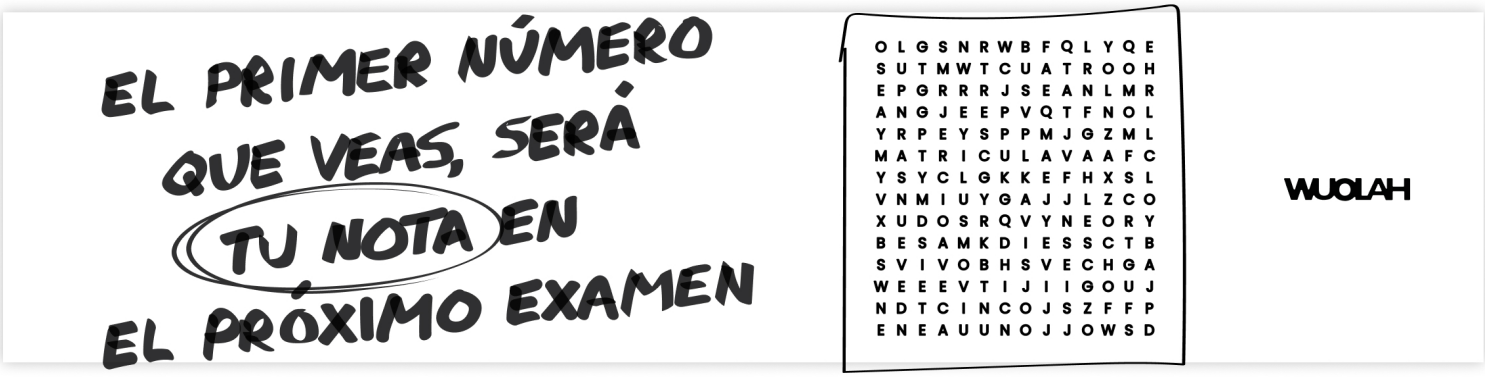
Parte 4.pdf

QuesoViejo_

Sistemas Operativos

2º Grado en Ingeniería Informática

Facultad de Informática
Universidad de A Coruña



Capítulo 1

La primera sesión con GNU/Linux, las órdenes y la otención de ayuda

1.1. Estructura de GNU/Linux

GNU/Linux es un sistema operativo multitarea y multiusuario que funciona en diferentes arquitecturas tales como Intel 386 y superiores, Alpha, Sun SPARK, Motorola 68K y PowerPC. Su estructura se puede representar gráficamente como se ve en la figura 1.1. Podemos distinguir tres capas:

- Núcleo.
- Intérprete de órdenes o *shell*.
- Utilidades.

El núcleo es el corazón del sistema GNU/Linux y reside en memoria siempre que el sistema está activo. Controla todos los recursos (memoria, disco, procesador, ...) y los asigna a los distintos procesos que compiten por ellos.

El intérprete de órdenes o *shell* es un programa que lee órdenes de la entrada, las traduce a un formato interno y las ejecuta. Las diferentes distribuciones de GNU/Linux suelen disponer de varios *shells*, entre los que se encuentran el C, de Korn, Z, bash, etc.

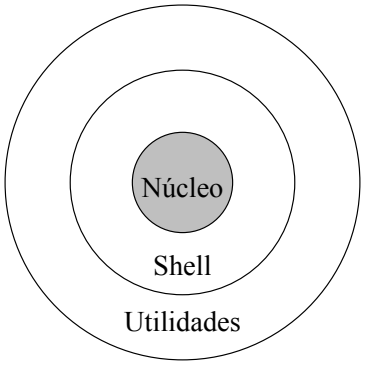


Figura 1.1: Estructura del Sistema Operativo GNU/LINUX

1.2. Cómo se accede a un sistema GNU/LINUX

En las primeras líneas hemos dicho que GNU/LINUX es un sistema multiusuario, lo que implica que más de una persona puede estar usando el ordenador a un tiempo, pero además engloba el concepto de **usuario.** Esto significa que GNU/LINUX debe ser capaz de identificar a los usuarios del sistema y distinguir entre ellos. Por tanto, antes de que una persona pueda acceder al sistema, éste debe reconocerla como usuario. Para ello el administrador debe haberle inscrito como usuario autorizado, asignándole un indentificador único, el nombre de usuario o nombre de *login*, así como una contraseña. Es importante mantenerla en secreto ya que es un elemento de seguridad muy importante, por ser la forma que tiene el sistema de saber que la persona que quiere acceder a él es realmente el usuario que dice ser.

Al iniciar una sesión el sistema nos pide nuestro nombre de usuario (*login*) y nuestra contraseña (*passwd*). Al introducir la palabra clave o contraseña no verá en la pantalla nada de lo que teclea; esto es debido a razones de seguridad, para que otra persona que esté cerca no pueda verla.

Una vez que el sistema reconoce el nombre de usuario introducido como el de uno autorizado, y comprueba que la palabra clave es correcta, se procede a la entrada al sistema.

Si estamos iniciando la sesión en modo gráfico bajo X Window, a continuación se ejecutará el administrador de ventanas o de escritorio asociado a nuestro usuario. En caso de iniciarla en modo texto, se ejecutará un *shell*. ¿Cuál de ellos? El que el administrador de su sistema haya elegido de entre todos los disponibles. En nuestro caso el elegido ha sido el *shell* bash (*Bourne again shell*).

ACOPOL

ACADEMIA OPOSICIONES POLICIALES

Academia de policía desde 1997

Oposiciones Policía Nacional

2023

NUEVO CURSO

ESCALA EJECUTIVA

Consigue tu sueño en nuestra academia

ÁVILA-VALLADOLID Y ONLINE

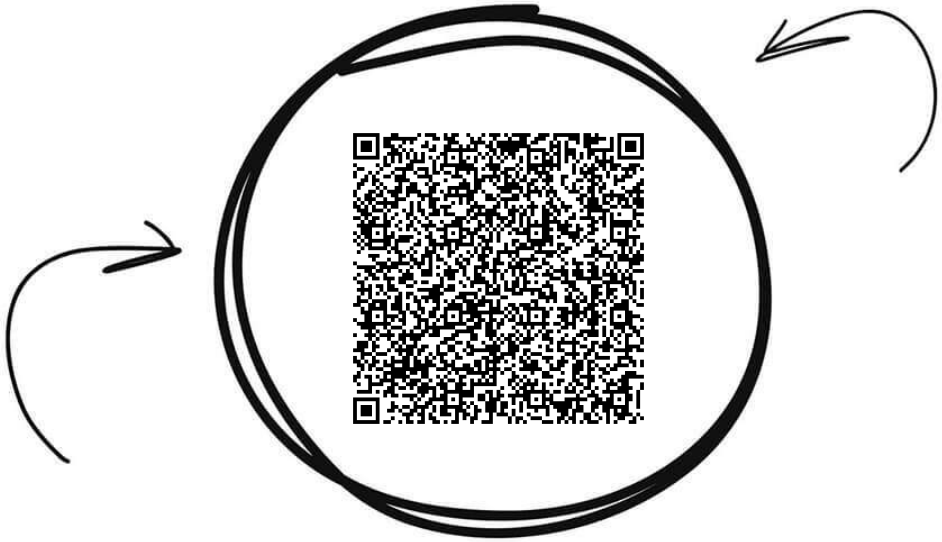
POR UN FUTURO MEJOR

C. Agustín Rodríguez Sahagún, 3, 05003 Ávila
TEL. 920 25 45 53

Calle Italia, 8, 47007 Valladolid
TEL. 983 74 88 85

WWW.ACOPOL.ES

Sistemas Operativos



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

1.3. El entorno de trabajo

En clase vamos a utilizar el sistema GNU/Linux bajo un entorno X Window. X Window es una interfaz gráfica de usuario independiente del sistema operativo, del hardware y de la red. Se trata de un sistema de ventanas, que se pueden manejar mediante un administrador de ventanas, o mediante los manejadores de escritorio. En el caso de GNU/Linux, los manejadores de escritorio más utilizados son KDE (*K Desktop Environment*) y GNOME (GNU *Network Object Model Environment*). Nosotros vamos a utilizar KDE.

1.3.1. Terminología

A continuación damos la definición de algunos términos que se utilizan frecuentemente en combinación con escritorios gráficos. El significado de algunos de estos términos puede variar de un entorno al otro y algunos de ellos se utilizan sólo al hablar de un determinado escritorio.

Escritorio El escritorio es el entorno de trabajo principal. Llena completamente la pantalla pero es más que una mera imagen de fondo. Por ejemplo, es posible colocar las aplicaciones y objetos más usados sobre él y así disponer de un acceso directo.

Botón de menú principal De forma similar al "botón de inicio"de MS Windows, los escritorios de Linux contienen normalmente un botón de menú en el extremo izquierdo para abrir el menú principal. Este menú alberga una estructura ordenada para acceder a funciones como "Buscar", "Terminar", y "Bloquear sesión".

Iconos del escritorio Los iconos del escritorio se encuentran en éste y representan archivos, directorios, aplicaciones y medios de almacenamiento extraíbles como CDs o DVDs. El icono de escritorio más conocido es probablemente la papelera que sirve para soltar sobre ella los archivos que se quieren borrar.

Panel El panel es una barra que se encuentra típicamente en el borde superior o inferior de la pantalla. Se compone de menús, un área de lanzamiento rápido, algunas herramientas y habitualmente una barra de tareas. El panel está concebido para proporcionar toda la información importante sobre las aplicaciones y el sistema y proporciona fácil acceso a numerosas funciones y aplicaciones importantes. KDE permite situar la barra en posición horizontal o vertical.

Barra de tareas La barra de tareas sirve para alternar entre ventanas abiertas, incluye también los escritorios virtuales y permite cambiar de uno a otro. La barra de tareas forma parte del panel.

Lanzador rápido El lanzador rápido es una parte del panel que alberga los iconos de las funciones y aplicaciones más importantes. Basta con pulsar el icono para lanzar la aplicación y hace que no sea necesario pasar por los menús de las aplicaciones.

Applet Un *applet* es una pequeña herramienta que se integra en el panel mientras que una aplicación es un programa completo que utiliza su propia ventana en el escritorio.

Escritorios virtuales El concepto de los escritorios virtuales es exclusivo de Linux y es comparable a disponer de varias mesas de trabajo en la oficina. Puede guardar objetos en todas las mesas pero sólo puede trabajar en una en cada instante. Por ejemplo, es posible utilizar los escritorios separados por tareas o utilizarlos sencillamente como espacio adicional. Gracias a los escritorios virtuales se puede tener múltiples ventanas abiertas simultáneamente pero sólo ver una o algunas de ellas. Las ventanas se mueven de un escritorio virtual al otro tan fácilmente como una hoja de una mesa a la otra. Todos los escritorios permiten configurar el número y el uso de los escritorios virtuales. KDE dispone de un mecanismo para cambiar de un escritorio virtual al otro.

Terminal Por terminal se entiende cualquier dispositivo que permite enviar órdenes a un computador. Por una parte existen terminales reales o físicos formados por un monitor, un teclado y una conexión al computador. Por otra parte, hay emuladores de terminal formados por una ventana en el escritorio que procesa órdenes.

Sesión La sesión en el escritorio comienza después de la autenticación. La validez de una sesión finaliza cuando se sale del entorno. El arranque y la terminación de ciertos programas forman parte de una sesión. Los programas y servicios que se inician pueden ser configurados por cada usuario.

1.3.2. Configuración de los componentes del escritorio de KDE

Es posible configurar prácticamente la totalidad de los componentes del escritorio de forma individual. Al pulsar con el botón derecho sobre un determinado elemento se abre siempre el menú contextual. KDE dispone de un centro de control para acceder a todas las opciones de configuración del escritorio. A continuación se muestran unos ejemplos para ilustrar el proceso.

Añadir una aplicación al área de lanzamiento rápido 1. Pulse con el botón derecho sobre una parte vacía del panel para añadir allí la aplicación nueva.

2. Seleccione en el menú emergente 'Añadir'+'Botón de aplicación'.
3. Seleccione la aplicación desde una de las categorías de los submenús.

Cambiar el fondo de escritorio

1. Pulse con el botón derecho sobre el escritorio.
2. Seleccione 'Configurar Escritorio'. Dentro de la ventana de diálogo que se abre a continuación puede configurar los siguientes aspectos del escritorio: 'Fondo', 'Comportamiento', 'Escritorios múltiples', 'Salvapantallas' y 'Pantalla'.
3. Después de seleccionar 'Fondo' defina si los cambios se deben aplicar en un solo escritorio o en todos. Puede seleccionar una imagen de fondo, deshabilitar las imágenes de fondo e incluso optar por una secuencia de imágenes. El cuadro 'Opciones' contiene varias posibilidades para posicionar la imagen de fondo, el color de fondo y el difuminado (mezcla) de los colores de fondo.
4. Pulse 'Aplicar' los cambios y salga del diálogo con 'Aceptar'.

Crear un nuevo icono de escritorio Se pueden crear iconos que representan diversos elementos: carpetas, ficheros, dispositivos, ... El procedimiento general es:

1. Pulse con el botón derecho sobre el escritorio para abrir el menú contextual. Seleccione 'Crear Nuevo'+ la opción adecuada según el tipo de elemento que quiera crear: carpeta, dispositivo, ...
2. Introduzca en la ventana emergente el nombre del nuevo elemento.
3. Podrá modificar algunas de las propiedades del nuevo elemento.

Por ejemplo para añadir el icono de un dispositivo nuevo:

1. Pulse con el botón derecho del ratón sobre el escritorio para abrir el menú contextual. Seleccione 'Crear Nuevo'+'Dispositivo'.
2. Seleccione el tipo de dispositivo adecuado para abrir el diálogo 'Propiedades'. El diálogo de 'Propiedades' se compone de cuatro pestañas: 'General', 'Permisos', 'Dispositivo' e 'Info Meta'. El nombre e icono para el dispositivo se definen en la pestaña 'General' mientras que los permisos de acceso se editan en 'Permisos'. La pestaña 'Dispositivo' sirve para configurar la ruta al dispositivo como por ejemplo /media/dvd para la unidad DVD u otras opciones. Seleccione 'Aplicar' los cambios y salga con 'Aceptar'.

1.4. Ejecución de aplicaciones X

Podemos ejecutar una aplicación X simplemente pulsando el icono que la representa, si éste existe, accediendo a ella a través del menú principal, o bien, desde la terminal escribiendo el nombre de la aplicación. En este último caso es conveniente escribir a continuación del nombre de la aplicación el carácter `&`, ya que esto nos a permitir seguir disponiendo de la terminal inmediatamente para lanzar nuevas aplicaciones. Si no lo hacemos así, no podremos introducir nuevas órdenes en la terminal hasta que no finalice la ejecución de la aplicación X.

Algunos ejemplos de aplicaciones X son:

- `acroread` ejecuta un visor de ficheros PDF.
- `kcalc` nos muestra una calculadora.
- `konqueror` llama al administrador de ficheros y navegador de KDE.

1.5. Cómo finalizar la sesión

Para finalizar una sesión X no nos basta con cerrar la ventana de la terminal sino que tendremos que ir cerrando todas las ventanas de las aplicaciones que estuvieran ejecutándose. Normalmente para cerrar la ventana de la terminal daremos las órdenes: `logout`, `exit` o también CTRL-D. Las ventanas de aplicaciones tienen normalmente una opción para salir. Asimismo, tend

Cuando quiera apagar el ordenador, tendrá antes que parar el sistema, ya que si apagamos el ordenador sin hacerlo, el sistema puede quedar en un estado inconsistente. Esta tarea sólo está permitida en los sistemas GNU/LINUX al administrador, sin embargo es posible habilitarla para los usuarios.

1.6. Las órdenes de GNU/LINUX

Una orden es una instrucción que le dice al sistema que realice una tarea determinada. Las órdenes en GNU/LINUX suelen ser vocablos de la lengua inglesa abreviados y siempre se escriben en minúsculas.

Las órdenes pueden ir seguidas de uno o más argumentos que pueden ser:

Opciones: modifican la forma en que actúa la orden. Son letras (mayúsculas o minúsculas) precedidas del carácter `-`. Muchas órdenes de GNU/LINUX

también admiten opciones en formato largo, en este caso la opción es una palabra y va precedida por `--`.

Ficheros: le indican a la orden sobre qué ficheros debe actuar.

Otro tipo de argumentos: pueden ser nombres de usuario, etc.

Ejemplos:

1. `$ ls`
   Damos la orden `ls` *sin ningún tipo de argumento.*
2. `$ ls -l`
   Damos la orden `ls` *con la opción* `l` *(muestra casi toda la información que tiene el sistema sobre los ficheros).*
3. `$ ls -li /etc/passwd`
   Damos la orden `ls` *con dos opciones:* `l` *e* `i` *(número de nodo índice), y le especificamos un nombre de fichero:* `/etc/passwd`.
4. `$ ls -li -a`
   En este ejemplo damos la misma orden con tres opciones divididas en dos grupos: `li` *(formato largo y nodo índice) y* `a` *(todos). Cada grupo de opciones va precedido por el carácter* `-`*, y el hecho de poner todas las opciones dentro de un mismo grupo o separarlas en diferentes grupos suele ser opcional. Normalmente la orden funciona igual independientemente de la forma que se elija.*
5. `$ ls --format=long --inode --all`
   Esta línea hace lo mismo que la anterior, pero las opciones se han dado en formato largo.

Por convenio, la mayoría de las órdenes de GNU/LINUX toman su entrada de la **entrada estándar** y envían su salida a la **salida estándar**. Cuando se trabaja de forma interactiva con el *shell* tanto la entrada como la salida estándar están ligadas a la terminal. Es decir, por omisión, la entrada estándar está asociada al teclado de la terminal y la salida estándar a la pantalla, pero esto no quiere decir que la entrada estándar sea el teclado y la salida estándar la pantalla, sino que inicialmente están ligadas a estos dispositivos. Los distintos *shells* permiten redirigir tanto la entrada como la salida estándar cuando el usuario lo cree necesario. Más adelante veremos cómo se hace esto.

1.6.1. Sintaxis de las órdenes

En la descripción de la sintaxis de las órdenes suelen utilizarse una serie de convenios:

- El texto entre corchetes indica que el contenido es opcional (se pueden poner o no). En ningún caso se escriben los corchetes. Si las opciones son pocas se suelen enumerar todas dentro de los corchetes, pero si son muchas es común dividirlas en varios bloques de corchetes o escribir el texto "opciones" y explicar a continuación las opciones que se aceptan.
- El texto en cursiva debe reemplazarse por el argumento apropiado, en el manual puede aparecer subrayado.
- Las opciones separadas por una barra vertical son excluyentes (no pueden usarse en la misma orden).
- Si es necesario especificar un valor para una opción los posibles valores aparecen entre llaves. Las llaves no escriben.
- Los puntos suspensivos indican que la expresión anterior se puede repetir. En el caso de las opciones suele darse por supuesto que se pueden combinar varias salvo que se especifique lo contrario.

Ejemplos:

La orden `touch` presenta la siguiente sintaxis:

`touch [ -acm] [-r` *fichero_referencia* `| -t` *fecha*`]` *fichero* `...`

Algunos ejemplos correctos de uso serían:

```
touch datos.txt
touch -c fichero3 fichero8
touch -c -r /etc/passwd .bashrc
touch -car /etc/passwd .bashrc
```

Algunos ejemplos de uso incorrecto serían:

`touch` *Falta un nombre de fichero.*

`touch [-c] fichero3` *Los corchetes de las opciones no se escriben.*

`touch -c -r .bashrc` *La opción -r obliga a acompañarla de un fichero, por lo que falta el otro fichero obligatorio de la orden.*

`touch -rc /etc/passwd .bashrc` *La opción -r debe de ir seguida de un fichero, así que la la opción -c debe ir delante o detrás del fichero con su propio guión.*

Shell bash

La línea de órdenes

La línea de órdenes que introducimos en la terminal es interpretada por el *shell*, que realiza una serie de operaciones antes de enviar a ejecutar la orden indicada.

Líneas de órdenes largas

El *shell* bash nos permite introducir órdenes que ocupen más de una línea de la terminal. Para ello podemos utilizar el carácter \.

Ejemplos:

1. ```
   $ ls --format=long \
   > --inode --all
   ```
   Cuando introducimos el carácter \, el shell bash muestra automáticamente en la línea siguiente el indicador secundario >.
2. ```
   $ ls \
   > --format=long \
   > --inode --all
   ```

El *shell* bash interpreta el carácter \ como "la orden continúa en la línea siguiente".

Modificación de la línea de órdenes

El que se puedan hacer o no modificaciones en la línea de órdenes va a depender del *shell* que estemos utilizando. Nosotros vamos a trabajar con el *shell* bash, y éste permite hacer estas modificaciones.

Esto es especialmente útil cuando nos equivocamos al escribir una orden, sobre todo si ésta era larga. También es útil hacer modificaciones en la línea de órdenes cuando tenemos que introducir líneas muy parecidas, en cuyo caso nos ahorramos escribir la línea completa simplemente modificando la anterior.

Para poder recuperar órdenes previas, movernos dentro de la línea de órdenes y modificarla usaremos:

CTRL-P o ↑ : Orden previa

CTRL-N o ↓ : Orden siguiente

CTRL-B o ← : Un carácter hacia atrás

CTRL-F o → : Un carácter hacia delante

CTRL-A : Movernos al principio de la línea

CTRL-E : Movernos al final de la línea

DEL : Borrar el carácter a la izquierda del cursor

CTRL-D : Borrar el carácter sobre el que está el cursor

CTRL-K : Borrar el texto desde la posición actual del cursor hasta el final de la línea

1.7. Las primeras órdenes

En este apartado vamos a ver algunas órdenes útiles y fáciles de usar que proporciona GNU/LINUX. La información que se dará sobre estas órdenes será muy resumida, usted podrá encontrar toda la información que necesite sobre ellas en el manual incorporado que proporciona el sistema; más adelante se le dirá cómo consultarlo.

1.7.1. Información sobre los usuarios

La orden `who` La orden `who` muestra información acerca de los usuarios que están actualmente utilizando el sistema, enviando los resultados a la salida estándar. La forma normal de la línea de `who` es

`who` [*opciones*]

Si damos `whoami` se obtiene la identificación del usuario que da la orden.

La orden `finger` Si queremos obtener información sobre cualquier usuario del sistema, esté o no conectado actualmente, podemos utilizar `finger`. También podemos obtener información acerca de usuarios de máquinas remotas.

El formato de `finger` es

`finger` [*opciones*] [*nombre*] ...[*usuario@máquina*] ...

donde *nombre* indica el usuario a ser buscado (puede darse su nombre de usuario o su nombre real o apellido); si no se da ninguno se obtiene información sobre todos los usuarios conectados actualmente. Si el usuario sobre el que se quiere la información está en una máquina remota habrá que especificar el nombre de ésta.

Las opciones nos permiten modificar la cantidad de información que se obtiene del usuario. Por omisión, se obtiene la forma larga cuando se especifica el usuario, y la forma corta si no se especifica éste.

1.7.2. Cambiar la contraseña

La orden **passwd** le permite cambiar su palabra clave una vez que ha sido registrado como usuario del sistema. Su formato es

`passwd` [*nombre*]

donde *nombre* es el *login* de un usuario. Si no se especifica *nombre*, toma el del usuario que está dando la orden. Sólo el superusuario (administrador del sistema) puede dar un nombre distinto del suyo para cambiar la contraseña de otros usuarios.

Una vez introducida la orden se nos pide la palabra clave actual y a continuación la nueva dos veces (a fin de comprobar si se ha introducido lo que realmente queríamos, ya que no se ve lo que escribimos por razones de seguridad).

A la hora de elegir una contraseña procure que no sea fácilmente adivinable por otros usuarios: no es aconsejable elegir como palabra clave su propio nombre de usuario, su nombre de pila o apellidos, palabras comunes del diccionario, etc. Procure elegir palabras no demasiado comunes o bien combine los caracteres en minúsculas y mayúsculas para que sea más difícil de adivinar. Quizás ahora no vea la razón de tantas medidas de seguridad, pero cuando lleve más tiempo trabajando con el sistema entenderá perfectamente el riesgo que corre si no elige bien su palabra clave.

1.7.3. Utilidades

La orden `date` La orden **date** muestra la fecha y la hora en la salida estándar. Tecleando simplemente **date** se obtiene esta información en el formato predeterminado

```
Thu Jun 5 10:36:03 CEST 1998
```

También es posible especificar el formato en que queremos obtenerla, para lo que le daremos a `date` el formato deseado:

`date` +*formato*

Ejemplo:

```
$ date +'Hoy es %d de %h de %Y'
```

La salida que produciría sería: `Hoy es 5 de Jun de 1998.`

El superusuario puede utilizar la orden `date` para establecer la hora y fecha del sistema.

La orden `cal` La orden `cal` produce un calendario para un solo mes o para un año entero y lo envía a la salida estándar. La forma de la orden `cal` es

`cal` [`-jy`] [[*mes*] *año*]

Hemos de tener en cuenta lo siguiente:

- Si no especificamos ningún argumento obtendremos el calendario del mes actual.
- Si especificamos dos argumentos (mes y año) obtendremos el calendario del mes especificado.
- Si sólo especificamos un argumento (año) obtendremos el calendario de ese año completo.
- Si damos la orden con la opción `-y`, nos dará el calendario del año en curso.
- Si utilizamos cualquiera de las formas anteriores con la opción `-j` nos mostrará los días del año en vez de los días del mes.

Ejemplo:

```
$ cal 9 1752
```

Obtendremos el calendario del mes de septiembre de 1752.

1.8. Cómo obtener ayuda

1.8.1. El manual de GNU/LINUX

La documentación de GNU/LINUX se publica en un formato estándar establecido por las primeras versiones del Manual del Programador de UNIX .

Existe una página del manual para describir cada orden de GNU/Linux, cada función de C, etc. El manual está dividido en varias secciones que están numeradas como sigue

1. Órdenes generales
2. Llamadas al sistema
3. Bibliotecas y rutinas
4. Ficheros especiales
5. Formatos de ficheros
6. Juegos
7. Miscelánea
8. Órdenes de administración del sistema y la red

Para acceder a la información contenida en el manual usamos la orden `man`, cuya forma general de uso es

`man` [*sección*] *título* ...

Sólo es necesario especificar la *sección* si hay varias páginas con el mismo título en secciones diferentes. Pongamos un ejemplo, existe una orden llamada `chmod` y una llamada al sistema con el mismo nombre; si queremos obtener la página del manual correspondiente a esta última no nos bastará con poner `man chmod`, porque entonces obtendríamos la información correspondiente a la orden, ya que la búsqueda se hace empezando por la sección 1. Así que la forma correcta de dar la orden sería `man 2 chmod`.

Cuando se trabaja en entorno X Window, una alternativa a `man` es el programa `xman`. Cuando lo llamamos aparece una ventana como la que puede verse en la figura 1.2 (cuando llame a cualquier aplicación X desde la terminal no olvide poner después del nombre `&`). Si pulsamos en Help obtendremos un texto de ayuda que describe cómo utilizar el programa. Para obtener información sobre un título, pulsaremos Manual Page. En este caso veremos una ventana como la que aparece en la figura 1.3. La forma más simple de buscar una página determinada del manual es seleccionar Sections, que nos muestra un menú con todas las secciones del manual; escogeremos la sección adecuada y elegiremos el título sobre el que queremos información.

Otra forma de obtener una página del manual es seleccionar el menú Options y después la opción `Search`. Nos aparecerá una ventana en la que

Figura 1.2: `xman`

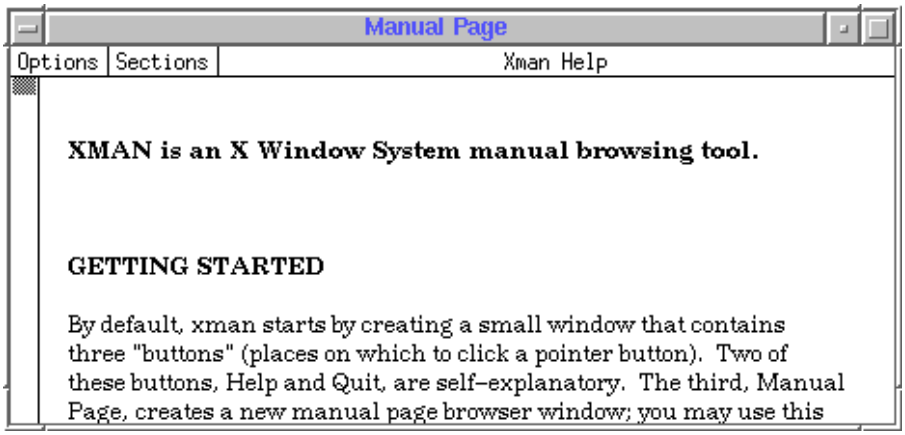


Figura 1.3: Ventana que se obtiene al pulsar Manual Page

podemos introducir el nombre del título en el que estamos interesados y después pulsaremos Manual Page.

Una forma muy cómoda de acceder al manual es mediante la aplicación X **`konqueror`**, esta aplicación es un navegador web y administrador de ficheros, pero también nos sirve para acceder al manual; para ello, simplemente debemos introducir en la barra de direcciones **`man:`***`página`* y nos mostrará la *página* deseada. En el caso de que la página introducida aparezca en varias secciones del manual, konqueror nos mostrará un menú con todas ellas para que elijamos la que queremos ver.

1.8.2. La orden apropos

En ocasiones sabemos que existe una orden pero no sabemos su nombre exacto, en este caso nos puede ser útil la orden **apropos**. Ésta nos muestra todo lo que haya en los títulos del Manual relacionado con cada palabra especificada. La forma de uso es

`apropos` *palabra* ...

Ejemplo:

```
$ apropos html
```

El sistema mostrará todas los títulos del manual, junto con su descripción, en los que aparece la cadena `html`.

También se puede hacer esto utilizando el programa **xman**, para ello procederemos a seleccionar el menú Options y la opción **Search**. Nos aparecerá una ventana donde introducimos la palabra en cuestión y pulsaremos Apropos.

1.8.3. La orden whatis

Cuando sólo queremos obtener una descripción breve de una entrada del manual podemos utilizar la orden **whatis**. La forma de uso de esta orden es

`whatis` *título* ...

esto nos muestra una breve descripción de cada título especificado.

Ejemplo:

```
$ whatis date
```

Nos mostrará una breve descripción de la entrada del manual correspondiente a la orden `date`.

1.8.4. El sistema info

El sistema `info` es otra forma de obtener ayuda. A veces es la única forma de conseguir la última información actualizada sobre una orden, o incluso de obtener información que es imposible localizarla en el manual, como los FAQ (*Frecuently Asked Questions*) de GNU/Linux, por lo que es necesario

conocer su funcionamiento. Para visualizar esta información podemos usar el programa `info` en una terminal.

La información está estructurada en forma de árbol (figura 1.4), de tal modo que existe un nodo directorio (D), a partir del cual existen nodos que tratan de temas diversos (T), a su vez a partir de estos existen nodos que tratan aspectos concretos de estos temas. Podemos movernos en dicha estructura visitando los nodos siguientes, superiores, previos, etc. Para movernos y localizar información dentro de esta ayuda, disponemos de las siguientes órdenes internas:

`m` Ir a una opción del menú. Pulsando TAB obtenemos la lista de elementos del menú.

`p` Volvemos al nodo previo.

`n` Nos situamos en el siguiente nodo de información.

`u` Ir al nodo superior.

`t` Ir al nodo principal del tema (T).

`d` Situarse en el nodo directorio (D), es decir, el que aparece cuando se ejecuta el programa `info`.

`l` Ir al nodo de información que anteriormente hemos visitado.

`i` Buscar en el índice. Pulsando TAB nos muestra los diferentes elementos que se encuentran en el índice, al cual podemos acceder.

En la figura 1.4 se representan las acciones que realizan estas órdenes.

El sistema `info` es una ayuda mediante hiperenlaces, es decir, dentro de un nodo de información existen ciertas palabras que nos permiten saltar a otros nodos de información. Pulsando la tecla `f` podemos saltar por los enlaces existentes en el nodo actual. Para ver los que existen podemos pulsar TAB.

Para finalizar hay que dar la orden `q` o bien CTRL-X + CTRL-C.

También se puede acceder al sistema info mediante la aplicación `konqueror`, introduciendo en la barra de direcciones: `info:`*`título`*.

1.9. El editor

Un editor es un programa que nos permite crear, modificar y visualizar ficheros de texto. Los editores pueden ser de dos tipos, de pantalla y de líneas.

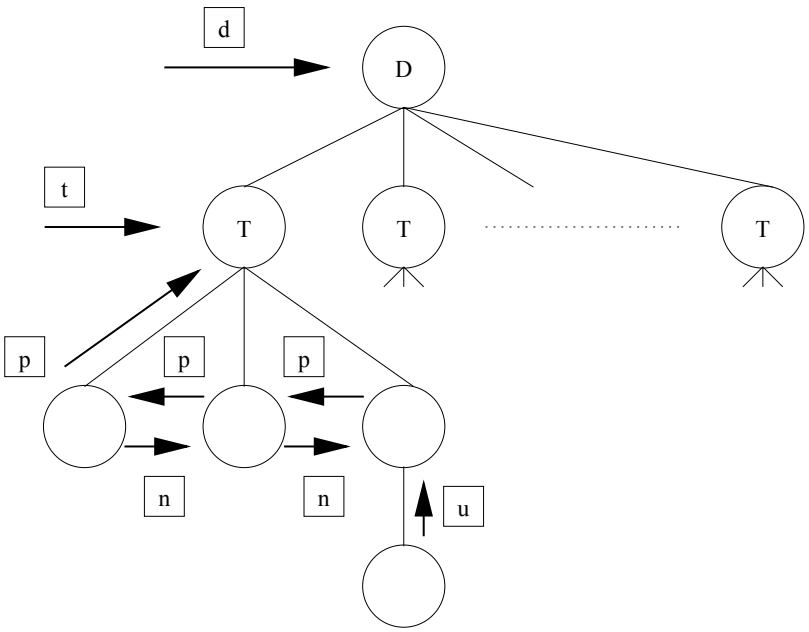


Figura 1.4: Estructura del sistema `info` y acciones posibles

En la actualidad los editores de pantalla han eclipsado casi totalmente a los de líneas.

Entre los editores de líneas están **`ed`** y **`ex`**, una versión extendida y mejorada de **`ed`**. Entre los editores de pantalla están **`vi`**, **`jed`**, **`emacs`**, **`xedit`**, **`pico`**, etc.

Emacs es un editor de pantalla extensible, adaptable y autodocumentado, escrito por Richard Stallman como parte del proyecto GNU. Emacs es un entorno de trabajo ya que además de sus recursos de edición de documentos, ofrece un correo, un editor de directorios, un depurador de LISP y muchos otros servicios. También proporciona modos para manejar diversos tipos de texto, tales como programas en distintos lenguajes de programación, etc. Emacs incorpora un lector de **`info`**, podemos acceder a él mediante CTRL-H + I, en este caso podemos utilizar el ratón para ir resaltando los diferentes nodos, enlaces y elementos de un menú; posicionándonos en ellos, y accediendo a su información pulsando el botón central del ratón.

QuesoViejo_

WUOLAH

si lees esto me debes un besito

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 ficheros 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.

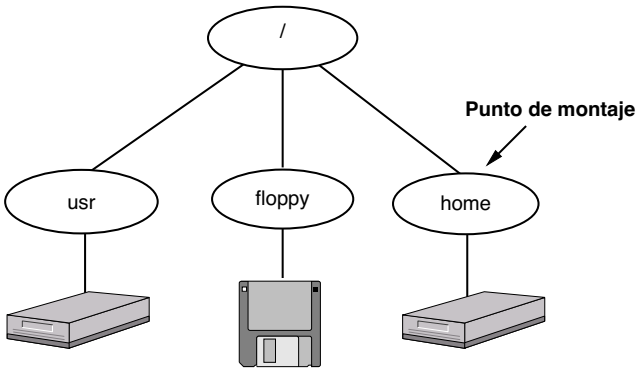


Figura 2.1: Jerarquía de directorios y ficheros

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` [`+`*númunidad* | `-`*númunidad*] [*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

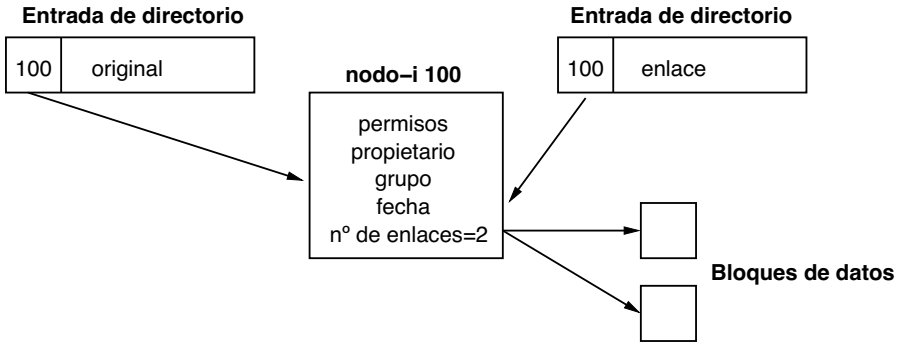
Una entrada de un directorio es un enlace a un fichero. En `ext2` puede haber múltiples enlaces a un mismo fichero, es decir, podemos tener un fichero con varios nombres. Todos los enlaces a un mismo fichero tienen el mismo número-i. Los enlaces pueden residir en directorios diferentes (figura 2.2 a).

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.

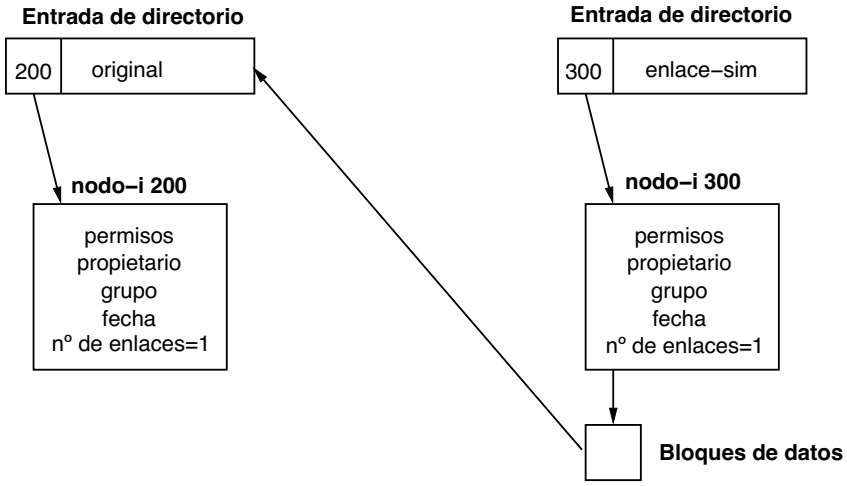
Estos enlaces se suelen denominar **enlaces duros** y presentan dos restricciones; no se pueden crear enlaces duros entre directorios, ni entre dos ficheros que estén en sistemas de ficheros diferentes. La razón para esta última restricción es que cada sistema de ficheros tiene su propio conjunto de 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 en *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 [15/10/2018]

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 número 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*

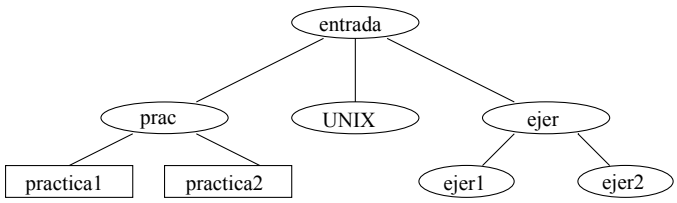
La orden `diff` obtiene una lista de instrucciones para transformar *fichero1* en *fichero2* enviándola a la salida estándar. El código de retorno es 0 si los ficheros coinciden, 1 si difieren, y mayor que 1 si se produce algún error.

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.

QuesoViejo_

WUOLAH

si lees esto me debes un besito

Capítulo 3

Permisos

3.1. Introducción

En GNU/LINUX cada fichero lleva asociados unos permisos que definen quién puede acceder a él y qué operaciones puede realizar. Es decir, mediante los permisos podemos hacer que un fichero no pueda ser leído por nadie o que otro pueda ser ejecutado por todo el mundo.

Existe un usuario especial en el sistema GNU/LINUX, llamado **superusuario**, que puede leer o modificar cualquier fichero en el sistema, independientemente de los permisos que tenga asociados. El nombre de *login* `root` posee privilegios de superusuario; éste lo emplean los administradores del sistema para realizar tareas de mantenimiento.

Cuando un usuario inicia su sesión, teclea un nombre y después confirma que efectivamente es esa persona tecleando una contraseña (*password*). El nombre se conoce como **login_id**. El sistema en realidad reconoce al usuario por medio de un número llamado identificador del usuario **UID**. Además del UID, al usuario se le asigna un identificador de grupo, o **GID**, que lo coloca en cierta clase de usuarios.

Esta información la podemos obtener utilizando la orden `id`, que nos muestra el nombre de usuario (login_id), el UID, y los grupos a los que éste pertenece, junto con los GID que les corresponden.

El sistema mantiene toda la información que necesita conocer sobre cada usuario en el fichero `/etc/passwd`. Se trata de un fichero de texto que contiene una línea por cada usuario autorizado a utilizar el sistema. El formato de la línea es:

```
login_id:clave_cod:UID:GID:varios:dir_de_entrada:shell
```

El primer campo contiene el nombre de *login* del usuario. El segundo está reservado para la contraseña del usuario, apareciendo ésta codificada[1]. Cuando un usuario le da su contraseña al programa `login`, éste la codifica y compara el resultado con la que tiene almacenada en el fichero `/etc/passwd`. Si ambas coinciden, se permite al usuario iniciar la sesión. Los campos tercero y cuarto contienen el identificador numérico del usuario (UID) y el del **grupo principal** al que éste pertenece (GID).

El campo `varios` puede contener cualquier cosa, el nombre completo, dirección, número de teléfono, etc. Esta información se puede obtener mediante la orden `finger`.

El sexto campo indica el directorio de entrada del usuario y el campo `shell` le dice al sistema qué *shell* deberá ejecutarse cuando el usuario inicie la sesión; si este campo se deja vacío, se ejecutará el *shell* de Bourne.

Un usuario puede pertenecer a más de un grupo. El grupo que aparece en el fichero `/etc/passwd` se llama grupo principal y el resto de los grupos a los que pertenece un usuario, **grupos secundarios.** El fichero `/etc/group` contiene la lista de los grupos que existen en el sistema con el siguiente formato:

```
nombre_grupo:x:GID:lista_de_usuarios
```

El cuarto campo de este fichero contiene la lista de los usuarios que pertenecen a un grupo como secundario. Así el sistema sabe a qué grupos pertenece un usuario.

En un sistema donde esté activado NIS (*Network Information System*) el fichero donde se almacena la información de los usarios se encuentra en el directorio `/var/yp/src`. NIS permite que un conjunto de máquinas conectadas en red compartan la misma información acerca de los usuarios, de forma que un usuario del sistema podrá acceder a cualquiera de las máquinas con un mismo nombre de usuario y contraseña, ya que esta información está centralizada en una de las máquinas. Asimismo cada máquina mantiene sus ficheros `/etc/passwd` y `/etc/group` locales que contienen información sobre usuarios y grupos locales. Estos ficheros tienen como primer campo de la última línea el carácter `+` para indicar que el resto de usuarios están listados en el fichero `/var/yp/src/passwd` y el resto de los grupos en `/var/yp/src/group` en la máquina que mantiene la base de datos NIS (servidor NIS).

[1] Esto es lo tradicional en los sistemas UNIX, sin embargo esto puede llegar a ser peligroso para la seguridad del sistema y en la actualidad algunos sistemas ocultan esta información, guardándose la contraseña codificada en un fichero aparte al cual no tienen acceso los usuarios. En el lugar de la contraseña codificada aparece entonces un `*` en el fichero `/etc/passwd`.

3.2. Permisos de ficheros y directorios

Tanto los ficheros regulares como los directorios pueden llevar asociados tres tipos de permisos: **lectura**, **escritura** y **ejecución**. La orden `ls` seguida de la opción `-l` nos permite conocer los permisos de un fichero. Veamos un ejemplo:

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root root 3215 Nov 28 13:26 /etc/passwd
```

En este caso hemos obtenido información del fichero `/etc/passwd`. La cadena `rw-r--r--` nos indica los permisos asociados al fichero. El permiso de lectura se representa mediante el carácter `r`, el de escritura mediante la `w` y el de ejecución mediante una `x`. Siempre que aparece uno de estos caracteres indica que el permiso está activado, si en su lugar aparece el carácter `-`, el permiso no está activado. Además, los usuarios pueden dividirse en tres clases: el propietario, el grupo y el resto de los usuarios. A cada clase de usuarios se le asocia un patrón del tipo `rwx` que indica lo que un usuario perteneciente a esa clase puede hacer con el fichero, como se muestra en la figura 3.1.

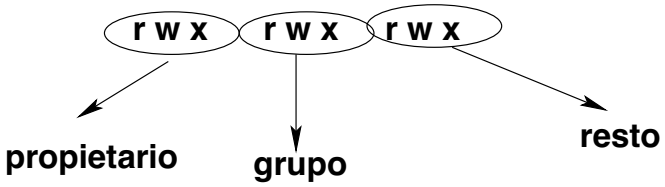


Figura 3.1: Permisos de un fichero

Para el fichero de nuestro ejemplo, el propietario (`root`) tiene permiso de lectura y escritura, los usuarios que pertenecen al grupo (`root`) sólo tienen permiso de lectura y lo mismo ocurre con el resto de los usuarios del sistema.

Si un fichero tiene el permiso de lectura activado se nos va a permitir examinar su contenido, si tiene activado el de escritura podremos modificarlo, y por último, el permiso de ejecución nos permite ejecutarlo, si se trata de un programa.

Los directorios también llevan permisos asociados. Se necesita permiso de lectura para ver el contenido de un directorio, permiso de escritura para añadir o borrar ficheros de un directorio, y permiso de ejecución para utilizarlo como parte de un camino.

Un directorio es un fichero que contiene información sobre otros ficheros. Esta información es básicamente una tabla de nombres de ficheros y su

localización en el disco (número índice).

`.`	104
`..`	200
`.profile`	2033
`bin`	130
`file1`	1025

Si pensamos que un directorio es un fichero que contiene esta información, podemos entender mejor cómo funcionan los permisos de lectura y escritura. El permiso de lectura nos permite ver el contenido del directorio; esto es justamente lo que hace la orden `ls`. El permiso de escritura permite a los programas realizar operaciones que alteran el fichero directorio. Borrar un fichero (`rm`), cambiar su nombre (`mv`) o crear uno nuevo, son operaciones que modifican el contenido del directorio y, por tanto, para realizarlas necesitamos permiso de escritura.

El permiso de ejecución de un directorio nos puede parecer un poco más extraño al principio. Se le suele llamar **permiso de búsqueda**, ya que nos permite buscar en el directorio el número de nodo índice que le corresponde al fichero, a partir de su nombre. Para ilustrar esto veamos cómo procede el sistema operativo GNU/LINUX cuando se abre un fichero. Cuando se le proporciona un nombre de fichero, el sistema debe localizar a partir de éste sus bloques en el disco. Supongamos que le damos el nombre `/usr/ast/mbox`.

En primer lugar busca `usr` en el directorio raíz con el fin de hallar el nodo índice del fichero `/usr`. A partir de este nodo-i, el sistema localiza el directorio `/usr` y busca el siguiente componente en él, `ast`. Cuando ha encontrado la entrada de `ast`, ésta tiene el nodo-i del directorio `/usr/ast`. A partir de este nodo-i se puede hallar el contenido del directorio y buscar `buzon`. Con este nodo-i podemos acceder a los bloques del fichero deseado. Este proceso de búsqueda se ilustra en la figura 3.2.

Los nombres de ruta relativos se buscan de la misma forma que los absolutos, sólo que comenzando desde el directorio de trabajo y no desde el directorio raíz. Todo directorio tiene entradas para los ficheros `.` y `..` que se colocan ahí cuando se crea el directorio. La entrada `.` tiene el número de nodo-i del directorio actual y la entrada `..` tiene el número de nodo-i del directorio padre. Por tanto, un procedimiento que busca a `../fps/prog.c` simplemente busca a `..` en el directorio de trabajo, halla el número de nodo-i del directorio padre y rastrea ese directorio hasta encontrar `fps`.

Hay que hacer notar que sólo cuando hay que abrir un fichero entran en juego sus permisos. Las órdenes `rm` y `mv` sólo requieren permisos de búsqueda y escritura en el directorio; los permisos del fichero no importan. Esto es

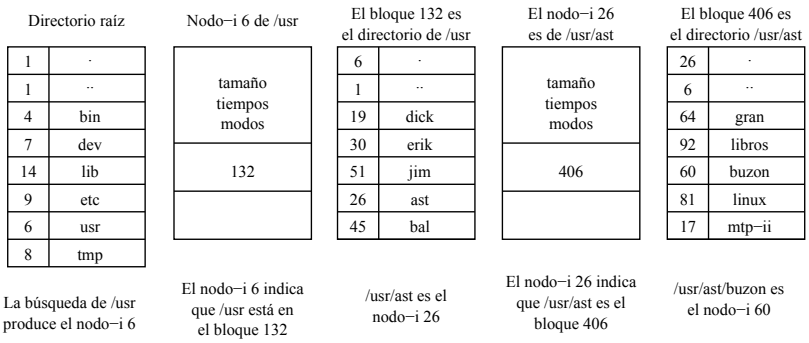


Figura 3.2: Las etapas en la búsqueda de `/usr/ast/buzon`

importante, porque alguien que no tenga acceso de lectura o escritura a nuestros ficheros puede borrarlos o reemplazarlos si tiene permiso de búsqueda y escritura en el directorio.

En el cuadro 3.1 se muestra un resumen de los permisos que deben tener activados los ficheros o directorios implicados para la realización de diferentes operaciones.

	Fichero origen	Fichero destino	Directorio origen	Directorio destino
`cp` (destino no existe)	r	-	x	wx
`cp` (destino existe)	r	w	x	x
`mv`	-	-	wx	wx
`ln`	-	-	x	wx
`ln -s`	-	-	-	wx
`rm`	-	-	wx	-

Cuadro 3.1: Permisos necesarios para hacer operaciones con ficheros

¿Qué permisos tiene activados un fichero cuando se crea? En principio un fichero regular tendrá activados los permisos de lectura y escritura para los tres tipos de usuarios, y los directorios tendrán activados todos los permisos (lectura, escritura y ejecución). Dado que esto no es conveniente desde el punto de vista de la seguridad del sistema, existe una orden que nos permite establecer limitaciones a los permisos de creación de los ficheros. Se trata de la orden `umask`, que estudiaremos con más detenimiento en el apartado 3.4. El administrador del sistema suele establecer mediante esta orden unos permisos adecuados para nuestros ficheros.

Ejercicios:

1. Suponga el siguiente fichero:

```
---xrwxrw- 1 juan alum 31 Jul 10 12:12 fichero
```

   ¿Qué operaciones puede realizar `juan` sobre `fichero`? ¿Y alguien que pertenezca al grupo `alum`?

3.3. Cambio de permisos

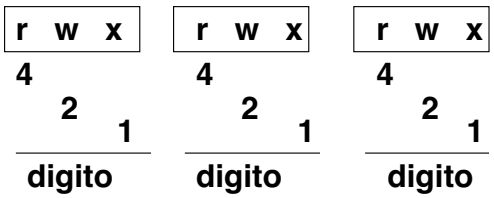
Para cambiar los permisos asociados a un fichero se utiliza la orden `chmod` cuyo formato es

`chmod` *modo fichero* ...

donde *modo* es la especificación de los permisos que se van a aplicar al *fichero*. El modo se puede especificar de dos formas, como números octales o por descripción simbólica; a continuación vamos a ver los dos métodos.

3.3.1. Números octales

El modo se representa como un número octal de tres dígitos, obteniéndose cada uno de ellos mediante la suma de los equivalentes numéricos de cada permiso.



Ejemplo:

```
$ chmod 640 prueba
```

Esta línea de órdenes hace que los permisos del fichero `prueba` *queden de la siguiente forma:* `rw-r---`

3.3.2. Descripción simbólica

El formato de la descripción simbólica del modo es:

[*quién*] *op permiso*

quién es una combinación de las letras u (para los permisos del propietario), g (grupo) y o (otros). Puede utilizarse una a para representar a los tres a la vez (ugo), y éste es el valor predeterminado.

op puede ser un + para añadir un permiso que actualmente no se tiene, un - para quitar un permiso que se tiene, o un = para asignar de forma absoluta un permiso.

permiso es una combinación de las letras r, w, x, s y t (s representa a los permisos *set user id* o *set group id* y la t al *sticky bit*, que estudiaremos más adelante).

Ejemplos:

1. `$ chmod a=rw prueba`
   `$ ls -l prueba`
   `-rw-rw-rw- .....`
   Se otorga a todos los usuarios los permisos de lectura y escritura.
2. `$ chmod go-w prueba`
   `$ ls -l prueba`
   `-rw-r--r-- .....`
   Al grupo y al resto de usuarios se le quitan el permiso de escritura.
3. `$ chmod u+x,g+w prueba`
   `$ ls -l prueba`
   `-rwxrw-r-- .....`
   Le damos permiso de ejecución al propietario y de escritura al grupo.

Hay que hacer notar que **los permisos de un fichero sólo pueden ser cambiados por su propietario o por el superusuario.**

Ejercicios:

1. ¿Qué significa que un fichero tenga los permisos 751? Represéntelos como una cadena de permisos. Dé los permisos de forma simbólica al fichero `prueba` para que tenga los permisos anteriores. ¿Cuáles son los requisitos para cambiar los permisos a un fichero?

2. Suponga que tenemos un fichero `permisos` sobre el que vamos realizando la siguiente secuencia de órdenes de forma consecutiva. Indique, de forma razonada, qué permisos tendrá tras la ejecución de cada una de ellas

   a) `$ chmod 365 permisos`

   b) `$ chmod g-r permisos`

   c) `$ chmod o=x permisos`

   d) `$ chmod u+r permisos`

3.4. La orden `umask`

La orden `umask` sirve para establecer los permisos con los que se crearán los ficheros y directorios a partir del momento en que la ejecutemos. El administrador del sistema puede establecer la máscara que desee poniendo la orden `umask` en un fichero que se lea al iniciar la sesión. Su formato es:

`umask` [*modo*]

donde *modo* indica la máscara predeterminada. Ésta se puede dar de forma octal, en cuyo caso indica los permisos que no se van a establecer, y en forma simbólica, que indica los permisos que se van a establecer.

Un ejemplo de máscara predeterminada podría ser `rw-r-----` para los ficheros regulares, y `rwxr-x---` para los directorios, correspondiendo esto a la orden `umask 027`. ¿A qué es debida esta diferencia? A un fichero regular no se le activa nunca el permiso de ejecución al crearlo; si queremos que lo tenga deberemos dárselo utilizando la orden `chmod`. Observe que la orden `umask` trabaja de forma opuesta a `chmod`; los valores que le damos le dicen al sistema qué permisos no se deben dar cuando se crea el fichero.

Un usuario podrá dar la orden `umask` en cualquier momento si quiere modificar su máscara por omisión, a partir de ese instante los ficheros que cree tendrán la nueva máscara de permisos. Esta orden será válida hasta que finalice la sesión. Si quiere que su máscara siempre sea esa tendrá que colocar la orden `umask` en un fichero que se lea al comienzo de la sesión.

Ejemplo: Damos la orden `umask 077` ¿Qué máscara de permisos tendrán los ficheros que se creen a continuación?

Si creamos un fichero regular tendrá los permisos: `rw----`*, y si creamos un directorio:* `rwx---`.

Ejercicios:

1. Supongamos que damos la orden `$ umask 026`, ¿qué permisos tendrán activados los ficheros ordinarios y directorios que se creen a partir de este momento?
2. ¿Qué diferencia hay entre dar la orden `umask` y `umask -S`?

3.5. Permisos especiales

Aparte de los permisos que ya hemos estudiado, existen otros sobre los que no hemos hablado todavía, los permisos SUID (*set user id*), SGID (*set group id*) y el bit *sticky*.

3.5.1. El bit SUID

Cuando se ejecuta un programa, al proceso creado se le asignan cuatro números que indican a quién pertenece el proceso. Estos son los UID y GID reales y efectivos. Normalmente, los UID y GID efectivos son los mismos que los reales. El sistema GNU/Linux utiliza los UID y GID efectivos para determinar los permisos de acceso de un proceso a los ficheros.

GNU/Linux determina los permisos de acceso de un proceso a un fichero de la siguiente forma; si el UID efectivo de un proceso es el mismo que el UID del propietario del fichero, entonces ese proceso tiene los permisos de acceso al fichero del propietario; si no es así, si el GID efectivo del proceso concuerda con el GID del grupo asociado con el fichero, entonces el proceso tiene los permisos de acceso del grupo; si no es así, el proceso tiene los permisos de acceso de los otros. Esto lo podemos ver en el algoritmo 3.1.

Algoritmo 3.1 Acceso de un proceso a un fichero

```
si UID_efectivo == UID_del_fichero
entonces
   Acceder al fichero como su propietario
si no si GID_efectivo == GID_del_fichero
entonces
   Acceder al fichero como miembro del grupo
si no
   Acceder al fichero como el resto de usuarios
fin si
```

El establecimiento del permiso SUID en un programa cambia su compor-

tamiento. Cuando este permiso está establecido, todos los procesos creados por ese programa tendrán el UID efectivo del propietario del programa y no el del usuario que lo ejecuta. Puesto que los permisos de acceso a los ficheros son determinados a partir del UID efectivo, no del real, el proceso correspondiente a un programa con el permiso SUID activado tiene los mismos permisos de acceso que el propietario del programa, sin importar quién lo ejecute.

Para comprender mejor su funcionamiento veamos un ejemplo. Consideremos los siguientes ficheros:

```
-rw-r--r--    1 root     root      /etc/passwd
-rwsr-xr-x    1 root     root      /usr/bin/passwd
```

El fichero `/usr/bin/passwd` es el programa ejecutable que nos permite cambiar la contraseña. Como ya sabemos, ésta se almacena codificada en el fichero `/etc/passwd`. Si nos fijamos en los permisos de este último (`-rw-r--r--`) podemos observar que sólo el propietario (`root`) tiene permiso de escritura. ¿Cómo es posible entonces que cualquier usuario pueda cambiar su contraseña y modifique el fichero `/etc/passwd`? Esto es posible gracias al permiso SUID que posee el fichero `/usr/bin/passwd`, que aparece como una `s` en el campo de ejecución del propietario. De este modo cuando cualquier usuario lo ejecute (tiene permiso de ejecución para todos los usuarios), su UID efectivo se hará igual al del propietario del fichero; en este caso, al del usuario `root`. Así cuando el proceso creado tenga que modificar el fichero `/etc/passwd` tendrá permiso de escritura en él.

Es importante tener en cuenta que el permiso SUID es independiente del de ejecución, de forma que el fichero ejecutable deberá tener ambos permisos. En el caso de que tengamos un fichero con el SUID activado y no tenga permiso de ejecución para el propietario, aparecería una `S` en el campo de ejecución del propietario.

Para establecer el permiso SUID se utiliza la orden `chmod` en notación simbólica:

`chmod u+s` *fichero*

o bien en notación octal[2]:

`chmod` 4*xxx* *fichero*

[2]Observe que ahora el modo se representa como un número octal de cuatro dígitos, donde el primero se emplea para los permisos especiales y el resto para el propietario, grupo y resto de usuarios.

Para quitarlo también se usa la misma orden:

`chmod u-s` *fichero*

3.5.2. El bit SGID

Al igual que hay un UID efectivo también existe un GID efectivo que se comporta de forma similar. Cuando un programa tiene el permiso SGID establecido, el proceso se ejecuta con los derechos de acceso del grupo asociado con el fichero.

El permiso SGID se establece mediante la orden `chmod` en notación simbólica:

`chmod g+s` *fichero*

o bien en notación octal:

`chmod` 2*xxx* *fichero*

El SGID se borra de la misma forma que el SUID:

`chmod g-s` *fichero*

El permiso SGID se puede activar también en los directorios, teniendo un significado diferente al que hemos visto. Esto se estudiará en el apartado 3.6.

3.5.3. El bit *sticky*

Nos queda un último permiso del que hablar: el bit *sticky*. Este bit se representa mediante una `t` en la máscara de permisos, apareciendo en el campo de ejecución de los otros. Se aplica a directorios de uso público, es decir, aquellos que tienen todos los permisos activados y por tanto, todo el mundo puede crear y borrar ficheros, esto puede ser un poco peligroso porque un usuario puede borrar los ficheros de otros; para evitarlo se activa el bit *sticky*. Con esto se consigue que cada usuario sólo pueda borrar los ficheros que son de su propiedad. Un directorio al que se le suele activar el bit *sticky* es `/tmp`.

Para establecerlo podemos dar:

`chmod` 1*xxx* *directorio*

o bien

`chmod o+t` *directorio*

De forma análoga a los permisos anteriores, para desactivarlo podemos dar:

`chmod o-t` *directorio*

3.6. Cambiar el propietario y grupo de un fichero

Mediante la orden `chgrp` se puede cambiar el identificador de grupo de uno o varios ficheros o un directorio. El formato de esta orden es el siguiente

`chgrp` [*opciones*] *grupo fichero* ...

donde *grupo* puede ser un número de identificador de grupo o un nombre de grupo y *fichero* se puede expresar como un camino absoluto o relativo.

Para poder cambiar el identificador de grupo de un fichero hay que ser el superusuario, o bien ser el propietario del fichero y pertenecer al nuevo grupo.

Cuando cambiamos el identificador de grupo de un directorio, los ficheros que están debajo de él no cambian de grupo, a menos que especifiquemos la opción `-R` de la orden `chgrp`. Los ficheros que se creen en ese directorio después de haber cambiado su grupo pertenecerán al nuevo grupo siempre que el directorio tenga activado el permiso SGID. Si no lo tiene, los nuevos ficheros creados pertenecerán al grupo principal del usuario que cree dichos ficheros.

También se puede cambiar el propietario de ficheros o directorios mediante la orden `chown`. La forma de uso de esta orden es:

`chown` [*opciones*] *usuario*[:*grupo*] *fichero* ...

donde *usuario* es el nuevo propietario del *fichero* y puede expresarse como nombre de *login* o UID, el *grupo* puede expresarse como nombre de grupo o GID.

Esta orden sirve para cambiar el propietario y el grupo de un fichero, pero sólo el superusuario puede cambiar el propietario. Un usuario normal sólo puede usarla para cambiar el grupo.

Cuando cambiamos el grupo o el propietario de un fichero mediante estas órdenes, se elimina automáticamente el permiso SUID si estaba activado.

Ejercicios:

1. Al ejecutar la orden `chmod 6621 ejercicio`, ¿qué permisos tendrá el fichero `ejercicio`?
2. Supongamos que tenemos la siguiente situación:

```
drwxrwxrwt  root  system  /usr/publico
-rw-------  pepe  system  /usr/publico/examen
-rw-rw-rw-  juan  system  /usr/publico/apuntes
-rw-r--r--  luis  system  /usr/publico/memoria
```

   Conteste a las siguientes preguntas, razonando la respuesta:

   a) ¿Podría borrar `juan` el fichero `/usr/publico/examen`?

   b) ¿Podría borrar `luis` `/usr/publico/apuntes`?

   `juan` y `luis` pertenecen al grupo `alumnos`.

3.7. Los permisos y las órdenes cp, mv y rm

cp Para poder copiar un fichero, éste debe tener el permiso de lectura activado. El comportamiento de la orden depende de la existencia del fichero de destino. Así, cuando se usa `cp` para copiar un fichero, y el fichero de destino no existe, los permisos del fichero fuente se copian también. Esto incluye al permiso SUID, sin embargo el SGID se elimina. Es decir, los permisos del fichero de destino serán iguales a los del fichero fuente, salvo si la máscara por omisión que tenemos definida para nuestros ficheros es diferente (más restrictiva), en cuyo caso le quitará aquellos permisos que se indique en esta máscara predeterminada, y el SGID. El propietario del nuevo fichero será el usuario que da la orden.

Resulta particularmente peligroso copiarnos un programa perteneciente a otro usuario, que tenga activado el bit SUID, puesto que la copia también tendrá activado este bit y nos pertenecerá.

Cuando el fichero de destino existe, tanto sus permisos como el propietario no cambian; el contenido del fichero fuente se copia en el de destino (suponiendo, por supuesto, que el fichero tiene permiso de escritura para el usuario que da la orden `cp`).

`mv` Para poder mover un fichero debemos tener permiso de escritura en el directorio donde está catalogado, los permisos del fichero no influyen en la operación. Su comportamiento depende de si los ficheros fuente y destino residen o no en el mismo sistema de ficheros.

En el primer caso, la orden `mv` simplemente cambia el nombre del fichero, no su contenido, permisos o propietario.

En el segundo, si la orden la da un usuario distinto del propietario del fichero fuente, el de destino tendrá como propietario el que da la orden. Los permisos del fichero de destino serán los mismos que los del fuente.

`rm` Para borrar un fichero no influyen los permisos de éste, debemos tener permiso de escritura en el directorio donde esté catalogado.

Cuando se borra un fichero que tiene el permiso SUID activado, es recomendable asegurarse de que sólo tenga un enlace, ya que cuando se borra el fichero lo que se van borrando son los distintos enlaces. Si borramos un fichero con SUID y existe un enlace al mismo en otro directorio, el fichero sigue existiendo; para asegurarnos de que nadie lo pueda usar lo que podemos hacer es cambiarle los permisos (quitarle SUID) antes de borrarlo, ya que al cambiar los permisos de un fichero cambian los de todos sus enlaces.

Ejercicios:

1. Supongamos que tenemos un fichero `enunciado`. A continuación damos la orden `chmod 751 enunciado`, y posteriormente la orden `cp enunciado solucion`. ¿Cómo afectaría la orden `umask 222` si se intercala entre las dos anteriores? Razone la respuesta.

3.8. Ejercicios

1. ¿Cómo puede saber cuál es su UID y GID? ¿Y los permisos asociados a un fichero?

2. ¿Podría ejecutar un programa que está en un directorio para el cual no tiene permiso de lectura? Justifique la respuesta.

3. Diga qué permisos son necesarios para borrar un fichero de un directorio. Razone su respuesta.

4. Estructura del fichero `/etc/passwd`. Importancia de este fichero en el sistema UNIX. ¿Sería muy grave que el fichero `/etc/passwd` tuviera como máscara de protección la siguiente: `rw-rw-rw-`? Explíquelo.

5. Suponga que está en su directorio principal y da la orden:

   ```
   $ chmod 000 .
   ```

   ¿Qué ocurriría durante la sesión? ¿Afectará esta orden al establecimiento de sesiones posteriores?

6. Considerando los permisos asociados al fichero `calc` ¿ve algún inconveniente en los permisos que se le han dado?

   ```
   -rws-wx-wx user1 grupo1 ....  calc
   ```

7. Existe en UNIX una orden llamada `chgrp`. El fichero `chgrp` se encuentra en el directorio `/usr/bin` y al hacer un listado largo del mismo se obtiene:

   ```
   -rwsr-xr-x 1 root 38912 Oct 19 1988 chgrp
   ```

   Explique la máscara de protección de este fichero.

8. A continuación se da una lista de ficheros con los permisos que llevan asociados, el nombre de su propietario y el grupo al que pertenecen los ficheros

   ```
   drwxr-xr--  root  als /home/als
   drwxr-xr-x  luis  als /home/als/luis
   drwxr-x--x  luis  als /home/als/luis/texto
   -r--rw-rw-  luis  als /home/als/luis/texto/notas
   ---xrwxrw-  luis  als /home/als/luis/texto/exe
   ```

`luis` y `pepe` son dos usuarios pertenecientes al grupo `als`.

Conteste de forma razonada si se puede hacer lo siguiente:

a) ¿Podrá `luis` desde el directorio `/home/als/luis/texto` hacer las siguientes acciones?

I. `cat nota?`
II. `rm notas`
III. Ejecutar el fichero `exe`
IV. Añadir una línea al fichero `notas`

b) ¿Y el usuario `pepe` desde `/home/als/luis/texto`?

I. `rm notas`
II. `cp /dev/null notas`
III. ejecutar el fichero `exe`
IV. `cat notas`

c) ¿Encuentra algún problema a la máscara de permisos del fichero `exe`?

9. Considere la siguiente información:

```
drwxr-x---  ppp  alum  /home/alum/ppp
drwxr-x---  ppp  alum  /home/alum/ppp/juegos
-rwsr-x--x  ppp  alum  /home/alum/ppp/juegos/tetris
-rw-r-----  ppp  alum  /home/alum/ppp/juegos/puntos
drwxr-x---  jjj  alum  /home/alum/jjj
```

a) ¿Podrá el usuario `jjj` del grupo `alum` copiar sin problemas el fichero `/home/alum/ppp/juegos/tetris` a su directorio de entrada? Justifique su respuesta. Si no lo pudiera hacer, ¿qué permisos habría que modificar para poder hacerlo?

b) Suponiendo que ya se ha copiado el fichero, ¿qué permisos tendrá y cuál será el propietario del nuevo fichero? ¿Cambiaría algo si hubiera dado antes la orden `umask 420`?

c) El programa `tetris` intenta escribir la puntuación obtenida en `/home/alum/ppp/juegos/puntos`. ¿Tendrá problemas el usuario `jjj` cuando ejecute el programa que acaba de copiarse para acceder a dicho fichero?

10. A continuación se da una lista de ficheros junto con los permisos que llevan asociados, el nombre de su propietario y el grupo al que pertenecen los ficheros:

```
drwxr-xr-x  root  alum  /home
dr-xr-x---  korn  alum  /home/korn
-r-xr-----  korn  alum  /home/korn/.profile
drwxr-----  korn  alum  /home/korn/C
-rw-r--r--  korn  alum  /home/korn/C/cal.c
-rwx--x--x  korn  alum  /home/korn/C/cal
```

`korn` es del grupo `alum`.

`ritchie` es del grupo `alum`.

`root` es del grupo `system`.

`thompson` es del grupo `system`.

¿Es posible lo siguiente?

a) Para el usuario `korn` en el directorio `/home/korn`
   I. `rm -f .profile`
   II. `mv .profile .kshrc`

b) Para el usuario `ritchie` en el directorio `/home/ritchie`
   I. ejecutar el fichero `~korn/C/cal`
   II. `more ~korn/.profile`

c) Para el usuario `root` en el directorio `/usr`
   I. ejecutar el fichero `~korn/C/cal`

11. Justifique la necesidad de que un sistema como UNIX tenga un permiso del tipo SUID con ejemplos que conozca.

12. A continuación se da una lista de ficheros junto con los permisos que llevan asociados, el nombre de su propietario y el grupo al que pertenecen.

```
drwxr-xr-- root  alf /home/alf
drwxr-xr-x perez alf /home/alf/perez
drwx------ fdez  alf /home/alf/fdez
drwxr--r-- perez alf /home/alf/perez/Cobol
-rws------ perez alf /home/alf/perez/Cobol/nomina
dr-x--x--- fdez  alf /home/alf/fdez/Cprog
-rwx--x--x fdez  alf /home/alf/fdez/Cprog/juego
-rw-r----- fdez  alf /home/alf/fdez/Cprog/juego.c
```

Responda razonadamente:

a) ¿Podría el usuario `perez` en su directorio `Cobol` hacer lo siguiente?

I. ejecutar el fichero `nomina`
II. `rm nomina`
III. `ls -l /home/alf/fdez/Cprog`

b) ¿Podría el usuario `fdez` en su directorio `Cprog` hacer lo siguiente?

I. `rm juego`
II. `ls -lg /home/alf/perez/Cobol`

13. Dados los siguientes ficheros y directorios:

```
drwxr-xr-x  manuel  al93  /home/a93/manuel
drw-r-x---  manuel  al93  /home/a93/manuel/dir1
-r--r-----  manuel  al93  /home/a93/manuel/dir1/fich1
drwxr-xr-x  manuel  al93  /home/a93/manuel/dir2
```

Conteste razonadamente a las siguientes preguntas: ¿Podría el usuario `manuel` hacer las siguientes acciones estando situado en su directorio de entrada?

a) `cp dir1/fich1 dir2/fich2`
b) `rm dir1/fich1`
c) `rm fich1`, el usuario se encuentra ahora en el directorio `dir1`
d) editar el fichero `fich1` para añadir una línea
e) editar el fichero `fich1` para añadir una línea, suponiendo que el usuario `manuel` se encuentra en su directorio `dir1`

14. Suponga que usted tiene por nombre de usuario `user1`, un compañero suyo tiene de nombre de usuario `user2`, y posee un fichero llamado `fich1` que tiene activado el permiso de escritura para el grupo y para los otros, pero no tiene permiso de lectura para ninguno de esos grupos. ¿Qué línea de órdenes daría usted para cambiar los permisos de dicho fichero, para que tenga permiso de lectura?

15. Suponga que tiene un fichero llamado `units` cuya máscara de permisos es: `rw-r--r--`. Su máscara por omisión para la creación de ficheros es 027. Si da la orden `cp units unidades` ¿qué permisos tendrá el nuevo fichero `unidades`?

16. Créese un enlace simbólico a un fichero ya existente. ¿Qué permisos tiene el nuevo fichero? Si utiliza la orden `chmod` para cambiar los permisos de este nuevo fichero, ¿qué ocurre?. Explique la respuesta.

17. Un usuario de UID=7 y GID=9 sitúa en su directorio `$HOME/bin` un programa llamado `cpal`. Este programa accede durante su ejecución a un fichero llamado `tpal` para modificarlo, que está situado en el mismo directorio. Los permisos asociados a dichos ficheros y al directorio son los siguientes:

```
rwx--x--x $HOME/bin
rwx--s--x $HOME/bin/cpal
rwx---r-- $HOME/bin/tpal
```

    a) ¿Podría un usuario de UID=18 y GID=9 ejecutar el fichero `cpal` sin problemas?
    b) ¿Qué ocurriría si el usuario de UID=21 y GID=15 intentara ejecutar el programa `cpal`?

18. Dados los ficheros:

```
drwxr-xr-x  root  al93  /home/al93
drwxr-xr-x  juan  al93  /home/al93/juan
drwxr--r--  juan  al93  /home/al93/juan/docs
-rw-rw-rw-  juan  al93  /home/al93/juan/docs/apuntes
```

    ¿Podría el usuario `pepe` perteneciente al grupo `al92` dar las órdenes siguientes?

    a) `ls ~juan`
    b) `ls ~juan/docs`
    c) `rm ~juan/docs/apuntes`

19. ¿Qué orden hay que dar para ver la máscara de permisos de forma simbólica?

20. Existen en nuestro sistema dos ficheros cuyos permisos, propietario y grupo se dan a continuación:

```
-?????????  daemon system /usr/games/rogue
-rw-r--r--  daemon system /usr/games/lib/rogue_roll
```

    El fichero `/usr/games/rogue` contiene el código ejecutable de un juego que se desea pueda ser ejecutado por cualquier usuario del sistema. Durante su ejecución, este programa intenta acceder al fichero `/usr/games/lib/rogue_roll` para escribir en él la puntuación que ha obtenido el usuario que ha jugado. Dé la línea de órdenes necesaria para establecer permisos estrictamente necesarios que debe tener el fichero `/usr/games/rogue` para que lo anterior sea posible. Explique por qué se dan cada uno de los permisos otorgados.

21. Considere los siguientes ficheros y los permisos asociados a los mismos:

```
fichero  propietario  grupo  permisos
prog     pepe         alum   rwxr-x--x
datos    pepe         alum   rw-rw-r--
```

Teniendo en cuenta que el programa `prog` durante su ejecución abre el fichero `datos` para lectura/escritura, conteste a las siguientes preguntas:

¿Podrá la usuaria `ana` ejecutar `prog` sin ningún tipo de problemas?

a) Si `ana` pertenece al grupo `alum`

b) Si `ana` pertenece a otro grupo distinto

Si existe algún problema ¿sería posible solucionarlo sin alterar los bits de protección del fichero `datos`?

22. Suponga que da la siguiente secuencia de órdenes:

```
$ mkdir temp
$ cp /etc/passwd temp
$ chmod 400 temp
$ ls temp
$ ls -l temp
```

¿Cuál será la salida de las dos últimas órdenes? Explique la razón.

23. ¿Qué significa la máscara de creación de ficheros 024?

24. ¿Qué contiene el fichero `/etc/group`? ¿Cómo podemos cambiar el grupo al que pertenece un fichero? Requisitos necesarios. ¿Qué ocurre si el fichero al que cambiamos de grupo es un directorio?

Capítulo 4

Redireccionamientos y filtros

4.1. Introducción

En el capítulo 6 se explicará a fondo el significado de *shell* así como algunas características del *shell* bash. En este capítulo nos centraremos en algunas características de éste relacionadas con los redireccionamientos y los filtros:

1. Manejo de la entrada/salida estándar.
2. Listas de órdenes.
3. Agrupación de órdenes.

4.2. Entrada/Salida estándar

Por convenio, la mayor parte de los programas de GNU/Linux toman su entrada de la entrada estándar, envían la salida que producen a la salida estándar y los mensajes de error a la salida de errores estándar.

Todos los *shells* manejan la E/S estándar básicamente de la misma forma. Cada programa que llamemos tiene inicialmente los tres canales de E/S estándar asignados; así la entrada estándar está asignada al teclado, y la salida estándar y de errores a la pantalla o a la ventana donde se ejecuta la orden (en sistemas X Window).

Cuando es necesario, podemos redirigir la entrada y la salida estándar a un fichero. Es decir, podemos hacer que un programa tome la entrada que necesite de un fichero, o bien que escriba la salida que proporciona en un fichero. Esta es una de las funciones del *shell*.

Asimismo podemos concatenar varios programas mediante interconexiones; en este caso, la salida del primer programa alimenta la entrada estándar del segundo, y así sucesivamente. Esto hace posible emplear las utilidades de GNU/LINUX para construir líneas de órdenes más complejas.

4.2.1. Redirección de la E/S

Una de las funciones del *shell* es gestionar la entrada y salida estándar. Las órdenes no necesitan saber de dónde les viene la entrada o a dónde va la salida, es el *shell* el que establece estas conexiones. Para ello el *shell* bash reconoce una serie de operadores, que podemos ver en el cuadro 4.1.

Símbolo	Función
orden > *fichero*	Escribe la salida de *orden* en *fichero*. Crea *fichero* si no existe, en otro caso lo reescribe.
orden < *fichero*	Toma la entrada para *orden* de *fichero*.
orden1 | *orden2*	Conecta la salida estándar de la *orden1* a la entrada estándar de la *orden2*.
orden >> *fichero*	Añade la salida de *orden* a *fichero*. Crea *fichero* si no existe.
orden 2> *fichero*	Escribe la salida de error estándar en *fichero*. Crea *fichero* si no existe. en otro caso lo reescribe
orden 2>> *fichero*	Añade la salida de errores a *fichero*.
orden &> *fichero*	Redirecciona las salidas estándar y de errores al mismo *fichero*.
orden >| *fichero*	Redirecciona la salida a *fichero*, incluso si éste existe y la opción `noclobber` está activada.

Cuadro 4.1: Operadores de redirección

El operador > *fichero*

La salida estándar de una orden está asignada por omisión al terminal. Usando el símbolo >, la salida estándar de una orden puede ser redirigida a un fichero. El fichero se abre para escritura y se asigna a la salida estándar.

Ejemplo:

```
$ echo hola > p.out
```

```
$ cat p.out
hola
```

La salida de la orden `echo` *se redirecciona al fichero* `p.out`*, de forma que el contenido de éste es la cadena* `hola`.

Si el fichero no existe, se crea. Cuando el *shell* encuentra en una línea de órdenes un redireccionamiento de salida hacia un fichero que ya existe, lo primero que hace es abrir el fichero para escritura. Esto significa que su contenido se borra siempre, fuera el que fuese. Seguidamente, establece como contenido del fichero la información de salida del mandato que redirigimos. Evidentemente esta operación requiere que el fichero tenga los permisos adecuados además de tener en cuenta la opción `noclobber` del *shell*.

Ejemplo:

```
$ chmod 444 p.out
$ echo hola otra vez > p.out
bash: p.out: Cannot clobber existing file
```

(sin permiso de escritura)

El símbolo `>` también se puede utilizar para crear un fichero vacío.

Ejemplo:

```
$ > vacio
$ ls -s vacio
0 vacio
```

La opción `noclobber`

Impide que se destruya la información al redirigir la salida estándar a un fichero existente. La opción se establece con `set -o noclobber`.

se quita con set +o noclobber

Ejemplo:

```
$ ls > fichero.salida
$ chmod 777 fichero.salida
$ set -o noclobber
$ ls > fichero.salida
bash: fichero.salida: Cannot clobber existing file
```

El operador >| *fichero*

El operador >| se usa para forzar la sobreescritura de un fichero aunque está establecida la opción `noclobber`.

Ejemplo:

```
$ ls >| fichero.salida
```

En este caso no tiene en cuenta si la opción `noclobber` *está establecida.*

El operador >> *fichero*

El operador >> *fichero* añade a *fichero* la salida estándar. Si no existe *fichero* lo crea. Opera de modo similar al operador >, excepto en que los datos se escriben al final del contenido actual de *fichero*. El operador >> es útil para acumular la salida de varias ejecuciones consecutivas de una orden en un mismo fichero.

Ejemplos:

1.
```
$ ls -l > lista
```
2.
```
$ cat lista
$ ls -l >> lista
$ cat lista
```

El operador < *fichero*

Para redirigir la entrada estándar de una orden se utiliza el operador <. El fichero se abre para lectura y se asigna a la entrada estándar.

Ejemplo:

```
$ mail juan < carta
```

Hace que el programa `mail` *tome su entrada del fichero* `carta`.

Interconexiones: el operador |

También es posible redirigir la salida de un programa a la entrada estándar de otro en vez de a un fichero. Esto lo podemos hacer mediante una **interconexión** o tubo (*pipe*) y para ello se utiliza el símbolo **|**.

orden1 | *orden2* | ...

El *shell* arranca todas las órdenes simultáneamente. La salida estándar de cada orden, excepto la última, se conecta con la entrada estándar de la siguiente. A la secuencia de una o más órdenes simples separadas por el carácter **|**, se le denomina **tubería** (*pipeline*).

La información escrita en una interconexión se mantiene en la memoria del sistema hasta que otro programa la lee. Normalmente ésta es abierta por dos programas diferentes al mismo tiempo, uno la abre para escritura y el otro para lectura.

Ejemplos:

1. `$ find . -user boabdil | wc -l`
   Muestra en la salida estándar el número de ficheros del directorio de trabajo de los que es propietario el usuario `boabdil`.
2. `$ ls -l | more`
   Muestra en la salida estándar un listado largo del directorio de trabajo de forma paginada.
3. `$ who | wc -l`
   Muestra en la salida estándar el número de usuarios conectados al sistema.

Mediante la interconexión de órdenes podemos construir líneas de órdenes con funciones nuevas.

La orden `tee` La orden `tee` nos suministra un modo de capturar el contenido de una interconexión en un fichero sin interrumpir el flujo de información de la interconexión. Su formato es:

orden1 | `tee` [`-a`] [*fichero* ...] | *orden2*

La orden `tee` hace que la entrada estándar, proveniente de la salida estándar de *orden1*, se copie en la salida estándar, convirtiéndose en la entrada estándar de *orden2*, y también en *fichero*. Si se le pasa la opción `-a` añade la información a éste. en vez de sobreescribir

Ejemplos:

1. `$ who | tee usuarios | wc -l`
   La salida de la orden `who` *se copia en el fichero* `usuarios` *y además pasa por la interconexión a la siguiente orden (*`wc -l`*) que contará el número de líneas de la entrada. Es decir, la entrada estándar se desvía al fichero especificado y continúa a la salida estándar.*
2. `$ cat modelo | tee uno dos tres | lpr`
   Crea tres copias del fichero `modelo` *con los nombres* `uno`, `dos` *y* `tres`*, y lo imprime.*

Descriptores de ficheros

Un descriptor de fichero es un número entero que utiliza el núcleo para identificar los distintos flujos de datos asociados a un proceso. Cuando un proceso arranca, usualmente tiene tres descriptores de ficheros abiertos, los correspondientes a la entrada estándar (0), salida estándar (1), y salida de errores estándar (2). Si un proceso abre ficheros adicionales para entrada o salida, a éstos se les asignan los siguientes descriptores disponibles, empezando por el 3.

A cualquiera de los operadores de redireccionamiento (de entrada/salida) se le puede anteponer como prefijo un número de descriptor de fichero (0-9). Si se especifica éste, es ese descriptor de fichero el que se abre (de lectura/escritura) en vez de la entrada o salida estándar. Si no especificamos descriptor de ficheros alguno, los operadores de redireccionamiento actúan sobre la entrada/salida estándar.

Los operadores 2> *fichero* y 2>> *fichero*

La mayoría de las órdenes escriben sus mensajes de error en la salida estándar de errores. Los mensajes de error de los mandatos se pueden redirigir usando el descriptor 2 con los símbolos de redireccionamiento ya comentados.

Ejemplos:

1. `$ ls tmp 2> error` *Si se produce algún error al ejecutar la orden, éste se escribiré en el fichero* `error`*. Es decir, se ha redirigido la salida de errores al fichero mencionado.*
2. `$ ls -R / 2> /dev/null`
   Cuando no nos interesa ver los mensajes de error, ni queremos almacenarlos en un fichero podemos redireccionar la salida de errores al dispositivo `/dev/null`.

El operador `&>` *fichero*

Redirige la salida estándar y de errores al mismo fichero. También se puede utilizar el operador `>&`, aunque de las dos formas se prefiere la primera.

Ejemplo:

```
$ programa &> salida
```

Redirige tanto la salida estándar como la de errores al mismo fichero.

Ejercicios:

1. ¿Qué hacen las siguientes órdenes?
   a) `$ ls -l lpr | lpr`
   b) `$ cat < cat`
   c) `$ find . -name "*.txt" >> more 2> /dev/null`
   d) `$ more less 2> cat | cat > lpr`
   e) `$ cat fichero | tee contenido | wc -l`

4.3. Listas de órdenes

Una **lista** es una secuencia de una o más tuberías separadas por unos de los operadores siguientes: `;`, `&`, `&&` o `||`, y terminada opcionalmente en uno de los siguientes: `;`, `&`, o `Nueva-Línea`.

A continuación veremos qué significado tienen los operadores `;`, `&&`, `||`. El estudio del operador `&` se deja para el apartado ??.

4.3.1. órdenes múltiples

El *shell* bash permite escribir varias órdenes en una misma línea separándolas con el carácter `;`. El resultado es equivalente a dar las órdenes de forma independiente, cada una en una línea.

Ejemplo:

Compare las salidas que producen:

1. `$ pwd ; ls texto ; echo hola`
2. ```
   $ pwd
   $ ls texto
   $ echo hola
   ```

4.3.2. Ejecución condicional

Cuando un programa termina su ejecución devuelve un valor de salida (status) que indica si se ha ejecutado correctamente o no. Un valor de salida cero significa que su ejecución ha sido correcta, y uno distinto de cero indica una ejecución incorrecta o que se ha producido algún error. Este valor de salida es usado por el *shell* bash mediante dos operadores condicionales simples, `&&` y `||`.

Si se separan dos órdenes mediante el operador `&&`, la segunda sólo se ejecutará si la primera se ejecuta correctamente.

Ejemplos:

1. ```
   $ ls temp && echo fichero temp existe
   temp
   fichero temp existe
   ```
2. ```
   $ rm temp
   $ ls temp && echo fichero temp existe
   ls: temp: No such file or directory
   ```

Si dos órdenes se separan por el operador `||`, la segunda se ejecutará sólo si la ejecución de la primera devuelve un valor de salida distinto de cero, es decir, si no se ejecuta correctamente.

Ejemplo:
   ```

```
$ ls temp || echo fichero temp no existe
ls: temp: No such file or directory
fichero temp no existe
```

4.4. Agrupación de órdenes

El *shell* bash proporciona dos formas de agrupar una lista de órdenes para que sea ejecutada como una unidad, mediante el uso de `{}`, o de `()`.

El *shell* bash trata sintácticamente un grupo como si fuera una orden simple a efectos de interconexiones, redirecciones u orden de evaluación en mandatos condicionales. Es útil cuando se quiere combinar la salida de varias órdenes.

Los caracteres `{ }` tienen que escribirse aislados, es decir, precedidos y seguidos de un espacio. Las órdenes que se incluyan dentro de `{ }`, si se dan en una sola línea deben terminar en `;`.

Ejercicios:

1. Explique la diferencia entre la salida que producen estas órdenes:

```
$ echo El fichero temp: ; cat temp | wc -l
$ { echo El fichero temp: ; cat temp ; } | wc -l
```

La agrupación de órdenes se puede usar también con los operadores de ejecución condicional.

Ejemplo:

```
$ ls temp && { echo Existe ... borrando ; rm temp ; }
```

La utilización de `( )` es similar, pero la lista se ejecuta en un *subshell*, concepto que se estudiará en el apartado ??.

4.5. Concepto de filtro

Los filtros son programas que hacen una selección o transformación del flujo de datos de la entrada estándar o de un fichero, en la salida estándar.

Los filtros están diseñados para leer la entrada estándar, procesarla de alguna manera y escribir el resultado en la salida estándar. Puede pensarse

que la información en bruto entra por un extremo de la orden, obteniéndose en el otro la información filtrada. Una ventaja importante de los filtros es que pueden usarse con redireccionamientos e interconexiones.

4.6. Ordenación de ficheros

La orden `sort` ordena las líneas de un fichero de texto. Como unidad de ordenación utiliza la línea, a no ser que se le indique otra cosa; `sort` ordena según distintos criterios, dependiendo de las opciones elegidas. Su formato es:

`sort` [*opciones*] [*fichero* ...]

Si no se especifica *fichero* se ordena la entrada estándar. Si se especifica un fichero, lo ordena. Si se especifican varios, primero los concatena y después ordena el resultado. Si incluimos en la lista de ficheros el carácter `-`, se tratará la entrada estándar como un fichero.

Ejemplo:

```
$ ls -lR > listavieja
$ who | sort listavieja - > nuevowho
```

`sort` *toma como entrada el fichero* `listavieja` *y la entrada estándar, que es la salida de* `who`*. La salida se almacena en el fichero* `nuevowho`.

Opciones:

`-o` *fichero* Escribe la salida en *fichero*. Puede ser el mismo que uno de los de entrada.

Ejemplo:

```
$ sort -o notas notas
```

Sustituye el fichero original por el fichero ordenado.

`-f` Indica que ignore las diferencias entre las mayúsculas y las minúsculas a la hora de ordenar.

`-n` Indica a `sort` que ordene los números por su valor numérico.

Ejemplo:

```
$ sort -n datos
```

Ordena el fichero `datos` *según el orden numérico normal.*

`-r` Invierte el orden de clasificación.

`-u` Elimina líneas repetidas.

`-d` Especifica ordenación tipo diccionario (letras, dígitos y blancos sólo).

`-b` Ignora los blancos iniciales.

4.7. Selección de caracteres o campos de un fichero

La orden `cut`, selecciona secciones específicas de cada línea de la entrada y las muestra. Su formato es:

`cut` [*opciones*] [*fichero* ...]

Si no se especifica *fichero*, toma su entrada de la entrada estándar; si se especifican uno o varios *ficheros*, tomará su entrada de éstos. Si se da el carácter `-` como *fichero*, éste especifica la entrada estándar.

Opciones:

La orden `cut` nos permite seleccionar caracteres o campos. Para ello proporciona dos opciones `-c` y `-f` que van seguidas de la lista de caracteres o campos que deseamos seleccionar. La lista es una secuencia de números o un rango. El rango se indica con un par de números separados por el carácter `-`. Los números deberán especificarse en orden creciente.

- Cómo seleccionar caracteres:

  `cut -c`*lista* [*fichero* ...]

  Ejemplo:

  ```
  $ cut -c1,3-6  /etc/group
  ```

  Selecciona de cada línea del fichero `/etc/group` *el primer carácter y del tercero al sexto.*

- Cómo seleccionar campos:

  `cut -f`*lista* [`-d`*caracter*] [`-s`] [*fichero* ...]

Ejemplo:

```
$ cut -f5,7-9  casa
```

Selecciona los campos 5.º, 7.º, 8.º y 9.º del fichero `casa`.

Los campos son una serie de caracteres separados, si no se le indica otra cosa, por el carácter `<tab>`. Para especificar un carácter separador de campos diferente, tendremos que dar la opción `-d`.

`-d`*char* *char* indica el carácter a utilizar como separador de campos.

Ejemplos:

1. `$ cut -f2,4 -d: /etc/passwd`
2. `$ who | cut -d" " -f1`
3. `$ cut -f5-9 -d" " otrofichero`

`-s` No muestra aquellas líneas que no contienen el carácter separador de campos.

4.8. Eliminación de líneas duplicadas

La orden `uniq` suprime o informa de las líneas duplicadas consecutivas que contiene un fichero. Su forma de uso es:

`uniq` [`-udc`] [`+`*n*] [`-`*n*] [*fich-entrada* [*fich-salida*]]

Los datos los toma de *fich-entrada* y los escribe en *fich-salida*. Si sólo se indica un fichero, el resultado se manda a la salida estándar. Si no se indica ningún fichero lee de la entrada estándar y escribe en la salida estándar.

Ejemplo:

```
$ uniq repetido unico
```

Elimina las líneas repetidas consecutivas del fichero `repetido` *y envía la salida al fichero* `unico`.

Opciones:

`-u` Sólo muestra aquellas líneas que no están repetidas.

`-d` Sólo muestra las líneas que están repetidas.

`-c` Muestra el número de veces que aparece cada línea en la entrada junto con dicha línea.

`-n` Ignora los primeros *n* campos de una línea. Los campos están separados por espacios o por tabuladores.

`+n` Ignora los *n* primeros caracteres de una línea a la hora de comprobar si está duplicada.

4.9. Borrado de columnas

La orden `colrm` elimina columnas de la entrada. Su formato es:

`colrm` [*columna_inicial* [*columna_final*]]

Si sólo se especifica la *columna_inicial*, borrará desde ésta hasta la última. Para que lea de un fichero habrá que especificar el operador de redirección de la entrada estándar.

Ejemplo:

```
$ colrm 1 3 < datos
```

Muestra en la salida el resultado de eliminar las columnas primera, segunda y tercera del fichero `datos`. *Observe que hemos redirigido la entrada estándar para que lea del fichero.*

4.10. Sustitución caracteres

La orden `tr` copia la entrada estándar en la salida estándar sustituyendo o borrando los caracteres seleccionados. Su uso más común es la conversión de mayúsculas y minúsculas. Su formato es;

`tr` [-d] [*cadena1* [*cadena2*]]

Opciones:

`-d` Borra todos los caracteres que se especifican en la *cadena1* de la salida estándar.

cadena1 Caracteres a sustituir o borrar.

cadena2 Caracteres en que se van a convertir.

Ejemplos:

1. `$ tr "a-z" "A-Z" < fuente`
   Pasa a mayúsculas todas las letras minúsculas del fichero `fuente` *y lo muestra en la salida estándar*.
2. `$ tr "A-Z"  "a-z" < fuente > resultado`
   Pasa a minúsculas todas las letras mayúsculas del fichero `fuente` *y lo graba en el fichero* `resultado`.

Ejercicios:

1. Escriba una línea de órdenes que muestre los nombres de los usuarios del sistema de forma ordenada y en mayúsculas.