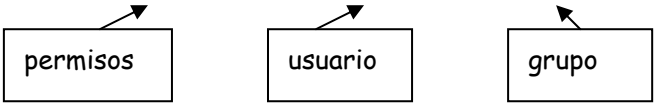
1 Permisos en Linux.

1.1 Introducción.

La gestión de permisos en Linux (proceso de autorización para uso de recursos) es bastante distinta de la que hemos estudiado anteriormente en Windows. Linux usa un esquema de permisos bastante más simple, ya que las ACL de los recursos sólo admiten 3 tipos de usuarios, y básicamente 3 tipos de permisos. Sin embargo, aunque es un sistema muy simple, con una buena planificación permite desarrollar políticas de seguridad lo suficientemente buenas para el uso cotidiano de un sistema.

En Linux cada recurso pertenece a un usuario, y a un grupo de usuarios.

```
joancadi@ubixito:~/permisos$ ls -lia carta
848814 -rw-r-----  1 joancadi joancadi 109 2006-03-08 20:05 carta
```



Podemos cambiar estas pertenencias con la orden chown como ya hemos visto.

```
chown usuario.grupo recurso
```

La columna permisos, consta de 10 caracteres y se divide de la siguiente manera:

- 1º carácter: Nos indica que tipo de fichero es, los valores posibles para este carácter son:
  - d : directorio
  - l : enlace simbólico (ver sección)
  - b : dispositivo de bloque
  - c : dispositivo de caracteres
  - s : socket
  - p : tubería (pipe)
  - - : fichero regular
- 2º 3º 4º carácter. Es el primer trío de permisos, y nos indican los permisos que el usuario (U) tiene sobre ese fichero. Este usuario es el que aparece en la línea del ls.
- 5º 6º 7º carácter. Es el segundo trío de permisos, y nos indica los permisos que el grupo (G) tiene sobre ese fichero. Este grupo es el que aparece en la línea del ls.

8º 9º 10º carácter. Es el tercer trío de permisos, y nos indica los permisos que los otros (O) tienen sobre ese fichero. Otros se refiere a cualquier usuario que no sea el usuario del fichero (U) ni pertenezca al grupo del fichero (G).

TIPO	USUARIO (U)			GRUPO (G)			OTROS (O)		
-	r	w	x	r	w	x	r	w	x
	READ	WRITE	EXEC	READ	WRITE	EXEC	READ	WRITE	EXEC
	Leer	Escri	Ejec.	Leer	Escri	Ejec.	Leer	Escri	Ejec.

Para cada uno de estos tres tríos vemos que existen tres tipos de permisos fundamentales:

- **r**: read (lectura). El usuario que tenga este permiso podrá si es un directorio listar los recursos almacenados en él, y si es cualquier otro tipo de fichero podrá leer su contenido.
- **w**: write (escritura). Todo usuario que posea este permiso para un fichero podrá modificarlo. Si se posee para un directorio se podrán crear y borrar ficheros en su interior.
- **x**: execute (ejecución). Este permiso para el caso de los ficheros permitirá ejecutarlos desde la línea de comandos y para los directorios, el usuario que lo posea tendrá acceso para realizar el resto de las funciones permitidas mediante los otros permisos (lectura y/o escritura). Si un usuario no tiene permiso de ejecución en un directorio, directamente no podrá entrar en el mismo, ni pasar por el.

Los tres tipos de permisos mencionados poseen una representación numérica basada en el sistema octal que parte de representar como ``1'' los bits de los permisos otorgados y ``0'' para los negados. Luego se transforma la representación binaria así obtenida en octal. De esta forma se obtienen para cada tipo de permiso los siguientes valores:

Permiso	R	W	X
Valor binario	4	2	1

La combinación de los tres tipos de permisos para un tipo de usuario oscila desde cero (ningún permiso) hasta siete (todos los permisos).

Ejemplos:

```
rw-   = 110 (4 + 2 + 0) (6 en octal)
rwx   = 111 (4 + 2 + 1) (7 en octal)
r-x   = 101 (4 + 0 + 1) (5 en octal)
r--   = 100 (4 + 0 + 0) (4 en octal)
```

Los permisos ``totales'' de un recurso constan de nueve indicadores, donde los tres primeros indican los permisos asociados al dueño, los otros tres, al grupo y los últimos, al resto de los usuarios.

Ejemplos:

```
rwx  r-x  ---     = 111 101 000 (750 en octal)
rw-   r--  r--    = 110 100 100 (644 en octal)
```

Sólo el dueño de un recurso siempre tendrá derecho a cambiar sus permisos, además del root, por supuesto.

Existen otros tipos de permisos más complejos:

- s y S : es un permiso que de no administrarse correctamente puede provocar problemas de seguridad. Para su representación a través de caracteres se utiliza el lugar del permiso de ejecución y de ahí la diferencia entre s y S: si es s (minúscula) significa que incluye además el permiso de ejecución a diferencia de S (mayúscula). Este permiso se puede asociar al dueño o al grupo del recurso. Si se asocia a un fichero significa que cuando este se ejecute por un usuario que tenga permisos para ello adquirirá los permisos de su dueño o grupo en depencia de a cual de los dos está asociado el permiso. Un ejemplo de fichero con este permiso es el comando passwd, el cual adquiere los permisos de root al ser ejecutado por los usuarios (sin argumentos) para poder modificar el fichero /etc/shadow que es donde se guardan las contraseñas de los usuarios. Para el caso de un directorio este permiso sólo tiene validez para el grupo del mismo permitiendo a los ficheros y a los subdirectorios que se creen en él heredar el grupo, los subdirectorios heredarán también el permiso s. Un ejemplo de directorio con este permiso es aquel donde se guardan los documentos de un sitio FTP anómimo. Este permiso se conoce como setuid bit o setgid bit, para el usuario y el grupo, respectivamente.

- t y T : cuando está asociado a un directorio junto al permiso de escritura para un grupo de usuarios, indica que estos usuarios pueden escribir nuevos ficheros en el directorio pero estos sólo podrán ser borrados por sus dueños o por root. Para un fichero el permiso expresa que el texto de este se almacena en memoria swap para ser accedido con mayor rapidez. Este permiso sólo se asocia al resto de los usuarios y para su representación se emplea el bit correspondiente al permiso de ejecución: si es t (minúscula) significa que incluye además el permiso de ejecución y T (mayúscula) no lo incluye . Ejemplo de un directorio con este permiso es /tmp donde todos los usuarios pueden escribir pero sólo los dueños pueden borrar sus ficheros, además de root. Este permiso se conoce también como sticky bit.

Para representar los permisos t y s en el sistema binario se utilizan tres bits adicionales: el primero para s en el dueño, el segundo para s en el grupo y el tercero para t. Estos se colocan al inicio de la cadena numérica de nueve bits vista anteriormente. En la cadena de caracteres se mezclan con el permiso de ejecución y de ahí la necesidad de emplear las mayúsculas y minúsculas.

Ejemplos:

```
rws rwS r--       = 110 111 110 100 (6764 en octal)
rwx rws -wT       = 011 111 111 010 (3772 en octal)
```

Vemos como el primer trío de bits, nos indica si deseamos activar los permisos s para el usuario, los permisos s para el grupo, y los permisos t para el resto. Los otros tres tríos de bits son los permisos tal como los hemos visto anteriormente.

1.2 Gestionar los permisos chmod.

Para cambiar los permisos de un recurso se utiliza el comando chmod.

Sintaxis: chmod [opciones] <permisos> <ficheros>

Las formas de expresar los nuevos permisos son diversas, se puede emplear la representación numérica o utilizando caracteres.

Utilizando caracteres, la orden chmod se usa asi:

Chmod (letra del trío a cambiar) (+ - o =) (permisos) fichero

Donde:

- Letra de trío a cambiar puede ser:
  - U - Usuario (1º trío)
  - G - Grupo (2º trío)
  - O - Otros (3º trío)
  - A - All, todos los tríos.
- El carácter puede ser:
  - + Otorga el permiso
  - - Quita el permiso
  - = deja el permiso exactamente igual (es decir, quitará todos los demas).
- Permisos puede ser cualquier combinación de:
  - r para leer
  - w para escribir
  - x para ejecutar

Utilizando la representación numérica, simplemente hay que poner chmod (representación numérica) fichero

Ejemplo:

$ chmod u+x clase.txt	**# añade el permiso de ejecución al dueño**
$ chmod g=rx program.sh	**# asigna exactamente los permisos de lectura y ejecución al grupo**
$ chmod go-w profile	**# elimina el permiso de escritura en el grupo y en otros**
$ chmod a+r,o-x *.ts	**# adiciona el permiso de lectura para todos los usuarios y elimina el de ejecución para otros**
$ chmod +t tmp/	**# adiciona el permiso especial t**
$ chmod 755 /home/pepe/dc/	**# asigna los permisos con representación octal 755 (rwx r-x r-x)**
$ chmod -R o+r apps/	**# adiciona el permiso de lectura a otros para un directorio de forma recursiva (incluyendo todo su contenido)**
$ chmod +x ./bin/*	**# adiciona el permiso de ejecución a todos los usuarios que les corresponde por defecto**
# chmod 4511 /usr/bin/passwd	**# asigna los permisos con representación octal 4511 (r-s--x--x)**

Para determinar cuales permisos se asocian por defecto a los ficheros o directorios creados, cada usuario posee una máscara de permisos. Esta se expresa en el formato numérico octal o sea, posee tres dígitos entre cero y siete (Ej. 166). La máscara indica que permisos no se desea que tenga el recurso creado. Por defecto esta máscara es 002 para los usuarios comunes y 022 para root. La máscara realmente se asocia al shell y se hereda por los subshells.

Para calcular los permisos finales dado que se tiene la máscara 022, se hace la siguiente operación por parte del sistema:

Ficheros = totales_para_ficheros – máscara = 666 - 022 = 644 = -rw-r--r--

Directorios = totales_para_directorios - máscara = 777 - 022 = 755 = drwxr-xr-x

Para colocar u observar la máscara se puede emplear el comando umask. Sintaxis: umask [-S] [máscara]

Ejemplos:

```
$ umask  # sin argumentos muestra la máscara actual en formato numérico
$ umask -S # muestra el complemento de la máscara en formato de caracteres u
```

```
$ umask 037  # asigna la máscara 037 (niega permisos de ejecución y de escritura para el grupo, y todos los permisos para el resto de los usuarios).
```

```
$ umask g=rx,o= # especifica el complemento la máscara utilizando el formato de caracteres
```

Para ser bien restrictivos se recomienda hacer: **$ umask 077**

Los nuevos directorios tendrán el permiso: 700 = drwx------

Los nuevos ficheros tendrán el permiso: 600 = -rw-------

Resumiendo, si queremos que a partir de este momento los nuevos ficheros que creemos, tengan como permiso por ejemplo rw- r-- ---, calculamos la representación numérica de estos permisos, que en este caso son 640. Ahora restamos a 666 el número obtenido, y obtenemos 026, pues este es el número que debemos usar en umask para conseguir lo que queremos. #umask 026

Si en lugar de pensar en ficheros, pensamos en directorios, habría que restar no 666, sino 777. (Se recomienda no trabajar en números impares con umask, es decir, obviamos trabajar con el bit del permiso de ejecución).

2 Gestión de procesos en Linux.

1.1 Introducción ps

Un proceso es una instancia de un programa en ejecución. En Linux se ejecutan muchos procesos de forma concurrente aunque realmente sólo uno accede al procesador en un instante de tiempo

determinado. Esto en esencia es lo que caracteriza a los sistemas multitarea como ya vimos en anteriores apuntes.

Cada proceso en el momento de su creación se le asocia un número único que lo identifica del resto. Además a un proceso están asociadas otras informaciones tales como:

- El usuario que lo ejecuta.
- La hora en que comenzó.
- La línea de comandos asociada.
- Un estado. Ejemplos: sleep, running, zombie, stopped, etc.
- Una prioridad que indica la facilidad del proceso para acceder a la CPU. Oscila entre -20 y 19, donde -20 es la mayor prioridad.
- La terminal donde fue invocado, para el caso de que este asociado a alguna terminal.

Para ver los procesos y sus características se emplea el comando **ps**. Una salida típica de este comando es:

```
# ps
PID   TTY          TIME         CMD
1035  pts/0        00:00:14     bash
1831  pts/0        00:00:00     ps
```

Como puede apreciarse para cada proceso se muestra su ID (identificación o numero), la terminal donde se invocó, el tiempo de CPU que se le ha asignado hasta el momento y el comando que lo desencadenó. Por defecto ps muestra en formato reducido los procesos propios del usuario y la terminal actual.

Algunas opciones de ps.

- x : muestra todos los procesos del usuario actual sin distinción de terminal.
- a : muestra todos los procesos de todos los usuarios.
- f : muestra las relaciones jerárquicas entre los procesos.
- e : muestra el entorno de cada proceso.
- l : utiliza un formato más largo (muestra más información).
- u : utiliza un formato orientado a usuario.

Ejemplos:

```
$ ps aux
$ ps e
$ ps xf
```

Otro comando para ver el estado de los procesos en ejecución es top, que permite hacerlo dinámicamente. top es más bien un programa interactivo con el cual se pueden observar los procesos más consumidores de CPU por defecto. Este comportamiento se puede modificar tecleando:

- M : ordenará según el empleo de la memoria.
- P : ordenará según el empleo de la CPU.
- N :ordenará por ID.
- A :ordenará por antigüedad.

Para observar todos los posibles comandos durante la interacción con top se puede pulsar h. Para salir se presiona q.

El programa top muestra además algunas estadísticas generales acerca del sistema:

- La hora actual y en la que se inició el sistema.
- La cantidad de usuarios conectados.
- Los promedios de carga de la CPU en los últimos uno, cinco y quince minutos transcurridos.
- Un resumen estadístico de la cantidad de procesos en ejecución y su estado (sleeping, running, zombie y stopped).
- Un resumen del empleo de la memoria física y virtual (swap).

Los tres primeros aspectos se pueden ver también con el comando uptime y el último con free.

1.2 Control de los procesos en bash & fg bg

El shell bash permite ejecutar los procesos en foreground (primer plano) o background (segundo plano). Los primeros son únicos por terminal (no puede haber más de un proceso en primer plano en cada terminal) y es la forma en que se ejecuta un proceso por defecto. Solo se retorna al prompt una vez que el proceso termine, sea interrumpido o detenido. En cambio, en background pueden ejecutarse muchos procesos a la vez asociados a la misma terminal. Para indicar al shell que un proceso se ejecute en background, se utiliza un símbolo ampersand (&) al final de la línea.

Ejemplo:

```
# updatedb &
```

Para modificar el estado de un proceso en foreground desde bash existen dos combinaciones de teclas muy importantes que este interpreta:

- Ctrl-c : trata de interrumpir el proceso en foreground. Si es efectivo, el proceso finaliza su ejecución (es asesinado).
- Ctrl-z : trata de detener el proceso en foreground. Si es efectivo el proceso continúa activo aunque deja de acceder al procesador (esta detenido y pasa a background).

Para ver los procesos detenidos o en background en un shell se emplea el comando integrado a bash **jobs**, que mostrará una lista con todos los procesos en dichos estados mediante los comandos asociados y un identificador numérico especial.

Ejemplo:

```
# jobs
[1]   Running   sleep 10000 &
[2]-  Stopped   cp /var/log/messages /tmp
[3]+  Stopped   updatedb
```

Los procesos detenidos se pueden llevar al background y estos a su vez pueden trasladarse al foreground. Para ello se emplean respectivamente los comandos integrados al bash: **bg** y **fg**, pasándoles como argumento el identificador especial del proceso. Si no se especifica un argumento se asumirá el trabajo marcado con un signo ``+'' que sería el último detenido o llevado al background.

Ejemplos:

```
$ bg 2
[2]- cp /var/log/messages /tmp &

$ fg
cp /var/log/messages /tmp
```

Si se comenzará a ejecutar un proceso y este se demora mucho y no interesan por el momento sus resultados se puede detener y enviarlo al background haciendo Ctrl-z y luego, bg.

1.3 Comunicación con los procesos kill

Un comando muy útil para interactuar con los procesos es kill. Este permite enviarles señales con significados muy diversos. Los programas o comandos deben estar preparados para atrapar y tratar estas señales, al menos las más importantes. Existen muchos tipos de señales, para verlas se puede escribir en Linux kill -l.

```
joancadi@ubixito:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
```

……

Por defecto kill envía la señal TERM que indica al proceso que debe terminar (15). La señal 9 o KILL lo finaliza forzosamente (es como una orden TERM pero imperativa). La señal HUP es interpretada por muchos comandos y programas como una indicación de que relean los ficheros de configuración correspondientes (que reinicien su ejecución).

Así, al ejecutar kill, hay que indicar que señal se quiere mandar, y a que número de proceso se le quiere mandar dicha señal.

Ejemplos:

```
$ kill 1000             # envía la señal 15 (TERM) al proceso 1000
$ kill -s 9 10101       # envía la señal 9 (KILL) al proceso 10101
$ kill -4 18181         # envía la señal 4 (ILL) al proceso 18181
# kill -HUP 199         # envía la señal HUP al proceso 199
$ kill %2               # envía la señal 15 (TERM) al trabajo 2 (en background o detenido)
```

También existe **killall** que permite enviar señales a los procesos a través de sus nombres. A diferencia del ID, el nombre de un proceso no es único, o sea pueden existir muchos procesos con el mismo nombre y de ahí la utilidad de este comando.

Sintaxis: killall [opciones] [-señal] <nombre>

Algunas opciones:

- -i : forma interactiva. Pregunta para cada proceso si se desea enviar la señal o no.
- -v : reporta si tuvo exito el envío de la señal.

Ejemplo:

```
$ killall -9 gdm
```

1.4 Prioridades y ámbito de los procesos nice renice nohup

En Linux existe la posibilidad de iniciar los procesos con prioridades distintas a las asignadas por parte del sistema. Para ello se puede emplear el comando **nice**. Este al invocarse sin argumentos imprime la prioridad asignada por defecto a los procesos del usuario actual. La prioridad máxima de un proceso en Linux es -20 (negativo) y la prioridad mínima de un proceso es 19 (positivo). La prioridad por defecto de todos los comandos es de 10.

La otra forma de emplear a nice es indicando la nueva prioridad precedida del signo "-" y la línea de comando que desencadena el proceso. Si no se indicara la prioridad se incrementa en 10 la por defecto. Sólo el usuario root puede asignar a sus procesos prioridades con valores inferiores a cero.

Ejemplos:

```
# nice tar cvf /tmp/etc.tgz /etc   # incrementa en 10 la prioridad por defecto del comando
```

```
# nice - 10 updatedb                # ejecuta un comando con prioridad 10
# nice - -10 updatedb                # ejecuta un comando con prioridad -10
```

Si se deseara reajustar la prioridad de un proceso ya en ejecución se puede utilizar **renice**. A este se le indican como argumentos la nueva prioridad y el identificador numérico del proceso (pueden ser varios). En este caso el valor de la prioridad no va precedido por el signo ``-'' como es en nice. También se puede cambiar la prioridad de todos los procesos de uno o de varios usuarios a la vez.

Ejemplos:

```
# renice -19 1001       # ajusta la prioridad de un proceso a -19
# renice 1 602          # ajusta la prioridad de un proceso a 1
# renice 10 -u pepe     # ajusta a 10 la prioridad de todos los procesos del usuario pepe
# renice 5 -g ppp uucp  # ajusta a 5 la prioridad de todos los procesos de los usuarios miembros de los grupos ppp y uucp
```

Las prioridades de los procesos sólo se pueden disminuir, nunca aumentar con excepción de root que puede hacerlo indistintamente.

Los procesos de los usuarios, por defecto, se asocian a la terminal actual. Es en ella donde muestran su salida estándar si esta no ha sido redireccionada. Si la sesión en una terminal termina, los procesos activos asociados a esta recibirán la señal HUP. En caso de que no trataran dicha señal se les enviará la señal TERM y por último, KILL. Para evitar este tratamiento por parte del sistema se puede emplear el comando nohup que ejecuta un comando cuyo proceso no recibirá la señal HUP correspondiente, una vez terminada la sesión. Por defecto nohup reduce la prioridad del proceso en 5 y envía su salida a un fichero llamado nohup.out.

Ejemplo:

```
$ nohup gran_calculo &
$ logout
```

En este caso, ejecutamos un proceso gran_calculo en segundo plano (background). Este proceso es hijo de nuestro shell que hemos creado al hacer login, y si hacemos logout mataremos nuestro shell y a todos sus hijos, con lo que mataríamos gran_calculo. Pero al haber lanzado gran_calculo con nohup, este no recibirá las señales hup term y kill del sistema, y seguirá ejecutándose aunque nos salgamos del sistema.

Esto es especialmente interesante si nos conectamos a un servidor mediante telnet o ssh. Una vez que hemos abierto sesión en el ordenador remoto, podemos ejecutar cualquier comando, pero en cuanto que cerremos la sesión, dicho comando morirá. Si ejecutamos el comando con nohup y & dicho comando seguirá corriendo en el sistema aunque cerremos la sesión.

3 Dispositivos de almacenamiento. Montaje.

En Linux los dispositivos físicos de la máquina en general y los de almacenamiento de información, en particular, son manipulados a través de ficheros especiales ubicados en el directorio /dev. Los discos duros, las particiones de estos, las unidades de disquete, los CD-ROM o los dispositivos de almacenamiento USB son ejemplos de estos con los cuales interactuamos constantemente. Pero trabajar directamente sobre los dispositivos representados de esa forma casi nunca es conveniente ni resulta cómodo, por lo que usualmente se incorporan al sistema de ficheros tradicional.

Esta acción se conoce como ``montar'', que en definitiva es asociar el dispositivo a un directorio determinado.

Como se explicó anteriormente las particiones de los discos en Linux se montan en directorios como /, /home y /usr. El sistema tiene un fichero llamado /etc/fstab en el cual se especifican donde y en que forma se montan los diferentes dispositivos.

joancadi@ubixito:~$ cat /etc/fstab

 # /etc/fstab: static file system information.

# <file system>	<mount point>	<type>	<options>	<dump>	<pass>
proc	/proc	proc	defaults	0	0
/dev/hd3	/	ext3	defaults,errors=remount-ro	0	0
/dev/hda7	none	swap	sw	0	0
/dev/hdc	/media/cdrom0	udf,iso9660	user,noauto	0	0

Cada línea en este fichero describe un dispositivo, indicando los siguientes aspectos para cada uno:

- Nombre del dispositivo o etiqueta. Ejemplos: /dev/hda1, /dev/sdc1, /dev/fd0, LABEL=/home, LABEL=/cursos, etc.
- Directorio donde se monta. Ejemplos: /, /mnt/floppy, /tmp, etc.
- Sistema de ficheros. Ejemplos: ext2, msdos, nfs, swap, iso9660, auto, etc.
- Opciones de montaje. Ejemplos: ro, rw, exec, auto, user, etc.
- Dos valores numéricos: el primero toma los valores 0 ó 1 indicando si al dispositivo se le hará dump (especie de backup) o no. El segundo número expresa la prioridad que tiene el dispositivo cuando se chequea la integridad del file system durante el inicio del sistema.

Las opciones de montaje son numerosas. Las más usadas se listan a continuación:

- auto : indica que el dispositivo se monta siempre que se inicie el sistema. La opuesta es noauto.
- rw: indica que el dispositivo se monta con permisos de lectura y escritura.
- ro: indica que el dispositivo se monta con permisos de lectura solamente.
- owner: indica que el usuario conectado al sistema localmente en primer lugar tiene derechos a montar y desmontar el dispositivo (se adueña de este).

- user : indica que cualquier usuario puede montar y solo el mismo usuario podrá desmontar el dispositivo. La opción opuesta es nouser.
- users : indica que cualquier usuario puede montar y cualquiera también, puede desmontar el dispositivo.
- suid : indica que el permiso ``s'' tenga efecto para los ejecutables presentes en el dispositivo. La opción opuesta es nosuid. (Todos los ejecutables del sistema se ejecutan como si fueran invocados por el root)
- exec : indica que los binarios ejecutables almacenados en el dispositivo se pueden ejecutar. La opción opuesta es noexec.
- async : expresa que todas las operaciones de entrada y salida se hacen de forma asíncrona, o sea, no necesariamente en el momento en que se invocan. La opción opuesta es sync.
- dev : indica que se interprete como tal a los dispositivos especiales de bloques y de caracteres presentes en el dispositivo. La opción opuesta es nodev.
- defaults : es una opción equivalente a la unión de rw, suid, dev, exec, auto, nouser y async.

Actualmente para cada dispositivo con sistema de ficheros ext2 en lugar de especificar su nombre en el fichero fstab se puede indicar una etiqueta o identificador asociado. La forma utilizada es LABEL=<etiqueta> o UUID=<uuid>. Esta posibilidad hace más robusta la configuración ante la realización de cambios en los discos duros ya sea porque se incluyan nuevos o se reordenen los existentes. Para ver o cambiar la etiqueta de un dispositivo se puede emplear el comando e2label.

Para montar y desmontar los dispositivos se emplean los comandos mount y umount respectivamente. Estos mantienen una lista de los dispositivos montados en el fichero /etc/mtab.

Sintaxis:

mount [opciones] [dispositivo] [dir]

umount [opciones] <dir>|<dispositivo>

Algunas opciones:

- -a : en el caso de mount monta todos los dispositivos que tienen la opción auto en el fichero fstab, y para umount desmonta todo lo que está en el fichero **/etc/mtab**.
- -t <tipo> : indica el tipo de file system a montar.
- -o <opciones> : especifica las opciones de montaje (separadas por comas).

Cuando se especifican en el fichero fstab las características del montaje de un dispositivo, para montarlo no es necesario indicarlo todo, basta con poner el nombre del dispositivo o el directorio donde se monta por defecto.

Ejemplos:

```
$ mount -a -t ext2                  # monta todos los dispositivos con file
system ext2 y con la opción auto en el fichero /etc/fstab

$ mount /dev/fd0 /floppy            # monta el disquete en el directorio
/floppy

$ mount /media/cdrom0               # monta el cdrom. Toma las
especificaciones del fichero /etc/fstab
```

```
$ mount /dev/hdc              # hace lo mismo que el anterior
$ umount -a -t ntfs           # desmonta todo los dispositivos con file
system ntfs especificados en /etc/mtab
$ umount /dev/fd0 /mnt/cdrom        # desmonta el cdrom y el disquete
```

Siempre que un dispositivo esté siendo utilizado por el sistema no se podrá desmontar. Este emitirá un mensaje de error como en el siguiente ejemplo:

```
$ umount /mnt/floppy
umount: /mnt/floppy: device is busy
```

Un dispositivo puede estar ocupado por el simple hecho de tener posicionado un shell en el directorio donde se montó, haber lanzado un ejecutable al background desde ese directorio, o haber montado otro dispositivo en un subdirectorio del mismo.

Para lograr el objetivo será necesario eliminar todos estos casos.

Siempre que se trabaje con los disquetes y llaveros USB en esta forma, sobre todo cuando se realizan operaciones de escritura no se debe olvidar desmontarlo antes de extraerlo del ordenador. El resultado puede ser que la información almacenada quede inconsistente. En el caso del CD-ROM esto no es posible pues su funcionamiento electrónico permite que el sistema pueda controlar que mientras no se desmonte el dispositivo, el usuario no pueda extraer el disco.

Algunos comandos útiles para el manejo de discos son:

fdformat : permite formatear un disquete a bajo nivel (sin ponerle un file system determinado).

Ejemplo: $ fdformat /dev/fd0

dd : permite duplicar ficheros o partes de estos ya sean regulares o especiales (hacer imágenes).

Sintaxis: dd [opciones]

Las opciones son en forma de pares <llave>=<valor>.

Algunas opciones:

if=<fichero> : especifica el nombre del fichero de entrada o origen.

of=<fichero> : indica el nombre del fichero de salida o destino.

bs=<n> : especifica la cantidad de bytes leídos y copiados a la vez o tamaño de bloque. Por defecto es 512.

count=<n> : indica la cantidad de bloques a copiar del origen al destino. Por defecto se copian todos los bloques presentes en el origen.

Ejemplos:

```
# dd if=/kernel-image of=/dev/fd0
# dd if=/dev/hda1 of=/mnt/floppy/boot_sector count=1 bs=512
# dd if=/dev/cdrom of=CDimage.iso
```

- eject : desmonta, si es necesario, y luego expulsa un dispositivo a nivel de software.

Opcionalmente recibe como argumento el nombre del dispositivo o el directorio donde se montó, asumiendo el CD-ROM por defecto. La opción -t introduce el dispositivo en lugar de expulsarlo.

Ejemplos:

```
$ eject
$ eject -t /dev/burner
```

- mkfs : se utiliza para crear un sistema de ficheros en un dispositivo.

Por defecto el file system creado es del tipo ext2. Sirve de interfaz para otros comandos más específicos como mkfs.msdos, mkfs.reiserfs mkfs.minix, mkfs.ext2, mkreiserfs, mkdosfs y mke2fs.

Ejemplos:

```
# mkfs /dev/hda10
# mkdosfs /dev/fd0
```

- fsck : chequea y repara un sistema de ficheros.

Sirve de interfaz para otros comandos más específicos como fsck.msdos, fsck.reiserfs, fsck.minix, fsck.ext2, e2fsck, dosfsck y reiserfsck.

Ejemplos:

```
# fsck /dev/hdc5
# dosfsck /dev/fd0
```

- mkbootdisk : se utiliza para crear un disco de carga dado el kernel que se desea cargar.

Ejemplo: $ mkbootdisk 2.4.2-2

(Para comprobar que versiones del kernel tenemos disponibles en el sistema, sacad un directorio de /lib/modules).

- Comandos tipo MS-DOS (mtools) : se utilizan para obtener compatibilidad con Ms-Dos.

Son un conjunto de herramientas que permiten el acceso a disquetes o particiones con formato msdos sin necesidad de montarlos o desmontarlos explícitamente. Ejemplos de estos comandos son:

mdir : lista el contenido de un disquete. Ej. $ mdir a:

mcopy : copia ficheros hacia el disquete. Ej. $ mcopy doc/tesis.txt a:

mdel : borra ficheros del disquete. Ej. $ mdel a:tesis.txt

mformat : formatea el disquete. Ej. $ mformat a:

Otros comandos son : mattrib, mbadblocks, mdeltree, mdu, minfo, mkmanifest, mlabel, mmount, mmove, mread, mren, mtoolstest y mtype.