestión de directorios.

pwd : Muestra el nombre del directorio actual.
Sintaxis : pwd

cd : Cambia el directorio actual.
Sintaxis : cd [directorio]
Directorio es un argumento opcional. Si no se especifica al acción por defecto es cambiar al directorio de usuario.

mkdir : Crea uno o varios directorios especificados.
Sintaxis : mkdir [-opciones] directorio1 directorio 2 .....

Opciones:
-m modalidad Establece los permisos de directorio a modalidad en el momento de creación.
-p Crea la ruta entera.

Ejemplo:
Crear la siguiente ruta de una sola vez *adios/pongo/por/testigo*
mkdir -p adios/pongo/por/testigo

rmdir : Elimina directorios que están vacíos.
Sintaxis : rmdir [-p] directorio1 directorio2 .....
Opciones:
-p Borra los directorios especificados que quedan vacios en el directorio padre.

2.7 Organización de Usuarios.

Como hemos podido comprobar hay una organización de directorios y otra que es el sistema de ficheros. Pues bien, superpuesta a estas dos organizaciones hay una tercera que es la organización de usuarios del sistema.

Los objetivos de la organización de usuarios son :

☒ Protección de uso de la Información, evitando de esta forma usos indebidos.
☒ Adaptación del entorno de trabajo de cada usuario a sus necesidades específicas.

El Unix es un sistema multiusuario, de manera que, un usuario puede ser entendido como 'Una unidad de uso del sistema'.

Cada usuario del sistema tiene asociado, asignado por el administrador del sistema (root) :

- ☒ Un identificador o nombre de Usuario : Login.
- ☒ Una clave de acceso al entorno : Password.
- ☒ Un directorio de trabajo : Home (La casa de ese usuario)
- ☒ Un buzón de comunicación con otros usuarios : mail.
- ☒ Un fichero de configuración de su entorno de trabajo : **.**profile (debe estar situado en el directorio home)
- ☒ Pertenece a un grupo de usuarios para compartir recursos.
- ☒ Un shell asignado en el momento de la creación del usuario por parte del Administrador del sistema o superusuario.

Unix reserva identificadores numéricos de usuario desde el 0 hasta el 500 para los usuarios del propio sistema. En otras versiones de Unix solamente hasta el 200.

El usuario 0 llamado superusuario (root) es el que tiene los máximos derechos de acceso y ejecución.

Cada grupo tiene asociado un identificador de grupo. Los identificadores numéricos de grupo, desde el 0 hasta el 100 (hasta el 50 en otras versiones), están reservados al sistema operativo. Los usuarios creados por defecto se les asigna el grupo 100 y el nombre es users , group en otras versiones.

Para cada usuario existe una casa (home), que es su directorio de trabajo. Cuando un usuario accede al sistema accede a este directorio. Un usuario puede crear más directorios hijos de este.

Los directorios de trabajo de cada usuario, en el sistema Linux, están situados en el directorio home (/home/nombre_usuario). En otros sistemas, como en el SCO System V, están situados en el directorio usr (/usr/nombre_usuario).

2.8 Los ficheros de configuración .profile y /etc/profile.

El fichero '.profile' es un shell-script de inicialización del entorno de usuario. Un shell-script o script es un fichero que contiene órdenes de shell, permitiendo explotación batch o por lotes.

Uno de los mejores ejemplos de script es, precisamente el .profile, que cada usuario posee individualmente en su directorio home.

El fichero .profile es un script estándar ejecutado por el sistema durante el acceso de un usuario. Se utiliza para configurar las sesiones de trabajo de acuerdo con las preferencias propias de cada usuario y para definir o crear algunas variables de entorno que las órdenes que utiliza ese usuario esperan encontrar. Cada usuario debe mantener este fichero según sus necesidades.

Aunque parece que solamente se ejecuta el script '.profile' del usuario, la realidad es que se ejecutan dos.

El script /etc/profile es propiedad del sistema y se ejecuta antes del .profile de cada usuario. Se utiliza para preparar el entorno de los usuarios, de acuerdo con unas características globales.

El fichero /etc/profile puede ser leído por los usuarios. Cualquier modificación de este fichero la debe realizar el Administrador del sistema, porque modificará el entorno de todos los usuarios globalmente.

2.9 Derechos de uso de ficheros. Permisos.

Para que un usuario pueda acceder a un determinado fichero, es necesario que tenga habilitado el permiso de acceso. Se puede tener alguno de ellos, todos o ninguno.

Los permisos de acceso o derechos de uso de ficheros son :

r . Permiso de lectura. Significa que se puede mirar el contenido. Si se trata de un directorio que se puede ver la lista de ficheros de ese directorio.

w. Permiso de escritura. Significa que se puede cambiar el contenido. Si es un directorio que se pueden borrar y crear ficheros dentro de ese directorio.

x . Permiso de ejecución. Se puede ejecutar el fichero, este permiso es imprescindible si se quiere ejecutar un script. Si es un directorio, quien tiene el permiso, entonces significa que se pueden buscar y copiar ficheros de ese directorio.

Las entidades que disfrutan de estos derechos son :
u . Usuario. También conocido como propietario, es el usuario que creó el fichero.
g . Grupo. Grupo al que pertenece el propietario. El grupo determina el dominio.
o . Otros. Son los demás usuarios que no pertenecen a este grupo.

Los permisos son atributos de cada fichero, están almacenados en la tabla de inodes. Los atributos de un fichero se pueden conocer mediante la orden : ls -l . Orden que muestra el listado largo de los ficheros del directorio y que produce la siguiente salida por pantalla :

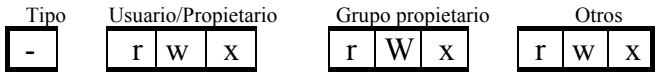
drwxr-xr-x	2	root	bin	2048	Oct 22 21:48	bin/
drwxr-xr-x	2	root	root	1024	May 10 00:29	boot/
drwxr-xr-x	2	root	root	8192	Nov 3 00:44	dev/
drwxr-xr-x	41	root	root	16384	Jan 1 1970	dosc/
drwxr-xr-x	9	root	root	2048	Nov 3 00:44	etc/
drwxr-xr-x	4	root	root	1024	Oct 22 21:12	home/

Dejemos la primera columna para el final y pasemos a estudiar con detalle esta salida :
2ª columna. Muestra el número de enlaces que tiene un fichero o directorio.
3ª columna. Muestra el propietario del fichero.
4ª columna. Indica a qué grupo pertenece el fichero.
5ª columna. Es el tamaño en bytes del fichero o directorio.
6ª columna. Es la fecha, y en algunos caso la hora, de la última modificación.
7ª columna. Es el nombre del fichero o directorio.

Volvamos a la primera columna. Muestra los atributos de cada fichero. Están representados mediante una cadena de 10 caracteres.

Los atributos de los ficheros son de dos tipos :
- Tipo de fichero. Está representado en el primer carácter.
- Permisos. Se representan por los siguientes nueve caracteres.

La cadena de atributos tiene la siguiente estructura :



El primer carácter de la cadena representa el tipo de fichero, que puede ser :
- El '-' (guión) indica que se trata de un archivo ordinario.
- La 'd' indica que es un fichero de entrada a un directorio.
- La 'b' indica que es un fichero driver de bloques (disquetera).
- La 'c' indica que es un fichero driver de caracteres.
- Existen otros que varían según la versión de Unix. Para conocerlos se deberá consultar el manual del administrador del sistema concreto.

Los restantes nueve caracteres son una representación simbólica de los derechos de uso o permisos de ese fichero, relativos a las tres entidades que disfrutan de esos derechos, usuario, propietario, grupo y otros.

Estos nueve caracteres se dividen en tres grupos, de tres caracteres cada uno, para representar los derechos de lectura, escritura o ejecución de cada entidad, que no son excluyentes.

Cada uno de estos grupos podrá tener, en las posiciones correspondientes :

- ☒ El '- ' (guión) representa que no está habilitado el permiso.
- ☒ La 'r' indica que tiene habilitado el permiso de lectura.
- ☒ La 'w' indica que tiene habilitado el permiso de escritura.
- ☒ La 'x' indica que tiene habilitado el permiso de ejecución.

Los derechos de uso de ficheros son, por tanto, combinaciones entre los tres tipos de permisos y las tres entidades que los disfrutan.

Veamos un ejemplo. Tenemos un fichero cuyo listado largo muestra :

-rwxr-x--- 4 primi1 users 1024 Oct 22 21:12 prueba

El fichero ordinario 'prueba' tiene los siguientes permisos :

- ☒ El propietario, que es primi1, puede : leer, modificar ejecutar el programa.
- ☒ Los miembros del grupo del propietario, que es users, solo pueden leer y ejecutar el programa.
- ☒ Los otros usuarios del sistema, no pueden acceder al fichero prueba.

Además tiene 4 enlaces, fue modificado por última vez el 22 de Octubre del presente año a las 21 :12 horas y tiene un tamaño de 1024 bytes.

2.10 Ordenes básicas de gestión de usuarios.

who : Muestra los identificadores de los usuarios que actualmente están trabajando con el sistema.

Sintaxis : who [-opciones]

La acción por defecto de esta orden es proporcionar una lista de quienes son los usuarios, que están conectados al sistema en ese momento.

Opciones:

- -u Informa de los usuarios que están actualmente en el sistema. Además muestra una columna que indica el tiempo que un usuario está inactivo. La columna se denomina IDLE
- -q Muestra sólo el nombre de los usuarios conectados y el número total de ellos.
- -T Indica si el terminal permite que los usuarios envíen mensajes al mismo. '+' indica que se puede escribir en el terminal. '-' indica que no se puede escribir en el mismo.
- -m Semejante al who i am.
- -H Muestra cabeceras de columnas

Existe una variante de who que es : who am i . Que como la traducción literal indica 'quién soy yo', muestra el usuario que ejecuta esta orden.

tty : Informa del nombre del dispositivo del terminal con el que actualmente está trabajando o prueba si la entrada estándar es un terminal .

Sintaxis : tty [-s]

Opciones:

- -s tty inhibe la impresión de la via de acceso al terminal, devolviendo únicamente el código de salida.

La acción por defecto es mostrar el nombre del dispositivo del terminal.

su : Cambia a un entorno de trabajo de otro usuario.

Sintaxis : su [nombre de usuario]

Opciones:

-l Ejecuta el .profile del usuario donde accedes. Si a continuación pones el nombre de otro fichero ejecutable también lo ejecuta.

La acción por defecto es cambiar al superusuario. En ambos casos preguntará la clave de acceso a ese usuario.

chown : Cambia el usuario propietario de uno o varios ficheros.

Sintaxis : chown [-opciones] propietario fichero1 fichero2 .....

Opciones:

-R Cambia de forma recursiva la propiedad de los directorios y sus contenidos.
-v Describe en detalle los cambios de propiedad.
-c Describe en detalle sólo los archivos cuya propiedad cambia.

La propiedad del archivo, en Linux, sólo la puede cambiar el superusuario.

chgrp : Cambia el grupo propietario de los ficheros especificados.

Sintaxis : chgrp [-opciones] nuevo_grupo fichero1 fichero2 .....

Opciones:

-R Cambia de forma recursiva la propiedad de los directorios y sus contenidos.
-v Describe en detalle los cambios de propiedad.
-c Describe en detalle sólo los archivos cuya propiedad cambia.

No se puede cambiar la propiedad de grupo de un archivo a no ser que sea el propietario del mismo o el superusuario.

chmod : Cambia los permisos de acceso a uno o varios ficheros o directorios.

Sintaxis : chmod [-opciones] modalidad fichero1 fichero2 .....

Opciones:

-R Cambia de forma recursiva los permisos de los directorios y sus contenidos.
-v Describe en detalle los permisos cambiados.
-c Describe en detalle sólo los archivos cuyos permisos cambian.

Existen dos formas de cambiar la modalidad a un archivo :

Modo simbólico : Existen tres niveles en el cambio de permisos, representados por : u,g,o,a.

Dónde:

‘**u**’ representa al propietario,
‘**g**’ representa al grupo ,
‘**o**’ representa a los otros usuarios del sistema
‘**a**’ los representa a todos

Para agregar un permiso se utiliza el signo + , para eliminar un permiso se usa el signo - y para dejar como estaba el permiso el signo =. De tal forma que, si queremos dar permiso de todo a un fichero, la forma sería :

chmod u+r+w+x, g+r+w+x, o+r+w+x fichero ó chmod a+rwx fichero ó chmod +x fichero

En este último caso lo único que estamos haciendo es darle permiso de ejecución al fichero. Es necesario recordar que un script no se puede ejecutar si no tiene activado este permiso.

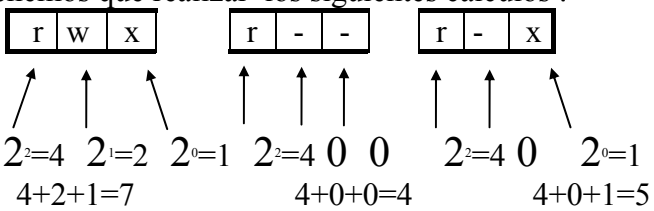
Entre el último permiso de la opción anterior y la letra del siguiente grupo de permiso debe haber únicamente una coma, no puede haber ningún espacio en blanco.
También se puede dar permisos o quitar de forma selectiva, por ejemplo :

chmod u+w, g-wx,o=r fichero

De esta forma damos permiso de escritura al propietario. Le quitamos al grupo los permisos de escritura, ejecución y a los otros les dejamos como están.

Modo octal : Esta es conocida también como modalidad numérica, consiste en dar 3 números, en otras ocasiones 4. Cada número, en octal , establece un byte en el campo de modalidad almacenado en la tabla de inode del sistema de archivos. Veamos como podemos calcular y utilizar esta modalidad :

Queremos obtener los siguientes permisos : rwxr—r-x para representar estos permisos en octal tenemos que realizar los siguientes cálculos :



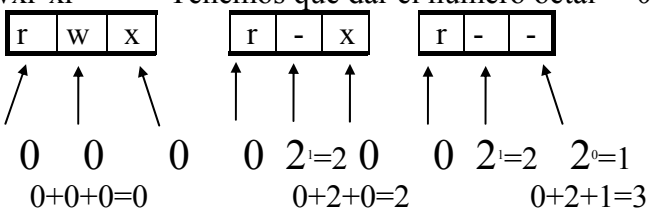
La orden que habría que dar para dar estos permisos sería : chmod 745 fichero ó directorio.

umask : Cambia la máscara de creación de ficheros. Se entiende por tal a un número de tres dígitos, en octal, cada unos de los cuales pertenece a un grupo de permisos

Sintaxis : umask [d1d2d3]

La acción por defecto que realiza es mostrar la máscara actual. La máscara queda activa hasta que finaliza la sesión de trabajo. El cálculo de estos tres dígitos se realiza de idéntica forma que en chmod, con la única diferencia que en esta orden hay que dar el número octal de los permisos que queremos quitar.

De tal forma que para poner los permisos :
rwxr-xr-- Tenemos que dar el número octal 023 Veamos cómo ;



Recordar que el permiso de ejecución para que sea visible, o tenga sentido debe ser un fichero ejecutable.

2.11 Dispositivos.

Hasta ahora se han estudiado los ficheros ordinarios y los directorios, pero existe un tercer tipo de ficheros. Los ficheros driver o controladores de dispositivos, o dispositivos.

La idea de fichero de dispositivo es una idea original de Unix. La definición podría ser la interfaz o conexión entre un dispositivo físico (hardware) y el sistema operativo, necesaria para la realización de operaciones con ese dispositivo. Unix la realiza mediante un fichero, que representa un canal de E/S.

En Unix, toda operación de E/S se realiza sobre un fichero de dispositivo. Por ejemplo, la orden :

tty muestra /dev/tty01

lo que devuelve es el nombre del canal de E/S asociado al terminal donde se ha ejecutado esta orden. Es decir, devuelve el nombre del fichero de dispositivo que representa ese terminal. En realidad, cuando escribamos sobre este fichero se estará escribiendo en el terminal. Este fichero se encuentra en /dev , como todos los ficheros de dispositivo. Al ejecutar la siguiente orden :

ls /dev/tty*

aparecerán todos los ficheros de dispositivo relativos a terminales. Nótese que el primer carácter de los atributos es una c, ya que son ficheros de dispositivo de caracteres.

Existen dos tipos de ficheros de dispositivos, porque existen dos tipos de dispositivos periféricos :

- Dispositivos de caracteres, (El atributo es una ‘c’) se accede carácter a carácter. Son los terminales, impresoras, líneas telefónicas, etc.
- Dispositivos de bloques, (el atributo es una ‘b’) se accede bloque a bloque. Son los dispositivos de almacenamiento : cintas, discos o disquetes. Los dispositivos de bloque permiten su utilización como sistema de ficheros. Al ejecutar la orden :

  ls /dev/fd*

  aparecerán todos los ficheros de dispositivo relativos a las unidades de disquetes. Obsérvese que, los ficheros /dev/fd* tiene una ‘b’ como primer carácter de los atributos.

Los ficheros de dispositivo tienen permisos como cualquier otro fichero y algunos tienen como propietario el usuario actual.

ORDENES DE LINUX (II)

ln

Establece un enlace o vínculo entre dos ficheros. De manera que al actualizar uno de ellos los cambios se producirán en los dos.
Sintaxis: ln [-opciones] fichero_existente fichero_nuevo
Opciones:

-v Imprime el nombre del archivo antes de enlazarlo.
-i Pide confirmación si fichero_nuevo existe en destino. Si existe fichero_nuevo y no ponemos esta opción no permite hacer el enlace.
-s Realiza un enlace simbólico.

Esta orden permite aumentar el número del 2º campo que se muestra con la orden ls –l referido al número de enlaces sobre un contenido concreto. De esta manera podemos tener varios nombres asignado a un mismo contenido sin que ocupe más espacio como si se realizara copias, con la característica de que utilizando cualquier nombre podemos modificar el contenido como si se hiciera con todos los demás a la vez.

La diferencia entre el enlace normal y el simbólico es la siguiente:

- Con el enlace normal, el funcionamiento aparente es como si se hiciera copias (pero sin ocupar espacio adicional), de tal modo que podemos borrar el fichero original y nos permitiría utilizar los enlaces que se hubieran realizado.
- Con el enlace simbólico, si se borrase el fichero original, no funcionaría el acceso desde el enlace que se haya realizado. En esto se parece a un acceso directo desde Windows.
- Realizando un listado con ls –l, se reconoce el enlace simbólico ya que al final de la línea haciendo referencia al nombre del fichero tiene esta forma: fich_original → fich_enlace

cmp

Compara el contenido de dos archivos.
Sintaxis: cmp [-opciones] archivo1 archivo2

Este comando compara los dos archivos especificados. Por defecto informa sobre la localización de la primera diferencia, si existe. Si los archivos son iguales, no aparece ningún comentario. Si son diferentes, indicará el octeto y el número de línea donde se encuentra la diferencia.
La orden facilita los códigos siguientes:

‘0’ si los archivos son idénticos
‘1’ si son diferentes
‘2’ si un archivo es inaccesible o no se ha especificado correctamente.

Opciones:

-l Imprime el número del octeto en decimal, y los octetos diferentes en octal por cada diferencia encontrada.
-s No imprime las diferencias encontradas.
-c Visualiza la diferencia en ascii y el caracter.

Ejemplo:
Crea dos ficheros con el cat exactamente iguales y utiliza el comando cmp luego cambia una letra y vuelve a ejecutar el cmp. Siempre utilizando las opciones.

diff

Produce un índice completo con todas las líneas que difieren entre dos ficheros.

find

Busca ficheros recursivamente a partir del nodo que se especifique.
Sintaxis: find nodo_ruta criterio_nombre_expresión acción

El nodo-ruta especifica a partir de dónde buscar los archivos (camino).
El criterio indica cómo seleccionar los archivos, si es por el nombre solamente o es por otro criterio cualquiera.
La acción es referida a, qué hacer con los ficheros seleccionados.

Criterios:

-name fichero	Busca ficheros con nombre fichero.
-type f	Busca ficheros ordinarios.
-type d	Busca directorios.
-user nombre	Busca ficheros cuyo propietario sea nombre.
-group nombre	Busca ficheros cuyo grupo sea nombre.
-links n	Busca ficheros con n enlaces.
-atime n	Busca ficheros a los que se ha accedido hace n días.

Donde n es un argumento numérico y podemos especificar

+n	mayor que n
-n	menor que n
n	igual a n

Acciones:

-print	Visualiza el nombre y la ruta del fichero. Opción por defecto.
-exec comando {} \;	El comando se ejecuta por cada archivo comparado.

Se utiliza la notación {} para indicar dónde debe aparecer el nombre del archivo en el comando ejecutado.

El comando debe terminarse por una barra inclinada inversa seguida de punto y coma. IMPORTANTE entre el caracter } y \ debe haber un espacio.

Ejemplos:

find . -name fic* -exec ls -l {} \; Busca ficheros llamados fic y ejecuta un listado largo de ellos. De tal forma que, el comando ls se ejecuta con el argumento -l y cada archivo se pasa a ls en la posición dónde se encuentran {}.

find . -name fic -type f -user bin -print Busca en todo el disco duro los ficheros ordinarios que se llamen fic y tengan como propietario a user y visualiza su nombre y la ruta.

sort

Permite ordenar y fusionar archivos de texto. La clasificación puede hacerse en campo de caracteres o numéricos y pueden especificarse varias claves de ordenación.
Por defecto siempre compara desde el principio de línea. Y el carácter delimitador es el espacio en blanco.

Sintaxis:

```
$sort   [opciones]  ficheros
```

Esta orden ordena línea a línea según el código ASCII.

Opciones:

-c	Comprueba si los archivos están ordenados, y si es así, no se genera ninguna salida.
-m	Fusiona los archivos especificados. Asume que ya están ordenados.
-u	Elimina las líneas repetidas.
-o archivo	Especifica el nombre del archivo de salida. Si no se especifica es la salida estándar.
-d	Solo considera letras, cifras y espacios en blanco. No caracteres especiales.
-f	Trata mayúsculas y minúsculas como iguales.
-i	Ignora los caracteres no imprimibles de las claves de ordenación.
-M	Trata la clave de ordenación como si fuera un mes. MAY es menor que JUN, que a su vez es menor que JUL, y así sucesivamente.
-n	Especifica que una clave es numérica, es decir, realiza la ordenación numérica.
-r	Invierte el orden de ordenación.
-tcar	Utiliza *car* como separador de campo en vez de tabuladores o blancos.
-b	Ignora blancos en la clave de ordenación.
+num	Comienza la ordenación por el campo de posición *num*. Recordar que el primer campo es el cero.
-num	Detiene la comparación antes del campo de posición *num*. Ese campo no lo utiliza para ordenar.

Ejemplos:
Antes de probar los ejemplos, generar un fichero que contenga un listado largo del directorio.

sort -M +5 -6 fichero	Ordena el fichero generado por el mes de creación de los ficheros que contiene.
sort -r -n +4 -5 fichero	Ordena el fichero anterior por número de bytes en orden inverso.
sort +0 -1 -n +4 -5 fichero >fichero3	Ordena el fichero anterior por permisos y número de bytes y deja el resultado en *fichero3*.

wc

Lee la entrada estándar y proporciona el número de líneas, palabras y caracteres de la entrada.
Sintaxis:

```
wc  [-opciones]  lista_de_archivos
```

opciones:

- -l visualiza sólo el número de líneas
- -w visualiza el número de palabras. Una palabra es una cadena separada por un espacio, un tabulador o una nueva línea.
- -c visualiza el número de caracteres. Concretamente cuenta el número de retornos de línea.

Ejemplos:

wc entrada1 entrada2 Cuenta en los ficheros especificados los caracteres, las palabras y las líneas.

```
$cat fichero  |   wc
4    50    310
```

wc -l entrada1 Cuenta y visualiza el número de líneas del fichero entrada1

tail

Permite visualizar el final de un archivo de texto o hacer el seguimiento del crecimiento de este.

Sintaxis:
```
tail [desplazamiento] [-f ] archivo
```

Opciones:

Desplazamiento: Es un número que representa a las líneas del archivo que se empieza a visualizar.
Si a este número le precede el signo **-**, el desplazamiento está relacionado con el final del archivo, es decir, visualiza ese número de líneas del final.
Si se utiliza el +, el desplazamiento está relacionado con el comienzo del archivo, es decir, visualiza desde ese número de línea al final.
Si no se especifica nada, muestra las 10 últimas líneas.

-f Cuando se utiliza esta opción y la entrada no es estándar, tail controla el crecimiento del archivo. Esto es un bucle de salida sin fin y tiene que ser terminado con la tecla de interrupción.

Ejemplos:
Antes de probar estos ejemplos generar un fichero que tenga numeradas sus líneas.

tail -20 fic1	Muestra las 20 últimas líneas del fichero fic1.
tail +10	Muestra desde la línea 10 del fichero fic1.
tail fic1	Muestra las 10 últimas líneas del fichero.
tail -f fic1	Suponiendo que el archivo fic1 ha sido creado por otro proceso, esto muestra lo que se ha construido hasta ahora y lo que se está generando. Hay que conducir la salida con el comando more en caso que se genere demasiado rápidamente para visualizarlo en la pantalla.

head

Muestra la porción inicial de un fichero.

Sintaxis:
```
head   [-líneas]   lista_de_archivos
```

Opciones

-líneas Es el número de líneas a imprimirse desde el principio del archivo. El valor predeterminado es 10.

lista de archivos que se quiere visualizar separados por blancos. Si no se especifica asume la entrada estándar.

Ejemplos:
Antes de probar los ejemplos siguientes generar un fichero numerado de la siguiente forma:

ls >fic ; pr -n fic >fic1

head fic1	Visualiza las primeras 10 líneas del fichero *fic1*.
head -3 fic1	Visualiza las primeras 3 líneas del fichero *fic1*

cut

En Linux los archivos de texto se consideran cada línea como un registro y los campos se corresponden con una cadena de caracteres separados por un delimitador. Los delimitadores son caracteres especiales como espacio en blanco, tabulador, etc.

La orden cut se encarga de extraer campos de un archivo de entrada volcándolos a una salida (un archivo o salida estándar). En esta orden se debe especificar el campo a extraer y el carácter que se va a utilizar como delimitador.

Sintaxis:

```
cut -cpos-car lista-archivos
cut -fcampos -ddelimitador -s lista-archivos
```

- **-cpos-car** Indica la posición de los caracteres que se van a separar. Puede se una lista separada por comas, un rango separado por un guión o una combinación de ambos.

  Ejemplo: `cut –c1,4,5` `cut -c1-5` `cut -c1,4,5,8-12`
- **-fcampos** Indica los campos a separar. Los campos se distinguen por un carácter delimitador.
- **-ddelimitador** Indica el carácter delimitador de campo. El delimitador puede ser cualquier carácter.
- **-s** Suprime las líneas que no poseen un carácter delimitador de campo cuando se usa la opción -f
- **lista-archvios** Es el archivo sobre el cual se va a realizar la separación de campos. Si no se indica nada toma la entrada standard.

Si *fichero* tiene el siguiente contenido:

Pepe Juan Pedro

Ramón Antonio Pedro

Pablo José María

Sofía

Ejemplo 1: cut –f1,3 –d" " fichero

Pepe Pedro

Ramón Pedro

Pablo María

Sofía

Ejemplo 2: cut –c1-3 fichero

Pep

Ram

Pab

Sof

Ejemplo 3: cut –f1,3 –d" " -s fichero

Pepe Pedro

Ramón Pedro

Pablo María

Otros ejemplos:

ls -l | cut -c1,11,15-22	Visualiza dichas columnas
ls -l | tr -s " " | cut -f5,9 -d" "	Muestra el tamaño y el nombre de los archivos.

paste

Produce una salida en forma de columna desde uno o más archivos donde cada archivo proporciona una columna de salida. Paste se utiliza frecuentemente con cut para reordenar columnas de un archivo.

Sintaxis
```
paste  -ddelim [opciones] lista-archivos
```

- **-ddelim** Especifica el carácter que se utiliza para delimitar cada columna (por defecto es el tabulador).
- **-'-s'** Primero visualiza un fichero separado por columnas y luego visualiza el otro.
- **lista-archvios** Una lista de archivos para agrupar. Puede ser el "-" (guión) para indicar entrada estándar.

Ejemplo:
```
paste  -d  fich1  fich2
```
En dos columnas va visualizando una línea de cada fichero, es decir, primera columna una línea de entr1 y en la otra columna la primera línea de entr2 y así sucesivamente

```
paste  -d:  fich1  fich2  fich3  fich4
```

tee

Divide la salida en una conducción a uno o más archivos. Esto permite al usuario que capture los que van a la salida estándar y saque esa salida en un archivo mientras sigue permitiendo que la salida fluya hacia la salida estándar.

Sintaxis:
```
tee   [opciones]  lista-archivos
```

- **lista-archivos** La lista de archivos separadas por espacios en blanco donde quiere capturar la salida.
- **-a** Los archivos de lista de archivos se añaden a la salida en lugar de ser sobreescritos.

Ejemplo :
```
ls  -l   |   tee listado   |   more
```

uniq

Quita las líneas que están repetidas, dejando la salida con líneas que no están repetidas. Para que funcione correctamente los ficheros deben estar ordenador, ya que trabaja con líneas adyacentes.

Sintaxis:
```
uniq   [-opciones]   entrada   salida
```

La entrada se refiere al nombre del archivo que se va a leer, si no se especifica asume la estándar.
La salida es el nombre del archivo resultante de la ejecución del filtro. Si no se especifica es la estándar. Los ficheros entrada y salida no pueden ser el mismo.

Opciones:

-u	Sólo muestra las líneas que no están repetidas.
-d	Muestra sólo las líneas que están repetidas.
-c	Muestra un informe en el que la columna de la izquierda ofrece el número de repeticiones y luego la línea en sí misma.
+n	Ignora los *n* primeros caracteres. Funciona exclusivamente con la opción '-c'
-n	Ignora los *n* primeros campos.

Ejemplos:

uniq -c entrada	Cuenta las líneas repetidas.
uniq -4 entrada	Ignora los cuatro primeros campos y compara si el resto es igual.

grep

El comando grep localiza una palabra, clave o frase en un conjunto de directorios, indicando en cuáles de ellos la ha encontrado. Este comando rastrea fichero por fichero, por turno, imprimiendo aquellas líneas que contienen el conjunto de caracteres buscado. Si el conjunto de caracteres a buscar está compuesto por dos o más palabras separadas por un espacio, se colocará el conjunto de caracteres entre apóstrofes simples (') o dobles (").

Su sintaxis es:

```
grep [-opcion] expresión_regular [archivo(s)...]
```

Las opciones principales son:

c lo único que se hace es escribir el número de las líneas que satisfacen la condición.

i no se distinguen mayúsculas y minúsculas.

l se escriben los nombres de los ficheros que contienen líneas buscadas.

n cada línea es precedida por su número en el fichero.

s no se vuelcan los mensajes que indican que un fichero no se puede abrir.

v se muestran sólo las líneas que no satisfacen el criterio de selección..A continuación se muestra una serie de ejemplos.

Las expresiones regulares son:

.	El punto representa un solo dígito en la expresión.
*****	El asterisco representa ninguno o varios dígitos de la expresión.
^	El circunflejo representa comienzo de línea.
$	El dólar representa final de línea.
[ ]	Los corchetes representan un conjunto de caracteres que validarían la expresión.
[-]	Los corchetes con guión delimitan rangos para validar la expresión.
[^]	El circunflejo con corchetes representa cualquier carácter no contenido entre corchetes.

Ejemplos:

grep '^d' fichero líneas que comienzan por d.

grep '^[^d]' fichero líneas que no comienzan por d.

grep -v '^C' fic1 > fic2 quita las líneas de fic1 que comienzan por C y lo copia en fic2.

tr

Traduce su entrada standard sustituyendo cada carácter del primer argumento por el que se encuentra en la misma posición en el segundo argumento, elimina caracteres repetidos.

Sintaxis:

```
tr  [-opcion]  argumento-1  [argumento-2]
tr  [-opcion]  argumento-1  [argumento-2] < fichero1  > fichero2
```

- **argumento-1** Es la cadena de caracteres desde la que se asigna los caracteres que se quieren traducir. Pueden utilizar las siguientes anotaciones especiales:
  [c1-cn] especifica un rango de caracteres desde cl hasta cn.
  [c*n] indica que el carácter c se repite n veces. Es muy útil para rellenar el segundo argumento.

- **[argumento-2]**Es la cadena de caracteres que se utiliza para asignar los nuevos caracteres que se van a traducir en el argumento-1.

Opciones:

- **-d** Suprime los caracteres asignados en el argumento-l.
- **-s** Filtra los caracteres repetidos en la salida por aquellos especificados en el argumento-2, dejando en la salida solo uno de los caracteres repetidos.
- **-c** Sustituye en la salida todos los caracteres excepto los indicados en el argumento por los especificados en el argumento2. (Argumento2 debe ser un solo carácter).

Formar cadenas:

- \NNN NNN valor del carácter en octal (1 a 3 dígitos en octal).
- CAR1-CAR2 Todos los caracteres desde CAR1 a CAR2 en orden ascendente
- [CAR1-CAR2] Igual a CAR1-CAR2, preferible utilizar este

Ejemplos:

echo “hola como estas”| tr -d o Suprime la letra o de todo el texto.
tr -d “\012\032” < fichero1 >fichero2 Esta es una manera de traducir archivos de texto DOS en un formato más adecuado para Linux. Suprime los retornos de carro y el marcador DOS de final de archivo ^Z.
echo “modifica las vocales”|tr aeiou 12345 Reemplaza las vocales por números.
tr aeiou 12345 < fichero Reemplaza las vocales por números de un fichero y muestra en la salida estándar el resultado.
tr a-z A-Z <fichero1 >fichero2 Cambia todas las letras minúsculas de fichero1 por mayúsculas y las deposita en fichero2.
echo “cammbio” | tr -s m Suprime las repeticiones consecutivas.
tr -c a $ <fichero Cambia todos los caracteres del fichero por el símbolo ‘$’ menos el carácter ‘a’. El contenido no lo modifica, cambia solo en la salida estándar.
tr -s “ “ < fichero Reemplaza múltiples espacios por un único espacio.
tr –s “\012” < fichero Deja un solo retorno por varios juntos. Elimina líneas en blanco.
tr –cd “\012[0-9]” < fichero Elimina todos los caracteres menos los números y retornos.

sed

Se trata de un editor de textos no interactivo. La orden que se le de a sed debe formar un argumento único, por que si contienen espacios en blanco son necesarias las comillas simples o dobles.

Operaciones:

- **Sustituir una cadena por otra:**

  Sintaxis: sed "s/cadena_antigua/cadena_nueva/" 1º coincidencia por línea.
  sed "s/cadena_antigua/cadena_nueva/g" Todas las coincidencias por línea.

  Ejemplos:

  sed s/Smith/White/ gente

  Sustituye la 1ª coincidencia encontrada por línea y visualiza el resultado

  sed "s/Sally Smith/Sally White/g" gente > nueva.gente

  Sustituye todas las coincidencias por línea y guarda el resultado en otro fichero

  Podemos establecer un críterio indicando que sustituya "cadena_antigua" por "cadena_nueva" en las lineas que contengan "cadena".

  Sintaxis: sed "/cadena/s/cadena_antigua/cadena_nueva/" 1º coincidencia por línea.
  sed "/cadena/s/cadena_antigua/cadena_nueva/g" Todas por línea.

  Ejemplos:

  sed /Sally/s/Smith/White/g gente.vieja > gente

  Sustituye en las líneas con Sally, Smith por White y guarda el resultado en otro fichero

  También podemos indicarle una línea o rango de líneas separadas por comas.

  Ejemplos:

  sed 4"s/Sally Smith/Sally White/g" gente

  Sustituye todas las coincidencias de la línea 4 y visualiza el resultado

  sed 2,5"s/Sally Smith/Sally White/" gente

  Sustituye la 1ª coincidencias desde la línea 2 hasta la 5 y visualiza el resultado

  Otros ejemplos:

  sed s/[a-z]/[A-Z]/g texto — Sustituye todas las minúsculas por mayúsculas

  sed "s/ */ /g" texto — Sustituye uno o más espacios por uno solo.
  2 espacios → 1 espacio — Deja un espacio en blanco entre cada palabra.

  sed "s/ */ /g" texto — Sustituye ninguno o más espacios por uno solo.
  1 espacio → 1 espacio — Deja un espacio en blanco entre cada carácter.

- **Eliminar las líneas que contengan la cadena:**

  Sintaxis: sed "/cadena/d" Elimina las líneas que contengan "cadena"

  Ejemplo:

  sed /Henry/d gente.vieja > gente ⇔ grep –v Henry gente.vieja > gente

  Elimina las líneas donde aparezca la cadena y guarda el resultado en otro fichero

Programación de Shell Scripts

INTRODUCCIÓN

Un **script** es un proceso por lotes. Es un fichero de texto cuyo contenido son mandatos del sistema operativo. Cuando se ejecute el script en el indicador del sistema se ejecutarán todos los mandatos que contiene.

Creamos script :	Ejecutamos script directamente con :	O haciendo primero :
cat > sc01		
clear	**sh sc01**	**chmod u+x sc01**
date	o :	y después :
who	**. sc01**	**sc01**
pwd		

MANEJO DE VARIABLES

Se pueden crear variables de tipo alfanumérico que podrán ser empleadas para manejar el entorno del sistema o en los procesos de usuario. Las variables de entorno suelen ser utilizadas por el sistema operativo y sus mandatos o por las aplicaciones instaladas en el sistema.

Se puede consultar las variables de entorno con la ordenes : **env** o **set**.
El usuario también puede crear nuevas variables así : nombrevariable=valor
Con los acentos graves se asigna el resultado de un mandato : nombrevariable=`orden`.
Para introducir el valor a guardar en la variable : **read** nombrevariable
Se puede hacer referencia al contenido de una variable con : **$**nombrevariable

PARÁMETROS DE UN SCRIPT

Son variables especiales que se utilizan con los scripts.

Nombre variable	Descripción
$0	Recoge el nombre del script que se está ejecutando.
$1 - $9	Parámetros adicionales con el script. $1 primero, $2 segundo, ...
$#	Contiene el número de parámetros que se pasan con el script.
$*	Contiene todos los parámetros que se pasan con el script.
$?	Código de retorno del último valor ejecutado en el script, 0 si es correcto.

```
echo  Introduce tu nombre
read  nombre
direc=`pwd`
echo  '$nombre con $LOGNAME se encuentra en $direc'
echo  "$nombre con $LOGNAME se encuentra en $direc"
```

```
echo  "Nombre del programa: " $0
echo  "Primero parámetro: " $1
echo  "Segundo parámetro: " $2
echo  "Número de parámetros: " $#
echo  "Todos los parámetros: " $*
```

SENTENCIAS DE PROGRAMACIÓN

El mandato exit

Además de terminar la sesión que se haya iniciado en el sistema, también se puede utilizar para terminar la ejecución de un script. De esta forma, se acompaña de un valor que se guarda en la variable $? para indicar el código de terminación del script.

El mandato if

El mandato if permite comprobar condiciones de un script.

```
if <condición>
then
  <mandato>
  <mandato>
  ...
fi
```

```
if <condición>
then
  <mandato>
  ...
else
  <mandato>
  ...
fi
```

```
if <condición1>
then
  <mandato(s)>
elif <condición2>
then
  <mandato(s)>
else
  <mandato(s)>
fi
```

```
if <condición> ; then <mandato> ; ... ; else <mandato> ; ...; fi
```

La condición debe tener una de las siguientes formas:

test <expresión>	**[** <expresión>]

Para la expresión se puede elegir diversos operadores:

Comparación de archivos:

-e <fichero>	Cierto si el fichero existe
-f <fichero>	Cierto si el fichero existe y es normal
-x <fichero>	Cierto si el fichero existe y es ejecutable
-r <fichero>	Cierto si el fichero existe y se puede leer
-w <fichero>	Cierto si el fichero existe y se puede escribir
-d <fichero>	Cierto si el fichero existe y es un directorio
-s <fichero>	Cierto si el fichero existe y no está vacio
-h <fichero>	Cierto si el fichero existe y es un enlace simbólico

Comparación de cadenas:

$cadena	Cierto si no es la cadena vacía
-z $cadena	Cierto si la longitud de la cadena es cero
-n $cadena	Cierto si la longitud de la cadena no es cero
"cadena1" = "cadena2"	Cierto si las dos cadenas son iguales
"cadena1" **!=** "cadena2"	Cierto se las dos cadenas son distintas
"cadena1" **<** "cadena2"	Cierto si cadena1 es menor que cadena2
"cadena1" **>** "cadena2"	Cierto si cadena1 es mayor que cadena2

Comparación de números enteros:

num1 **–eq** num2	Cierto si los dos números son iguales
num1 **–ne** num2	Cierto si los dos números son distintas
num1 **–gt** num2	Cierto si num1 es mayor que num2
num1 **–ge** num2	Cierto si num1 es mayor o igual que num2
num1 **–lt** num2	Cierto si num1 es menor que num2
num1 **–le** num2	Cierto si num1 es menor o igual que num2

Comparación lógica:

! <expresión>	Sirve para negar una expresión lógica
<expr1> **–a** <expr2>	Realiza un AND lógico. Cierto si todas las expresiones son ciertas.
<expr1> **–o** <expr2>	Realiza un OR lógico. Cierto si al menos una expresión es cierta.

```
mkdir $1
if test $? –eq 0
then
  echo El subdirectorio $1 ha sido creado
fi
```

```
if [ -f $1 –a –f $2 ]
then
  echo Existen los dos
fi
```

```
if [ -f $1 –a –f $2 ] ; then
  echo Existen los dos
fi
```

El mandato case

Alternativa múltiple que compara el contenido de una variable o expresión con una serie de patrones.
La última opción < *) > se elegirá cuando la variable no coincida con ningún patrón indicado
En cada patrón se puede indicar más de un valor separados por una barra < | >.
También se pueden indicar rangos de valores entre corchetes < [ ] >.

```
case <variable> in
<patron1>)
        mandato(s)
        ;;
<patron2>)
        mandato(s)
        ;;
*)
   mandato(s)
   ;;
esac
```

```
case $1 in
L|l)    echo  "Lunes "
        ;;
M|m)    echo  "Martes"
        ;;
S|s|D|d) echo  "Fin de semana"
        ;;
*)  echo "No sé que día es"
    ;;
esac
```

```
echo  -e "Introduce un caracter : \c"

read  car
case $car in
[A-Z])  echo $car " es mayúscula" ;;
[a-z])  echo $car " es minúscula" ;;
[0-9])  echo $car " es un número" ;;
*)  echo $car " no es válido" ;;
esac
```

El mandato for

Permite crear un bucle en el que una variable irá tomando todos los valores de una lista indicada.

```
for <variable> in <lista_de_valores>
do
    <mandato>
    <mandato>
    ...
done
```

```
echo Has introducido :
for var in $*
do
   echo $var
done
```

```
cuenta=0
for fichero in *
do
   cuenta=`expr $cuenta + 1`
done
echo Hay $cuenta ficheros
```

El mandato while

Permite crear un bucle de mandatos que se ejecutarán mientras la condición especificada sea cierta.

```
while <condición>
do
    <mandato>
    <mandato>
    ...
done
```

```
num=1
while test $num –le 5
do
   touch curso$num
   num=`expr $num + 1`
done
```

```
echo –e "Teclea $1 : \c"
read cadena
while [ "$cadena" != "$1" ]
do
    echo –e "Mal. Teclea $1 : \c"
    read cadena
done
```

El mandato until

Permite crear un bucle de mandatos que se ejecutarán hasta que la condición especificada sea cierta.

```
until <condición>
do
    <mandato>
    <mandato>
    ...
done
```

```
num=1
until test $num –gt 5
do
   touch curso$num
   num=`expr $num + 1`
done
```

```
echo –e "Teclea $1 : \c"
read cadena
until [ "$cadena" = "$1" ]
do
    echo –e "Mal. Teclea $1 : \c"
    read cadena
done
```