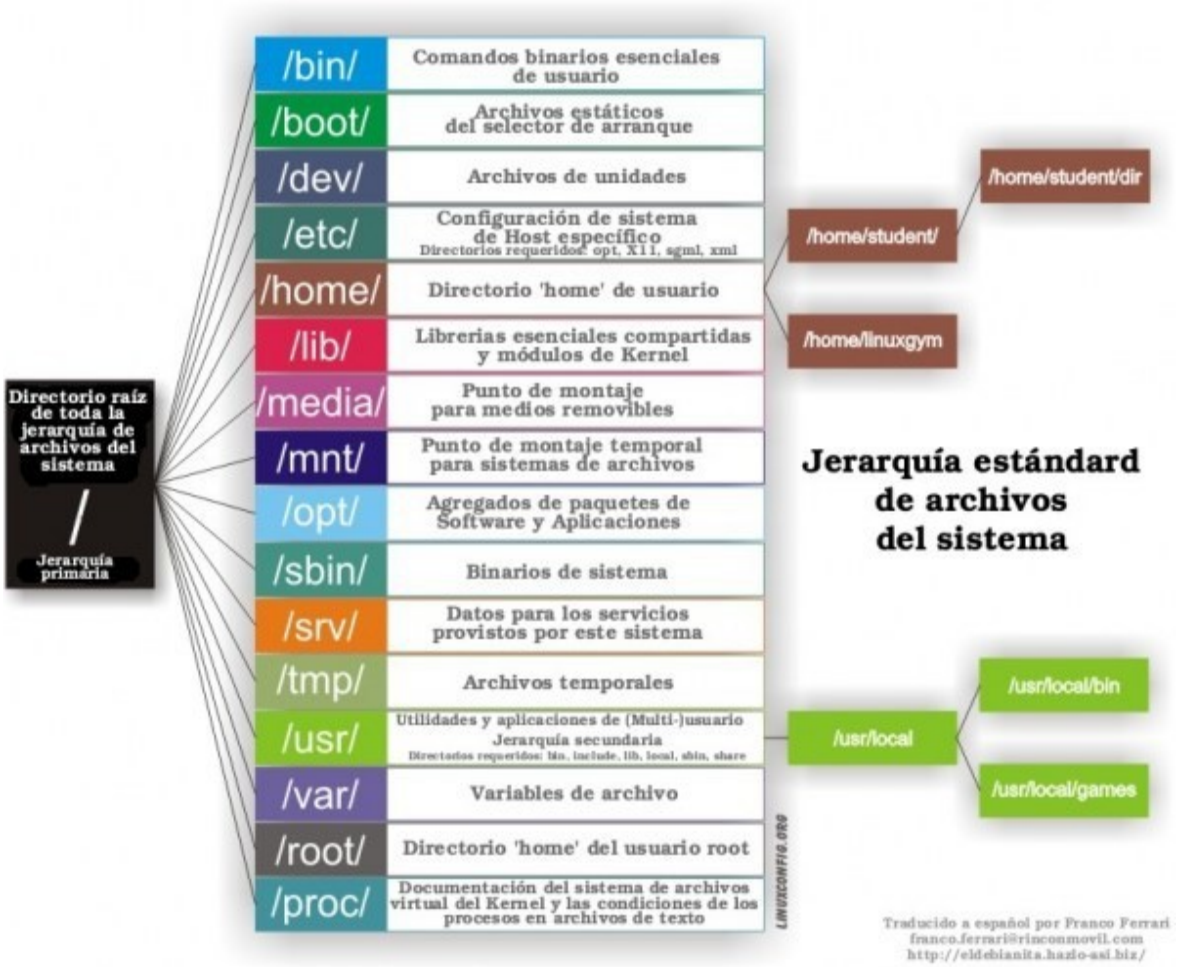
Tema 10: Administración GNU/LINUX

ESTRUCTURA DE DIRECTORIOS DE LINUX

Es **“muy importante”** conocer la estructura de directorios en Linux, ya que ello nos permite saber que tipo de información contiene.

Esto nos ayudará para tener una mejor visión de **cómo está organizado el sistema operativo GNU/Linux**, además con ello sabremos **dónde localizar lo que necesitamos**.

En esta imagen podemos ver una breve descripción de la estructura de directorios de linux:



/: (Raíz) Es el nivel más alto dentro de la jerarquía de directorios (PRINCIPAL) . De aquí parten los demás directorios y subdirectorios.

/bin: (Binarios) Contiene los binarios básicos, que son los ejecutables del sistema operativo.

/boot: (Arranque) Aquí se encuentran todos aquellos archivos necesarios para que el sistema inicie.

/dev: (Dispositivos) En esta carpeta se encuentran todos los archivos que nos permiten interactuar con los dispositivos hardware de nuestro ordenador. Por ejemplo los usb, sda (o hda) con la información de cada uno de ellos.

/etc: (Etcétera) Aquí se guardan los ficheros de configuración de los programas instalados.

/home: (Hogar) Contiene las carpetas por defecto de los usuarios, (sería algo así como el "Documents and Settings" del sistema operativo Windows).

/lib: (Bibliotecas) Contiene las librerías del sistema y los drivers.

/lost+found: (Perdidos y encontrados) información que se guardó de manera incorrecta debido a algún fallo del sistema, se crea en cada partición independientemente.

/media: (Media/medios) Directorio donde puede ser utilizado como punto de montaje para las Unidades Extraíbles. Por ejemplo, los dispositivos USB, disqueteras, unidades de CD/DVD.

/mnt: (Montaje) Es un directorio que se suele usar para montajes temporales de unidades. Por ejemplo, Directorios compartidos dentro de una red, alguna partición de Windows, etc.

/opt: (Opcionales) Destinado para guardar/instalar paquetes adicionales de aplicaciones.

/proc: (Procesos) Información de los datos del kernel del sistema de ficheros de Linux, Virtualización y procesos en ejecución.

/root: (Casa del Administrador) Es el directorio /home del administrador. Es el único **/home** que no está incluido -por defecto- en el directorio HOME.

/sbin (Binarios del Sistema): Son los ejecutables de administración, tales como mount, umount, etc.

/srv: (Servicios) En este directorio residen las carpetas accesibles por el programa cliente de un determinado servicio ofrecido por algunos servidores configurados en el sistema. Por ejemplo Apache, ProFtpd, etc.

/sys: (Sistema) Información sobre los dispositivos tal y como los ve el kernel Linux.

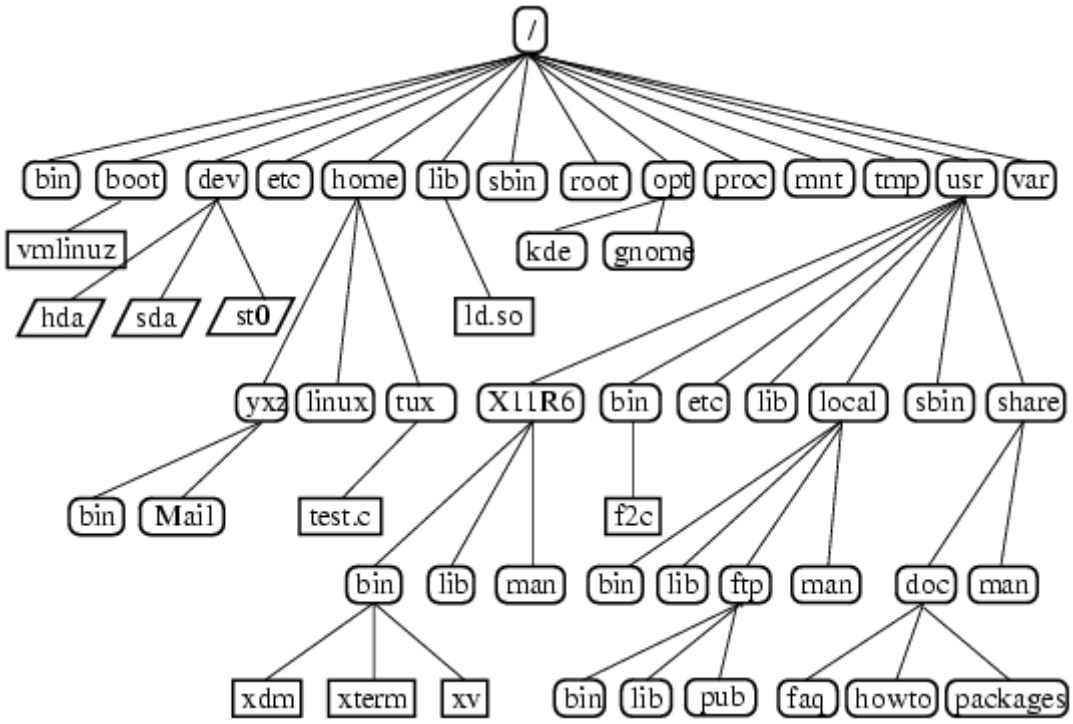
/tmp: (Temporales) Es un directorio donde se almacenan ficheros temporales. Cada vez que se inicia el sistema este directorio se borran.

/usr: (Compartido de Usuarios) Es el directorio padre de otros subdirectorios de importancia. Se encuentran la mayoría de los archivos del sistema, aplicaciones, librerías, manuales, juegos… Es un espacio compartido por todos los usuarios del sistema.

/var: (Variables) Ficheros y datos que cambian frecuentemente (logs, bases de datos, colas de impresión…)

Hasta aquí hemos visto una descripción de los directorios principales, lo cual no quiere decir que dentro no haya subdirectorios.

Observemos algo un poco más completo:



Dentro de la categorización de los directorios encontramos:

- **Estáticos:** Contiene archivos binarios, bibliotecas, y otros archivos están en Read Only (Solo lectura) que no cambian sin la intervención del administrador. **/bin, /sbin, /boot, /opt**.

- **Dinámicos:** Son aquellos que los archivos dentro de estos van cambiando. Generalmente se encuentra como Read-Write (Lectura-Escritura). **/var/spool, /var/lock, /var/mail, /home.**

- **Compatible**s: Se pueden encontrar archivos comunes que van a estar en cualquier distribución. **/usr/bin, /opt**

- **No Compatibles:** Contiene archivos que no son compatibles con otras distribuciones. **/etc, /boot, /var/run, /var/lock.**

FLUJOS DE DATOS

En Linux al igual que en Unix todos los procesos (programas en ejecución) tienen asociados tres flujos (streams) de datos principales. Estos son:

- La entrada estándar. (stdin) Es donde un proceso puede tomar los datos que maneja y que no se indican mediante argumentos u opciones. Por defecto se toma a partir del teclado.
- La salida estándar. (stdout) Es donde un proceso escribe los resultados de su ejecución. Por defecto es la terminal (pantalla) donde se invocó el programa correspondiente.
- La salida de errores. (stderr) Es donde un proceso escribe los posibles errores durante su ejecución. Por defecto es la terminal (pantalla) donde se invocó el programa correspondiente.

Los flujos de datos se almacenan en descriptores de ficheros que se identifican por un número en la forma siguiente:

- 0: representa la entrada estándar.
- 1: representa la salida estándar.
- 2: representa la salida de errores.

En bash, al igual que en otros shells, los flujos de datos se pueden redireccionar libremente hacia distintos ficheros. En esencia este mecanismo consiste en que la salida de un proceso (estándar o de errores) puede escribirse en un fichero en lugar de la terminal asociada, así como la entrada puede tomarse también a partir de un fichero en lugar de utilizar lo escrito mediante el teclado.

Para indicar un redireccionamiento se utilizan los signos de comparación < y >. De esta forma se generan dos tipos de construcción:

- instrucción > salida: indica el redireccionamiento del flujo de datos de la instrucción al fichero nombrado salida.
- Instrucción < entrada: indica el redireccionamiento del contenido del fichero nombrado entrada como flujo de datos de entrada para la instrucción.
- Instrucción 2> salida: indica el redireccionamiento del flujo de datos DE LOS ERRORES de la instrucción al fichero nombrado salida.

Ejemplos:

# cat /etc/shadow > cont	Se redirecciona la salida estándar hacia un fichero con nombre cont
$ write pepe < msj.txt	Toma la entrada en lugar de desde el teclado, desde el fichero msj.txt
$ cat > telefonos	Toma de la entrada estándar (teclado) y lo envía a la salida que será el fichero teléfonos. (Terminar la entrada con Ctrl D).
$ ls > /dev/null	Se redirecciona la salida estándar al dispositivo

	especial de caracteres /dev/null provocando su desaparición.
$ ls /tmp 2> fich-errores	Se redirecciona la salida de errores hacia un fichero fich-errores.
$ find / -iname “carta” > sa 2> err	Ejecuta el comando find, mandando la salida normal de la orden al fichero sa y mandando la salida de errores al fichero err.

Siempre que se emplee la forma [x]>salida, si el fichero salida existe se sobrescribirá y si no, se creará. Para añadirle algo más a su contenido anterior (append) en lugar de sobrescribirlo, se puede emplear >>.

$echo “Hola” >> fichero	Se redirecciona el texto Hola hacia fichero. Si fichero ya existe se añadirá el texto al final, si no existe, se creará fichero y luego se le añadirá el texto.

TUBERÍAS

Las tuberías (pipes) son un poderoso mecanismo del shell en Unix. Este en esencia permite tomar la salida de un comando y pasársela como entrada a otro.

Muchos de los comandos mencionados anteriormente, que reciben como argumento un fichero, en caso de omitirse este, utilizan su entrada estándar. Esta entrada puede provenir a su vez de la salida de otros comandos. Gracias a esto las tuberías permiten realizar filtrados de los más diversos tipos

Las tuberías pueden estar formadas por un número ``ilimitado'' de comandos. Estos no se ejecutan secuencialmente, o sea no se espera a que termine uno para ejecutar el siguiente, sino que se va haciendo de forma concurrente. El carácter que se emplea para separar un comando de otro mediante una tubería es |. (Alt Gr + 1 ó Alt + 124)

Ejemplos:

$ ls -l /dev | less	Toma la salida de ls y la envia como entrada a la orden less, con lo que se consigue un listado paginado.
$ ls -l /dev | grep ma	Toma la salida de la orden ls –l /dev y la filtra, realizando sobre ella una búsqueda de la cadena ma, de modo que solo aparecerán en la pantalla los ficheros que contengan dicha cadena.
$ ls -R /etc | sort	Lista recursivamente el contenido del directorio /etc y ordena la salida.
$ ls -R /etc | sort | more	Lista recursivamente el contenido del directorio /etc y ordena la salida, paginándola.
$ ls -R /etc | sort | more | uniq	Lista recursivamente el contenido del directorio /etc y ordena la salida paginándola y

	mostrando únicamente las líneas únicas. (Sin líneas duplicadas).
$ locate user | grep bin	Busca en la jerarquía de ficheros a aquellos que contengan en su nombre la cadena "user" y de estos selecciona los que contengan además "bin"
$ grep /bin/bash /etc/passwd | wc -l	Cuenta los usuarios que tienen como shell el bash

PERMISOS

Cada uno de los elementos del sistema de ficheros de Linux posee permisos de acceso de acuerdo a tres tipos de usuarios:

♦ U. Su dueño (casi siempre el creador) representado por la letra u (**USUARIO** o USER).
♦ G. Su grupo representado por la letra g (**GRUPO** o GROUP).
♦ O. El resto de los usuarios que no son el dueño ni pertenecen al grupo. Se representa con o (**OTROS** u OTHER).

Nota: Para representar a todos los tipos de usuarios se utiliza la letra a (all).

Para cada uno de estos tres grupos de usuarios 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).

Para poder realizar operaciones sobre cualquier **directorio** (leer o escribir) será necesario siempre, tener otorgado además el permiso de ejecución. Para acceder a un recurso de cualquier forma (ejecución, lectura o escritura) se deben tener permisos de ejecución para todos los directorios que contienen al recurso directa e indirectamente.

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. Los permisos siempre van formando tríos, de la forma rwx.

De esta forma se obtienen para cada tipo de permiso los siguientes valores:

Permiso	r	w	x
Valor	4	2	1

r = 100 (4 en octal) (r--)
w = 010 (2 en octal) (-w-)
x = 001 (1 en octal) (--x)

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

Ejemplos:

r	w	x	Binario	Decimal	Permisos
r	w	-	110	6	Lectura y escritura, no ejecución.
r	-	x	101	5	Lectura y ejecución.
r	-	-	100	4	Solo lectura.
-	-	-	000	0	Ningún permiso.

Los permisos ``totales'' de un recurso constan de nueve indicadores, donde los tres primeros indican los permisos asociados al **usuario** dueño, los otros tres al **grupo** y los últimos 3 a los **otros**, al resto de los usuarios.

Permisos (ugo)			Valor octal	Permisos al usuario	Permisos al grupo	Permisos a otros
rw-	rw-	rw-	6 6 6	Lectura y escritura	Lectura y escritura	Lectura y escritura
rwx	rwx	---	7 7 0	Todos	Todos	Ninguno
rw-	r--	r--	6 4 4	Lectura y escritura	Lectura	Lectura
rwx	r-x	---	7 5 0	Todos	Lectura y ejecución	Ninguno
r--	---	---	4 0 0	Lectura	Ninguno	Ninguno

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

Aparte de r w x 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 (x y s) a diferencia de S que incluye solo el permiso (s) y no el x. 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 según a que esté 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 anónimo. 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.

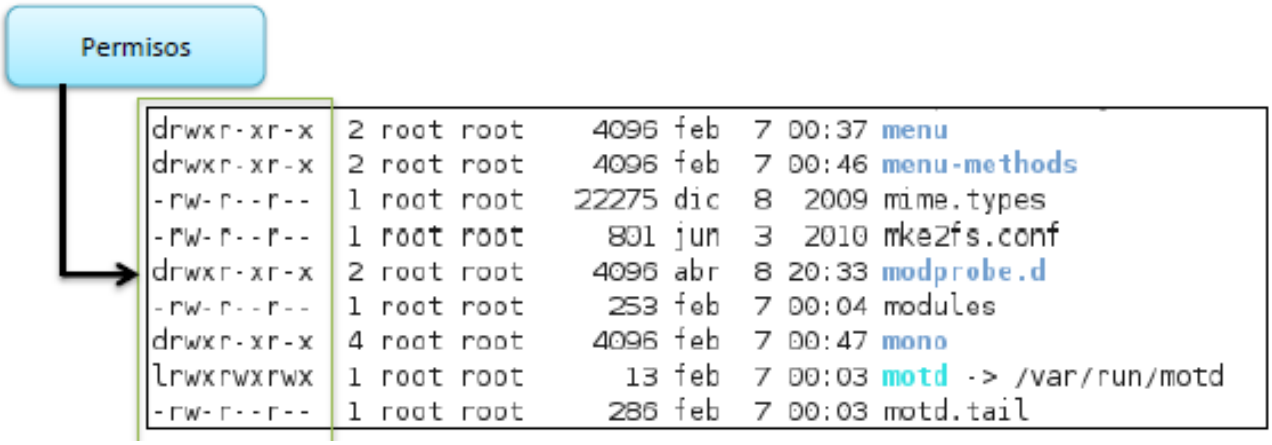
Para representar los permisos t y s en el sistema se utilizan tres bits adicionales: el primero para s en el dueño, el segundo para s en el grupo y el tercero para t. Estos se colocan al inicio de la cadena numérica de nueve bits vista anteriormente. En la cadena de caracteres se mezclan con el permiso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

Estos permisos s y t son una causa frecuente de problemas, y son inseguros de por si, por lo que es muy recomendable no usarlos a menos que sea estrictamente necesario. De hecho la mayoría de las distribuciones actuales ya no utilizan estos permisos.
Después de toda esta introducción a los permisos en Linux veamos cómo estos se muestran, modifican y se les asigna un valor por defecto.

Posiblemente el comando más empleado en Linux es aquel que muestra el contenido de un directorio, llamado ls. Este comando con la opción -l permite observar los permisos que tienen asociados los recursos listados, además de otras características. Los permisos se muestran a través de una cadena de 10 caracteres:



1) el primer carácter indica el tipo de recurso, y puede ser:

- d : directorio
- l : enlace
- b : dispositivo de bloque
- c : dispositivo de caracteres
- s : socket

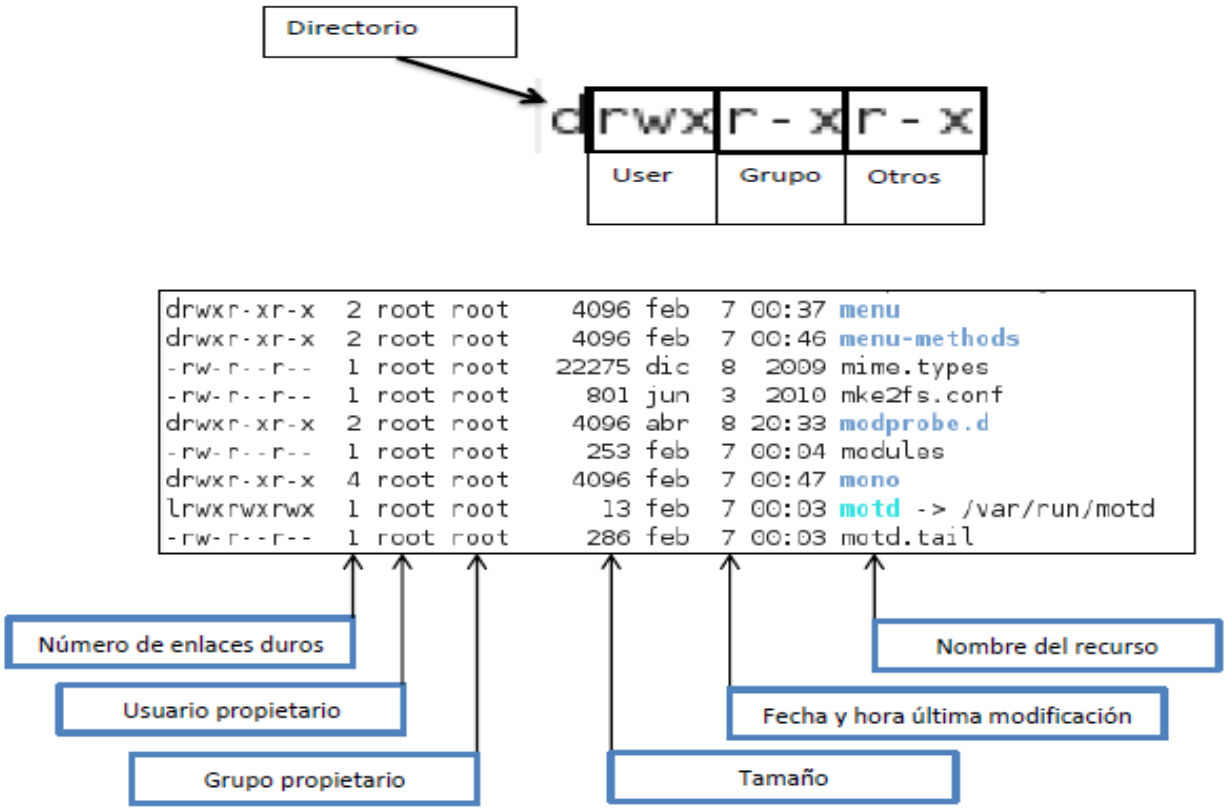
♦ p : tubería (pipe)

♦ - : fichero regular

2) Los caracteres 2,3 y 4 indican los permisos para el usuario dueño del recurso.

3) Los caracteres 5,6 y 7 indican los permisos para el grupo dueño del recurso.

4) Los caracteres 8,9 y 10 indican los permisos para el resto de usuarios, es decir, los usuarios que no son



El resto de las columnas de la salida representan para cada elemento:

- El número de enlaces duros que posee. En el caso de un directorio, indica la cantidad de subdirectorios que contiene contando a los directorios especiales ``." y ``.."
- El identificador del dueño.
- El identificador del grupo.
- El tamaño en bytes si es un fichero y si es un directorio el tamaño en bloques que ocupan los ficheros contenidos en él.
- La fecha y hora de la última modificación. Si la fecha es seis meses antes o una hora después de la fecha del sistema se coloca el año en lugar de la hora.
- El nombre del recurso.

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 pueden usar números o caracteres para indicar los permisos. Podremos comprender mejor cómo funciona la orden mirando directamente algunos ejemplos:

$ chmod u+x clase.txt	Añade el permiso de ejecución (+x) al usuario dueño (u) del fichero clase.txt.
$ chmod g=rx program.sh	Asigna exactamente los permisos de lectura y ejecución (rx) al grupo (g) sobre el fichero program.sh.
$ chmod go-w profile	Elimina el permiso de escritura (-w) en el grupo y en otros (go) del fichero o directorio profile.
$ chmod a+r,o-x *.ts	Añade el permiso de lectura (+r) para todos los usuarios (a) y elimina el de ejecución (-x) para otros (o) en todos los ficheros terminados en .ts.
$ chmod +t tmp/	Añade el permiso especial t al directorio tmp.
$ chmod 755 /home/pepe/doc/	Asigna los permisos con representación octal 755 (rwx r-x r-x) al fichero /home/pepe/doc.
$ chmod -R o+r apps/	Añade el permiso de lectura a otros en el directorio apps y además lo hace de forma recursiva, añadiendo dicho permiso también en todos los ficheros y directorios contenidos en apps.
# chmod 4511 /usr/bin/passwd	Asigna los permisos con representación octal 4511 (r-s--x--x)
$ chmod 644 *	r w – r - - r - - Lectura y escritura para el usuario, lectura para el grupo y lectura para otros.

Como ejercicio, cread un directorio copia en vuestro home de usuario, y dentro copiad todos los ficheros que existen en el directorio /etc. Modificad los permisos de los ficheros copiados, probando tanto la orden chmod de forma numérica como mediante caracteres.

Si creamos un nuevo fichero, veremos como es creado con unos permisos por defecto, normalmente 644.

Para determinar estos permisos que se asocian por defecto a los ficheros o directorios creados, cada usuario posee una **máscara** de permisos. Esta mascara por defecto suele ser 022 para los usuarios comunes.
Para calcular los permisos finales conociendo la máscara, se hace la siguiente operación por parte del sistema:

Tipo de ficheros	Operación	Desarrollo	Resultado
Ficheros normales	666 - máscara	(666 – 022 = 644)	644
Directorios y Ficheros ejecutables	777 - máscara	(777 – 022 = 755)	755

Para ajustar 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
$ umask 037	asigna la máscara 037
$ umask –S u=rwx,g=rwx,o=rx	Directorios y archivos ejecutables se crearán con los permisos 775 y los archivos comunes con los permisos 664.
$ umask 077	Los nuevos directorios tendrán el permiso 700 y los nuevos ficheros tendrán el permiso 600.

Como vemos, si usamos el formato numérico en umask, tendremos que restar la máscara al total de permisos para saber que permisos se asignarán por parte del sistema. Sin embargo, si usamos el formato simbólico (-S) de umask, asignaremos directamente los permisos que el sistema asignará.

Para dejar este umask fijo, conviene agregarlo al fichero .bash_profile o al fichero .bash_rc de nuestro directorio de inicio.

ENTORNOS DE TRABAJO EN BASH

Un entorno de trabajo en Linux no es más que la configuración que posee un usuario durante su interacción con el sistema y más específicamente con el shell. Las características del entorno para el caso de bash pueden ser la forma en que se muestra el prompt, los alias y funciones definidos y las variables del entorno de forma general, ya que estas definen el comportamiento de muchos programas y comandos. Las facilidades para cambiar el entorno de trabajo dependen de las capacidades del shell que se utilice.

VARIABLES DEL ENTORNO

Las variables del entorno almacenan valores que describen las propiedades del entorno de trabajo. Para dejarlo fijo en la sesión, entonces conviene agregarlo a .bash_profile o .bash_rc de nuestro directorio de inicio. Un usuario puede definir sus propias variables o modificar las ya existentes.

Para asignarle valor a una variable en el shell se emplea el operador de asignación tradicional (=) entre el nombre de la variable y el valor asignado (no deben haber espacios intermedios).

```
MENSAJE="Hola Mundo"
```

Para acceder al valor de una variable en el shell se emplea el carácter $ seguido por el nombre de la variable. Para imprimir en la terminal el valor de una variable se puede utilizar el comando echo.

```
echo $MENSAJE
```

Dentro de un shell se pueden ejecutar otros shells que serían hijos del primero (subshells) heredando todo o parte del entorno de trabajo del padre. Para que una variable mantenga su valor en los shells hijos es necesario indicarlo explícitamente mediante el comando export.

```
$ export MENSAJE
```

Tanto la asignación del valor como el exportar una variable se pueden hacer a la vez:

```
$ export MENSAJE="Hola amigo"
```

Para ver las variables del entorno definidas se puede emplear el comando **set**. Este además se relaciona con las opciones que es otra forma de regir el comportamiento del shell. Las opciones se activan (on) o desactivan (off). Estas se utilizan para indicar propiedades del shell muy específicas por lo que no nos vamos a detener en ellas. Si se desea conocer más al respecto se puede hacer $ help set | less.

Para eliminar el valor de una variable se emplea el comando unset.

```
$ unset MENSAJE
```

Algunas variables del entorno en bash son:

♦ **PATH**: guarda la secuencia de caminos donde el shell busca los programas que se intenten ejecutar en la línea de comandos cuando no se utilizan los caminos absolutos. Estos caminos se separan por el carácter : (dos puntos)

Ejemplo: $ echo $PATH
/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin

Para añadir un nuevo camino puede hacerse:
export PATH=$PATH: /bin

♦ **USER**: contiene el login del usuario actual.

♦ **PAGER**: almacena el paginador utilizado por defecto por algunos programas. Por ejemplo, el comando man utiliza esta variable para determinar que paginador empleará, aunque primero chequea otra variable llamada MANPAGER y si esta no tiene valor entonces acude a PAGER, que de no tener valor tampoco, asumirá al less como paginador por defecto. También asociada a man existe la variable MANPATH donde se especifican los caminos de los manuales que despliega man, y LANG para indicar el idioma3.2.
Ejemplo: $ export PAGER=more
$ man bash

♦ **HOME**: guarda el directorio base del usuario actual.
Ejemplo: $ echo $HOME

♦ **EDITOR**: contiene el editor por defecto del usuario actual. De no tener valor asociado se utiliza vi. En entornos gráficos se pueden indicar editores visuales aunque para ello se prefiere emplear la variable VISUAL.

♦ **PS1**: almacena la estructura del prompt principal del usuario. Permite una serie de macros.
\d : contiene la fecha del sistema.
\h : contiene el nombre de la máquina.
\T : contiene la hora del sistema.
\u : contiene el login del usuario.
\w : contiene el nombre completo del directorio de trabajo actual.
\W : contiene la base del directorio actual (Ejemplo: para /home/pepe/doc la base es doc).
\$: si el ID del usuario es 0 (root) contiene el valor # y sino, $.
\# : contiene el número del comando actual desde la creación del shell.

El prompt principal por defecto tiene la forma: "[\u@\h \W]\$ "

Ejemplo:
$ export PS1="[\T,\u\#]\$ "

♦ **PS2**: guarda el prompt secundario. Este es el que se presenta cuando el shell no puede interpretar lo que se ha escrito hasta el momento.

♦ **PWD**: contiene el directorio de trabajo actual. Esta variable la pueden emplear algunos programas para saber desde donde se invocaron.

♦ **SECONDS**: almacena la cantidad de segundos transcurridos desde la invocación del shell actual.

FICHEROS PERFILES

Para cada usuario existen tres ficheros muy importantes que permiten definir en gran medida las características del shell durante la interacción con este. Estos constituyen shells scripts y se conocen como ficheros perfiles:

.bash_profile ó .profile : se ejecuta siempre que se abra una nueva sesión en el sistema. Cada vez que un usuario se conecte al sistema se ejecutará el script .bash_profile, en el caso de que se utilice a bash como shell. Para ser compatible con sus versiones anteriores, bash en caso de que no existiera .bash_profile, ejecuta .bash_login, o sino, .profile. En este fichero se pueden colocar las asignaciones a las variables del entorno siendo debidamente exportadas a través de export. También se puede establecer la máscara de permisos usando umask.

.bash_logout: se ejecuta al terminar una sesión de trabajo.

.bashrc: se ejecuta siempre que se invoque un nuevo shell. Por lo general en él se colocan las definiciones de funciones y los alias de comandos.
Cuando se crea un nuevo usuario se le colocan en su directorio base estos tres ficheros cuyos patrones están el directorio /etc/skel. Estos ficheros son los que ajustan los perfiles para el shell. Aparte, existen en el home de cada usuario otros ficheros que cargan perfiles para los entornos gráficos como el gnome, perfiles para el open office, etc.

EJECUCIÓN EN BASH

Existen dos formas de ejecutar un shell script (fichero con instrucciones bash):

1. Invocándolo directamente por su nombre. Para esto el camino al fichero debe encontrarse en la variable PATH del entorno, sino será necesario especificar el camino completo del fichero. El fichero debe además, poseer los permisos de ejecución adecuados. Lo que ocurrirá en este caso es que se creará un shell hijo que será el que realmente ejecute (interprete) al script.
2. Utilizando el comando source. De esta forma se ejecutará el script en el shell actual. En este caso no será necesario que el fichero correspondiente posea permisos de ejecución. El comando se puede sustituir por el carácter punto. Por ejemplo, si se modificara el .bash_profile no será necesario, para activar los cambios, desconectarse y conectarse nuevamente al sistema, simplemente se puede hacer:

   $ source .bash_profile ó $ . .bash_profile

Un ejercicio interesante puede ser hacer un shell script (editar un fichero) llamado

program.sh con la siguiente línea: echo $SECONDS

Ejecutar luego: $. program.sh (que es lo mismo que $ source program.sh)
por último asignarle al fichero permisos de ejecución para ejecutarlo de la forma tradicional:

$ chmod a+x program.sh
$ ./program.sh

¿Podéis explicar las diferencias en la salida?

PROCESOS

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 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:

```
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 desde 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:
♦ 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 axu $ 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.

CONTROL DE LOS PROCESOS EN BASH

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 la construcción:

<línea de comando> &

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 (1º plano). Si es efectivo, el proceso finaliza su ejecución (se le mata).

- Ctrl-z : trata de detener el proceso en foreground. Si es efectivo el proceso continúa activo aunque deja de acceder al procesador (está detenido y pasa a background ó 2º plano).

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 comenzara 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. Se puede volver a traerlo a 1º plano con fg.

COMUNICACIÓN CON LOS PROCESOS

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 hacer:

$ kill –l (un listado de todos los tipos de señales)

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
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO

Sintaxis:
kill -l [señal]
kill [-s señal] <id>
kill [-señal] <id>

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).

Ejemplos:

```
$ kill 1000          # envía la señal 15 (TERM) al proceso 1000
$ kill -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 fue exitoso el envío de la señal.

Ejemplo: $ killall -9 ssh

PRIORIDADES DE LOS PROCESOS

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 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.

DISPOSITIVOS DE ALMACENAMIENTO. MONTAJE Y DESMONTAJE.

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 y de CD-ROM son ejemplos de estos dispositivos 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 dónde y en qué forma se montan los diferentes dispositivos. Veamos un ejemplo de dicho fichero:

```
usuario@debian6:~$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
# / was on /dev/sda1 during installation
UUID=e6f32ac6-334c-4422-b2d2-b9648b017211 /               ext3    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=514aa442-92ef-4bB4-8020-b5cb526f0326 none            swap    sw              0       0
/dev/scd0       /media/cdrom0   udf,iso9660 user,noauto     0       0
/dev/fd0        /media/floppy0  auto    rw,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. También puede aparecer codificada como UUID.
- 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 primer usuario distinto de root conectado al sistemalocalmente

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.

- 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.

- errors : indica que hacer si el dispositivo presenta errores. Así por ejemplo errors=remount-ro indica que se debe montar el sistema de ficheros como de solo lectura (read only).

Actualmente para cada dispositivo con sistema de ficheros ext 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.

Ejemplos de líneas en el fichero /etc/fstab:

LABEL=/	/	ext4	defaults	1 1
/dev/hda7	/home	ext2	defaults	1 2
/dev/cdrom	/mnt/cdrom	iso9660	noauto,owner,ro	0 0
/dev/fd0	/mnt/floppy	auto	noauto,owner	0 0
/dev/hda9	swap	swap	defaults	0 0
/dev/hdc1	/alien	ext2	ro,noexec,owner,noauto	1 0
/dev/hda1	/mnt/win	msdos	ro,async,nouser,auto	0 0

Si queremos que un dispositivo se monte automáticamente cada vez que el sistema se inicie, basta con colocar una línea apropiada en el fichero /etc/fstab.

Si lo que queremos es montar o desmontar dispositivos directamente, podemos emplear 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 /dev/cdrom # monta el cdrom. Toma las especificaciones del fichero /etc/fstab
$ mount /mnt/cdrom # hace lo mismo que el anterior
$ umount -a -t ntfs # desmonta todo los dispositivos con file system ntfs especificados en /etc/mtab
$ umount /mnt/cdrom # desmonta el cdrom
$ mount /dev/sda5 /home/usuario/llavero # monta el dispositivo sda5 (supongamos que es una memoria USB) en el directorio llavero del home del usuario usuario.

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 abierto 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 medios extraíbles en esta forma, sobre todo cuando se realizan operaciones de escritura no se debe olvidar desmontarlo antes de extraerlo del ordenador. El resultado de extraer un dispositivo sin desmontarlo antes 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.

Hoy en día la mayoría de las distribuciones cuentan con algún tipo de programa monitor en memoria que se encargará de montar automáticamente cualquier dispositivo externo que conectemos al sistema. El desmontaje de los dispositivos sin embargo debe ser manual.

Un comando muy útil en Linux es **df**. Este se emplea para conocer información acerca de las particiones y dispositivos montados actualmente en el sistema. Para cada dispositivo se muestra por defecto su tamaño, el espacio empleado, que porcentaje esta empleado, así como el directorio donde se ha montado.

Sintaxis: df [opciones] [directorio]

Algunas opciones:
♦ -h : (human readable view) por defecto los tamaños se muestran en bytes y con esta opción se hace de forma más legible (Ej. G para gigabytes y M para megabytes).
♦ -T : muestra además el tipo de file system de cada dispositivo.
♦ -i : describe la utilización de los i-nodos de cada partición, en lugar del espacio.

Otros 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.

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.

GESTORES DE PARTICIONES.

En Linux el particionador estándar es el fdisk. Este posee una interfaz texto que permite crear, modificar y borrar particiones de diversos tipos (Linux, FAT12/16/32, NTFS, minix, Linux Swap, HPFS, Novell, etc). Funciona en modo interactivo y para ejecutarlo se le pasa como argumento el disco duro a particionar a través del dispositivo correspondiente.
Ejemplo:
fdisk /dev/hda

Desde el prompt que muestra fdisk es posible ejecutar comandos que permiten realizar las distintas operaciones. Algunos de estos comandos se listan a continuación:

♦ m : muestra la ayuda de todos los posibles comandos.
♦ p : imprime la tabla de particiones del disco duro actual. Su salida es similar a la del siguiente ejemplo:
♦ n : añade una nueva partición. Para ello se debe indicar si esta será lógica o primaria, el sector inicial (se ofrece uno por defecto) y el sector final o el tamaño total de la partición. Por defecto las particiones se crean del tipo Linux.
♦ d : permite borrar una partición.
♦ l : lista todos los tipos de particiones reconocidas por fdisk. Los tipos se identifican utilizando números hexadecimales.
♦ t : permite cambiar el tipo de una partición.
♦ w : actualiza la tabla de particiones y termina. Hasta que no se ejecute este comando los cambios realizados no tendrán validez.
♦ q : permite salir sin salvar los cambios realizados hasta el momento.

Un uso habitual que hacemos del fdisk es utilizarlo para listar todos los dispositivos de almacenamiento del sistema, a fin de poder montarlos. Esto lo conseguimos con la opción –l (ele) y sin indicar ningún dispositivo.

Existen otros programas particionadores en Linux, tanto pensados para ejecutarlos desde terminal como para ser lanzados desde el entorno gráfico. Algunos de los más usados son: cfdisk, parted, qtparted, gparted, etc. CREACIÓN DE PARTICIONES Y FICHEROS SWAP

Como se explicó anteriormente en Linux es usual crear una o varias particiones especiales para intercambiar las zonas de memoria almacenadas en la RAM. Este proceso permite el mecanismo conocido como memoria virtual y es imprescindible para el funcionamiento de los sistemas operativos modernos. Las particiones con ese propósito son conocidas como particiones Swap y se pueden crear durante la instalación del sistema o posteriormente.
Para crear una partición Swap después de instalado Linux se deben seguir los siguientes pasos:

1) Crear la partición usando algún particionador, por ejemplo fdisk.

2) Utilizar el comando mkswap para indicar que la partición se empleará para hacer Swap.
Ejemplo:
mkswap /dev/hda8

3) Utilizar el comando swapon para activar la partición, para desactivarla se puede emplear swapoff. Ejemplo:
swapon /dev/hda8
swapoff /dev/hda8

Para hacer intercambio de memoria también se pueden utilizar uno o varios ficheros (como hace Windows). No obstante esto no es lo más recomendado salvo en el caso de que no se disponga de espacio en disco sin particionar. Para crear un fichero Swap de 64MB nombrado swapfile en el directorio raíz se deben seguir los siguientes pasos:

1) Crear el fichero de 64MB lleno de caracteres nulos ejecutando el comando:
dd if=/dev/zero of=/swapfile bs=1024 count=6553

2) Indicar que el fichero se empleará como Swap ejecutando:
mkswap /swapfile

3) Comenzar a utilizar el fichero para hacer Swap ejecutando:
swapon /swapfile

Para que una partición o fichero Swap se active siempre que el sistema arranque, se debe colocar la entrada correspondiente en el fichero /etc/fstab. Ejemplos:
/dev/hda9 swap swap defaults,pri=1 0 0
/swapfile swap swap defaults,pri=2 0 0

La opción pri=<n> indica la prioridad con que se utiliza la partición o fichero para hacer Swap.

Para que el kernel pueda balancear adecuadamente el uso de más de una partición o fichero Swap estos deben colocarse en discos distintos, con controladores distintos, y asignarles la misma prioridad.

En el caso del ejemplo se le asigna una prioridad mayor a la partición que al fichero por razones obvias (asumiendo que están en el mismo disco duro).

El comando swapon permite además mostrar un sumario de las particiones y ficheros Swap activos, y su utilización actual, a través de la opción -s. La opción -a activa todas las particiones o ficheros Swap con la opción de montaje auto en /etc/fstab.

MANEJO DE PAQUETES EN LINUX.

Como ya se expresó anteriormente una distribución de Linux ofrece básicamente un conjunto de paquetes que contienen aplicaciones, utilidades, documentación y el propio kernel del sistema.

Estos paquetes no son más que ficheros con cierto formato que, manipulados por un comando u otra aplicación son capaces de instalarse en el sistema de acuerdo a las especificaciones que determinó su fabricante. Entre las especificaciones puede citarse a los ficheros y directorios que crea el paquete, el tamaño que ocupa una vez instalado, una

descripción breve y otra más amplia de su utilidad, las dependencias que puede haber respecto a otros paquetes, etc.

Algunos paquetes son dependientes de plataforma como es el caso del que contiene al kernel, otros por el contrario se pueden instalar en cualquier arquitectura como son los que agrupan documentación. Usualmente los programas fuentes también se agrupan en paquetes independientes de los compilados.

En Linux no se utiliza un único sistema de paquetes, sino que podemos encontrar varias alternativas. Veamos ahora las más usadas:

RPM

Existen varios tipos de formato para definir un paquete. Red Hat introdujo la forma RPM (RedHat Package Manager) la cual ha sido adoptada por otras distribuciones como SuSE y Mandrake. En cambio Debian y Slackware hacen sus paquetes siguiendo otra técnica. Los paquetes RPM por lo general poseen extensión rpm. Estos pueden tener un grado mayor o menor de compatibilidad con las distribuciones que emplean el mismo formato.

En Red Hat la herramienta por excelencia para administrar paquetes es el comando rpm que posee muchísimas opciones. Internamente rpm se basa en una base de datos con información acerca de lo instalado, las dependencias existentes, etc. Esta base de datos es actualizada transparentemente por rpm cuando se instala, actualiza o desinstala un paquete. También puede ser consultada para conocer información diversa acerca de lo ya instalado en el sistema.

De forma general el comando rpm brinda las siguientes posibilidades:
♦ Instalar, actualizar y desinstalar paquetes
♦ Construir y compilar paquetes
♦ Inicializar y reconstruir la base de datos
♦ Hacer consultas sobre paquetes instalados o no
♦ Verificar dependencias
♦ Adicionar firmas y chequearlas (mecanismo que permite asegurar la validez e integridad de un paquete)

A continuación se listan las opciones que se emplean más a menudo:

♦ -i : permite instalar los paquetes. Como argumento se colocan los nombres de los ficheros rpm a instalar. Internamente el comando chequea las dependencias funcionales, y de ser posible, reordena los ficheros rpm para que se satisfagan dichas dependencias durante la instalación.
♦ -e : permite desinstalar los paquetes. Como argumento se indican los nombres de los paquetes a desinstalar. El nombre de un paquete una vez instalado puede tener dos formas: una breve y otra más amplia que incluye el número de la versión. Algunos paquetes permiten que se instalen simultáneamente distintas versiones del mismo. Tal es el caso del paquete que contiene al kernel cuyo nombre breve siempre es kernel, mientras que pueden haber varios nombres ampliados como kernel-2.4.2-2 ó kernel-2.4.3-12.
♦ -U : permite actualizar los paquetes de versiones inferiores a superiores. Al igual que en la instalación, se colocan como argumentos los nombres de los ficheros rpm que se

desean actualizar. También aquí el comando realiza el ordenamiento más adecuado para satisfacer dependencias. De no existir una versión inferior previamente instalada, el proceso sería equivalente a una instalación de la versión superior.
♦ -F : es igual a la opción anterior salvo en que hace la actualización sólo si hay una versión inferior del paquete previamente instalada.
♦ -q : permite hacer consultas diversas a los paquetes instalados. Para ello se emplean como argumento los nombres de los paquetes: de forma breve o no. El tipo de consulta se expresa a través de una segunda opción. Las más importantes son:
♦ -a : lista los nombres (incluyendo la versión) de todos los paquetes instalados en el sistema.
♦ -i : brinda información acerca de un paquete: nombre, versión, tamaño, descripción, fabricante, fecha de instalación, licencia, clasificación, etc.
♦ -f : indica dado un fichero del file system, el nombre del paquete que lo creó.
♦ -l : lista los nombres de todos los ficheros que instaló un paquete.
♦ -p : permite hacer consultas a un paquete no instalado a partir del fichero rpm que lo contiene. Se puede combinar con las opciones anteriores -i y -l.

Otras opciones interesantes:

♦ -v : activa el modo explicativo durante la instalación o actualización. Si se indica dos veces, o sea-vv entonces se imprime información detallada de todas las operaciones efectuadas en la base de datos de RPMs durante el proceso.
♦ -h : muestra hasta 50 caracteres ``#'' que expresan el progreso del proceso de instalación o de actualización. Usualmente se combina con -v.
♦ -R :es una opción de consulta que muestra las depencias que tiene un paquete.
♦ -d: es una opción de consulta que lista los ficheros de documentación que instala un paquete.
♦ -changelog : es una opción de consulta que muestra los logs que expresan los cambios efectuados en el paquete a partir de su versión anterior.
♦ -scripts : es una opción de consulta que muestra los scripts que se ejecutan al intalar y desinstalar un paquete.

MANIPULACIÓN DE PAQUETES EN DEBIAN.

En el principio existían los .tar.gz. Los usuarios tenían que descomprimir, destarear y compilar cada programa que quisieran usar en su sistema GNU/Linux. Cuando Debian fue creado, se decidió que el sistema incluyera un programa que se encargara de manejar los paquetes (programas) instalados en el ordenador. Este programa se llamó **DPKG**. Así fue como nació el primer "paquete" en el mundo GNU/Linux, aún antes de que RedHat decidiera crear su propio sistema de paquetes "rpm".

Rápidamente llegó un nuevo dilema a las mentes de los creadores de GNU/Linux. Ellos necesitaban un modo fácil, rápido y eficiente de instalar programas, que manejara automáticamente las dependencias (programas que dependen de otros) y se hiciera cargo de la configuración mientras se actualizan. Nuevamente Debian fue pionera y creó el **APT**, Herramienta Avanzada de Empaquetamiento (Advanced Packaging Tool).

Como parte de su funcionamiento, APT utiliza un archivo que contiene las direcciones de varios servidores (repositorios) que se encuentran en Internet que contienen los paquetes

candidatos a ser instalados. También indicamos en este fichero si vamos a cargar paquetes desde medios locales como un cdrom, un directorio compartido de la red, etc. Este archivo es /etc/apt/sources.list. Veamos el contenido de un archivo sources.list:

```
usuario@debian6:~$ cat /etc/apt/sources.list
#

# deb cdrom:[Debian GNU/Linux 6.0.0 _Squeeze_ - Official Multi-architecture amd64/i386 NETINST
#1 20110205-14:45]/ squeeze main

#deb cdrom:[Debian GNU/Linux 6.0.0 _Squeeze_ - Official Multi-architecture amd64/i386 NETINST #
1 20110205-14:45]/ squeeze main

deb http://ftp.es.debian.org/debian/ squeeze main
deb-src http://ftp.es.debian.org/debian/ squeeze main

deb http://security.debian.org/ squeeze/updates main
deb-src http://security.debian.org/ squeeze/updates main

deb http://ftp.es.debian.org/debian/ squeeze-updates main
deb-src http://ftp.es.debian.org/debian/ squeeze-updates main
usuario@debian6:~$
```

Cada línea en este fichero es un repositorio de software, que almacena paquetes que podemos descargar e instalar.

La primera palabra en cada línea, deb o deb-src, indica el tipo del archivo que se almacena en el repositorio: ya sea que contenga paquetes binarios (deb), esto es, los paquetes pre-compilados que normalmente se usan, o los paquetes fuente (deb-src), que son los códigos originales o fuentes.

A continuación de esta palabra viene la dirección donde se encuentra el repositorio, que puede ser bien una URL de red o un indicador como cdrom.

A continuación, viene el nombre de la distribución de Linux que queremos manejar con el repositorio. Normalmente un repositorio almacena paquetes para varias versiones de Linux, así por ejemplo el fichero anterior indica al repositorio que queremos los paquetes de la versión squeeze (Debian 6.0).

También se indica el tipo de repositorio, que puede ser normal, de actualización, de seguridad, etc.

Cada línea termina indicando el tipo de repositorio que queremos usar. Vemos que en el ejemplo todos nuestros repositorios son main (general).

En debian normalmente se pueden indicar repositorios para stable, testing, unstable y experimental, que son las distintas versiones de desarrollo ofrecidas por Debian.

En Ubuntu podemos indicar main, restricted, universe y multiverse, dependiendo de si nos queremos bajar los paquetes oficiales y libres de Ubuntu, los paquetes no enteramente libres, paquetes no oficiales, etc.

Siempre que se modifica el archivo sources.list, hay que ejecutar el comando apt-get

update. Debe hacer esto para permitir a APT obtener la lista de paquetes desde las fuentes que especificamos.

Si queremos utilizar el CD-ROM actual introducido para instalar los paquetes o para actualizar el sistema con APT, lo podemos agregar al archivo sources.list. Para hacerlo, podemos utilizar el programa apt-cdrom así:

apt-cdrom add

Si tenemos en la unidad de cdrom un cd con paquetes debian, esta instrucción lo montará, y buscará la información de los paquetes en el CD.

El sistema de paquetes utiliza una base de datos para llevar un control sobre los paquetes instalados, los no instalados y cuales están disponibles para su futura instalación. El programa apt-get utiliza esta base de datos para averiguar cómo instalar los paquetes que son requeridos por el usuario y para indagar sobre que paquetes adicionales serán requeridos para que el paquete seleccionado funcione correctamente. Esta base de datos se actualiza con la orden apt-get update.

INSTALAR PAQUETES

Con el archivo sources.list listo y la lista de paquetes disponibles al día, todo lo que necesitamos es ejecutar apt-get para tener el paquete que queramos instalar. Por ejemplo, si ejecutamos:

apt-get install dopewars

APT buscará en su base de datos para encontrar la versión más reciente del paquete dopewars y lo descargará del servidor correspondiente especificado en sources.list. Si este paquete necesitara otro para funcionar APT resolverá las dependencias e instalará los paquetes necesarios. Observemos este ejemplo:

```
usuario@debian6:~$ sudo apt-get install 3dchess
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  xaw3dg
Se instalarán los siguientes paquetes NUEVOS:
  3dchess xaw3dg
0 actualizados, 2 se instalarán, 0 para eliminar y 21 no actualizados.
Necesito descargar 195 kB de archivos.
Se utilizarán 607 kB de espacio de disco adicional después de esta operación.
¿Desea continuar [S/n]?
```

Como vemos hemos indicado que queremos instalar el paquete 3dchess, apt se ha puesto en contacto con el repositorio indicado en sources.list, le ha pedido información sobre dicho paquete y ha encontrado que tiene una dependencia no resuelta, es decir, que le hace falta un paquete para funcionar que no tenemos instalado en nuestro sistema (en el caso del ejemplo, el que falta es el paquete xaw3dg).

Vemos como apt automáticamente marca dicho paquete para instalarlo y nos pide confirmación. APT se encargará de bajar ambos paquetes, descomprimirlos, instalarlos en el sistema y configurarlos para que funcionen juntos.

APT sólo pregunta por confirmación cuando se van a instalar paquetes que no fueron especificados en la línea de comando.

Las siguientes opciones de apt-get podrían ser útiles:

- h Ayuda de la orden.
- d Solo descarga los paquetes, no instala nada en el sistema.
- f Continua aunque haya fallos de integridad en los ficheros bajados.
- y Asume SI a todas las preguntas.
- u Muestra una lista de paquetes modificados.

Pueden seleccionarse varios paquetes para instalar en una sola línea. Los archivos descargados son almacenados en el directorio /var/cache/apt/archives para su instalación posterior.

Si en lugar de un paquete queremos descargarnos el código fuente del paquete podemos hacerlo usando el comando apt-get source paquete.

REINSTALAR UN PAQUETE

Si queremos reinstalar un paquete, (por qué se haya estropeado, hayamos tocado la configuración y ya no funcione, etc.) podemos usar la opción –reinstall:

#apt-get install paquete --reinstall

```
usuario@debian6:~$ sudo apt-get install mc --reinstall
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
0 actualizados, 0 se instalarán, 1 reinstalados, 0 para eliminar y 21 no actualizados.
Se necesita descargar 0 B/2173 kB de archivos.
Se utilizarán 0 B de espacio de disco adicional después de esta operación.
(Leyendo la base de datos ... 129719 ficheros o directorios instalados actualmente.)
Preparando para reemplazar mc 3:4.7.0.9-1 (usando .../mc_3%3a4.7.0.9-1_i386.deb) ...
Desempaquetando el reemplazo de mc ...
Procesando disparadores para man-db ...
Procesando disparadores para menu ...
Configurando mc (3:4.7.0.9-1) ...
Procesando disparadores para menu ...
usuario@debian6:~$
```

ELIMINAR UN PAQUETE

Si ya no necesitamos utilizar cierto paquete, podemos eliminarlo de nuestro sistema utilizando APT. Para realizar esta tarea sólo debemos escribir:

#apt-get remove paquete

```
usuario@debian6:~$ sudo apt-get remove mc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los siguientes paquetes se ELIMINARÁN:
  mc
0 actualizados, 0 se instalarán, 1 para eliminar y 21 no actualizados.
Se liberarán 6603 kB después de esta operación.
¿Desea continuar [S/n]? s
(Leyendo la base de datos ... 129718 ficheros o directorios instalados actualmente.)
Desinstalando mc ...
update-alternatives: utilizando /usr/bin/vim.tiny para proveer /usr/bin/view (view) en modo aut
omático.
Procesando disparadores para menu ...
Procesando disparadores para man-db ...
usuario@debian6:~$
```

ACTUALIZAR UN PAQUETE

Las actualizaciones de los paquetes pueden realizarse con tan sólo un comando:

#apt-get upgrade.

Podemos utilizar esta opción para actualizar los paquetes de la distribución actual, o bien para actualizar a una nueva distribución, aunque el comando apt-get dist-upgrade es una mejor opción.

Es muy útil utilizar este comando con la opción -u. Esta opción muestra la lista completa de paquetes que APT actualizará. Sin ella, se estaría actualizando a ciegas. APT descargará las versiones más recientes de cada paquete y las instalará de la manera más apropiada. Es muy importante ejecutar siempre apt-get update antes de probar esto.

```
usuario@debian6:~$ sudo apt-get upgrade -u
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se actualizarán los siguientes paquetes:
  apache2.2-bin bind9-host dnsutils gdm3 host iceweasel isc-dhcp-client isc-dhcp-common
  libbind9-60 libdns69 libisc62 libisccc60 libisccfg62 liblwres60 libmodplug1 libmozjs2d
  libnss3-1d libtiff4 tzdata x11-xserver-utils xulrunner-1.9.1
21 actualizados, 0 se instalarán, 0 para eliminar y 0 no actualizados.
Necesito descargar 16,0 MB de archivos.
Se utilizarán 98,3 kB de espacio de disco adicional después de esta operación.
¿Desea continuar [S/n]?
```

BUSCAR UN PAQUETE

Existen algunas interfaces para el APT que lo hacen más fácil de utilizar. Pero nuestro objetivo aquí es aprender a manejar APT puro. Así que, ¿cómo podríamos saber el nombre de un paquete que queremos instalar?
Tenemos numerosos recursos para realizar esa tarea. Empezaremos con apt-cache. Este programa es utilizado por APT para mantener su base de datos. Nosotros sólo veremos un poco de sus aplicaciones. Por ejemplo, supongamos que queremos encontrar un

emulador de la nintendo DS, y después bajar algunos juegos. Podríamos realizarlo así:

```
usuario@debian6:~$ apt-cache search nintendo
desmume - Nintendo DS emulator
fceu - FCE Ultra - a nintendo (8-bit) emulator
libgme-dev - Playback library for video game music files - development files
libgme0 - Playback library for video game music files - shared library
gbsplay - A Gameboy sound player
gnome-nds-thumbnailer - Nintendo DS roms thumbnailer for GNOME
kamefu-data - Data files for Kamefu
kamefu - KDE All Machine Emulator Frontend for Unix - binary files
libkamefu-dev - Development headers for Kamefu
libkamefu0 - Libraries for Kamefu
```

Veremos que la orden nos devuelve una cantidad muy grande de paquetes relacionados de alguna u otra manera con nintendo. Para refinar nuestra búsqueda, podríamos utilizar una orden como la siguiente:

```
usuario@debian6:~$ apt-cache search nintendo | grep -i 'nintendo ds"
desmume - Nintendo DS emulator
gnome-nds-thumbnailer - Nintendo DS roms thumbnailer for GNOME
usuario@debian6:~$
```

Parece que hay un paquete prometedor, el desmume. Para ver información sobre dicho paquete usamos el comando apt-cache show:

```
usuario@debian6:~$ apt-cache show desmume
Package: desmume
Priority: extra
Section: games
Installed-Size: 5048
Maintainer: Debian Games Team <pkg-games-devel@lists.alioth.debian.org>
Architecture: i386
Version: 0.9.6-1-1
Depends: libasound2 (>> 1.0.18), libc6 (>= 2.3.6-6~), libgcc1 (>= 1:4.1.1), libgl1-mesa-glx
ibgl1, libglade2-0 (>= 1:2.6.1), libglib2.0-0 (>= 2.14.0), libglu1-mesa | libglu1, libgtk2.0-
(>= 2.14.0), libosmesa6 (>= 6.5.2-1) | libgl1-mesa-glide3, libpango1.0-0 (>= 1.14.0), libsdl1
debian (>= 1.2.10-1), libstdc++6 (>= 4.4.0), zlib1g (>= 1:1.1.4)
Filename: pool/main/d/desmume/desmume_0.9.6-1-1_i386.deb
Size: 1554672
MD5sum: 5045bf82886ac1a2bd8a7de83fe2b441
SHA1: 96a6291b73aa4bed33e672fc325cf16ff3825247
SHA256: 899292aa8b5fa01519d3728066fd452c4ffb99b44bd51ba0d4fe05f91d8afdb0
Description: Nintendo DS emulator
 DeSmuME is a Nintendo DS emulator running homebrew demos and commercial games.
```

Como vemos obtenemos mucha información sobre el paquete, como la descripción, el tamaño, las dependencias, etc. Una vez comprobado que es el paquete que realmente queremos, bastaría con instalarlo con un apt-get install.

MANEJO DE PAQUETES CON DPKG.

Aparte de manejar paquetes con apt, también podemos manejarlos a bajo nivel con los comandos dpkg. Veamos algunos de ellos:

Comando dpkg	Explicación.
dpkg –i paquete.deb	A veces podemos bajar directamente un paquete desde internet, normalmente con la extensión .deb. Para instalar uno de estos paquetes basta con usar dpkg con la opción –i.
dpkg –r paquete	Elimina un paquete (remove).
dpkg –P paquete	Elimina un paquete, y además elimina también todos sus archivos de configuración, temporales, etc.
dpkg -l	Orden importante. Nos muestra un listado de todos los paquetes instalados en nuestro sistema.
dpkg –L paquete	Nos muestra información de un paquete ya instalado en el sistema, indicando que ficheros se instalaron y donde.

Un comando que se suele usar a menudo es dpkg-reconfigure. Este comando nos permite reconfigurar un paquete completo del sistema. Algunos usos de este comando habituales suelen ser:

Comando dpkg-reconfigure	Explicación.
dpkg-reconfigure locales	Configuración del idioma usado en los terminales de Debian.
dpkg-reconfigura xserver-xorg	Configura el sistema gráfico.