Tutorial de Linux

1 Comandos de Linux

1.1 Comandos básicos

Los comandos son esencialmente los mismos que cualquier sistema UNIX. En la tablas 1 y 2 se tiene la lista de comandos mas frecuentes. En la tabla 3 se tiene una lista de equivalencias entre comandos Unix/Linux y comandos DOS.

Comando/Sintaxis	**Descripción**	**Ejemplos**
cat *fich1* [...*fichN*]	Concatena y muestra un archivos	cat /etc/passwd
	archivos	cat dict1 dict2 dict
cd [*dir*]	Cambia de directorio	cd /tmp
chmod *permisos fich*	Cambia los permisos de un archivo	chmod +x miscript
chown *usuario:grupo fich*	Cambia el dueño un archivo	chown nobody miscript
cp *fich1...fichN dir*	Copia archivos	cp foo foo.backup
diff [-*e*]*arch1 arch2*	Encuentra diferencia entre archivos	diff foo.c newfoo.c
du [-*sabr*] *fich*	Reporta el tamaño del directorio	du -s /home/
file *arch*	Muestra el tipo de un archivo	file arc_desconocido
find *dir test acción*	Encuentra archivos.	find . -name ‘‘.bak” -print
grep [-*cilnv*] *expr archivos*	Busca patrones en archivos	grep mike /etc/passwd
head -*count fich*	Muestra el inicio de un archivo	head prog1.c
mkdir *dir*	Crea un directorio.	mkdir temp
mv *fich1 ...fichN dir*	Mueve un archivo(s) a un directorio	mv a.out prog1
mv *fich1 fich2*	Renombra un archivo.	mv .c prog_dir
less / more *fich(s)*	Visualiza página a página un archivo.	more muy_largo.c
	less acepta comandos vi.	less muy_largo.c
ln [-*s*] *fich acceso*	Crea un acceso directo a un archivo	ln -s /users/mike/.profile .

ls	Lista el contenido del directorio	ls -l /usr/bin
pwd	Muestra la ruta del directorio actual	pwd
rm *fich*	Borra un fichero.	rm foo.c
rm -r *dir*	Borra un todo un directorio	rm -rf prog_dir
rmdir *dir*	Borra un directorio vacío	rmdir prog_dir
tail -*count fich*	Muestra el final de un archivo	tail prog1.c
vi *fich*	Edita un archivo.	vi .profile

Comandos Linux/Unix de manipulación de archivos y directorios

Comando/Sintaxis	**Descripción**	**Ejemplos**
at [-*lr*] *hora* [*fecha*]	Ejecuta un comando mas tarde	at 6pm Friday miscript
cal [[*mes*] *año*]	Muestra un calendario del mes/año	cal 1 2025
date [*mmddhhmm*] [+*form*]	Muestra la hora y la fecha	date
echo *string*	Escribe mensaje en la salida estándar	echo ``Hola mundo''
finger *usuario*	Muestra información general sobre	finger nn@maquina.aca.com.co
	un usuario en la red	
id	Número id de un usuario	id usuario
kill [-*señal*] *PID*	Matar un proceso	kill 1234
man *comando*	Ayuda del comando especificado	man gcc
		man -k printer
passwd	Cambia la contraseña.	passwd
ps [*axiu*]	Muestra información sobre los procesos	ps -ux
	que se están ejecutando en el sistema	ps -ef
who / rwho	Muestra información de los usuarios	who

	conectados al sistema.	

Comandos Linux/Unix más frecuentes

Linux	**DOS**	**Significado**
cat	type	Ver contenido de un archivo.
cd, chdir	cd, chdir	Cambio el directorio en curso.
chmod	attrib	Cambia los atributos.
clear	cls	Borra la pantalla.
ls	dir	Ver contenido de directorio.
mkdir	md, mkdir	Creación de subdirectorio.
more	more	Muestra un archivo pantalla por pantalla.
mv	move	Mover un archivo o directorio.
rmdir	rd, rmdir	Eliminación de subdirectorio.
rm -r	deltree	Eliminación de subdirectorio y todo su contenido.

Equivalencia de comandos Linux/Unix y DOS

1.2 Comandos en background

Linux, como cualquier sistema Unix, puede ejecutar varias tareas al mismo tiempo. En sistemas monoprocesador, se asigna un determinado tiempo a cada tarea de manera que, al usuario, le parece que se ejecutan al mismo tiempo.

Para ejecutar un programa en background, basta con poner el signo ampersand (&) al término de la línea de comandos. Por ejemplo, si se quisiera copiar el directorio */usr/src/linux* al directorio */tmp*:

```
#cp -r /usr/src/linux /tmp &
#
```

Cuando ha terminado la ejecución del programa, el sistema lo reporta mediante un mensaje:

```
#
  [Done] cp -r /usr/src/linux /tmp
#
```

Si se hubiese ejecutado el programa y no se hubiese puesto el ampersand, se podría pasarlo a background de la siguiente manera:

1. Se suspende la ejecución del programa, pulsando *Ctrl+Z*.
2. Se ejecutamos la siguiente orden: `bg`

1.3 Interprete de comandos: Shell

El interprete de comandos es el programa que recibe lo que se escribe en la terminal y lo convierte en instrucciones para el sistema operativo.

En otras palabras el objetivo de cualquier intérprete de comandos es ejecutar los programas que el usuario teclea en el *prompt* del mismo. El *prompt* es una indicación que muestra el intérprete para anunciar que espera una orden del usuario. Cuando el usuario escribe una orden, el intérprete ejecuta dicha orden. En dicha orden, puede haber programas internos o externos: Los programas internos son aquellos que vienen incorporados en el propio intérprete, mientras que los externos son programas separados (ej: aplicaciones de */bin,/usr/bin,...*).

En el mundo Linux/Unix existen tres grandes familias de Shells como se muestra en la tabla a continuacion. Estas se diferencian entre sí básicamente en la sintaxis de sus comandos y en la interacción con el usuario.

Tipo de Shell	**Shell estándar**	**Clones libres**
AT&T Bourne shell	sh	ash, bash, bash2
Berkeley "C" shell	csh	tcsh
AT&T Korn shell	ksh	pdksh, zsh
Otros interpretes	--	esh, gush, nwsh

`Interpretes de comandos en Linux/Unix`

1.3.1 Sintaxis de los comandos

Los comandos tienen la siguiente sintaxis:

```
# programa arg1 arg2 ... argn
```

Se observa que, en la ``línea de comandos", se introduce el *programa* seguido de uno o varios *argumentos*. Así, el intérprete ejecutará el *programa* con las opciones que se hayan escrito.

Cuando se quiere que el comando sea de varias líneas, se separa cada línea con el carácter *barra invertida ()*. Además, cuando se quiere ejecutar varios comandos en la misma línea, los separa con *punto y coma (;)*. Por ejemplo:

```
# make modules ; make modules_install
```

En los comandos, también se puede utilizar los comodines:

- El *asterisco ()* es equivalente a uno o más caracteres en el nombre de un archivo. Ejm: `ls *.c` lista todos los archivos con extensión c.
- El *signo de interrogación (?)* es equivalente a un único carácter. Ejm: `ls curso.te?` lista el archivo curso.tex completando el último carácter.
- Un *conjunto de caracteres entre corchetes* es equivalente a cualquier carácter del conjunto. Ejm: `ls curso_linux.t[aeiou]x` lista curso_linux.tex seleccionando la *e* del conjunto. .

1.3.2 Variables de entorno

Una *variable de entorno* es un nombre asociado a una cadena de caracteres.

Dependiendo de la variable, su utilidad puede ser distinta. Algunas son útiles para no tener que escribir muchas opciones al ejecutar un programa, otras las utiliza el propio shell (*PATH*, *PS1*,...). La tabla muestra la lista de variables más usuales.

Variable	**Descripción**
DISPLAY	Donde aparecen la salidas de X-Windows.
HOME	Directorio personal.
HOSTNAME	Nombre de la máquina.
MAIL	Archivo de correo.
PATH	Lista de directorios donde buscar los programas.
PS1	Prompt.
SHELL	Intérprete de comandos por defecto.
TERM	Tipo de terminal.
USER	Nombre del usuario.

Variables de entorno más usuales

La forma de definir una variable de entorno cambia con el interprete de comandos, se muestra tcsh y bash siendo los dos mas populares en el ámbito Linux:

bash:

```
export VARIABLE=Valor
```

tcsh:

```
setenv VARIABLE Valor
```

Por ejemplo, para definir el valor de la variable DISPLAY:

bash:

```
export DISPLAY=localhost:0.0
```

tcsh:

```
setenv DISPLAY localhost:0.0
```

1.3.3 Alias

Un ``alias" es un nombre alternativo para un comando. Así, en lugar de escribir el comando propiamente dicho, escribiríamos el *alias* de dicho comando.

Un *alias* se puede definir por varios motivos, por ejemplo:

- Dar nombres familiares a comandos comunes: `alias md='mkdir'` Crearía un alias para el comando *mkdir*, similar al de DOS.
- Dar nombres a comandos largos: `alias tbz2='tar -cv --use-compress-program=bzip2 -f'` Crearía un alias para el comando *tar* para que use el compresor *bzip2* en lugar de *gzip*.

Para no tener que escribir todos los alias siempre que entremos al sistema, escribiríamos dicho alias en el archivo */.bash_profile*

1.3.4 Redireccionamiento de E/S

La filosofía de Linux/Unix es en extremo modular. Se prefieren las herramientas pequeñas con tareas puntuales a las meta-herramientas que realizan todo. Para hacer el modelo completo es necesario proveer el medio para ensamblar estas herramientas en estructuras mas complejas. Esto se realiza por medio del redireccionamiento de las entradas y las salidas.

Redirección de Entrada y Salidas fig_redirect width=5cmimages/standard_io.eps

Todos los programas tiene por por defecto una entrada estándar (teclado) y dos salidas: la salida estándar (pantalla) y la salida de error (pantalla). En ellos se puede sustituir la entrada y salidas estándar por otro dispositivo utilizando los caracteres ``" y ``", es decir, hacer que se lea un archivo que contenga las opciones a ejecutar y un archivo de salida, respectivamente. Por ejemplo:

Entrada:

Se desea realizar una transferencia de archivos por ftp automática. Para ello se va a utilizar el programa *ncftp* con unas determinadas instrucciones preestablecidas.

Se crea un archivo *entrada* con dichas instrucciones:

```
open linuxcol.uniandes.edu.co
cd /pub/linux/utils
get *
quit
```

y se ejecuta el programa: `ncftp < entrada .`

Salida:

Se quiere saber los archivos que empiezan por i o I y almacenarlo en un archivo:

```
ls [iI]* > listado.txt
```

Es importante resaltar que el carácter de redirección de salida ``" destruirá el archivo al cual apunta, si este existe, para ser reemplazado por uno nuevo con los resultados del proceso. Si se desea anexar la información a uno ya existente debe usarse doble carácter ``" :

1.3.5 Tuberías o pipes

La filosofía de Linux/Unix es en extremo modular. Se prefieren las herramientas pequeñas con tareas puntuales a las meta-herramientas que realizan todo. Para hacer el modelo completo es necesario proveer el medio para ensamblar estas herramientas en estructuras mas complejas. Esto se realiza por medio del redireccionamiento de las entradas y las salidas.

Redirección de Entrada y Salidas fig_redirect width=5cmimages/standard_io.eps

Todos los programas tiene por por defecto una entrada estándar (teclado) y dos salidas: la salida estándar (pantalla) y la salida de error (pantalla). En ellos se puede sustituir la entrada y salidas estándar por otro dispositivo utilizando los caracteres ``" y ``", es decir, hacer que se lea un archivo que contenga las opciones a ejecutar y un archivo de salida, respectivamente. Por ejemplo:

Entrada:

Se desea realizar una transferencia de archivos por ftp automática. Para ello se va a utilizar el programa *ncftp* con unas determinadas instrucciones preestablecidas.

Se crea un archivo *entrada* con dichas instrucciones:

```
open linuxcol.uniandes.edu.co
cd /pub/linux/utils
get *
quit
```

y se ejecuta el programa: `ncftp < entrada` .

Salida:

Se quiere saber los archivos que empiezan por i o I y almacenarlo en un archivo:

```
ls [iI]* > listado.txt
```

Es importante resaltar que el carácter de redirección de salida ``" destruirá el archivo al cual apunta, si este existe, para ser reemplazado por uno nuevo con los resultados del proceso. Si se desea anexar la información a uno ya existente debe usarse doble carácter ``" :

1.3.6 Programación shell

La programación del shell es una de las herramientas mas apreciadas por todos los administradores y muchos usuarios de Linux/Unix ya que permite automatizar tareas complejas, comandos repetitivos y ejecutarlas con un solo llamado al *script* o hacerlo automáticamente a horas escogidas sin intervención de personas.

La programación *shell* en Unix/Linux es, en cierto sentido, equivalente a crear archivos .*BAT* en DOS. La diferencia es que en Unix/Linux es mucho mas potente. Estos *scripts* pueden usar un sinnúmero de herramientas como:

- Comandos del sistema Linux/Unix (ejm: ls, cut)
- Funciones intrínsecas del shell (ejm: kill, nice)
- Lenguaje de programación del shell (ejm: if/then/else/fi) (ver tabla de comandos)
- Programas y/o lenguajes de procesamiento en línea. (ejm: awk, sed, Perl)
- Programas propios del usuario escritos en cualquier lenguaje.

El lenguaje de programación de cada shell provee de una amplia gama de estructuras de control como se muestra en la tabla de commandos de la shell

- `for name [ in word; ] do list ; done`
- `select name [ in word ; ] do list ; done`
- `case word in [ pattern [ | pattern ]\ldots ) list ;; ]\ldots esac`
- `if list then list [ elif list then list ]\ldots [ else list ] fi`
- `$while list do list done`
- `$until list do list done`
- `[ function ] name () { list; }`

Instrucciones *bash* para programación *shell* tbl_instr_bash

Un sencillo ejemplo es realizar un backup de solo ciertos directorios (prog_dir1 y prog_dir2), luego comprimirlos usando bzip2 y enviarlos a un area de almacenamiento (digamos una unidad ZIP previamente montada en /mnt/zipdrive), y además con que el nombre del archivo contenga la fecha del día. Suena difícil? Realmente no lo es.

Se crea un archivo texto con cualquier nombre, por ejemplo *mibackup* que contenga las instrucciones que se desea ejecutar.

```
#!/bin/sh
#
echo "---------- Captura fecha -----------"
fecha=‘date +%Y%m%d‘
#
echo "---------- Haciendo Tar  -----------"
tar cvf backup$fecha.tar prog_dir1 prog_dir2
#
echo "----------- Comprimiendo -----------"
bzip2 backup$fecha.tar
#
echo "---------- Enviándolos a zip -------"
cp ./backup$fecha.tar /mnt/zipdrive
#
echo "----------- Limpiando --------------"
rm -f ./backup$fecha.tar
#
echo "----------- Final      -------------"
```

Luego, se le asigna permisos de ejecución con el comando

```
chmod +x mibackup
```

y esta listo para ser ejecutado.

En el *script* aquí mostrado es importante resaltar varios principios importantes: la primera línea determina el shell que se esta usando (sh o bash); las variables no se declaran solo se asignan; su valor es retornado usando el símbolo $.

Si desea mas información acerca de programación en shell se puede consultar los manuales en línea del shell: *bash* o *tcsh*

1.3.7 Re-utilización de comandos

El shell almacena una historia de los comandos que el usuario ha escrito. Por medio de esta historia es posible volver a ejecutar una orden que ya se ha escrito anteriormente sin tener que escribirla de nuevo.

El comando history muestra la secuencia de comandos, con un número a su izquierda. Con este número es posible llamar de nuevo el comando utilizando el carácter admiración ``!"; Por ejemplo history retorna

```
1  history
2  ls
3  cd public_html
4  ls
5  rm *.bak
6  history
```

y para ejecutar nuevamente el comando `rm *.bak` solo es necesario escribir `!5`. También se puede pedir el último ``rm'' que se ha ejecutado escribiendo `!rm`.

El último comando se repite con doble admiración ``!!''. Es posible también editar el último comando utilizando el carácter ``^'' pero este conocimiento se esta volviendo poco útil, ya que los nuevos shells permiten viajar por la ``historia'' y editar los comandos usando únicamente las flechas del teclado.

1.3.8 Archivos de bash

Cada shell posee ciertos archivos donde mantiene su configuración. Estos tiene una jerarquía que va desde el archivo general de configuración del sistema para todos los shells, pasando por el archivo propio del shell, hasta los archivos personales del usuario.

A continuación, en la tabla siguiente, se muestran los archivos utilizados para especificar opciones dentro de *bash*. Es importante aclarar que no es necesario que todos estos archivos existan dentro del directorio personal, el sistema posee su configuración por defecto.

Archivo	**Descripción**
/bin/bash	Ejecutable *bash*.
/etc/profile	Archivo de inicialización utilizado por los shells.
/.bash_profile	Archivo(s) de inicialización personal
/.profile	utilizado por los shells
/.bash_login	Ejecuta cuando entra al shell
/.bash_logout	Ejecuta cuando sale del shell
/.bashrc	Archivo personal de inicialización del shell.
/.inputrc	Archivo de inicialización individual.

Archivos de *bash*

2 Sistema de Archivos

2.1 Organización de los directorios

Linux organiza la información en archivos, los cuales están contenidos en directorios. Un directorio puede contener subdirectorios, teniendo así una estructura jerárquica, como en cualquier otro sistema operativo.

Las nuevas versiones de Linux (incluido Red Hat) siguen el estándar FSSTND (*Linux Filesystem Standard*) el cual estipula los nombres, la ubicación y la función de la mayoría de los directorios y los archivos del sistema. La siguiente tabla muestra la estructura básica del sistema de archivos de Linux como es definida por FSSTND.

Estructura de directorios de Linux según FSSTND fig_struct_dir width=10cmimages/struct_dir.eps

Conociendo esta estructura básica, el usuario/administrador podrá moverse más fácilmente por los directorios, ya que la mayoría de éstos, tienen un determinado uso. En la siguiente tabla se tiene la descripción de los directorios más comunes.

Directorio	**Descripción**
/	Raíz (*root*), forma la base del sistema de archivos.
/boot	Archivos del kernel (compilados).
/bin	Archivos ejecutables esenciales para todos los usuarios.
/dev	Archivos de dispositivos.
/etc	Archivos de configuración.
/etc/rc.d	Archivos de inicialización (Red Hat).
/home	Generalmente, directorios de los usuarios.
/home/ftp	Contenido del servidor FTP.
/home/httpd	Contenido del servidor WWW.
/lib	Librerías esenciales y módulos del kernel.
/mnt	Directorios donde ``montar'' diversos dispositivos temporalmente.
/mnt/cdrom	Directorio donde se ``monta'' el CD-ROM.
/mnt/floppy	Directorio donde se ``monta'' el disquete.
/proc	Información sobre partes del sistema.
/root	Directorio del usuario principal del sistema.

/sbin	Archivos ejecutables para tareas de administración.
/tmp	Temporal.
/usr	Programas, documentación, fuentes,...compartidos por todo el sistema
/var	Archivos variables del sistema, bitácoras, temporales,...

Directorios de Linux más frecuentes

La tabla muestra los principales subdirectorios del directorio *usr*.

Directorio	**Descripción**
/usr/X11R6	Paquete XFree86 (X-Windows) Release 6.
/usr/bin	Archivos ejecutables para usuarios.
/usr/dict	Listados de palabras (diccionarios).
/usr/doc	Documentación.
/usr/doc/FAQ	F.A.Q. (o P.U.F.).
/usr/doc/HOWTO	HOWTO's.
/usr/etc	Archivos de configuración del sistema.
/usr/games	Juegos.
/usr/include	Archivos de encabezado.
/usr/info	Sistema de información GNU info.
/usr/lib	Librerías
/usr/local	Jerarquía de archivos locales.
/usr/man	Manuales.
/usr/sbin	Archivos ejecutables de administración no vitales.
/usr/share	Datos independientes de la arquitectura.
/usr/src	Código fuente.
/usr/src/linux	Código fuente del *kernel* de Linux.

Subdirectorios de usr

La tabla muestra los principales subdirectorios del directorio *var*.

Directorio	**Descripción**

/var/catman	Manuales formateados localmente.
/var/lib	Información del estado de aplicaciones.
/var/local	Variables del aplicaciones en /usr/local.
/var/lock	Archivos de cerrojo.
/var/log	Bitácoras del sistema.
/var/named	Archivos del DNS.
/var/nis	Base de datos para NIS (Network Inf. Service).
/var/preserve	Archivos de respaldo después de una caída para vi o ex.
/var/run	Archivos relevantes a programas corriendo.
/var/spool	Colas de trabajos para realizar mas tarde.
/var/spool/at	Archivos creados por comando *at*.
/var/spool/cron	Archivos creados por comando *crontab*.
/var/spool/lpd	Archivos de impresora.
/var/spool/mail	Archivos de correo de cada usuario.
/var/spool/mqueue	Archivos de correo de salida.
/var/spool/news	Archivos de noticias de salida.
/var/spool/smail	Archivos de correo de *smail*.
/var/tmp	Temporal.

Subdirectorios de var

2.2 Permisos

Linux, como cualquier sistema Unix, es multiusuario, por lo que, los permisos de los archivos están orientados a dicho sistema. Los permisos de cualquier archivo tienen tres partes: permisos del propietario, permisos del grupo y permisos del resto. Así, se ve que un archivo pertenece a un determinado propietario y a un determinado grupo y, dependiendo de los permisos que tenga asociado dicho archivo, se podrá tener acceso a él o no.

Los permisos son de lectura (r), escritura (w) y ejecución (x). Estos se definen mediante letras (parecido al comando *attrib* de DOS). Con el signo ``" añadimos un permiso y con el signo ``" se lo quitamos. Además, los permisos pueden ser generales o no, es decir, si se pone sería permiso de ejecución a todos, sin embargo, si se pone sólo el propietario podría ejecutarlo. De este modo, se tiene: para propietario, para grupo y para el resto.

Ejemplo: Se tiene una serie de archivos después de listar con `ls -l` el cual muestra el propietario (skina), el grupo (users) y los permisos de izquierda a derecha: propietario, grupo y el resto.

```
[]$ ls -l
-rw-r--r--   1 skina  users   17680 Nov 29 16:52 GNU-GPL.tex
-rw-r--r--   1 skina  users    2573 Nov 30 19:52 Makefile
-rw-r--r--   1 skina  users    1562 Nov 29 13:47 autorizacion.txt
-rw-r--r--   1 skina  users     461 Oct 24 21:43 biblio.tex
drwxr-xr-x   2 skina  users    1024 Nov 23 01:02 bin/
-rw-r--r--   1 skina  users     949 Nov 30 19:26 creditos.tex
-rw-r--r--   1 skina  users     550 Nov 30 19:48 curso_linux.tex
drwxr-xr-x   2 skina  users    3072 Nov 30 22:55 images/
drwxr-xr-x   3 skina  users    1024 Nov 30 00:09 install/
-rw-r--r--   1 skina  users   61566 Oct 26 22:29 lista_paquetes_mdk.tex
-rw-r--r--   1 skina  users   53227 Nov 23 01:08 lista_paquetes_rh.tex
-rw-r--r--   1 skina  users    3864 Nov 30 19:56 parte1.tex
parte3_sistemas_archivos.tex
[]$
```

Estos permisos llevan ``asociado'' un número: el cuatro para lectura, el dos para la escritura y el uno para la ejecución. De esta manera, un archivo tiene tres números asignados: propietario, grupo y resto. Por ejemplo, si tenemos un fichero con los permisos 644 correspondería a: el propietario puede leer/escribir, el grupo sólo puede leer y el resto sólo puede leer. Vemos que, para asignar lectura y escritura, basta con sumar lectura(4)+escritura(2).

El comando para modificar los permisos es `chmod` y tiene la siguiente sintaxis: `chmod permisos archivo(s)`. Por ejemplo se desea que todos las personas puedan ver escribir sobre el archivo creditos.tex, entonces

```
# chmod a+w creditos.tex
```
o su equivalente en números
```
# chmod 666 creditos.tex
```

2.3 Montaje de un sistema de archivos

Ya se ha visto que Linux accede a los dispositivos mediante archivos (directorios de */dev*), y, por este motivo, en Linux no hay el concepto de unidades, ya que todo está bajo el directorio principal . En Linux no se accede a la primera disquetera mediante la orden *A:* como en DOS sino que hay que ``montarla''.

De este modo, tenemos dos conceptos nuevos:

‘‘montar’’
: Decirle a Linux que se va a utilizar un determinado dispositivo con un determinado sistema de archivos y estará en un directorio específicado. En la siguiente tabla se muestran los sistemas de archivos más comunes en Linux.

Tipo	Descripción
ext2	Sistema de archivos de Linux.
msdos	Sistema de archivos de DOS.
vfat	Sistema de archivos de Windows 9X (nombres largos).
iso9660	Sistema de archivos de CD-ROM.
nfs	Sistema de archivos compartido por red (‘‘exportado”).

Sistemas de archivos más comunes

‘‘desmontar”

Decirle a Linux que se ha dejado de utilizar un determinado dispositivo.

Para ‘‘montar” un determinado sistema de archivos de un dispositivo, se utiliza el comando *mount*. La sintaxis es la siguiente:

```
# mount -t sistema_archivos dispositivo directorio [-o opciones]
```

donde: *sistema_archivos* puede ser cualquiera de los que aparece en la tabla anterior; *dispositivo* puede ser cualquier dispositivo del directorio */dev* o, en el caso de nfs, un directorio de otro ordenador; *directorio* es el directorio donde estará el contenido del dispositivo y *opciones* pueden ser cualquiera de la tabla siguiente, en el caso de no poner ninguna opción, `mount` utilizará las opciones por defecto. Una vez ``montado" el dispositivo, si no se va a volver utilizar se puede ``desmontarlo" con el comando *umount* con la siguiente sintaxis:

```
# umount directorio
```

Siempre, después de utilizar un dispositivo hay que ``desmontarlo", para que se almacenen correctamente los datos en dicho dispositivo. Un ejemplo de ello, es el hecho de que, un lector de CD-ROM, que haya sido ``montado", no se abrirá hasta que no se ``desmonte".

Opción	Descripción
rw	Lectura/escritura.
ro	Sólo lectura.
exec	Se permite ejecución.
user	Los usuarios pueden ``montar"/``desmontar".
suid	Tiene efecto los identificadores de propietario y del grupo.
auto	Se puede montar automáticamente.

async	Modo asíncrono.
sync	Modo síncrono.
dev	Supone que es un dispositivo de caracteres o bloques.

Opciones del comando *mount*

Se muestran unos cuantos ejemplos:

1. Disquete de DOS:
   ```
   mount -t msdos /dev/fd0 /mnt/floppy -o rw,noexec
   umount /mnt/floppy
   ```
2. Disquete de Windows 9X:
   ```
   mount -t vfat /dev/fd0 /mnt/floppy -o user,rw
   umount /mnt/floppy
   ```
3. CD-ROM:
   ```
   mount -t iso9660 /dev/cdrom /mnt/cdrom -o ro
   umount /mnt/cdrom
   ```
4. Directorio exportado de host2:
   ```
   mount -t nfs host2:/tmp /mnt/host2
   umount /mnt/host2
   ```

2.4 /etc/fstab

En ocasiones, cuando se tienen varios dispositivos que se suelen ``montar'', se puede ahorrar tener que escribir continuamente la oportuna orden *mount*, simplemente incluyendo una línea en el archivo */etc/fstab*.

El archivo */etc/fstab* contiene líneas donde se indica qué dispositivo debe ``montar'', el lugar donde ``montarlo'', así como el sistema de archivos y las opciones (en este archivo, se pueden poner dos opciones más: *auto* y *noauto*, que indican si se debe ``montar'' automáticamente al arrancar el sistema o no, respectivamente).

Un ejemplo de */etc/fstab* puede ser:

```
# Dispositivo  Directorio     FS       Opciones

/dev/hda1      /              ext2     defaults    1 1
/dev/hda2      /home          ext2     defaults    1 2
/dev/hda3      /tmp           ext2     defaults,noexec
/dev/hda4      none           swap     defaults

none           /proc          proc     defaults

/dev/fd0       /mnt/floppy    ext2     noauto,user,noexec,rw
/dev/fd0       /mnt/msdos     vfat     noauto,user,noexec,rw
/dev/cdrom     /mnt/cdrom     iso9660  noauto,user,noexec,ro
```

```
/dev/sda4      /mnt/iomegazip vfat     noauto,user,noexec,rw

host2:/tmp     /mnt/host2     nfs      defaults
```

Con un archivo */etc/fstab* como el anterior, cualquier usuario podría hacer:

```
$ mount /mnt/msdos+
$ umount /mnt/msdos+
```

para ``montar'' y ``desmontar'' un disquete, respectivamente. Sin embargo, sólo el administrador podría ``montar'' y ``desmontar'' el directorio */mnt/host2*

2.5 Uso de mtools

El hecho de tener que ``montar'' y ``desmontar'' puede ser un poco engorroso a la hora de utilizar determinados dispositivos (comúnmente, la disquetera). Por ello, se dispone de las herramientas *mtools* (ver tabla siguiente). Dichas herramientas, utilizan los dispositivos sin tener que ``montar'' y ``desmontar''; y su sintaxis es parecida a la de los programas de DOS.

Comando	**Descripción**
mdir	Muestra el contenido del dispositivo *dir*.
mcopy	Copia archivos *copy*.
mdel	Borra archivos *del*.
mformat	Formatea la unidad *format*.
mcd	Cambia de directorio *cd*.
mmd	Crea un directorio *md*.
mrd	Borra un directorio *rd*.

Herramientas *mtools*

3 El compilador GCC

(obtenido de: http://iie.fing.edu.uy/~vagonbar/gcc-make/gcc.htm)