# *Comandos básicos LINUX*

**WHO**. El comando who muestra por pantalla información de los usuarios que están conectados en ese momento al sistema operativo. Aparecen 4 columnas. En la primera columna aparece el *login* del usuario que está activo en el sistema. La segunda indica que *terminal* está utilizando. La tercera y cuarta indica *día, mes y hora de conexión*. Su sintaxis es:

**who [opciones]**

Donde las **opciones** pueden ser:

**-u** visualiza la siguiente información: nombre del usuario, terminal desde el que se está conectado, hora a la que accedió al sistema, horas y minutos transcurridos desde la última actividad del usuario (un punto indica actividad en el último minuto), número de identificación del proceso shell del usuario (PID), y puede aparecer por último un comentario.

**am i** muestra en pantalla el login del usuario que esta activo en el terminal.

*$ who*

*root tty04 Jan 29 19:15*

*marta tty02 Jan 29 19:17*

*alumxyz tty03 Jan 29 19:18*

*ttyXX es un identificador del terminal desde el que se está accediendo al sistema.*

*Ejemplo:*

*$ who am i*

*alumxyz tty02 Jan 29 19:17*

**DATE.** Visualiza el día y la hora del sistema.

*Ejemplo:*

*$ date*

*Mon Jan 31 12:25:28 GMT 2005*

Lo primero que aparece es el *día* de la semana, escrito con las tres primeras letras del día, en inglés.

A continuación aparece el *mes*, al igual que el día, con las tres primeras letras del mes y en inglés.

Después del día y del mes, lo siguiente que aparece es la *hora*, con hora:minuto:segundo con un horario de 24 horas. GWT indica que pertenece al meridiano de Greenwich, y por último, el *año*.

**CAL.** Se utiliza para saber en que día de la semana caerá una fecha determinada. El formato de la orden es el siguiente:

**cal [mes] año**

Donde:

**mes** debe ser introducido con un número del 1 al 12 ya que algunas versiones no admiten la introducción del mes en letras ( de admitirla, sería en inglés y sólo las tres primeras).

**año** se introducirá mediante cuatro dígitos.

Sirva de ejemplo para ver el mes de diciembre de 2004.

*$ cal 12 2004*

*December 2004*

*S M Tu W Th F S*

*1 2 3 4*

*5 6 7 8 9 10 11*

*12 13 14 15 16 17 18*

*19 20 21 22 23 24 25*

*26 27 28 29 30 31*

Antes se comentó que no hacía falta indicar el mes; si no se introduce el mes, Linux mostrará todo el año completo.

**MAN.** Orden que nos proporciona el acceso al manual interactivo de Linux.

Permite visualizar información acerca de cualquier orden o programa Linux. Basta con teclear la orden man seguida del nombre de la orden de la cual se desea obtener información. Su sintaxis es:

**man nombre_comando**

*Ejemplo:*

*$ man cd*

**CLEAR**. Limpia La pantalla, dejándola simplemente con el prompt del sistema en la parte superior izquierda de la pantalla.

**MAIL**. Mecanismo de correo electrónico o e-mail, que permite enviar mensajes de unos usuarios a otros. Su sintaxis es:

**mail [usuarios]**

Si se utiliza mail con un usuario sirve para enviar un mensaje a un usuario. No es necesario que el usuario destinatario esté conectado en ese instante, ya que toda la correspondencia será depositada en su buzón que podrá ser consultada posteriormente. Si tenemos correo pendiente puede aparecer un mensaje como el siguiente: "*You have new mail*"

Si se utiliza mail sin parámetros, se visualizan en pantalla los diferentes mensajes con su correspondiente remitente que se tengan en el buzón. Para pasar de un mensaje a otro se pulsa enter.

*Ejemplo:*

*$ mail alumxyz*

*Subject : Asistencia a clase*

*Te recuerdo que es obligatoria la asistencia a clase.*

*CTRL+D*

*$ mail*

*SCO Openserver release 5.0. Type ? for help.*

*"/usr/spool/mail/alumlcg": 1 message 1 new*

*>N 1 pepe Thu Feb 21 11:25 13/445 "AYUDA"*

Cuando se está leyendo el correo se puede utilizar a partir del indicador & los siguientes comandos:

**x** Salir de mail sin alterar el buzón, sin grabar. Si salimos de este modo es como si no hubiésemos entrado

**q** Salir grabando

**p** Imprimir mensaje

**s** Guardar el mensaje en un archivo se pulsa s

**t** Leer el mensaje especificado, por ejemplo si se escribe t2 se lee el mensaje 2.

**d** Eliminar mensajes.

**d1** Elimina mensaje 1.

**d*** Elimina todos los mensajes.

**d 2-5** Elimina los mensajes del 2 al 5.

**u** Sin salir de mail se recuperan mensajes.

**u1** Recupera el mensaje 1.

**u*** Recupera todos los mensajes.

**u 3 7** Recupera los mensajes 3 y 7.

**u 3-7** Recupera los mensajes del 3 al 7.

**r** Responder a un mensaje.

**h** Mostrar lista de mensajes.

*$ mail*

*>N 1 alumabc Thu Feb 21 11:25 13/445 AYUDA*

*U 2 alumxyz Thu Feb 21 11:34 13/402 dos*

**N** Indica nuevo mensaje.

**U** Indica mensaje recibido anteriormente y no leído.

**>** Mensaje activo, si se pulsar intro es el que se visualiza.

**WRITE**. Esta orden se utiliza para comunicarnos con otros usuarios que estén en ese momento conectados a nuestro sistema. Para terminar hay que pulsar ctrl-d. Si se intenta enviar un mensaje a un usuario no conectado, se advertirá de que dicho usuario no se encuentra en sesión. Puede ocurrir que el usuario al que se le envíe el mensaje tenga desactivados los mensajes, en cuyo caso write fallará. Si hubiera un usuario conectado a varios terminales habría que especificar por cual sacar el mensaje. La sintaxis es:

**write usuario [tty]**

*$ write elisa*

*Message from alumlcg*

*Nos vemos en el jamaicano. ¿Estás ahí?*

*EOF*

Lo normal es que cuando iniciamos una comunicación con otro usuario este responda invocando a write, de tal manera que se establece una comunicación bidireccional. Si el sistema está muy cargado la salida write se puede ver retrasada.

*$ write alumlcg*

*Message from elisa*

*Claro que estoy y nos vemos*

*EOF*

**MESG**. Esta orden se utiliza para modificar los derechos de escritura por parte de otros usuarios en nuestro terminal, de tal manera que si alguien nos quiere enviar

un mensaje y tenemos desactivados estos derechos, no seremos interrumpidos. La prohibición de acceso de escritura no afecta al administrador del sistema. La orden **mesg** sin parámetros nos dirá si tenemos o no activa la recepción de mensajes. Si se tienen los mensajes desactivados, no recibiremos ninguno aunque alguien nos los envíe. Estos mensajes se perderán aunque después se vuelva a habilitar la posibilidad de recibirlos. Su sintaxis es:

**mesg [y/n]**

*$ mesg*

*is y*

*$ mesg n*

*$ mesg*

*is n*

**PWD.** Presenta por pantalla el camino absoluto del directorio actual. Para las personas habituadas a trabajar con MS-DOS el prompt del sistema les muestra constantemente el directorio donde están trabajando. En Linux no ocurre lo mismo generalmente, por lo que es bueno saber en qué directorio estamos trabajando si en algún momento estamos perdidos.

**CD**. Permite cambiar de un directorio a otro. Su sintaxis es:

**cd [camino]**

Cada vez que un usuario se conecta al sistema se sitúa en su directorio personal o home. Si se utiliza la orden cd sin argumento, le sitúa directamente en el directorio home o de conexión, con independencia del directorio en que se encuentre.

Si se desea entrar en un directorio determinado se sustituye el argumento "camino" por el camino absoluto o relativo del directorio al que se desea acceder.

*Ejemplos:*

*$ cd*

*Nos sitúa en el directorio home.*

*$ cd ..*

*Nos sitúa en el directorio padre del directorio actual.*

*$ cd /home/i82fecoe/mail*

*Nos sitúa en el directorio mail utilizando un camino absoluto.*

**LS**. Permite ver el contenido de un directorio; muestra todos los archivos del directorio. Puede ocurrir que un directorio no esté disponible para la lectura, por lo que no se podrá ver su contenido hasta que no sea desprotegido. Su sintaxis es:

**ls [opciones] [directorio]**

Si se utiliza la orden ls sin argumentos visualiza el contenido del directorio actual.

Si se desea visualizar el contenido de un directorio concreto se incluirá su camino como argumento.

*Ejemplo:*

*$ ls mail*

*Nota: Cuando se utiliza la orden ls sin opciones sólo se visualizan los nombres de los ficheros y/o subdirectorios que contiene el directorio.*

Las **opciones** más importantes de la orden ls son:

**- l** listado en formato largo incluyendo permisos, usuario, grupo, etc.

**-t** ordenado por fecha de modificación.

**-a** lista todos los ficheros, incluso los que empiezan por punto, ordenados alfabéticamente.

**-s** tamaño de los ficheros en bloques.

**-i** informa del número de inodo de cada fichero.

*Ejemplo:*

*$ ls –l*

Este listado suele incluir siete columnas de información:

(1) Contiene diez caracteres. El primer carácter indica si el archivo es de tipo ordinario (-) o directorio (d). Los nueve caracteres restantes dan una representación simbólica de los derechos de acceso actuales que tiene el archivo.

Estos nueve caracteres se dividen en tres columnas con tres caracteres cada una, aunque no estén separadas por espacios. La primera columna representa al usuario, la segunda al grupo de trabajo al que pertenece el usuario (en el caso de que pertenezca a alguno), y la tercera al resto de los usuarios del sistema.

Cada uno de los tres grupos de permisos consta de tres caracteres. Estos representan los permisos de lectura (r), escritura (w) y de ejecución (x).

(2) Contiene el número de enlaces en el sistema de archivos a este archivo. Hay casos en los que interesa que un grupo de usuarios en el mismo sistema UNIX

puedan compartir un archivo mientras trabajan en un proyecto conjunto. La compartición del archivo permite su actualización por todos los usuarios en cualquier instante. Para que esto sea posible se crean enlaces al archivo que va a estar sujeto a modificaciones mediante la orden **ln**, que se verá más adelante. De esta forma los usuarios que hayan creado enlaces podrán acceder al archivo como si este hubiera sido creado en su propio directorio de trabajo.

(3) Muestra el propietario del archivo (el usuario que creó inicialmente el archivo).

(4) Visualiza el tamaño del archivo en caracteres (o bytes).

(5) Muestra la fecha en la que el archivo fue modificado por última vez.

(6) Muestra la hora en la que el archivo fue modificado por última vez.

(7) Contiene el nombre del archivo.

**MKDIR**. Permite crear uno o varios directorios. Su sintaxis es:

**mkdir [-p] [camino] / nombre/s_directorio/s**

La opción **–p** permite que los directorios padre que no existan sean creados.

*Ejemplo:*

*$ mkdir –p ../ejercicios/tema6*

*Crea el directorio ‚tema6`. Al utilizar la opción –p también crea el directorio padre ‚ejercicios`, que no existe.*

*Ejemplo:*

*$ mkdir tema5*

*Crea el directorio ‚tema5` sobre el directorio actual.*

*Ejemplo:*

*$ mkdir /home/i82fecoe/ejercicios/tema1 tema2 tema3 tema4*

**RMDIR**. Borra uno o varios directorios. Para poder borrar un directorio tiene que estar vacío. Su sintaxis es:

**rmdir [camino] / nombre/s_directorio/s**

Su funcionamiento es igual que la orden mkdir.

*Ejemplo:*

*$ rmdir tema1 tema2 tema3*

*Ejemplo:*

*Si en el directorio tema4 creamos un directorio ,parte2` e intentamos borrar ,tema4`, nos aparecer{ un mensaje de error y no se borrar{ el directorio, ya que no está vacío. Por tanto, para poder borrarlo tendremos que borrar el directorio ,parte2` que acabamos de crear y a continuación proceder al borrado del directorio ,tema4`.*

**CAT**. Visualiza los ficheros especificados como parámetros

*Ejemplo:*

*$ cat coches motos*

**CP**. Se utiliza para copiar ficheros. Su sintaxis es:

**cp [opciones] fichero1 fichero2**

Copia el contenido de "fichero1" en "fichero2". También se puede especificar el camino absoluto o relativo de cada uno de los ficheros.

*Ejemplo:*

*$ cp coches ejercicios/cars*

*Copia el fichero ,coches` del directorio actual en el directorio ejercicios, con el nombre ,cars`.*

Las **opciones** más importantes son:

**-i** pedirá confirmación si el fichero destino ya existe antes de sobrescribirlo.

**-r** copia un directorio junto con sus ficheros y subdirectorios en otro directorio. Si el directorio destino no existe, lo crea.

*Ejemplo:*

*$ cp coches mbox ejercicios*

*Copia los archivos ,coches` y ,mbox` sobre el directorio ,ejercicios`. Cuando se utilizan varios ficheros para copiar, el destino tiene que ser un directorio.*

*Ejemplo:*

*$cp –i coches ejercicios*

*Copia el archivo ‚coches` en el directorio ‚ejercicios`, no sin antes haber pedido la confirmación para sobrescribirlo, puesto que ya hay un archivo con igual nombre en el directorio ‚ejercicios`.*

*Ejemplo:*

*$ cp –r ejercicios so/pruebas*

*Copiamos el directorio ‚ejercicios` y todo su contenido al directorio ‚so`, en un nuevo directorio que creamos al hacer la copia, llamado ‚pruebas`.*

**MV**. Cambia un fichero de un directorio a otro. Sus sintaxis son:

**a) mv [opciones] fichero1 fichero2**

**b) mv [opciones] directorio1 nuevo_directorio**

**c) mv [opciones] fichero/s directorio**

**d) mv [opciones] directorio1 directorio2**

En los dos primeros casos, la orden mv lo que hace es cambiar el nombre de un fichero o un directorio, según el caso. Mientras que en los siguientes lo que realiza es un cambio de localización. Es decir, se desplaza un fichero, o bien todo el contenido de un directorio, a otro directorio.

Las **opciones** más importantes son:

**-i** si el fichero destino existe, pide confirmación antes de sobrescribirlo.

**-f** es la opción por defecto. Si el fichero existe, lo sobrescribe sin pedir confirmación.

*Ejemplos:*

*$ mv coches carros*

*Cambia el nombre del fichero ‚coches` por ‚carros`.*

*$ mv so sistemas_operativos*

*Cambia el nombre del directorio ‚so` por ‚sistemas_operativos`.*

*$ mv carros list sistemas_operativos*

*Mueve los ficheros ‚carros` y ‚list` al directorio ‚sistemas_operativos`.*

*$ mv pruebas sistemas_operativos*

*Mueve los directorios y ficheros del directorio ‚pruebas` al directorio ‚sistemas_operativos`.*

**RM.** Se utiliza para borrar ficheros. Su sintaxis es:

**rm [opciones] fichero/s**

Sus **opciones**:

**-r** borra recursivamente. Se emplea para borrar directorios.

**-f** borra todos los ficheros sin confirmación.

**-i** pide confirmación para borrar cada fichero.

*Ejemplos:*

*$ rm –f carros list*

*Borra los ficheros ‚carros` y ‚list` del directorio actual sin pedir confirmación.*

*$ rm –r pruebas*

*Borra el directorio ‚pruebas` y todo su contenido recursivamente. Solicita la confirmación.*

**FIND**. La orden find permite efectuar búsquedas de archivos. Lleva como argumentos los nombres de directorios seguidos de varias opciones posibles que especifican el tipo y el criterio de la búsqueda. Find busca dentro de los directorios indicados y en sus subdirectorios archivos que cumplan con el criterio. Su sintaxis es:

**find directorio [condiciones] acción**

Donde:

**directorio** es el directorio a partir del cual comienza la búsqueda.

**condiciones** se puede substituir por:

***–name nombre_fichero*** Búsqueda por el nombre del fichero.

***-perm*** **NNN** Búsqueda por la máscara de permisos en octal.

***-user nombre_usuario*** Búsqueda por el nombre de usuario.

***-size n*** Fichero con tamaño n bloques.

***-mtime n*** Ficheros modificados hace n días.

***-atime n*** Ficheros cuyo último acceso fue hace n días.

**acción** indica la acción a realizar con los ficheros encontrados.

*Ejemplo:*

*$ find informes –name lunes*

*Se busca en el directorio informes el fichero lunes*

*Ejemplo:*

*$ find programas –name '*.c' –print*

*Se busca en el directorio programas ficheros que tengan extensión .c y se muestra la ruta de los ficheros.*

Se pueden usar los caracteres comodín del shell como parte del patrón del criterio de búsqueda de archivos. El carácter comodín debe ir delimitado con el fin de evitar su evaluación por el shell.

*Ejemplo:*

*$ find / -user ,profes`*

**LN**. Un *inodo* es la estructura de datos en disco que describe y almacena los atributos del fichero y su localización. Cada inodo tiene un número distinto y cada fichero tiene su inodo. Cuando se crea un fichero un inodo se le asigna a él y este es único. La información que se guarda en un inodo es: propietario y grupo, tipo de fichero, permisos, fecha de creación y modificación, número de links al fichero, tamaño del fichero,… Los inodos guardan toda la información del fichero excepto su localización en el directorio y su nombre.

Un enlace es un mecanismo que permite referirse con distinto nombre de fichero a un mismo fichero en disco. Existen dos tipos de links: *físico*,(duro o hardlink) y *simbólico* (blando o softlink).

**Enlace duro**: Crea un enlace, asociando uno o más nombres de fichero con el mismo inodo. Se realizan mediante la orden ln. Su sintaxis es:

**ln archivo destino**

Crear un enlace consiste en hacer que un mismo fichero aparezca bajo diferentes nombres, en diferentes lugares del árbol de directorios. Físicamente sólo existirá un fichero, pero se podrá acceder a él bajo diferentes nombres, de modo que si se modifica el fichero independientemente del nombre con el que se haya accedido, todos sus enlaces resultan modificados. No se puede hacer de directorios.

*Ejemplo:*

*$ ln index hlink*

*Crea un link con nombre hlink con el mismo inodo que el fichero index*

**Enlace simbólico**: Asocia un nombre a un nuevo inodo, que indica la posición en disco donde se guarda la información del path donde está el fichero. Se realizan mediante la orden ln con el parámetro –s. Su sintaxis es:

**ln –s archivo destino**

*Ejemplo:*

*$ ln -s index slink*

*Asocia al nombre slink un nuevo inodo. Este inodo indica la posición en disco donde se guarda el path completo del fichero index.*

Cuando se hace un enlace simbólico se le asigna un nº de inodo nuevo. En ningún caso se incrementa el nº de enlaces. Sí se puede hacer de directorios. Los enlaces simbólicos tienen su propio tipo de archivo que se representa por una l y ocupan mucho menos que el archivo original porque solamente almacenan el nombre de la ruta de otro archivo.

Los dos tipos de enlaces permiten visualizar el contenido de Index utilizando los nombres de hlink e slink pero existen diferencias en la estructura interna. Casos:

1) Si no ocurre nada, ambos link visualizan el fichero index

*$ cat index*

*Este es index de Elisa*

*$ cat hlink*

*Este es index de Elisa*

*$ cat slink*

*Este es index de Elisa*

*$ ls -li*

*total 20*

*36409 -rw-r--r-- 1 librada8 users 24 Feb 25 10:46 index*

*36409 -rw-r--r-- 2 librada8 users 24 Feb 25 10:46 hlink*

*36852 lrwxrwxrwx 1 librada8 users 5 Feb 25 10:56 slink -> index*

*34568 drwxrw_rw_ 3 librada8 users 5 Feb 25 10:56 ultimo*

2) Si se mueve index que significa solo añadir una nueva entrada en el nuevo directorio y borrarla del antiguo no se afecta el hardlink pues el inodo del fichero

es el mismo. Por el contrario softlink apunta a algo que no existe pues ya no está en el mismo directorio. Lo mismo ocurre si se borra el fichero index.

*$ mv index seg/*

*$ ls –li*

*total 20*

*36409 -rw-r--r-- 2 librada8 users 24 Feb 25 10:46 hlink*

*36852 lrwxrwxrwx 1 librada8 users 5 Feb 25 10:56 slink -> index*

*cat hlink*

*Este es index de Librada*

*$ cat slink*

*cat: slink: No such file or directory*

*3)Si creamos un nuevo index*

*$ cat > index*

*Este es el nuevo index de Librada*

*$ cat hlink*

*Este es index de Librada*

*$ cat slink*

*Este es el nuevo index de Librada*

*$ ls -li*

*total 20*

*36485 -rw-r--r-- 1 librada8 users 24 Feb 25 10:46 index*

*36409 -rw-r--r-- 2 librada8 users 24 Feb 25 10:46 hlink*

*36852 lrwxrwxrwx 1 librada8 users 5 Feb 25 10:56 slink -> index*

ls -l identifica los diversos tipos de ficheros que existen en LINUX. Los más comunes son:

- ficheros regulares

d directorio

l link simbolico

**MORE.** Orden que muestra el contenido de un archivo paginando la salida por pantalla. Su sintaxis es:

**more fichero1 ....**

Pulsando <Enter> aparece una nueva línea.

Pulsando barra espaciadora aparece una nueva pantalla.

Pulsando q finalizamos la visualización.

*Ejemplo:*

*$ more fich1*

*Ejemplo:*

*$ ls | more*

**LESS.** Orden que muestra el contenido de un archivo paginando la salida por pantalla y permitiendo moverse tanto hacia al final del fichero como hacia el principio. Su sintaxis es:

**less fichero1 ....**

Pulsando <Enter> aparece una nueva línea.

Pulsando barra espaciadora aparece una nueva pantalla.

Pulsando q finalizamos la visualización.

*Ejemplo:*

*$ less fich1*

*Ejemplo:*

*$ ls | less*

**TAIL**. Visualiza el final de un fichero, por defecto visualiza las diez últimas líneas. Su sintaxis es:

**tail [opciones] [fichero]**

Donde las **opciones** pueden ser:

**+n** empieza a visualizar a partir de *n* líneas del principio del fichero.

**- n** empieza a visualizar a partir de *n* líneas del final del fichero.

*Ejemplo:*

*$ tail –10c fich*

*Obtiene los 10 últimos caracteres de fich.*

*Ejemplo:*

*$ ls / tail*

*Obtiene las 10 últimas líneas del listado obtenido mediante la orden ls..*

**HEAD**. Obtiene en la salida estándar las primeras líneas de un fichero, por defecto las 10 primeras líneas. Su sintaxis es:

**head [-n] [fichero(s)]**

Donde:

**- n** obtiene las **n** primeras líneas.

*Ejemplo:*

*$ cat prueba*

*Buenos días*

*esto es una prueba*

*esta ya no se va a ver al hacer head*

*$ head –2 prueba*

*Buenos días*

*esto es una prueba*

*Ejemplo:*

*$ ls | head*

*Obtiene las 10 primeras líneas del listado obtenido mediante la orden ls..*

**WC**. Cuenta el número de caracteres, palabras y líneas de uno o más *fichero(s),* incluye espacios en blanco y caracteres de salto de línea. Cuando se indica el nombre de más de un fichero **wc** proporciona los contadores individuales y su totalidad y se presentan los de cada fichero etiquetados con su nombre. Si no se especifica ningún fichero, leerá de la entrada estándar. Su sintaxis es:

**wc [opciones] [fichero(s)]**

Donde las **opciones** pueden ser:

**-c** visualiza sólo el número de *caracteres* del fichero.

**-l** visualiza sólo el número de *líneas* del fichero.

**-w** visualiza sólo el número de *palabras* del fichero.

*Ejemplo:*

*$ cat prueba*

*Buenos dias*

*esto es una prueba*

*$ wc prueba*

*2 6 31 prueba*

*En este ejemplo, el comando imprime el número total de líneas (2), de palabras (6), y de caracteres (31) (incluye espacios en blanco y nueva línea) del fichero prueba.*

*Ejemplo:*

*$ wc c**

*4 4 26 coche*

*4 4 22 coche2*

*8 8 48 total*

*Visualiza los ficheros que empiezan por c y suma el total.*

También puede formar parte de una tubería para contar las palabras o líneas de la entrada:

*Ejemplo:*

*$ ls | wc –l*

*Cuenta las líneas del directorio activo.*

**GREP**. Es un filtro que permite buscar cadenas de caracteres (patrón) en los archivos que se le indica y muestra la línea del fichero que contiene el patrón. Su sintaxis:

**grep [opciones] patrón [fichero(s)]**

Donde las **opciones** pueden ser:

**-c** visualiza el nº total de líneas donde se localiza el patrón.

**-i** elimina la diferencia entre mayúsculas y minúsculas.

**-l** visualiza sólo el nombre de los ficheros donde se localiza el patrón.

**-n** visualiza el nº de línea donde aparece el patrón, así como la línea completa.

**-v** visualiza las líneas del fichero donde no aparece el patrón.

*Ejemplo:*

*$ cat pruebagrep*

*hola voy a probar a*

*ver si me dice en que lugares*

*aparece la letra a*

*el comando grep*

*y este no tiene la letra*

*$ grep ,letra` pruebagrep*

*aparece la letra a*

*y este no tiene la letra*

*$ grep -c ,letra` pruebagrep*

*2*

*$ grep -n ,letra` pruebagrep*

*3:aparece la letra a*

*5:y este no tiene la letra*

*$ grep -v -n ,letra` pruebagrep*

*1:hola voy a probar a*

*2:ver si me dice en que lugares*

*4:el comando grep*

**Caracteres especiales dentro del patrón. Expresiones regulares**

Una expresión regular es un patrón que define a un conjunto de cadenas de caracteres.

**^** Indica que la cadena buscada debe encontrarse al principio de línea.

**$** La cadena deberá encontrarse al final de la línea.

*$ cat semana.doc*

*Lunes*

*Martes*

*Miércoles*

*$ grep ,^M` semana.doc*

*Martes*

*Miércoles*

*$ grep ,es$` semana.doc*

*Lunes*

*Martes*

*Miércoles*

**.** Se corresponde con cualquier carácter, pero sólo uno y en la posición marcada por el punto.

*$ cat nombres*

*López*

*Gómez*

*Montes*

*$ grep ,.ó.ez` nombres*

*López*

*Gómez*

**[a...z]**Coincide con cualquier carácter de los que están incluidos en la lista.

*$cat países*

*Portugal*

*Francia*

*Panamá*

*Lisboa*

*España*

*$ grep ,[P]` países*

*Portugal*

*Panamá*

*$ grep ,[Pp]` países*

*Portugal*

*Panamá*

*España*

*$ grep ,[PF]` países*

*Portugal*

*Francia*

*Panamá*

*$ grep ,[n]` países*

*Francia*

*Panamá*

**[x-y]** Sustituyen a un carácter y puede ser cualquiera de los comprendidos en el rango definido entre [].

*$ grep ,[o-s]` países*

*Portugal*

*Francia*

*Lisboa*

*España*

**[^]** Indica exclusión de los caracteres incluidos entre corchetes.

*$ grep ,^[^P]` país*

*Francia*

*Lisboa*

*España*

*$ cat dicho*

*Cada fracaso*

*enseña al hombre*

*algo que*

*debería haber*

*aprendido.*

*El único hombre*

*que nunca se equivoca es*

*el que nunca*

*hace nada*

*$ grep ,^a` dicho*

*algo que*

*aprendido.*

*$ grep ,qu[aeiou]` dicho*

*algo que*

*que nunca se equivoca es*

*el que nunca*

*$ grep ,equ[aeiou]` dicho*

*que nunca se equivoca es*

**SORT**. El filtro sort se utiliza para ordenar líneas. Esta clasificación la realiza por orden alfabético o numérico teniendo en cuenta que cada línea del fichero es un registro compuesto por varios campos, los cuales están separados por un carácter denominado separador de campo (tabulador, espacio en blanco, dos puntos…). Su sintaxis es:

**sort [opciones] [+campo] [fichero/s]**

Donde las **opciones** pueden ser:

**-f** ignora mayúsculas y minúsculas.

**-n** ordena campos numéricos por valor no por caracteres.

**-r** orden inverso.

**-u** suprime líneas repetidas en el fichero de salida

**-d** ignora caracteres especiales (solo letras y números)

**-t** indica el delimitador de campos, por defecto el espacio en blanco.

**-o** para que el fichero ordenado se almacene en el propio fichero. Funciona como una redirección.

**+campo** cuando se quiere ordenar por un campo determinado del registro. Por defecto un campo es cualquier secuencia de caracteres separados por espacios en blanco. Los campos del registro se numeran a partir del 0. El número que sigue al **+** es el número de campo por el que se quiere comenzar la ordenación. Para parar la ordenación en otro campo se pone **–** nº de campo.

**ficheros** nombres de los ficheros a ordenar.

*$ cat ciudades*

*barcelona*

*oviedo*

*Paris*

*Alicante*

*alicante*

*OVIEDO*

Si se ordenan, las mayúsculas por defecto van primero:

*$ sort ciudades*

*OVIEDO*

*Paris*

*alicante*

*alicante*

*barcelona*

*oviedo*

*$ sort -o ciudades ciudades*

*El fichero ordenado se almacena en el propio fichero.*

*$ sort -f ciudades*

*alicante*

*alicante*

*barcelona*

*OVIEDO*

*oviedo*

*Paris*

*$ sort -f -u ciudades*

*alicante*

*barcelona*

*OVIEDO*

*Paris*

*$ sort -fur ciudades ⇔ $ sort –f-u-r ciudades*

*Paris*

*OVIEDO*

*barcelona*

*alicante*

*$ sort –u f1 f2 f3 > lista*

*Fusiona los 3 ficheros eliminando las líneas repetidas.*

*$ cat numeros*

*101*

*112*

*10*

*373*

*64*

*$ sort numeros*

*10*

*101*

*112*

*373*

*64*

*$ sort –n numeros*

*10*

*64*

*101*

*112*

*373*

Se puede indicar que utilice algún tipo de separador específico utilizando la opción –t y a continuación el separador e indicar que se ordene por una determinada columna.

*$ cat ciudades*

*España:Alicante*

*España:Barcelona*

*Alemania:Berlín*

*Grecia:Atenas*

*$ sort –t: +1 ciudades*

**CUT**. El filtro cut se usa para cortar columnas en ficheros y pasar a la salida estándar las columnas o campos de la entrada estándar o del archivo especificado. Por defecto el separador de campos es <TAB>. Su sintaxis es:

**cut [opciones] lista [fichero(s)]**

Donde las **opciones** pueden ser:

**-c** corta caracteres.

**-f** corta campos.

**-d** especifica el carácter de separación entre los distintos campos (delimitador).

**lista** especifica que columnas se van a cortar. Estas se identifican por su número a partir del 1 y caben las siguientes posibilidades:

***Un número***: corta desde ese campo hasta el final de la línea. *Ej. $cut -f1* (corta la primera columna)

***Una lista de campos separados por comas***: corta esos campos. *Ej. $cut -f2,3* (corta los campos 2 y 3).

***Un rango de campos o caracteres***: corta todos los elementos incluidos en el rango. *Ej. $cut -c 1-72* (corta los 72 caracteres de cada línea).

*Ejemplo:*

*$ cat rios.dat*

*Alemania:Berlín:Danubio*

*Francia:París:Sena*

*España:Madrid: Manzanares*

*Portugal:Lisboa:Tajo*

*$ cut –f2 -d`:` rios.dat*

*Berlín*

*París*

*Madrid*

*Lisboa*

*$ cut –f1,3 –d`:` rios.dat > rios1*

*$ cat rios1*

*Alemania :Danubio*

*Francia:Sena*

*España: Manzanares*

*Portugal: Tajo*

*$ cut –c 1-2 rios.dat*

*Al*

*Fr*

*Es*

*Po*

**PASTE**. Este comando une las columnas de distintos ficheros en otro fichero de salida. Su sintaxis es:

**paste [-d separador] fichero1, fichero2, ... ficheroN**

Donde:

**-d** especifica el carácter que se desea para separar los campos.

*$ cat moneda.dat $ cat pais.dat*

*alemania:marco:20 alemania*

*francia:franco:30 francia*

*españa:peseta:180 españa*

*portugal:escudo:240 portugal*

*$ paste –d ,:` moneda.dat pais > estado*

*$ cat estado*

*alemania:marco:20:alemania*

*.....*

**TEE**. En ciertas ocasiones puede interesar además de redireccionar la salida de una orden a un archivo, visualizar los resultados obtenidos. La orden tee se creó con esta intención. Su sintaxis es:

**tee fichero**

*$ ls –l | tee fichero*

*Visualiza por pantalla y al fichero.*

*$ ls –l | tee dirlist | wc*

*Visualiza por pantalla, lo envía al fichero y cuenta las líneas.*

*$ ls –l | wc | tee dirlist*

*Visualiza por pantalla, cuenta las líneas y lo envía al fichero.*

**USERADD**. Permite crear usuarios en Linux. Hay que tener privilegios de *root* para ejecutar este comando. Su sintaxis es:

**useradd nombre_usuario [-u uid] [-g gid] [-d home-directory] [-m ] [-s shell] [-g grupo] [-c comentarios]**

Donde:

**-u uid** sirve para especificar un UID al usuario. Debe ser un número positivo y único. Por defecto, el sistema asigna el primer UID libre.

**-g gid** selecciona el GID del grupo principal del usuario. Este GID deberá existir.

**-d home_directory** para especificar un directorio personal concreto. Si no se especifica, se asume el directorio.

**-m** crea el directorio home para el nuevo usuario.

**-s shell** especifica un shell de inicio para el usuario.

**-c comentarios** introduce un comentario para tener más información del usuario.

**USERDEL**. Permite borrar cuentas de usuarios en Linux. Hay que tener privilegios de *root* para ejecutar este comando. Su sintaxis es:

**userdel [-r] nombre_usuario**

Donde:

**-r** para eliminar todos los archivos y subdirectorios contenidos dentro del directorio de trabajo del usuario a eliminar.

**nombre_usuario** identifica al usuario que se va a borrar

**USERMOD**. Una vez creada una cuenta de usuario se puede cambiar cualquier característica editando directamente el fichero /etc/passwd o usando el comando usermod. *Por ejemplo si quieres modificar el UID de la cuenta usr1 desde 501 al 502 puedes usar el comando: usermod –u 502 usr1.*

Sintaxis:

**usermod [-c comentarios] [-d home_directory] [-e final de datos]**

**[-g grupo inicial] [-G otros grupos *,…++*-l nombre de login]**

**[-s shell] [-u uid] [-o]] login**

Donde:

**-c comentarios** modifica el comentario acerca del usuario.

**-d home_directory** para especificar un directorio personal concreto. Si no se especifica, se asume el directorio.

**-e final de datos** fecha en la que la cuenta de usuario será desactivada. Formato MM/DD/YY

**-g grupo inicial** nuevo número de grupo o nombre de grupo. El grupo deberá existir.

**-G otros grupos** lista de grupos suplementarios de los que el usuario es miembro. Cada grupo se separa del siguiente por coma, sin espacio en blanco. Los grupos deben existir. Si el usuario es miembro de un grupo que no esté en esta lista, se borrará el grupo.

**-l nombre de login** cambio de login

**-s shell** cambio de shell

**-u uid** cambio de número de usuario. Este valor será único a menos que se use la opción **–o**. No admite valores negativos. Cualquier **uid** de fichero que esté en el directorio origen del usuario se cambiará automáticamente al nuevo uid, pero no pasa así con los ficheros que estén en otro directorio, que deberán ser cambiados manualmente (usando chmod, chgrp, chown).

**login** login actual del usuario que vamos a modificar.

**GROUPADD**. Permite crear grupos en Linux, al igual que podría hacerse modificando manualmente el fichero **/etc/shadow**. Su sintaxis es:

**groupadd [-g gid] [-o] [-r] grupo**

Donde:

**-g gid** especifica el GID del grupo que se va a crear.

**-o** indica que el grupo puede tener asociado más de un GID.

**-r** crea un grupo de sistema.

**grupo** nombre asignado al grupo que se va a crear.

*Ejemplo:*

*$ groupadd –g 500 claseasi*

*Añade el grupo 500 de nombre claseasi*

**GROUPDEL**. Permite borrar grupos en Linux. Su sintaxis es:

**groupdel grupo**

Donde:

**grupo** nombre asignado al grupo que se va a borrar.

*Ejemplo:*

*$ groupdel claseasi*

**GROUPMOD**. Permite modificar grupos en Linux. Su sintaxis es:

**groupmod [–g nuevo_gid] [- n nuevo_nombre] grupo**

Donde:

**-n nuevo_nombre** especifica el nuevo nombre del grupo.

**-g nuevo_gid** especifica el nuevo GID del grupo.

**grupo** nombre del grupo que se va a modificar

*Ejemplo:*

*$ groupmod clasenueva claseasi*

*Cambia el nombre del grupo claseasi a clasenueva.*

**NEWGRP**. El fichero **/etc/group** contiene los grupos del sistema. Cuando se accede al sistema siempre se hace con el mismo grupo. Pero un usuario puede pertenecer a más de un grupo aunque sólo está activo uno. El usuario puede necesitar cambiarlo por otro de los que pertenezca para crear nuevos ficheros. Para ello usará el comando que nos ocupa. Ha de ser el usuario desde su login quien ejecute esta operación. Su sintaxis es:

**newgrp nuevo_grupo**

Si el usuario desea visualizar la lista de nombres de los grupos a los que pertenece, bastará con que ejecute la orden **groups**.

**CHMOD**. Permite modificar los permisos de uno o varios ficheros. Su sintaxis es:

**chmod mascara fichero/s**

donde la **máscara** puede ser *octal* o *simbólica*.

*Ejemplo:*

*$ chmod 652 practica*

*El propietario de practica tendrá permiso de lectura y escritura (4+2=6), el grupo de lectura y ejecución (4+1=5) y otros de escritura (2).*

*Si quisiéramos expresar lo mismo mediante una máscara simbólica, pondríamos:*

*$ chmod u=rw, g=rx, o=w practica*

La máscara simbólica está formada por tres códigos:

**a) Clases de usuarios:**

Símbolo Significado

u propietario

g grupo

o otros

a todos

**b) Permisos:**

Símbolo Significado

r lectura

w escritura

x ejecución

**c) Operación:**

Símbolo Significado

+ añadir permisos

- quitar permisos

= asignar permisos

*Ejemplo:*

*$ ls –l*

*-rw-r-x-w-1 cbt staff 512 Feb 09 18:55 practica*

*$ chmod u+x, g-x practica*

*$ ls –l*

*-rwxr---w-1 cbt staff 512 Feb 09 18:55 practica*

*Da permiso de ejecución del fichero al propietario y le quita el permiso de ejecución al grupo.*

**CHOWN**. Este comando cambia el propietario de un fichero. La sintaxis es:

**chown nuevo_propietario fichero/s**

Donde:

**nuevo_propietario** se sustituye por el nombre o el identificador de usuario del nuevo propietario.

*Ejemplo:*

*$ chown pepe carta*

**CHGRP.** Este comando cambia de grupo a un fichero. La sintaxis es:

**chgrp nuevo_grupo fichero/s**

Donde:

**nuevo_grupo** se sustituye por el nombre o el identificador de grupo del nuevo grupo.

*Ejemplo:*

*$ chgrp usuarios1 carta1 carta2*

**PS**. Permite conocer lo que está ocurriendo en el sistema. Esta orden lista todos los procesos activos en ejecución en la máquina. Su sintaxis es:

**ps [opciones]**

Si se utiliza sin opciones, se obtendrá información referente a los procesos asociados con el terminal en el que estamos. Gracias a esta orden se muestran datos como el PID de cada proceso, el terminal en el que se están ejecutando, el tiempo que llevan en ejecución y el nombre del propio proceso.

Los números de proceso se asignan secuencialmente cuando se generan nuevos procesos hijos. El proceso 0 es el primero que se genera al principio de conectar con el sistema; el proceso 1 es el proceso *init*, a partir de cual se generan todos los restantes. El resto de procesos empezarán por el 2 como identificador. Lo normal es que no se lancen tantos procesos como para que el sistema no pueda ejecutarlos.

La lista de opciones del comando **ps** es demasiado larga, pero mostraremos algunos de los parámetros más importantes mediante ejemplos.

*Ejemplos:*

*$ ps -f -u usuario01*

*Vemos los procesos que está ejecutando el usuario01.*

*$ ps -f -u tty07*

*Vemos los procesos que están ejecutándose en el Terminal asociado al dispositivo tty07.*

**KILL**. Este comando mata a un proceso cuando se está ejecutando, es decir, finaliza su ejecución.

Para terminar el proceso, es suficiente con conocer el número identificativo asignado por el sistema.

*Ejemplo:*

*$ kill 201*

Cuando el comando **kill** se utiliza sin argumentos (de los 20 que tiene aproximadamente), se envía una señal al proceso de tipo 15. Esta señal indica al sistema que tiene que detener el proceso en ejecución. Si queremos eliminar un proceso, sea el que sea, y de la forma anterior no se elimina, es debido a que el

sistema no permite detener el proceso en ejecución. Si queremos terminarlo de todas formas, enviaremos el valor 9 como parámetro del comando kill.

*Ejemplo:*

*$ kill -9 201*

Se ha de tener en cuenta que el administrador 'root', podrá matar todos los procesos que estén ejecutando otros usuarios, así también, los usuarios podrán matar sus propios procesos, y ningún usuario podrá matar otro proceso distinto del suyo.

Supongamos que a un usuario se le ha quedado bloqueado su terminal; éste no podrá hacer nada, ni siquiera ejecutar el comando kill. En este caso, el administrador del sistema puede solucionar el problema eliminando desde el servidor o desde cualquier ordenador operativo sobre los que tenga los derechos de administrador, una orden de matar el *Shell* del ordenador bloqueado, es decir, cancelar el proceso *Shell* que se está ejecutando por ese usuario.

Si lo que deseamos es terminar todos los procesos existentes en el sistema, ejecutaremos el comando **kill** con el valor **0**.

**SLEEP**. Este comando sirve para detener, por un intervalo de tiempo especificado en segundos, un proceso determinado. Suele utilizarse en programación *Shell*, similar a la programación BATCH.

**WAIT**. Permite detener un proceso ejecutándose en segundo plano mientras se está ejecutando un proceso hijo.

**wait [pid]**

**TIME.** Calcula el tiempo que un proceso tarda en ejecutarse.

**time [opciones] [comando] [argumentos]**