Capítulo 2

El sistema de ficheros ext2

Un sistema operativo necesita para seguir la pista de los ficheros que se encuentran en el disco o dispositivo de almacenamiento masivo un modo de organización. Éste es el sistema de ficheros. GNU/Linux proporciona distintos sistemas de ficheros tales como `ext2`, `msdos`, `minix`, etc. En este capítulo vamos a profundizar en el estudio del sistema `ext2`, que es el nativo de GNU/Linux.

2.1. Conceptos básicos

El sistema de ficheros `ext2` consiste en un conjunto de ficheros. Cada fichero puede tener uno o más nombres. Hay tres clases de ficheros:

Ficheros ordinarios Contienen datos.

Ficheros especiales Proporcionan acceso a dispositivos de E/S.

Directorios Contienen información acerca de conjuntos de ficheros y se utilizan para localizar un fichero a partir de su nombre.

Como la mayoría de los sistemas operativos modernos, GNU/Linux organiza su sistema de ficheros como una jerarquía de directorios. La jerarquía de directorios se suele denominar **árbol de directorios**. Un directorio especial, llamado **directorio raíz**, está situado en la parte más alta de la jerarquía. Más adelante en este capítulo veremos órdenes para navegar por el árbol de directorios y modificarlo.

Un directorio puede incluir tanto ficheros como otros directorios (subdirectorios). Cuando se muestra el contenido de un directorio aparecen tanto los ficheros como los directorios.

2.1.1. Identificadores de ficheros

Un identificador de fichero le da un nombre dentro de un directorio. El número máximo de caracteres que puede tener un identificador de fichero depende del sistema concreto, en `ext2` puede ser hasta 255. Un identificador de fichero puede contener cualquier carácter distinto de `/`, aunque algunos caracteres tales como `-` y espacio en blanco pueden dar problemas debido al significado especial que tienen dentro de la línea de órdenes. Las letras (tanto mayúsculas como minúsculas del alfabeto inglés), los dígitos, y los caracteres especiales `+`, `=`, `_` y `:`, son normalmente seguros. Por convenio, el carácter '`.`' al principio de un identificador de fichero indica que se trata de un fichero de inicialización o soporte para un programa particular. Al listar el contenido de un directorio no suelen aparecer estos ficheros, a menos que se indique explícitamente. También son tratados de forma especial durante la expansión de comodines, que se verá más adelante.

Los identificadores de ficheros distinguen entre mayúsculas y minúsculas; por ejemplo, `programa`, `Programa` y `PROGRAMA` son tres identificadores diferentes.

2.1.2. Nombres de ficheros

Un nombre o camino de fichero consta de una secuencia de identificadores de ficheros separados por el carácter `/`. Los identificadores de ficheros son los componentes del nombre del fichero. Hay dos clases de nombres de ficheros: **absolutos** y **relativos**. Un nombre de fichero absoluto comienza por el carácter `/`, uno relativo no.

Nos podemos referir a un fichero que está en cualquier lugar del árbol de directorios dando su nombre o camino absoluto. El nombre absoluto especifica la secuencia de subdirectorios que debemos atravesar para ir desde el directorio raíz hasta el fichero. El carácter `/` que siempre da comienzo a un nombre absoluto designa al directorio raíz. Por ejemplo `/home/alum/pepe/copias/back1` es un nombre absoluto.

Cada proceso lleva asociado un directorio llamado **directorio de trabajo** o **actual**, que puede servir como punto de partida para los nombres de ficheros. Un nombre de fichero que no empieza por `/` se denomina nombre relativo y se toma de forma relativa al directorio de trabajo. Si estamos trabajando en el directorio `/home/alum/pepe`, el nombre relativo del fichero anterior será `copias/back1`.

El caso más simple y más común de un nombre relativo es el de un solo identificador de fichero usado como nombre de fichero. Este nombre de fichero se refiere a uno que está en el directorio de trabajo.

El padre de un directorio (distinto del raíz) es el que está justo por encima de él en la jerarquía. Existe una notación para designar al directorio de trabajo y a su directorio padre. Al primero se le denomina '`.`', y a su padre '`..`'. Por ejemplo, si el directorio de trabajo es `/home/alum/pepe`, '`..`' se refiere a `/home/alum` y '`../..`' se refiere a `/home`. El directorio raíz es un caso especial en el que su padre es él mismo.

Cada usuario tiene un **directorio de casa**, en el que es situado automáticamente al iniciar su sesión. El nombre de este directorio varía dependiendo del sistema. Los ficheros creados por un usuario se almacenarán a partir de este directorio.

Ejercicios:

1. Diga si los siguientes nombres de fcheros son absolutos o relativos: `/tmp`, `temario`, `~/practicas/so1`, `/`, `alumnos/juan`.
2. La orden `dirname` quita del nombre de un fichero el sufijo que no sea parte del nombre del directorio. Es decir, da el camino sin el identificador del fichero. ¿Qué resultado proporcionará la orden `dirname /usr/mrm/datos.Z`? ¿Y `dirname ~`? Razone las respuestas.
3. La orden `basename` permite extraer el identificador del fichero de un camino. ¿Qué puede decir del nombre (relativo o absoluto) que se le pasa a la orden y el resultado que proporciona? Indíquelo sobre un ejemplo.

2.1.3. Cómo almacena `ext2` los ficheros

El sistema `ext2` almacena los ficheros en bloques de disco que pueden estar dispersos, por lo que el sistema necesita disponer de algún mecanismo que le permita saber qué bloques de disco son los que forman parte de un fichero. Esto lo consigue mediante los **nodos índice** o **nodos-i**.

Entre la información que se guarda en el nodo índice está: qué bloques de disco componen el fichero, su tamaño, propietario, permisos, número de enlaces, fecha de creación, etc. Cada nodo-i viene identificado por un **número-i**.

Los directorios son ficheros que catalogan otros ficheros. La estructura abreviada de una entrada de un directorio en `ext2` es la siguiente:

ident-fichero	número-i

El número de bloques de disco que se dedican a almacenar nodos índice está limitado, por tanto también lo está el número de ficheros que se pueden

crear. Por esta razón es conveniente limitar para cada usuario el número máximo de ficheros que puede crear. Los sistemas GNU/Linux permiten establecer una cuota para cada usuario que limita tanto el número de ficheros como el número de bloques de datos que éste puede utilizar. La orden `quota` nos muestra estos valores.

***Shell* bash**

Expansiones El *shell* es una interfaz entre el usuario y el sistema operativo GNU/Linux, es decir, se encarga de traducir las líneas de órdenes que introduce el usuario en instrucciones que pueda entender el sistema operativo. Para ello realiza una serie de acciones sobre la línea de órdenes.

1. Parte la línea en palabras.
2. Realiza expansiones.
3. Determina el tipo de cada palabra: orden, argumento, etc.
4. Envía a ejecutar las órdenes.

El *shell* bash realiza distintos tipos de expansiones, entre ellas la expansión de nombres de ficheros, la expansión del carácter ~ y la expansión de llaves.

Expansión de nombres de ficheros

A veces necesitamos ejecutar una orden sobre más de un fichero, podemos hacer esto de una forma fácil haciendo uso de caracteres especiales o comodines. Estos caracteres especiales tienen un significado especial para el *shell* bash.

Los caracteres especiales que reconoce el *shell* bash aparecen en el cuadro 2.1.

Lo dicho anteriormente es válido para todos los caracteres excepto / y '.' cuando va al principio de un identificador de fichero, que no son sustituidos por los comodines.

A continuación damos algunos ejemplos de comodines y los ficheros a los que se refieren:

Los caracteres comodines son interpretados por el *shell*; cuando ve un nombre de fichero que contiene comodines, lo traduce a una secuencia de nombres de ficheros específicos. Por ejemplo, si a una orden de GNU/Linux se le pasa como parámetro el nombre de fichero `quijote*`, y en el directorio actual existen los ficheros `quijote1`, `quijote2` y `quijote3`, lo que recibirá la orden en cuestión es la lista formada por los tres nombres, separados por el carácter espacio.

Expansión de llaves

Carácter	Significado
*	Concuerda con cualquier conjunto de cero o más caracteres. Un * sustituye a los nombres de todos los ficheros excepto aquellos que comienzan por el carácter punto (.). El punto debe de indicarse de modo explícito.
?	Concuerda exactamente con cualquier carácter simple.
[]	La construcción [*caracteres*] sustituye a cualquier carácter simple en el conjunto *caracteres*. Este conjunto se puede escribir como una secuencia o como pares de caracteres. Un par de caracteres tiene la forma c1-c2; y denota los caracteres comprendidos entre c1 y c2 en el código de caracteres de la máquina. Si ponemos ! delante de la secuencia, esto denota a todos los caracteres que no están en ella.

Cuadro 2.1: Caracteres comodines del *shell* bash

Nombre	Ficheros
gn*.1	gnu.1, gn.1, gname.1; no concuerda con gn/x.1
~/.[a-zA-Z]*	~/.profile, ~/.mailrc
/num	uno/num1, dos/num1.c, dos/num.h; pero no con num
zz?	zz1, zza; pero no con zz12
[!0-9]*	A1a, Maria, Jose; pero no con 1barco, ni 123
*.[acAC]	fichero.a, fichero.c; pero no con .a

La expansión de llaves es un mecanismo por el que se generan cadenas arbitrarias. Este mecanismo es similar a la expansión de nombres de ficheros, pero los nombres de los ficheros generados no tienen que existir. Los patrones a expandir tienen la siguiente forma: un prefijo opcional, seguido por una serie de cadenas separadas por comas y encerradas entre llaves, seguido por un sufijo que también es opcional. El prefijo se antepone a cada cadena contenida dentro de las llaves, y el sufijo se añade al final de cada cadena resultante, expandiéndose de izquierda a derecha.

Ejemplos:

1. `a{d,c,b}e`
   Se expande a ade ace abe.
2. `/usr/local/src/bash/{old,new,dist,bugs}`
   Esta línea se expande a:
   `/usr/local/src/bash/old,`

```
/usr/local/src/bash/new,
/usr/local/src/bash/dist y
/usr/local/src/bash/bugs.
```

Expansión del carácter ~

Cuando le damos al *shell* bash el carácter ~ aislado en un línea de órdenes, éste lo expande al directorio de entrada del usuario que da la orden.

Si el carácter ~ va seguido de un nombre de usuario, el *shell* bash lo expande al directorio de entrada del usuario especificado.

Si el carácter ~ va seguido del carácter +, el *shell* bash lo expande al directorio de trabajo.

Si el carácter ~ va seguido del carácter -, el *shell* bash lo expande al directorio de trabajo previo.

2.2. Visión del sistema de ficheros

Los sistemas de ficheros residen normalmente en particiones de disco duro, aunque pueden estar en cualquier dispositivo de almacenamiento. Cada partición contiene un único sistema de ficheros.

El usuario ve una única jerarquía de ficheros y directorios, con un único directorio raíz. Realmente esa jerarquía está formada por uno o varios sistemas de ficheros que pueden residir en dispositivos diferentes y cada uno comienza en su propio directorio raíz. Esto se consigue de la siguiente manera:

- Existe un único sistema de ficheros raíz que se monta[1] cuando el sistema arranca.
- Los demás sistemas de ficheros se montan en directorios que deben existir previamente y se denominan **puntos de montaje**. Este es el único modo que tiene el usuario para acceder a los ficheros que hay en una partición de disco duro, CD-ROM, disco flexible, cinta, etc.

De este modo, el usuario ve un único sistema de ficheros, pero éste está realmente compuesto por varios sistemas de ficheros físicos, cada uno de ellos

[1] **Montar** consiste en integrar en la jerarquía de directorios.

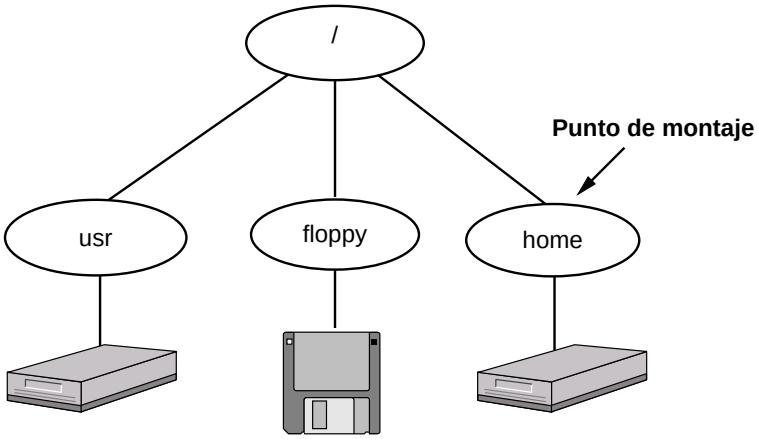


Figura 2.1: Jerarquía de directorios y ficheros

situado en un dispositivo o partición diferente. La figura 2.1 muestra esta situación.

2.2.1. Montaje de un sistema de ficheros

Para que los ficheros estén disponibles para los usuarios es necesario montar el sistema de ficheros en la jerarquía de directorios. Si el punto de montaje elegido tiene ya un contenido, éste queda oculto hasta que se desmonte el sistema de ficheros.

Cuando el sistema operativo arranca monta automáticamente el sistema de ficheros raíz. Si queremos tener disponibles otros sistemas de ficheros los tendremos que montar manualmente o bien indicarle que nos los monte durante el arranque. Esto último lo conseguimos mediante el fichero `/etc/fstab`.

Este fichero proporciona información al sistema sobre el procedimiento de montaje de diferentes sistemas de ficheros, indicándole el dispositivo donde se encuentran, el punto de montaje, el tipo de sistema de ficheros, así como opciones de montaje, copias de seguridad y verificación. Cada línea del fichero posee el siguiente formato:

```
dispositivo punto_de_montaje tipo opciones copia ver
```

Cuando en las opciones aparezca `auto` o `defaults` se montará automáticamente al arrancar el sistema.

El montaje manual se realiza mediante la orden `mount`, que está reservada generalmente al administrador del sistema. Sin embargo, un usuario puede hacer uso de ella bajo ciertas condiciones. Un usuario podrá montar un sistema de ficheros que esté especificado en el fichero `/etc/fstab` cuando en el campo opciones aparezca `users`. Para ello dará la orden:

`mount` *punto de montaje*

donde el punto de montaje es el indicado en el fichero `/etc/fstab`.

Cuando se para el sistema se desmontan todos los sistemas de ficheros. Si hemos montado un sistema de ficheros que reside en un disquete o en un CD-ROM y queremos sacar éste, habremos de desmontarlo previamente. Este es un paso importante, porque en el caso del disquete si no se hace se podría perder información. Para desmontar un sistema de ficheros disponemos de la orden `umount`, cuyo formato es:

`umount` *punto de montaje*

2.2.2. Jerarquía típica de un sistema GNU/Linux

Un sistema GNU/Linux contiene un conjunto de fiheros y directorios estándar, organizados según la Jerarquía estándar del sistema de ficheros (FSH, *Filesystem Hierarchy Standard*), propuesta en 1994, que forma parte del poyecto Linux *Standard Base* (LSB). Según este existen una serie de directorios estándares con el siguiente contenido:

`/` Éste es el directorio raíz. Aquí comienza todo el árbol de directorios.

`/boot` Contiene los ficheros necesarios para el proceso de arranque.

`/dev` Contiene ficheros especiales correspondientes a dispositivos.

`/etc` Contiene los ficheros de configuración del sistema y de todas las aplicaciones instaladas.

`/home` Contiene los directorios de casa de los usuarios.

`/media` Directorio que contiene los distintos puntos de montaje de sistemas de ficheros en dispositivos tales como disquetera, CD-ROM, etc.

`/mnt` Lo utiliza el administrador como punto de montaje temporal.

`/lib` Contiene bibliotecas de código objeto frecuentemente usadas, incluyendo las bibliotecas dinámicas.

`/opt` Sirve para instalar paquetes de software adicionales, que deben estar ubicados en `/opt/nombre-paquete/bin`.

`/proc` Contiene información sobre el sistema y los procesos que se están ejecutando actualmente.

`/sbin` y `/usr/sbin` Contiene programas de mantenimiento del sistema que sólo debe usar el administrador.

`/bin` y `/usr/bin` Contienen los ficheros binarios del sistema.

`/tmp` Contiene ficheros temporales generados por el sistema y por aplicaciones.

`/usr/X11R6` El sistema X Window.

`/usr/bin/X11` Es el lugar tradicional donde se colocan los ejecutables X11; en GNU/Linux, normalmente es un enlace simbólico al directorio `/usr/X11R6/bin`.

`/usr/local` Aquí se suelen colocar los programas que son locales.

`/usr/man` Aquí se sitúan las páginas del Manual de Referencia.

`/usr/src` Ficheros fuente de distintas partes del sistema operativo.

`/var/log` Suele contener ficheros de registro, que son importantes para el administrador del sistema.

`/var/spool` Se sitúan los ficheros creados por programas que envían trabajos a colas.

`/var/tmp` Una lugar alternativo para colocar ficheros temporales.

Una descripción más detallada de la jerarquía de directorios la puede obtener visualizando la página del manual correspondiente a `hier`.

2.3. Creación de un sistema de ficheros

Antes de poder utilizar un dispositivo como un sistema de ficheros, es necesario inicializarlo, y crear las estructuras de datos necesarias. Los pasos a seguir dependen del tipo de dispositivo. Si se trata de un disco duro es

necesario crear las particiones, y posteriormente el sistema de ficheros en cada una. Si se trata de un disquete, habrá que formatearlo a bajo nivel y posteriormente crear el sistema de ficheros.

Para particionar el disco duro se utilizan las órdenes `fdisk` o `cfdisk`.

Para formatear a bajo nivel un disquete se usa la orden `fdformat`. Su formato es:

`fdformat` *dispositivo*

donde *dispositivo* indica la unidad donde se encuentra el disquete. Así, `/dev/fd0` es la unidad A, `/dev/fd1` la unidad B, y así sucesivamente.

Para crear un sistema de ficheros se dispone de la orden `mkfs`, cuyo formato es:

`mkfs`[`-t` *tipo*][*opciones_del_SF*] *dispositivo* [*bloques*]

Opciones:

`-t` ***tipo*** Especifica el tipo del sistema de ficheros. Si no se especifica crea uno de tipo `ext2`.

bloques El tamaño en bloques del sistema de ficheros. Si no se especifica intenta detectarlo, aunque algunos tipos de sistemas de ficheros requieren este parámetro.

opciones_del_SF Son las opciones que se pasan al constructor del sistema de ficheros. Puede consultar el manual para ver las distintas opciones que posee.

Ejemplo:

Como resumen de todo lo visto anteriormente vamos a describir los pasos que tendría que dar un usuario para preparar un disquete para su uso con un sistema de ficheros `ext2`.

1. Formatear el disquete a bajo nivel.

   ```
   $ fdformat /dev/fd0
   ```

2. Crear un sistema de ficheros `ext2`.

   ```
   $ mkfs -t ext2 /dev/fd0
   ```

3. Montar el disquete para su utilización, sabiendo que en el fichero `/etc/fstab` aparece el punto de montaje `/floppy` para que el usuario pueda usar la disquetera.

   ```
   $ mount /floppy
   ```

4. Desmontar el sistema de ficheros para sacar el disquete.

   ```
   $ umount /floppy
   ```

2.3.1. Creación de un sistema de ficheros FAT

Aunque el capítulo se dedica al estudio del sistema de ficheros `ext2`, es conveniente conocer cómo se puede crear un disquete con sistema de ficheros FAT, el sistema nativo de Windows. Para ello, se formatea con la orden `mformat`, cuyo formato es:

`mformat` [*opciones*] *disquetera*

donde *disquetera* hace referencia a la unidad de disquetes, `a:` o `b:`. Esta orden además de formatear el disquete crea de forma automática un sistema de ficheros FAT, por lo que el disquete estará listo para ser usado.

Mediante las *opciones* se pueden especificar aspectos concretos de la estructura física del disquete.

2.4. Creación de ficheros

Antes de ver qué operaciones podemos hacer sobre los ficheros vamos a aprender a crearlos. Podemos crear ficheros de distintas formas:

Mediante un editor de texto Si queremos crear un fichero que contenga un texto determinado: una carta a un amigo, el código de un programa, los apuntes de clase, etc., utilizaremos un editor.

Con la orden `touch` Cuando damos la orden `touch` seguida del nombre de un fichero que no existe, se nos crea este fichero vacío. Si el fichero ya existe, `touch` nos cambia la hora de acceso y modificación a la actual o a la que se le diga.

Ejemplo:

```
$ touch hola
```

Crea un fichero vacío llamado `hola` *(si no existía previamente).*

Mediante la redirección de la salida estándar Más adelante veremos con mayor detenimiento cómo se redirige la salida estándar de una orden. Aquí nos basta con decir que podemos hacerlo utilizando el metacarácter `>`.

Ejemplos:

1. `$ ls > lista`
   Nos creará un fichero llamado `lista` *que contendrá la salida de la orden* `ls`.
2. `$ > hola`
   Crea un fichero vacío llamado `hola`.

2.5. Operaciones con directorios

2.5.1. Creación de directorios

La orden `mkdir` crea uno o más directorios nuevos. Su formato es

`mkdir` [*opciones*] *directorio* ...

Opciones:

-p Crea los directorios padres intermedios de cada argumento, si fuera necesario.

Ejemplos:

```
$ mkdir docs docs/texto docs/figuras
$ mkdir -p docs/texto docs/figuras
```

Ambas órdenes crean el directorio `docs`*, y los directorios* `texto` *y* `figuras` *catalogados en él.*

2.5.2. Cambio de directorio

La orden `cd` cambia el directorio de trabajo. La forma de la línea de órdenes es

`cd` [*directorio*]

donde *directorio* es el camino del nuevo directorio de trabajo. Si se omite *directorio*, se nos sitúa en el directorio de entrada. Ésta es una forma rápida de volver al directorio de casa. La mayoría de los *shells* admiten especificar - como argumento, esto nos hace volver al directorio anterior.

Ejemplos:

1. `$ cd docs/figuras`
   Nuestro nuevo directorio de trabajo es `docs/figuras`.
2. `$ cd -`
   Cambiamos al directorio de trabajo previo (es equivalente a `cd ~-`*).*
3. `$ cd`
   Nuestro directorio de trabajo vuelve a ser el directorio de entrada.

2.5.3. Visualización del camino del directorio de trabajo

La orden `pwd` envía el camino completo del directorio actual (o directorio de trabajo) a la salida estándar. Su formato es:

`pwd`

2.5.4. Visualización del contenido de un directorio

La orden `ls` lista un conjunto de ficheros, el contenido de un directorio, los contenidos de un árbol de directorios, o cualquier combinación de ellos. Se puede usar para ver si un fichero existe o para examinar sus características. Su formato es el siguiente:

`ls` [*opciones*] [*nombre*] ...

Se pueden especificar uno o varios *nombres* que pueden ser ficheros o directorios; los nombres sucesivos deben ir separados por blancos. Si no se da ningún nombre, toma el directorio actual. Los ficheros que empiezan por '.' siempre se omiten a no ser que se especifique la opción `-a` o `-A`.

La orden `ls` cuenta con muchas opciones sobre las cuales podrá encontrar información en su página del manual. Una de las opciones más importantes es `-l`, que proporciona el listado en formato largo.

Supongamos que damos la orden `ls -l`; obtendremos un listado de este tipo:

```
drwxrwxr-x   3  clio  musas  176 Mar 15 12:06  .
drwxrwxr-x  13  clio  musas  944 Feb 16 19:39  ..
drwxr-xr-x   2  clio  musas   48 Apr  4 13:01  d1
-rwxr-xr-x   1  clio  musas    9 Mar 18 11:23  texto
-rw-r--r--   1  clio  musas  124 May 12 10:15  memo
```

El listado se interpreta de la siguiente forma:

- El primer carácter de la línea indica el tipo de fichero:

  - Fichero ordinario

  d Directorio

  l Enlace simbólico

  c Dispositivo de caracteres

  b Dispositivo de bloques

- Las letras que le siguen indican los permisos asociados al fichero para distintos tipos de usuarios.

- El siguiente número indica el número de enlaces duros que tiene el fichero. Si se trata de un directorio, indica el número de entradas correspondientes a directorios catalogadas en el primero.

- Los dos elementos siguientes indican el usuario y el grupo a los que pertenece el fichero.

- A continuación aparece el tamaño del fichero en bytes.

- Luego viene la fecha y la hora en que el fichero fue modificado por última vez.

- Por último aparece el nombre del fichero.

Ejemplos:

1. `$ ls -l /`
   Se obtiene un listado en formato largo del directorio raíz.
2. `$ ls -la .`
   Se obtiene un listado en formato largo de los ficheros del directorio de trabajo, incluyendo los que empiezan por el carácter '.'.
3. `$ ls *.tex`
   Si en el directorio de trabajo existen ficheros cuyo identificador termina en '.tex', muestra los identificadores.

En la página del manual correspondiente a esta orden encontrará más ejemplos de uso de la misma.

Ejercicios:

1. Obtenga un listado de su directorio de entrada. Obtenga el mismo listado anterior donde aparezcan todos los ficheros que hay actualmente en su directorio de entrada (incluyendo los que empiezan por `.`)
2. ¿Qué línea de órdenes daría para obtener el nombre del directorio donde usted se encuentra actualmente?
3. ¿Cómo podría saber los permisos que lleva asociado su fichero `.profile`? ¿Y si además quiere saber su número de nodo-i?

2.6. Visualización de ficheros

2.6.1. Paginadores

Un paginador es un programa que permite visualizar el contenido de un fichero o la salida de un mandato, página a página. Un sistema puede disponer de varios paginadores tales como `more`, `less`, etc.

Los paginadores suelen tener órdenes internas que nos permiten hacer búsquedas dentro de un fichero, movernos página a página o por unidades mayores, ir hacia atrás en el fichero, etc. De todos los paginadores que tenemos en nuestro sistema el más completo es `less`. Dispone de una ayuda incorporada que se obtiene con la orden interna `h`.

Para conocer las órdenes y las posibilidades que proporciona cada paginador puede consultar las páginas correspondientes del manual.

2.6.2. Concatenación de ficheros

La orden `cat` debe su nombre al hecho de que copia y concatena ficheros. Su formato es:

`cat` [*opciones*] [*fichero*] ...

Los ficheros se concatenan y se copian en la salida estándar. El resultado es una copia del primer fichero, seguida de una copia del segundo, etc. Si no se especifica ningún nombre de fichero tomará su entrada de la entrada estándar.

Las opciones disponibles para esta orden puede verlas en el manual.

Ejemplo:

```
$ cat /etc/passwd /etc/group
```

Muestra en la salida estándar los ficheros `/etc/passwd` *y* `/etc/group` *concatenados.*

2.6.3. Extracción del final de un fichero

Para extraer la última parte de un fichero podemos usar la orden `tail`. Su formato es:

`tail` [`-c` *núm* | `-n` *núm*] [*fichero* ...]

`tail` [`+`*núm unidad* | `-`*núm unidad*] [*fichero* ...]

La orden `tail` copia el contenido de *fichero* a la salida estándar, empezando en la posición especificada por los argumentos que preceden a *fichero*, que pueden ser los siguientes:

núm Un entero (por omisión 10). Puede ir precedido de un signo que indica desde dónde se empieza a contar (`+` cuenta desde el principio del fichero, `-` cuenta desde el final).

`-c`,`-n` Indicador de unidad (`-c` carácter, `-n` línea)

unidad Indicador de unidad, puede ser `l` (línea), `b` (bloques de 512 bytes), `c` (caracteres).

Por omisión se muestran las últimas diez líneas del fichero. Si se omite *fichero* se supone que la entrada proviene de la entrada estándar.

Ejemplos:

1. `$ tail /etc/passwd`
   Muestra las diez últimas líneas del fichero `/etc/passwd`.
2. `$ tail -n +5 /etc/group`
   Muestra a partir de la 5.ª línea del fichero `/etc/group`.
3. `$ tail -10c /etc/passwd`
   Muestra los diez últimos caracteres del fichero `/etc/passwd`.

Existe una orden relacionada llamada `head`, que muestra las primeras líneas o caracteres de un fichero; para más información consulte la página correspondiente del manual.

Ejercicios:

1. ¿Qué línea de órdenes daría para obtener en la salida estándar las 5 primeras líneas del fichero `/etc/passwd`? ¿Y si quisiéramos ver las 3 últimas? ¿Y para ver todo el fichero?

2.7. Operaciones con enlaces

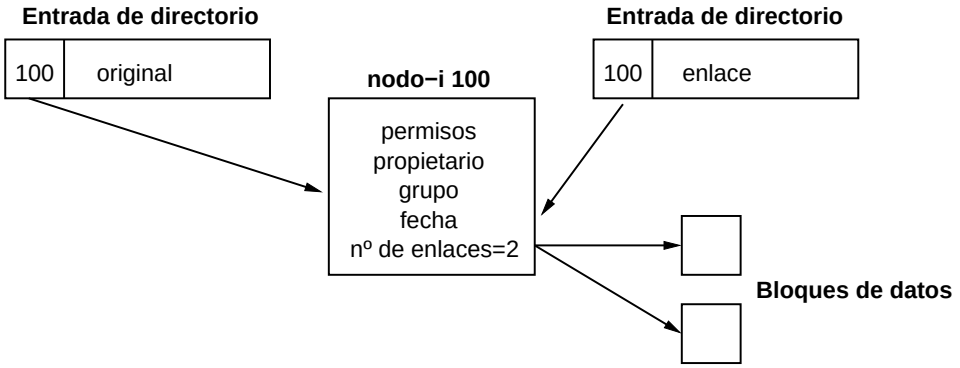
Una entrada de un directorio es un enlace a un fichero. En `ext2` puede haber múltiples enlaces a un mismo fichero, es decir, podemos tener un fichero con varios nombres. Todos los enlaces a un mismo fichero tienen el mismo número-i. Los enlaces pueden residir en directorios diferentes (figura 2.2 a).

Como sólo hay una copia del fichero, cualquier cambio que se haga en éste a través de cualquiera de sus enlaces es visible para todos los demás.

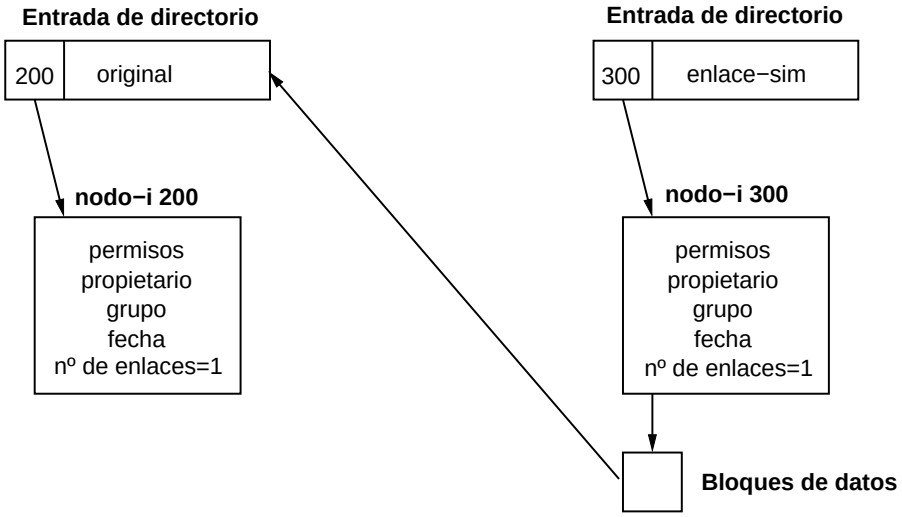
Estos enlaces se suelen denominar **enlaces duros** y presentan dos restricciones; no se pueden crear enlaces duros entre directorios, ni entre dos ficheros que estén en sistemas de ficheros diferentes. La razón para esta última restricción es que cada sistema de ficheros tiene su propio conjunto de nodos-i, por tanto, los números-i sólo son únicos dentro de un mismo sistema de ficheros.

El sistema `ext2` proporciona otra forma de enlace, el simbólico. Cuando se crea un **enlace simbólico** a un fichero se está creando un nuevo fichero, cuyo contenido es el camino del fichero enlazado. Si este camino tiene menos de 60 caracteres se guardará en el propio nodo-i; en caso contrario le asignará un bloque de datos. Así, cuando hacemos referencia al enlace simbólico el sistema averigua el camino del fichero al que realmente vamos a acceder (figura 2.2 b).

Los enlaces simbólicos no presentan las restricciones de los enlaces duros a la hora de crearlos. Se pueden establecer entre ficheros que estén en sistemas de ficheros diferentes, y también entre directorios, lo cual permite a los administradores de sistemas mover parte de la jerarquía de directorios de un sitio a otro de forma transparente para los usuarios.



a) Enlace duro



b) Enlace simbólico

Figura 2.2: Enlaces duros y simbólicos en GNU/Linux

Las órdenes `ln` (*link*), `mv` (*move*), `cp` (*copy*) y `rm` (*remove*) están estrechamente relacionadas. Todas ellas actúan sobre los enlaces.

- La orden `ln` crea un nuevo enlace a un fichero ya existente, lo que significa crear otro nombre para el fichero.
- Mover un fichero (`mv`) significa cambiar uno de sus nombres.

- Al copiar un fichero (`cp`) se replica su contenido con otro nombre.
- La orden `rm` borra un enlace a un fichero.

Estas órdenes, a diferencia de otras, no usan la entrada o la salida estándar de forma predeterminada.

2.7.1. Enlazar ficheros

La orden `ln` crea enlaces a uno o más ficheros existentes. Las posibles formas de usarla son:

`ln` [*opciones*] *fich_ fuente fich_ destino*

`ln` [*opciones*] *fich_ fuente ... dir_ destino*

En la primera forma se crea un enlace de *fich_ destino* a *fich_ fuente*. Después de haberse creado el nuevo enlace, tanto *fich_ fuente* como *fich_ destino* son nombres del mismo fichero.

En el segundo caso se crea un enlace en *dir_ destino* a cada fichero de la lista, dándole como nombre el mismo identificador actual.

Por ejemplo, suponga que `pedro` y `juan` son dos subdirectorios de su directorio actual y el directorio `pedro` contiene dos ficheros: `calc.c` y `calc`. Entonces la orden

```
ln pedro/* juan
```

construye los enlaces `juan/calc.c` y `juan/calc`. Después de haber ejecutado la orden `ln`, `pedro/calc.c` y `juan/calc.c` se refieren al mismo fichero y lo mismo ocurre con `pedro/calc` y `juan/calc`.

Cuando `ln` crea un enlace, el "nuevo" fichero tiene los mismos atributos que el fichero original. Esto lo podemos comprobar mediante la orden `ls -li`.

```
112 -rwxr-xr-- 2 luis luis 170 oct 18 14:11 juan/calc
112 -rwxr-xr-- 2 luis luis 170 oct 18 14:11 pedro/calc
100 -rwxr-xr-- 2 luis luis 170 oct 18 13:24 juan/calc.c
100 -rwxr-xr-- 2 luis luis 170 oct 18 13:24 pedro/calc.c
```

La orden `ln` proporciona múltiples opciones, algunas de las más importantes son:

`-s` Crea enlaces simbólicos. Si no se especifica, crea enlaces duros.

`-f` Si *fich_ destino* existe, fuerza su borrado.

Ejemplo:

```
$ ln -s ~juan/docs/ ~pedro/docs
```

Si el usuario juan tiene un directorio llamado `docs` *en su directorio de entrada, se crea un enlace simbólico que apunta a éste en el directorio de entrada del usuario pedro, que también se llama* `docs`.

2.7.2. Mover ficheros

Podemos pensar que la orden `mv` es una especie de renombrador de ficheros. Esta orden presenta tres formas de uso:

`mv` [*opciones*] *fich_ fuente fich_ destino*

`mv` [*opciones*] *fich_ fuente ... dir_ destino*

`mv` [*opciones*] *dir_ fuente ... dir_ destino*

En la primera forma, se crea un nuevo enlace a *fich_ fuente* que se va a llamar *fich_ destino* y se borra el enlace original; en otras palabras, tendremos el mismo fichero anterior pero con un nuevo nombre.

En la segunda forma, cada fichero de la lista es movido a *dir_ destino*. Los identificadores de los ficheros serán los originales. Veamos un ejemplo; supongamos que `pedro` y `juan` son subdirectorios de su directorio actual y `pedro` contiene los ficheros `calc.c` y `calc`. Entonces la orden

```
mv pedro/* juan
```

mueve `pedro/calc.c` a `juan/calc.c` y `pedro/calc` a `juan/calc`. Después de hacer esto el directorio `pedro` estará vacío y el directorio `juan` tendrá los dos ficheros ya nombrados.

La tercera forma permite mover directorios completos a otro directorio diferente, si este último existe previamente; si no existe, renombrará el primer directorio. Así por ejemplo, la línea de órdenes

```
mv dir1 dir2
```

mueve todo el contenido del directorio `dir1` al directorio `dir2`, si éste existe; si `dir2` no existía antes de dar la orden simplemente renombrará `dir1`.

2.7.3. Copiar ficheros

La orden `cp` copia uno o más ficheros. Al contrario de `ln` y `mv`, `cp` no se limita a hacer una reorganización de las entradas del directorio, sino que también copia los datos.

Las formas de uso de `cp` son:

`cp` [*opciones*] *fich_fuente fich_destino*

`cp` [*opciones*] *fich_fuente* ... *dir_destino*

`cp` [*opciones*] `-r` *fich_fuente* | *dir_fuente* ... *dir_destino*

En el primer caso se hace una copia de *fich_fuente* con el identificador *fich_destino*, es decir, tendremos dos ficheros con el mismo contenido y distintos nombres.

En el segundo caso uno o varios ficheros se copian en un directorio diferente.

Una copia puede ser recursiva o no. Para que sea recursiva se deberá especificar la opción `-r`. Esto sólo afecta a la copia de directorios. A continuación veremos las diferencias entre ambas:

Copia no recursiva En este caso el directorio de destino debe existir antes de dar la orden de copia, y cada uno de los ficheros fuentes debe ser un fichero ordinario y no un directorio. Cada fichero fuente se copia en el directorio de destino, permaneciendo con su identificador original.

Copia recursiva El comportamiento de la copia recursiva depende de si el directorio de destino existe antes de dar la orden de copia o no:

- Si *dir_destino* no existe, sólo se puede especificar un *dir_fuente*. En primer lugar se creará *dir_destino*, y a continuación se copiarán todos los ficheros y subdirectorios del directorio fuente en él.
- Si *dir_destino* existe previamente, se copia cada fichero fuente al directorio de destino, al igual que se hace en una copia no recursiva. Cada directorio fuente es reproducido como un subdirectorio del directorio de destino, incluyendo sus ficheros y subdirectorios.

Todas las órdenes estudiadas en esta sección tienen más opciones de las aquí comentadas; para más información acerca de éstas consulte la página correspondiente del manual.

2.7.4. Borrar ficheros

GNU/Linux proporciona dos órdenes para borrar ficheros, `rm` y `rmdir`. La orden `rm` puede borrar tanto ficheros ordinarios como directorios (siempre que se especifique la opción `-r`), mientras que `rmdir` sólo puede borrar directorios.

Borrar un fichero, ya sea un fichero ordinario o un directorio, significa borrar un enlace del fichero, más que borrar el fichero en sí mismo. Si hay otros enlaces al fichero, éstos permanecen y los datos también. Un fichero sólo se borra cuando lo hace el último enlace al mismo. Un directorio sólo se borra cuando no hay ficheros catalogados en él.

La orden `rm`

La orden `rm` borra enlaces a ficheros, ya sean ficheros ordinarios o directorios (opción `-r`). Un fichero se borra cuando se han borrado todos los enlaces a él. La forma de uso es:

`rm` [*opciones*] *fichero* ...

Una de las opciones que admite `rm` es `-r`; si se incluye esta opción se le pueden pasar a `rm` nombres de directorios; en este caso `rm` borrará todos los ficheros incluidos en el directorio especificado y por último borrará el propio directorio.

Ejemplos:

1. `$ rm mifichero`
   Borra un enlace a `mifichero`*; si existe otro enlace a ese mismo fichero, éste permanecerá; si no es así, se borrará.*
2. `$ rm -ir manual`
   Borra recursivamente el contenido del directorio `manual`*, y luego borra el propio directorio (opción* `-r`*), preguntando si quiere borrar cada fichero y directorio (opción* `-i`*).*

La orden `rmdir`

La orden `rmdir` borra enlaces a directorios. La forma de usarla es:

`rmdir` [*opciones*] *directorio* ...

A diferencia de `rm -r`, `rmdir` no borra un enlace a un directorio a menos que éste esté vacío.

Tanto la orden `rm` como `rmdir` tienen más opciones; si necesita información acerca de ellas puede consultar el manual.

1. ¿Qué diferencias existen entre las siguientes órdenes? Razone la respuesta.

   a)
   ```
   $ cp preguntas resultado
   $ rm preguntas
   ```

   b)
   ```
   $ mv preguntas resultado
   ```

2. Suponga que se da la siguiente orden

   ```
   $ ls -li texto
   3218 -rw-r----- 1 amm alum 3898 Abr 15  1998 ejer7
   3195 -rw-r----- 1 amm alum 6960 Abr 12  1998 ejer8
   ```

   Explique de forma razonada qué ocurre al dar de forma consecutiva las siguientes órdenes. Indique el resultado de la orden `ls` anterior después de dar cada una de ellas.

   a) `$ ln texto/ejer7 texto/ejer9`

   b) `$ ln -s ejer9 texto/ejer10`

   c) `$ rm texto/ejer7`

   d) `$ cat texto/ejer9 texto/ejer10`

   e) `$ cd texto`

   f) `$ cat ejer9`

   g) `$ cat ejer10`

2.8. Compresión de ficheros

Se puede reducir el espacio ocupado por un fichero almacenándolo en formato comprimido. Así, si tenemos establecida una cuota de bloques de datos y estamos cerca del límite, puede ser conveniente comprimir aquellos ficheros que no podamos borrar.

Existen diversos algoritmos de compresión de datos, como por ejemplo, el de Huffman, Lempel-Ziv (LZ77 y LZ78), Burrows-Wheeler, etc; con diferente grado de efectividad. GNU/Linux suele disponer de varios programas que usan algunos de estos algoritmos, tales como `gzip`, que emplea el algoritmo

LZ77, o `bzip2`, que usa el de Burrows-Wheeler, que obtiene generalmente mejores resultados que el anterior.

A continuación se estudiará la orden `gzip` por estar actualmente más extendida. Puede encontrarse más información sobre `bzip2` en el manual.

2.8.1. La orden `gzip`

Su formato es:

`gzip` [*opciones*] *fichero* ...

donde *fichero* es el nombre del fichero que deseamos comprimir. Por cada fichero original se crea un fichero nuevo, denominándose *fichero*`.gz`, con el contenido comprimido, borrándose el original.

Opciones:

`-c` Escribe el fichero comprimido en la salida estándar, dejando el original inalterado.

`-d` Descomprime un fichero comprimido.

`-l` Lista para cada fichero comprimido dado como argumento los siguientes campos:

- Tamaño del fichero comprimido.
- Tamaño del fichero sin comprimir.
- Tasa de compresión.
- Nombre del fichero descomprimido.

`-r` Si se le pasa un nombre de directorio, desciende recursivamente y comprime todos los ficheros que encuentra.

Ejemplos:

1. `$ gzip fincas`
   Comprime el fichero fincas, creando el fichero comprimido `fincas.gz`.
2. `$ gzip -l fincas.gz`
   Da información acerca de los tamaños del fichero comprimido y sin comprimir, la tasa de compresión y el nombre del fichero descomprimido.

3. `$ gzip -d fincas.gz`
   Descomprime el fichero `fincas.gz`*, obteniéndose el fichero* `fincas` *original.*

2.8.2. La orden `gunzip`

La orden `gunzip` descomprime ficheros, siendo equivalente a `gzip -d`. Su formato es:

`gunzip` [*opciones*] *fichero* ...

donde *fichero* es el nombre del fichero que queremos descomprimir. Cada uno se reemplaza por su versión descomprimida, siendo su nombre el mismo que el de la versión comprimida pero sin `.gz`.

También descomprime ficheros comprimidos con otros programas, pero exige que su nombre termine en `.gz`. Sin embargo es posible pasarle un fichero comprimido que no termine en `.gz` siempre que utilicemos la opción `-S` seguida de la terminación del fichero.

Ejemplos:

1. `$ gunzip fincas.gz`
   Descomprime el fichero `fincas.gz`*. Es equivalente a* `gzip -d fincas.gz`.
2. `$ gunzip -c info.gz`
   Nos muestra el fichero descomprimido en la salida estándar, pero no llega a descomprimirlo realmente (opción `-c`*).*
3. `$ gunzip -S .Z datos.Z`
   Descomprime el fichero `datos.Z`*.*

2.8.3. La orden `zcat`

Nos muestra en la salida estándar el fichero descomprimido, pero sin llegar a descomprimirlo. Es equivalente a `gunzip -c`. Su formato es:

`zcat` [*opciones*] [*fichero* ...]

Ejemplo:

```
$ zcat fincas.gz
```

Es equivalente a `gunzip -c fincas.gz`.

`gzip`, `gunzip` y `zcat` son un solo programa, que se comporta de una forma u otra en función de la opción que se le pase o del nombre con que se le llame.

2.9. Archivar un conjunto de ficheros

Llamamos **archivo** a un fichero que contiene muchos otros ficheros así como información sobre éstos. El archivo contiene el nombre de los ficheros, su propietario, tamaño, y otros atributos.

Entre las utilidades que tiene la creación de archivos están:

- Como copia de seguridad de un grupo de ficheros, para poder restaurarlos en caso de pérdida o deterioro.
- Para agrupar un conjunto de ficheros con el fin de proceder a su transmisión a otro ordenador.
- Para guardar un conjunto de ficheros como si se tratara de uno solo. Esto es útil cuando estamos cerca del límite del número máximo de ficheros que nuestra cuota nos permite crear.

GNU/Linux suele suministrar varios programas para archivar. Uno de ellos es `tar`, que estudiaremos a continuación.

2.9.1. La orden `tar`

El programa `tar` (*tape archiver*) fue inicialmente pensado para leer y escribir archivos en cinta magnética, aunque también es posible almacenarlos en otros dispositivos (disco duro, disquete, ...). Nos permite grabar ficheros en un archivo y recuperarlos posteriormente. Los ficheros que forman parte de un archivo se suelen denominar **miembros**. Una vez creado el archivo se pueden realizar las siguientes operaciones sobre él:

- Añadir más miembros.
- Borrarlos (esto sólo es válido para archivos que residan en disco).
- Actualizar los miembros.

- Extraer alguno o todos los miembros del archivo, es decir, copiarlos desde éste a un directorio.
- Ver su contenido.

La orden `tar` escribe generalmente de una forma acumulativa, agregando los miembros nuevos al final del archivo. Esto es debido a que la "filosofía" de funcionamiento de `tar` es la de actuar sobre una cinta magnética (acceso secuencial, se añade por el final, etc.).

Es importante señalar que cuando se crea un archivo no desaparecen los ficheros originales; del mismo modo, cuando se extraen los miembros de un archivo éstos no desaparecen de él.

Su formato es:

`tar` *función* [*modificador* ...] [*fichero* ...]

donde *función* indica la operación a realizar, *modificador* las características de ésta y *fichero* el nombre de los que vamos a escribir en el archivo o a extraer de él. Las operaciones que podemos realizar aparecen en el cuadro 2.2 y algunos de los modificadores aparecen en el cuadro 2.3.

Función	Acción
`-c`	Crea un archivo nuevo.
`-r`	Añade ficheros al final de un archivo.
`-x`	Extrae miembros de un archivo.
`-t`	Lista el contenido de un archivo.
`-u`	Actualiza el archivo, es decir, si un fichero no es miembro de él o se ha modificado desde que se copió se añade al final.
`--delete`	Borra miembros del archivo (no se puede usar con cintas magnéticas).

Cuadro 2.2: Funciones de la orden `tar`

Ejemplos:

1. `$ tar -cf documentos.tar docs/`
   Crea un archivo llamado `documentos.tar` *con los ficheros contenidos en el directorio* `docs`.
2. `$ tar -xf documentos.tar docs/tema1.txt`
   Extrae de `documentos.tar` *el miembro* `docs/tema1.txt`.

Modificador	Acción
`v`	Visualiza el nombre de cada fichero que procesa.
`f` *fichero*	Utiliza *fichero* como nombre del archivo con el que vamos a operar. Si no se especifica este modificador toma por omisión la entrada o salida estándar, según proceda.
`M`	Crea, lista o extrae un archivo multivolumen.
`z`	Comprime el archivo con `gzip`.
`I`	Comprime el archivo con `bzip2`.

Cuadro 2.3: Modificadores de las funciones de `tar`

3. `$ tar -cvMf /dev/fd0 listados/`
   Crea un archivo en el dispositivo `/dev/fd0` *(disquete) con el contenido del directorio* `listados`*, va mostrando el nombre de los ficheros que va procesando (opción* `v`*). Si es necesario irá pidiendo varios disquetes (opción* `M`*).*
4. `$ tar -tMf /dev/fd0`
   Visualiza la lista de miembros del archivo que reside en el dispositivo `/dev/fd0`.

Ejercicios:

1. Imagine que tiene un fichero llamado `trabajo.so` y quiere comprimir su contenido y almacenarlo en un fichero llamado `copia` ¿qué línea de órdenes tendría que dar para conseguirlo?
2. ¿Qué hacen cada una de las siguientes órdenes?
   a) `$ tar -cvzf /dev/fd0 apuntes`
   b) `$ tar -xvf /dev/fd0 apuntes/tema1`
   c) `$ tar -cMf /dev/fd0 ~`

2.10. Buscar ficheros

El programa `find` busca ficheros, en la parte especificada del sistema de ficheros, que concuerden con un criterio. La forma de la línea de órdenes es:

`find` [*directorio* ...] [*criterio* ...]

donde *directorio* es el punto de partida de la búsqueda y *criterio* es la condición que deben cumplir los ficheros que estamos buscando. Si no se especifica ningún directorio se toma el directorio de trabajo; si no se proporciona ningún criterio, se muestran todos los ficheros.

Esta orden genera una lista de todos los ficheros incluidos directa o indirectamente en los directorios especificados y comprueba si cumplen o no los criterios impuestos; la lista de aquellos ficheros que cumplen todos los criterios especificados se envía a la salida estándar.

Ejemplo:

```
$ find /usr -name "v*.h"
```

Esta línea envía a la salida estándar una lista de todos los ficheros contenidos en `/usr` *y sus subdirectorios cuyo nombre empieza por* `v` *y termina en* `.h`.

A continuación vamos a enumerar algunos de los criterios más importantes que se pueden aplicar a los ficheros, clasificados según su función:

- Comprobación de las propiedades de los ficheros

  `-name` *fichero* Es verdadero si el identificador del fichero actual concuerda con *fichero*. Si se introducen comodines en *fichero* asegúrese de ponerlos entre comillas para que sean interpretados por `find` y no por el *shell*.

  `-type` *c* Es verdadero si el fichero actual es de tipo *c*, donde *c* puede ser uno de los siguientes caracteres:

  f fichero ordinario

  d directorio

  b dispositivo de bloques

  c dispositivo de caracteres

  l enlace simbólico

  `-links` *n* Es verdadero si el fichero actual tiene *n* enlaces.

  `-perm` [-] *p* Es verdadero si los permisos del fichero actual (en el capítulo **??** veremos qué son los permisos) vienen dados por el numero octal *p*. Se requiere una concordancia exacta a menos que *p* vaya precedido por `-`. En este caso *p* pueden ser un subconjunto de los permisos del fichero.

  `-user` *usuario* Es verdadero si el propietario del fichero actual es *usuario*.

`-group` *grupo* Es verdadero si el fichero actual pertenece al grupo *grupo*.

`-size` *n* Es verdadero si el fichero actual tiene un tamaño de *n* bloques (512 bytes). Si ponemos detrás de *n* una `k`, la unidad será el KiB y si ponemos una `c` la unidad será el byte.

`-newer` *fichero* Es verdadero si el fichero actual ha sido modificado más recientemente que el fichero especificado.

`-inum` *n* Es verdadero si el número-i del fichero actual es *n*.

`-maxdepth` *niveles* Especifica el número de niveles máximo que debe descender a partir del directorio inicial. '`maxdepth 0`' significa que sólo se apliquen las pruebas al directorio inicial.

`-ls` Se imprime en la salida estándar además del nombre del fichero actual los atributos asociados a él, tales como número de nodo índice, tamaño, modo de protección, número de enlaces, propietario, etc.

- Aplicación de órdenes a ficheros

`-exec` *orden* Es verdadero si la *orden* devuelve un status de salida 0; es decir, si se ejecuta correctamente. El final de la orden y sus argumentos se debe marcar con un punto y coma; éste se debe proteger con el carácter \ para que el *shell* no lo interprete. Para especificar que la orden actúe sobre el fichero actual se utiliza la notación `{}`.

Ejemplo:
```
$ find prog -name "*.ok" -type f -exec rm {} \;
```
Esta línea borra todos los ficheros que concuerden con `*.ok` *en el directorio* `prog` *y sus subdirectorios.*

`-ok` *orden* Funciona como `-exec`, excepto en que la orden generada se envía a la salida estándar de errores con una interrogación detrás. Si se contesta afirmativamente, se ejecutará la orden; si se contesta cualquier otra cosa, no se ejecutará. Una orden no ejecutada produce la condición "falso".

- Significado de los valores numéricos en las pruebas

Cuando aparece un valor numérico *n* en una prueba éste se puede dar de tres formas:

n indica exactamente el valor *n*.

-*n* indica un valor menor que *n*.

+*n* indica un valor mayor que *n*.

- Combinaciones lógicas de pruebas

  Las pruebas se pueden combinar de las siguientes formas:

  1. Encerrando varias entre paréntesis (`find` trata estos paréntesis como argumentos, por lo que tienen que estar rodeados por espacios, y como los paréntesis tienen significado para el *shell* deberán estar protegidos por \).
  2. Se puede negar una prueba precediéndola con `!`. Sólo se aceptarán los ficheros que no satisfacen la prueba. Al igual que los paréntesis, `!` debe estar rodeado por espacios.
  3. Se pueden escribir dos pruebas una a continuación de la otra. En este caso la combinación se satisface sólo si se cumplen las dos pruebas individuales. El mismo efecto tiene poner `-a` entre ambas.
  4. También se puede poner `-o` entre dos pruebas. En este caso la combinación se satisface si se cumple alguna de ellas. Si se cumple la primera no se verifica la segunda.

Ejemplos:

1. `$ find ~ -size 0 -ok rm {} \;`
   Busca de entre todos los ficheros que estén en el directorio de entrada y en sus subdirectorios, aquéllos cuyo tamaño es cero y los borra pidiendo antes confirmación.
2. `$ find . \( -user pepe -o -user antonio \) -type d`
   Lista todos los subdirectorios del directorio actual cuyo propietario sea `pepe` *o* `antonio`.

1. El editor `emacs` guarda una versión anterior del fichero que acaba de editar por motivos de seguridad. Cuando se tiene poco espacio libre de la cuota de disco que se tiene asignada, una forma de conseguir liberar espacio es borrando estas copias de seguridad. `emacs` nombra a estos ficheros añadiéndoles al final de su identificador el carácter `~`. ¿Cómo podría borrar, dando una sola línea de órdenes, todos los ficheros de este tipo que tenga en su cuenta?

2. Escriba una línea de órdenes que copie los ficheros que hay en el directorio `/tmp` cuyo nombre empiece por `z` y termine en un dígito, tengan dos o más enlaces y que no sean de nuestra propiedad, al directorio `~/copias`.

2.11. Impresión de ficheros

En el sistema GNU/Linux las impresoras se gestionan a través de **colas de impresión**, que son controladas por un proceso del sistema denominado demonio de impresión. Una misma impresora puede tener asociadas varias colas.

Una petición de impresión puede incluir múltiples ficheros. A cada petición se le asigna un identificador que se puede usar cuando se requiera información acerca de su estado o cuando se quiera cancelar ésta.

2.11.1. La orden `lpr`

Envía ficheros a la cola de impresión. La forma de la línea de órdenes es:

`lpr` [*opciones*] [*fichero* ...]

Si no se especifica ningún fichero, `lpr` toma su entrada de la entrada estándar; por tanto, se puede usar `lpr` para imprimir la salida de una orden, utilizando una interconexión (en el capítulo **??** veremos qué son las interconexiones).

Algunas opciones importantes de `lpr` son:

`-P` *cola* Imprime el trabajo en la impresora asignada a la cola de impresión indicada. Si no se especifica esta opción imprimirá en la cola 0.

`-#`*n* Imprime *n* copias.

`-h` Suprime la página de cabecera.

`-p` Formatea el fichero antes de la impresión, dividiéndolo en páginas y poniéndole a éstas la fecha y el nombre del fichero.

`-r` Borra el fichero después de imprimirlo.

2.11.2. La orden `lpq`

Nos muestra el estado actual de la cola de impresión. Entre la información que muestra está: la lista de trabajos pendientes de imprimir, el estado de cada uno de ellos, el identificador que le corresponde, etc.

El formato de la orden `lpq` es:

`lpq` [*opciones*]

Algunas opciones importantes de `lpq` son:

`-P` *cola* Da información sobre la cola indicada.

`-l` Formato largo.

2.11.3. La orden `lprm`

Borra trabajos de una cola de impresión. Su formato es el siguiente:

`lprm` [`-P` *cola*] [*ident_ trabajo* ...]

donde *ident_ trabajo* identifica el trabajo que se quiere borrar, obtenido a partir de la orden `lpq`.

Opciones:

`-P` *cola* Se indica la cola de la cual hay que borrar el trabajo.

2.12. Otras operaciones con ficheros

2.12.1. Contar palabras, líneas o caracteres de ficheros

La orden `wc` cuenta el número de caracteres, palabras o líneas en uno o más ficheros. Se considera que una palabra es una secuencia de caracteres delimitada por blancos (espacios, tabuladores, saltos de línea y de página).

La forma de la línea de órdenes es

`wc` [`-lwc`] [*fichero*] ...

Por omisión `wc` muestra las tres cosas: caracteres, palabras y líneas; si queremos que sólo muestre una de ellas podemos usar las opciones `-c` (caracteres), `-w` (palabras) o `-l` (líneas). Estas opciones también pueden combinarse entre sí.

Ejemplo:

```
$ wc -l /etc/passwd
```

Cuenta el número de líneas del fichero `/etc/passwd`.

2.12.2. Clasificación de ficheros con `file`

La orden `file` intenta clasificar un fichero examinando los primeros bytes de éste. El formato de `file` es:

`file` [*opciones*] *fichero* ...

Los tipos de ficheros que reconoce `file` incluyen binarios ejecutables y de datos, ficheros de texto y *shell scripts*.

Las opciones disponibles para esta orden puede verlas en el manual, así como una descripción más detallada de cómo reconoce cada tipo de fichero.

Ejemplo:

```
$ file /bin/date
```

2.12.3. Comparación de ficheros

La orden `diff` analiza las diferencias entre dos ficheros. La forma de la línea de órdenes es:

`diff` [*opciones*] *fichero1 fichero2*

La orden `diff` obtiene una lista de instrucciones para transformar *fichero1* en *fichero2* enviándola a la salida estándar. El código de retorno es 0 si los ficheros coinciden, 1 si difieren, y mayor que 1 si se produce algún error.

1. Suponga que tiene un fichero de texto y quiere imprimirlo de forma paginada ¿qué línea de órdenes tendría que dar? Si una vez dada la orden se arrepiente de haberlo mandado a imprimir y quiere eliminar el trabajo de impresión ¿qué órdenes tendría que dar?

2. ¿Qué línea de órdenes tendría que dar para saber cuántas líneas tiene el fichero `/etc/passwd`?

2.13. Ejercicios

1. ¿Qué hacen las siguientes órdenes?

   a) `$ touch fichero`

   b) `$ > fichero`

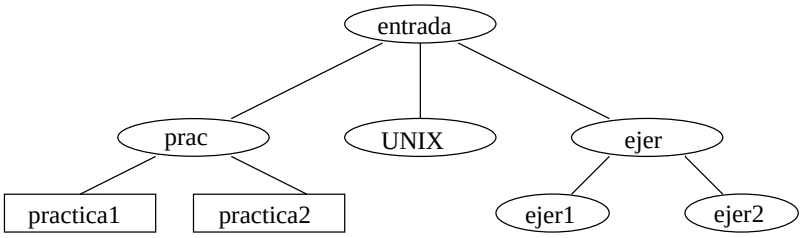
2. Suponga que quiere visualizar el contenido de un fichero ¿Qué diferencia hay entre utilizar la orden `cat` y las órdenes `more` o `less`? ¿Cuál de las dos formas es la más adecuada?

3. Una utilidad de la orden `basename` es la de borrar cualquier cadena de caracteres del final del identificador. Para los siguientes nombres de ficheros, extraiga su nombre sin la extensión de los mismos:

   a) `/usr/hamlet/capitulo.b.1`

   b) `usr/hamlet/capitulo.b`

4. Considere la estructura de directorios de la figura a partir de su directorio de entrada:



   a) Escriba la secuencia exacta de órdenes que debería haber dado para crear esta estructura, suponiendo que inicialmente sólo existía el directorio `entrada`.

   b) Si quisiera mover el directorio `ejer` junto con sus dos subdirectorios al directorio `UNIX` ¿qué línea de órdenes tendría que dar? ¿Cómo quedaría la nueva estructura de directorios?

   c) Suponga que quiere crear un enlace de los ficheros `practica1` y `practica2` al directorio `UNIX`. ¿Qué línea de órdenes daría? ¿Cuál sería la nueva estructura de directorios?

   d) ¿Qué línea de órdenes daría para borrar el directorio `prac`? ¿Cómo quedaría la estructura de directorios ahora?

   e) Suponiendo que su directorio actual es el de entrada y que el camino absoluto de éste es `/home/alum/entrada`, dé los nombres absolutos y relativos de todos los ficheros y directorios de la estructura inicial, así como después de cada uno de los pasos que se han dado en los apartados anteriores.

5. ¿Cómo podría averiguar los ficheros que hay en el sistema que pertenecen al grupo `root` y tienen un número de enlaces superior a 2?

6. Explique qué hace la siguiente línea de órdenes:

   ```
   $ find / -name "*data" -user fmp
   ```

   ¿En qué diferiría la salida de éste de la del siguiente?

   ```
   $ find . -name "*data" -user fmp
   ```

7. ¿Cómo expande el *shell* bash lo siguiente?

   a) `~usuario`

   b) `~-`

   c) `~+`

8. Suponga que su directorio de trabajo es `/usr/users/alum/ppp`. Escriba los resultados que faltan en la siguiente secuencia de órdenes:

   ```
   $ ls
   prueba examen notas C Cobol
   $ ln examen parcial
   $ ls
   ..........
   $ mkdir examenes
   $ ln examen parcial examenes
   $ ls
   ..........
   $ cd examenes
   $ ls
   ..........
   $ pwd
   ..........
   ```

9. Si se teclea `rm *` ¿por qué `rm` no puede advertir al usuario que está a punto de borrar todos sus ficheros?

10. Suponga que da las siguientes órdenes:

    ```
    $ ls -li texto
    3218 -rw-r----- 2 amm alum 3898 Feb 12  1996 tema7
    3195 -rw-r----- 1 amm alum 6860 Feb  2  1996 tema8
    $ ls -li backup
    3218 -rw-r----- 2 amm alum 3898 Feb 12  1996 tema7.bak
    ```

    ¿Cuál será el resultado de dar las mismas órdenes anteriores después de dar: `$ rm texto/tema7`?

11. ¿Qué línea de órdenes daría usted para mover todos los ficheros que se encuentran en el directorio `practicas` del usuario `juan` (teniendo en cuenta que `practicas` es un subdirectorio de `~juan`) a un directorio también denominado `practicas` que es un subdirectorio de su directorio de entrada?

12. ¿Qué línea de órdenes daría para obtener una lista de todos los ficheros del sistema que tienen tamaño 0? ¿Y si sólo quiere saber cuáles de los que le pertenecen tienen tamaño 0? ¿Y si quiere borrarlos?

13. ¿Cómo podemos saber el tipo del contenido de un fichero?

14. ¿Qué orden es necesario dar para visualizar el contenido de un fichero comprimido?

15. ¿Qué orden daría para hacer lo siguiente?

    a) Realizar un listado de su directorio de entrada donde aparezca el número de nodo-i de cada fichero.

    b) Listar todos los ficheros de `/usr` y `/lib` cuyos nombres terminen en `.1`.

    c) Listar todos los ficheros de su propiedad que han sido modificados en la última semana.

    d) Borrar todos los ficheros de su directorio de entrada y de los directorios que hay justo debajo de él, que tienen un tamaño superior a 10 bloques y tienen un enlace en otro lugar.

    e) Listar todos los subdirectorios de su directorio de entrada que no le pertenecen.

16. Al dar la orden `quota`, el usuario `pepe` obtiene la siguiente información:

```
Filesystem  blocks  quota  limit  files  quota  limit
/home/pepe   34949  35000  35100    379    400    410
```

    ¿Qué tipo de problemas puede tener el usuario `pepe` en un futuro cercano? ¿Qué acciones recomienda que realice el usuario para evitarlos? Indique las órdenes correspondientes, teniendo en cuenta que todos sus ficheros son importantes.