Aprendiendo Linux: Gestión de Ficheros

En este tema se pretende describir con rigor algunas de las órdenes de uso más frecuente en UNIX relacionadas con el sistema de …

Tabla de Contenidos

Autor: Lina García Cabrera & Francisco Martínez del Río

Copyright: 
Este material está sometido a las condiciones de una Licencia Creative Commons Attribution-Noncommercial-No Derivative Works 3.0.

1 Introducción

En este módulo se pretende describir con rigor algunas de las **órdenes de uso** más frecuente en UNIX relacionadas con **el sistema de ficheros**. Éstas permiten realizar tareas como **copiar ficheros** dentro de la estructura jerárquica o **buscar la ubicación de un fichero** dentro de la jerarquía de ficheros.

En esta sección se describen algunos conceptos de UNIX, como **los enlaces**, claves para entender este módulo.

1.1 Sistemas de ficheros subsidiarios

UNIX permite que ciertas partes de la jerarquía de ficheros, llamadas **sistemas de ficheros**, residan en dispositivos de almacenamiento o particiones de disco aparte. Hay dos razones para hacerlo:

- En primer lugar, el tamaño del disco u otro dispositivo que contenga los directorios y ficheros puede ser insuficiente para almacenar toda la jerarquía, siendo necesario distribuir la jerarquía en varios dispositivos.
- En segundo lugar, los dispositivos de almacenamiento quizá no estén conectados al ordenador de forma permanente. Un ejemplo de sistema de ficheros es un disco flexible que contiene su propia estructura de ficheros.

UNIX gestiona los sistemas de ficheros relacionándolos con **puntos de montaje**. Un punto de montaje es un

directorio en un sistema de ficheros que se corresponde con el directorio raíz de otro sistema de ficheros.

El ***sistema de ficheros primario*** es el que surge del directorio raíz verdadero y se denomina '**/**'. Los ***sistemas de ficheros secundarios*** se enlazan con el sistema primario a través de la orden **mount**, a la que se le da la trayectoria del punto de montaje y la ubicación del sistema de ficheros secundario. Su efecto es hacer que la raíz del sistema de ficheros secundario corresponda al punto de montaje. Una vez que se ha montado un sistema de ficheros, se puede hacer referencia a cualquier fichero o directorio que contenga utilizando una trayectoria que pase por el punto de montaje. El desmontado (mediante la orden **umount**) de un sistema de ficheros invalida todas las trayectorias que pasan por el punto de montaje.

Usted, como usuario, debe ser consciente de los límites del sistema de ficheros por dos razones:

- En primer lugar, existen restricciones para crear enlaces fuera de un sistema de ficheros (lo veremos a continuación).
- En segundo lugar, cuando se desmonta un sistema de ficheros se pierde todo acceso a los ficheros que contiene hasta que se vuelve a montar.

1.2 Nodos i

En UNIX todo **fichero** tiene asociado una pequeña zona de información en el disco (una estructura de datos) denominada **nodo i**. Un nodo i contiene toda la información importante para gestionar un fichero, entre otros:

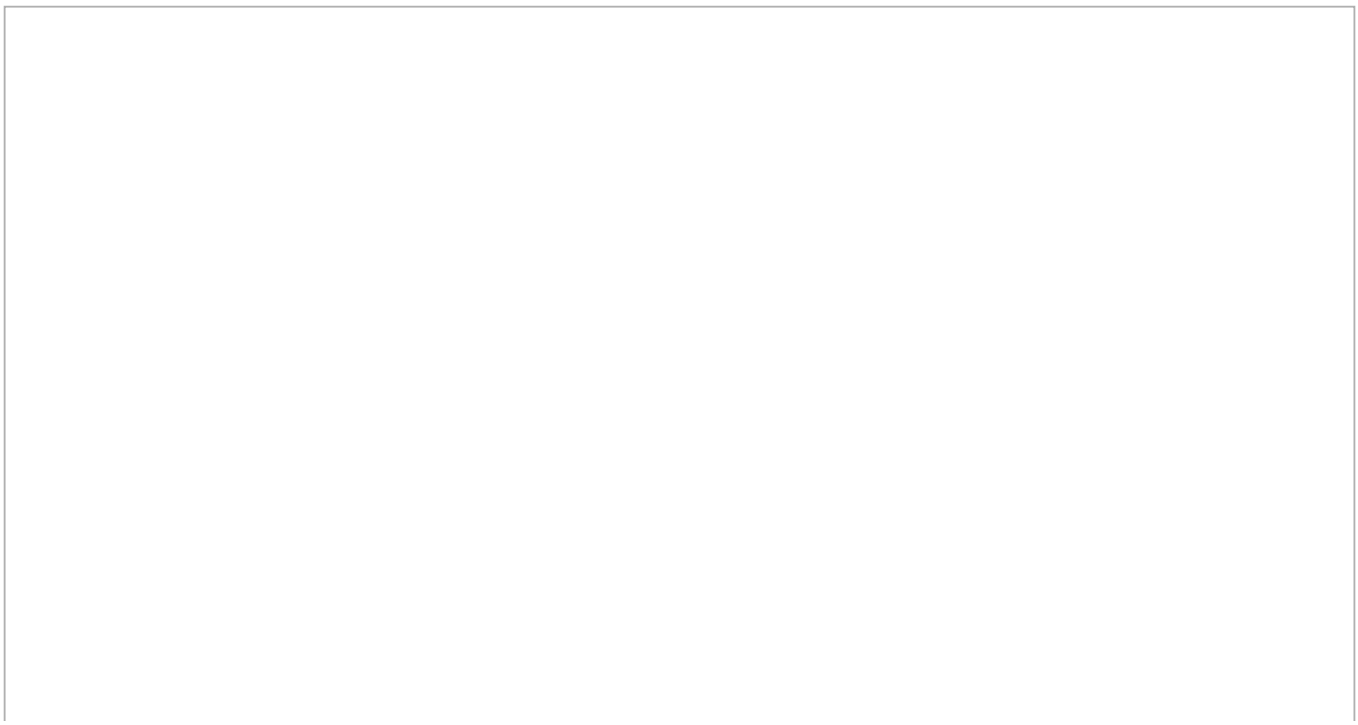
- la **identidad del propietario** del fichero y del grupo (su *uid* y *gid*)
- los **permisos** del fichero
- el **tamaño**
- la **fecha de creación y de última modificación**
- las **zonas del disco** donde se ubica el fichero
- la cantidad o **números de enlaces duros** al fichero

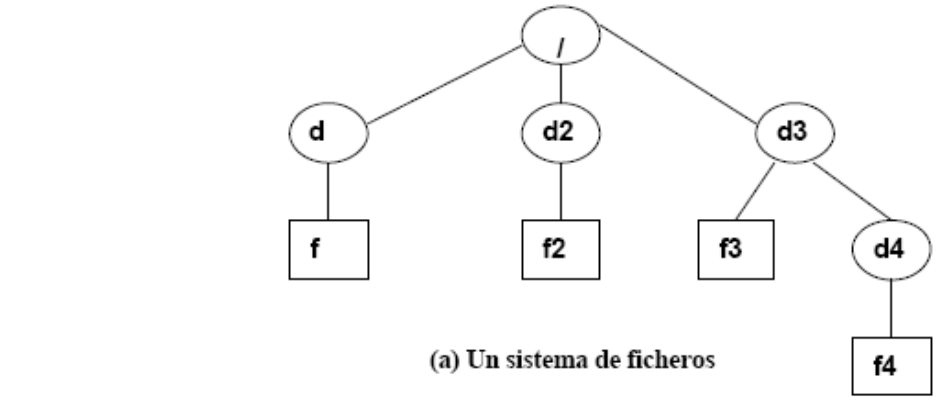
Cada sistema de ficheros tiene su conjunto de nodos i para guardar la información asociada a los ficheros del sistema. Los nodos i se numeran del 0 hacia delante.

1.3 Directorios y enlaces

En UNIX un **directorio** es un **tipo especial de fichero** que contiene una serie de **entradas** llamadas **enlaces a ficheros**. Un ***enlace es un par formado por un nombre***, que nombra al enlace en el directorio, y ***un número de nodo i***, que indica con qué fichero está asociado el enlace (nos dice la estructura de datos que contien información sobre el fichero).

Por ejemplo, la figura 1.b) muestra el contenido de los directorios del sistema de ficheros representado en la figura 1.a).





(a) Un sistema de ficheros

/	
.	0
..	0
d	1
d2	2
d3	3

/d	
.	1
..	0
f	4

/d2	
.	2
..	0
f2	5

/d3	
.	3
..	0
d4	6
f3	7

/d3/d4	
.	6
..	3
f4	8

(b) Contenido de los directorios

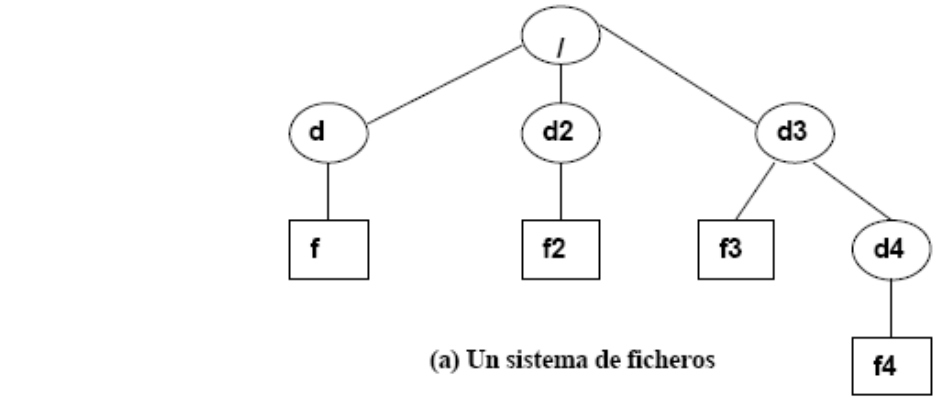
Figura 1. (a) Un sistema de ficheros y (b) el contenido de sus directorios

1.4 Apertura de un fichero

1. Los procesos solicitan el uso de los ficheros mediante una llamada al sistema. A esta solicitud se le llama **apertura del fichero**. La llamada precisa como parámetro una **trayectoria del fichero** que se quiere abrir.
2. Como respuesta a la solicitud el sistema operativo **localiza el nodo i del fichero en el disco** y, si el proceso tiene permisos para abrir el fichero, **carga su nodo i en memoria principal**.
3. A partir de entonces, cuando el proceso solicite leer y/o escribir del fichero el sistema operativo obtiene la información necesaria para realizar la lectura o escritura del nodo i almacenado en memoria principal, en lugar de hacerlo de disco, ya que el tiempo de acceso a memoria principal es inferior al tiempo de acceso a disco.

Veamos ahora ***cómo localiza el sistema operativo el nodo i de un fichero a partir de la trayectoria proporcionada por un proceso***.

> Por ejemplo, supongamos que se emplea la trayectoria ***/d2/f2*** para el sistema de ficheros de la figura.
>
> 1. Como la trayectoria comienza por **/** el sistema operativo inicia su búsqueda en el **directorio raíz**, el cual tiene asociado un nodo i fijo: el 0.
> 2. Se consulta el nodo i número 0 para saber a partir de qué dirección de disco se almacena **/**, tras obtener la dirección se lee el contenido de **/** (obsérvese en la figura 1.b), y se busca la entrada de nombre ***d2***.
> 3. Se accede ahora al nodo i de ***d2*** (el 2) para localizar a partir de qué dirección de disco se almacena su contenido.
> 4. Una vez descubierto se lee ***d2*** y se busca la entrada de nombre ***f2***. Ésta está asociada al nodo i 5.
> 5. Se consultan en el nodo i 5 los permisos del fichero, y si el usuario que ejecuta el proceso tiene permitido el acceso al fichero se guarda el nodo i en memoria principal.



(a) Un sistema de ficheros

/	
.	0
..	0
d	1
d2	2
d3	3

/d	
.	1
..	0
f	4

/d2	
.	2
..	0
f2	5

/d3	
.	3
..	0
d4	6
f3	7

/d3/d4	
.	6
..	3
f4	8

(b) Contenido de los directorios

Figura 1. (a) Un sistema de ficheros y (b) el contenido de sus directorios

Con las trayectorias relativas se trabaja de forma análoga, sólo que la búsqueda comienza en el **directorio de trabajo** del proceso, en lugar de comenzar en el directorio raíz. El nodo i asociado al directorio de trabajo de un proceso se almacena en su descriptor o bloque de control de proceso BCP.

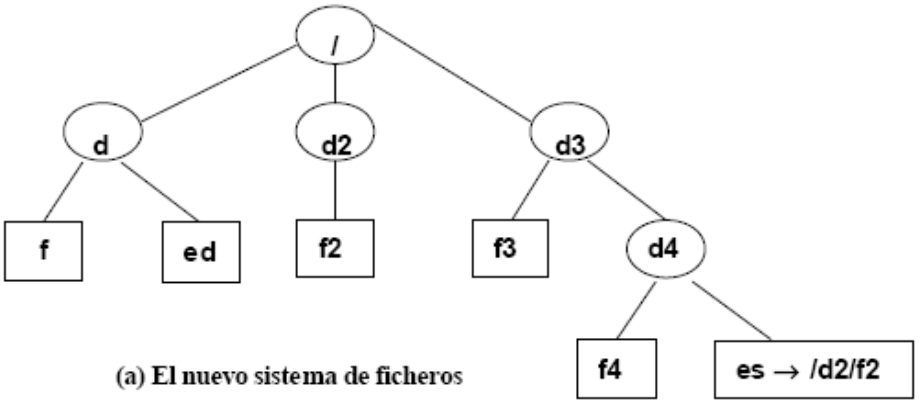
En la figura 1 puede observar que todo directorio posee al menos dos entradas: **.** y **..**, que hacen referencia al **propio directorio** y al **padre de éste respectivamente**. Estas entradas se añaden automáticamente cuando se crea un directorio. Su utilidad reside en permitir **construir trayectorias relativas al directorio de trabajo y al directorio padre** de éste.

> Por ejemplo, si el directorio de trabajo es ***/d3/d4*** la trayectoria ***../f3*** hace referencia a ***/d3/f3***. Piense que si el directorio de trabajo está situado a cierta profundidad en el sistema de ficheros y queremos utilizar una entrada de un "directorio hermano" al directorio de trabajo el uso de una trayectoria que incluya el **..** será mucho más corta que la trayectoria absoluta.

2 Enlace de ficheros

UNIX permite que exista más de un enlace a un fichero o directorio, lo que posibilita el acceso a un mismo fichero a través de distintas trayectorias. Existen dos tipos de enlaces: los enlaces tradicionales o **enlaces duros** (hard o físicos) y los **enlaces simbólicos** (soft o suaves).

> La figura 2.a) representa el mismo sistema de ficheros que el de la figura 1.a), salvo que se han añadido dos enlaces más: ***ed*** y ***es***.
>
> - ***ed*** es un **enlace duro** al fichero asociado al enlace ***f2*** del directorio ***/d2***, es decir, al fichero cuyos datos se almacenan en el nodo i 5. Como puede observar tanto la entrada ***ed*** del directorio ***/d*** como la ***f2*** del directorio ***/d2*** hacen referencia al mismo nodo i: el cinco.
> - Si se tiene en cuenta lo que hace UNIX en el proceso de apertura de un fichero (comentado en la sección anterior) se deduce que la apertura de las trayectorias ***/d/ed*** y ***/d2/f2*** producen el mismo resultado: solicitar el fichero asociado al nodo i 5. En terminología UNIX este fichero posee dos enlaces duros.
>
> Todos los enlaces del sistema de ficheros de la figura 2.a), salvo **es**, son duros. Observe que, por ejemplo, el directorio ***/d3*** posee 3 enlaces duros: el enlace ***d3*** de ***/***, el enlace **.** de ***/d3*** y el enlace **..** de ***/d3/d4***.



(a) El nuevo sistema de ficheros

/	
.	0
..	0
d	1
d2	2
d3	3

/d	
.	1
..	0
f	4
ed	5

/d2	
.	2
..	0
f2	5

/d3	
.	3
..	0
d4	6
f3	7

/d3/d4	
.	6
..	3
f4	8
es	9

(b) Contenido de los directorios

Figura 2. (a) El nuevo sistema de ficheros y (b) el contenido actualizado de sus directorios

El nodo i de un fichero almacena el número de enlaces duros que posee el fichero.

Los **enlaces simbólicos**, como los duros, sirven para aportar trayectorias adicionales a un fichero, pero se implantan de manera distinta, por lo que exhiben propiedades diferentes. Un enlace simbólico hace referencia a un pequeño fichero cuyo contenido es una trayectoria del sistema de ficheros.

> Por ejemplo, en la figura 2.a) es hace referencia a un fichero que se almacena en el nodo i 9, de contenido ***/d2/f2***. Cuando se solicita la apertura de una trayectoria como ***/d3/d4/es*** UNIX procede de forma normal, sin embargo, cuando localiza su nodo i asociado (el 9) se comprueba (leyendo un bit del nodo i) si el fichero se utiliza como enlace simbólico. Si es así UNIX, lee su contenido para obtener la trayectoria enlazada, en el ejemplo ***/d2/f2***, y localizar su nodo i, es decir, el 5. El resultado final es acceder al nodo i 5, y por tanto, abrir el mismo fichero que se hubiera obtenido empleando la trayectoria ***/d2/f2***. Sin embargo, observe que aunque el proceso de apertura de ***/d2/f2*** y ***/d2/d4/es*** produce el mismo resultado, la segunda trayectoria implica una apertura más lenta, pues implica más accesos a disco por parte del sistema operativo.

Otra diferencia entre enlaces duros y simbólicos viene dada por la posibilidad de realizar enlaces a sistemas subsidiarios. No es posible crear un enlace duro a un fichero de un sistema subsidiario. Esto se debe a que **cada sistema de ficheros numera su conjunto de nodos i del cero en adelante**. Si creamos un enlace duro al nodo i 7 nos referimos al nodo i 7 de nuestro sistema de ficheros. Sin embargo, podemos crear un enlace simbólico a un fichero de un sistema subsidiario usando una trayectoria de enlace que pase por un punto de montaje.

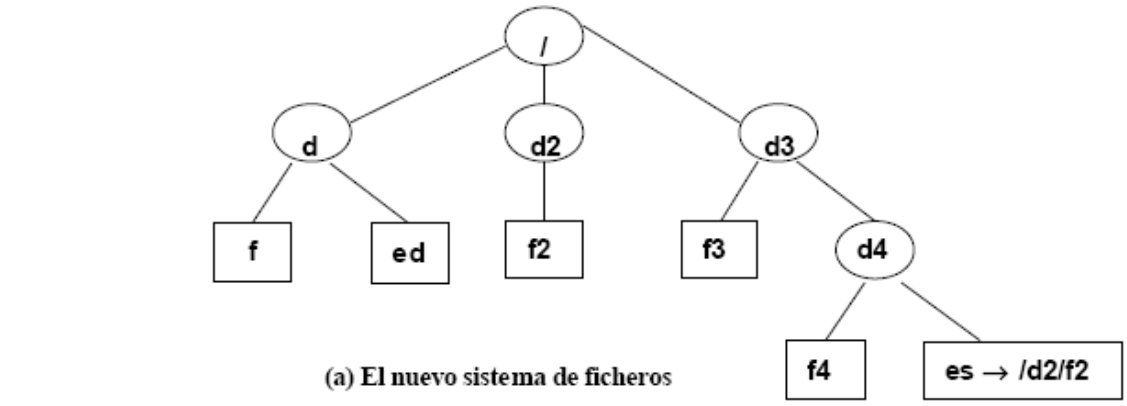
3 La orden ln

La orden **ln** (de *link*) crea un enlace con uno o más ficheros existentes. Tenemos dos formatos de uso:

$ ln [opciones] fich1 fich2	ln -s f2/fich1 fich3
$ ln [opciones] fichero ... dir	ln -s origen destino

En la primera forma se crea un enlace entre *fich2* y el fichero enlazado a *fich1*. Si *fich2* ya era un enlace a un fichero se destruye el enlace antiguo al crear el nuevo.

En la segunda forma se crea un enlace en *dir* con un fichero en fichero ... En este caso, **ln** crea un enlace para cada fichero en el directorio *dir*, asignando al enlace el mismo nombre que tiene el fichero. Como sucede con la primera forma de **ln**, se destruye todo enlace existente que tenga el mismo nombre.



(a) El nuevo sistema de ficheros

/	
.	0
..	0
d	1
d2	2
d3	3

/d	
.	1
..	0
f	4
ed	5

/d2	
.	2
..	0
f2	5

/d3	
.	3
..	0
d4	6
f3	7

/d3/d4	
.	6
..	3
f4	8
es	9

(b) Contenido de los directorios

Figura 2. (a) El nuevo sistema de ficheros y (b) el contenido actualizado de sus directorios

Cuando ln crea un enlace, el nombre del enlace destino tiene el mismo conjunto de permisos y el mismo propietario y grupo que el enlace del fichero fuente.

No se permite crear un enlace entre un directorio de un sistema de ficheros y un fichero de otro sistema, a menos que el enlace sea simbólico. Por omisión **ln** crea enlaces duros y solicita confirmación cuando reemplaza un enlace con un fichero que no tiene permiso de escritura. Se puede cambiar este comportamiento con las siguientes opciones de la línea de órdenes:

- **-s** Crea enlaces simbólicos en lugar de enlaces duros.
- **-f** No solicita confirmación, incluso si se reemplaza un enlace sin permiso de escritura.

4 Traslado de ficheros: la orden mv

La orden **mv** pasa uno o más ficheros de un directorio a otro. Si los dos directorios están en el mismo sistema de ficheros, lo único que hace **mv** es cambiar sus enlaces, sin transferir datos. Si están en sistemas de ficheros distintos, **mv** copia los datos de los ficheros del primer sistema en el segundo, crea los enlaces necesarios en el segundo sistema de ficheros, y luego elimina los enlaces en el primer sistema. El traslado se puede considerar como un cambio de nombre de los ficheros. Cuando se traslada un fichero permanecen intactos su modo de protección y su pertenencia, a menos que el movimiento sea de un sistema de ficheros a otro; en este caso, los ficheros pertenecen al usuario que ejecutó la orden **mv**. Las formas de utilizar **mv** en la línea de órdenes son:

$ mv [opciones] fich1 fich2

$ mv [opciones] fichero ... dir

En la primera forma se crea un enlace entre *fich2* y el fichero enlazado a *fich1*, y se elimina el enlace original de *fich1*. Si *fich2* ya estaba enlazado a un fichero, se destruye el enlace anterior al crearse el nuevo (esto no es lo mismo que destruir el fichero, ya que se conservan sus demás enlaces).

En la segunda forma, cada fichero en fichero ... se cambia a *dir*. El traslado conserva los nombres de los ficheros que se han movido.

Por omisión, **mv** solicita confirmación cuando reemplaza un enlace con un fichero que carece de permiso de escritura. Se puede modificar este comportamiento con las siguientes opciones de la línea de órdenes:

- **-i** Solicita confirmación al reemplazar cualquier enlace, sin importar sus permisos (sólo BSD UNIX).
- **-f** No solicita confirmación al reemplazar un enlace que carece de permiso de escritura.

5 Copia de ficheros: la orden cp

La orden **cp** copia uno o más ficheros. A diferencia de **ln** y **mv**, **cp** sí copia los datos y no sólo reacomoda las entradas en el directorio. Los formatos de **cp** de la línea de órdenes son:

$ cp [opciones] fichentrada fichsalida

$ cp [opciones] nomentrada ... dirsalida

donde *fichentrada* es un fichero de entrada, *fichsalida* es un fichero de salida, cada *nomentrada* es un fichero o directorio de entrada y *dirsalida* es un directorio de salida. Los elementos de entrada se denominan "fuentes"; los elementos de salida se conocen como "destinos".

- El primer formato se aplica cuando el destino es un fichero. En este caso sólo puede haber una fuente, un fichero. El fichero de entrada se copia en el fichero de salida.
- El segundo formato se aplica cuando el destino es un directorio.

Las copias pueden ser recursivas o no recursivas, aunque la versión de **cp** en System V sólo permite copias no recursivas. La opción **-r** especifica la recursividad de la copia, la cual sólo afecta a la copia de directorios.

5.1 Copiado no recursivo

Para una copia no recursiva es necesario que exista el directorio dirsalida y que cada nomentrada nombre un fichero, y no un directorio. Cada uno de los ficheros de entrada se copia en el directorio de salida y se le asigna el mismo nombre base que tenía originalmente. Por ejemplo, suponga que /usr/homer y burt son directorios y que /usr/homer contiene los ficheros lisa y marge. Entonces, la orden

$ cp /usr/homer/* burt

copia */usr/homer/lisa* en un fichero de nombre *burt/lisa* y */usr/homer/marge* en un fichero llamado *burt/marge*.

Si el fichero de destino no existe, se crea un fichero con ese nombre. Si el fichero de destino ya existe, se elimina el enlace especificado por el fichero de destino y se crea un enlace nuevo. Enseguida se copian los datos de cada fichero fuente en su fichero de destino correspondiente. Después de la copia, los ficheros fuente y destino son independientes, o sea, las modificaciones de un fichero no afectan al otro.

Un fichero creado por **cp** adquiere el dueño, el modo y los permisos del fichero de destino si dicho fichero existía antes de la copia; de lo contrario se asignan los del fichero fuente. Si una operación de copia va a destruir un enlace existente con un fichero con permiso de escritura desactivado, **cp** presentará el modo de protección del fichero y solicitará confirmación de la copia. Si se teclea cualquier cosa que comience con 'y' se permitirá la copia; cualquier otra cosa la cancelará.

5.2 Copiado recursivo

El comportamiento de la copia recursiva depende de la existencia del directorio de salida *dirsalida*:

- Si *dirsalida* no existe, la entrada debe consistir en un sólo directorio. Después de crear *dirsalida*, **cp** copia todos los ficheros y subdirectorios del directorio de entrada en *dirsalida* para que sea una copia del directorio de entrada.
- Si *dirsalida* ya existe, cada fichero de entrada se copia en el directorio de salida de la misma manera que se hace en la copia no recursiva. Los directorios de entrada se reproducen como subdirectorios del directorio de salida, incluyendo los ficheros y subdirectorios que contengan.

6 Eliminación de ficheros y directorios: las órdenes rm y rmdir

Las dos órdenes para eliminar ficheros son **rm** y **rmdir**. **rm** puede eliminar ficheros ordinarios y directorios, pero **rmdir** sólo puede eliminar directorios. Por otra parte, **rmdir** puede eliminar directorios padre que queden vacíos como resultado de eliminaciones anteriores, algo que no puede hacer **rm**.

La "eliminación" de un fichero, sea un fichero ordinario o un directorio, significa eliminar enlace de un directorio y no

eliminar el fichero en sí. Si hay otros enlaces duros con el fichero, se conservan, lo mismo que el fichero. Un fichero sólo se elimina cuando se borra su último enlace duro. Como un directorio es un tipo especial de fichero, está sujeto a las mismas reglas.

6.1 Eliminación de ficheros: la orden rm

La orden **rm** elimina los enlaces con los ficheros, sean ordinarios o directorios. Un fichero se elimina cuando se han borrado todos sus enlaces duros, de manera que la eliminación del único enlace con un fichero borra al propio fichero. La sintaxis de **rm** es:

```
$ rm [opciones] fichero ...
```

donde cada fichero en *fichero* ... puede incluir enlaces a directorios si está presente la opción **-r**. Por omisión, **rm** pide una confirmación si se intenta eliminar un enlace con un fichero que no tiene permiso de escritura. La pregunta de confirmación indica los permisos del fichero como un número octal de tres dígitos. Se puede cambiar este comportamiento con las opciones **-i** y **-f** que se describen más adelante.

rm devuelve un código de salida cero si tiene éxito en la eliminación de todos los enlaces especificados, y un código distinto de cero en caso contrario.

Opciones de la línea de órdenes:

- **-i** Solicita interactivamente la confirmación de cada eliminación. Una respuesta que comience con 'y' o con 'Y' indica que debe eliminarse el fichero; cualquier otra respuesta indica lo contrario. **rm** solicita esta confirmación incluso si la entrada estándar no es un terminal.
- **-r** Elimina recursivamente los enlaces con directorios, es decir, elimina los enlaces con los directorios y con todos los ficheros y subdirectorios que contengan, directa o indirectamente. De hecho, esta opción elimina toda una rama del árbol de directorios. **rm** pide que se confirme la eliminación de enlaces sin permiso de escritura, pero esta solicitud se omite si la entrada estándar no proviene del terminal, o si se especificó la opción **-f**. Si usted rechaza la eliminación de un enlace, **rm** no eliminará los directorios que contengan ese enlace de forma directa o indirecta.
- **-f** No solicita la confirmación de las eliminaciones, incluso en el caso de enlaces que carecen de permiso de escritura. La opción **-f** tiene precedencia si se especifica junto con la opción **-i**.
- **Nota**: No se puede eliminar enlaces de un directorio para el cual no se tenga permiso de escritura, sin importar las opciones empleadas.

6.2 Eliminación de directorios: la orden rmdir

La orden **rmdir** elimina los enlaces con directorios. Su sintaxis es:

```
$ rmdir [opciones] dir ...
```

donde *dir* ... es una lista de directorios.

> Por ejemplo,
>
> **$ rmdir denver**
>
> hace que **rmdir** elimine el enlace al subdirectorio *denver* del directorio actual. A diferencia de **rm -r**, **rmdir** no eliminará un enlace con un directorio si éste no está vacío.
>
> **rmdir** devuelve un código de salida de cero si tiene éxito en la eliminación de todos los enlaces especificados y un código distinto de cero en caso contrario.

Opciones de la línea de órdenes:

- **-p** Si la eliminación de un enlace con un directorio hace que el padre directo quede vacío, también se elimina el enlace del directorio padre. **rmdir** sigue eliminando los enlaces con directorios vacíos, dirigiéndose hacia la raíz, hasta encontrar un directorio que no esté vacío. **rmdir** envía un mensaje al error estándar anunciando cada enlace que elimina y cada intento de eliminación que fracasa.
- **-s** Suprime los mensajes generados por **rmdir**.

No todos los sistemas cuentan con estas opciones.

7 Creación de directorios: la orden mkdir

La orden **mkdir** crea uno o más directorios nuevos. Su sintaxis es:

> ***$ mkdir [opciones] dir ...***

Aquí, cada *dir* es la trayectoria de un directorio que se desea crear. Cada directorio de nueva creación tiene dos entradas: '.' para el directorio en sí y '..' para su directorio padre. Los permisos del directorio se asignan como *777* y se modifican teniendo en cuenta el valor **umask** actual, aunque se puede especificar permisos distintos con la opción **-m**. El propietario y el grupo del nuevo directorio serán los del proceso bajo cuyo auspicio se creó el directorio. La creación de un directorio, al igual que la creación de un fichero, requiere permiso de escritura en el directorio padre.

Opciones de la línea de órdenes:

- **-m** *n* Asigna *n* como los permisos de cada directorio creado.
- **-p** Si en una trayectoria faltan componentes, crea los directorios intermedios necesarios.

> Por ejemplo, suponga que */usr/luis* no tiene subdirectorios y que el usuario *luis* emite la orden
>
> **$ mkdir -p ~/a/x**
>
> Entonces, **mkdir** crea un directorio *a* intermedio con el directorio *x* como entrada inicial.

No todos los sistemas reconocen estas opciones.

8 Localización de ficheros: la orden find

El programa **find** busca en las partes especificadas del sistema de ficheros de UNIX los ficheros que cumplen cierto criterio. Este programa tiene multitud de opciones que permiten aplicar criterios complejos como: *aquellos ficheros que ocupen más de 10 bloques y su propietario sea distinto del que ejecuta el programa* ***find***. En esta sección sólo se comenta cómo localizar ficheros que coincidan con un patrón de búsqueda, si quiere ampliar sus conocimientos sobre **find** consulte un manual o **man**.

Para buscar los ficheros que coinciden con un patrón utilizaremos la sintaxis:

> ***$ find listrayec -name patrón -print***

listrayec es una lista de trayectorias de ficheros y directorios (por lo general, sólo de directorios), indica los ficheros y directorios en los que se debe realizar la búsqueda, si es un directorio la búsqueda será recursiva.

patrón indica el patrón de búsqueda que deben cumplir los ficheros para que **find** mande su trayectoria absoluta a la salida estándar. En *patrón* puede especificar comodines, tal como se explicaron en el tema segundo.

> Por ejemplo, la orden
>
> **$ find /usr -name "v*.h" -print**
>
> envía a la salida estándar una lista de todos los ficheros que están en */usr* o sus subdirectorios, y que además comiencen con 'v' y terminen con '.h'. La lista podría verse como sigue:
>
> ```
> /usr/spool/uucppublic/src/vlimit.h
> /usr/include/sys/var.h
> /usr/include/sys/vt.h
> /usr/include/sys/vtoc.h
> /usr/include/values.h
> /usr/include/varargs.h
> ```

Siempre que se utilicen comodines en el patrón, éste se debe entrecomillar, ya que los metacaracteres que van entre una expresión entrecomillada no los interpreta el *shell*. Con esto se logra que sea **find** la que interprete los comodines.

9 Ejemplos

En este apartado se muestran una serie de ejemplos que aclaran las diferencias entre enlace duro y simbólico, y el uso de algunas órdenes. En ellos se empleará el programa cambia descrito en el tema anterior.

A continuación se muestran los ejemplos:

$ cd *Nos desplazamos a nuestro directorio base*

$ echo abc >f *Creamos el fichero f con el contenido abc*

$ cat f *Efectivamente, f contiene abc*

abc

$ ls –li f *Consultamos información sobre el fichero asociado al enlace de nombre f*

```
2059 -rw-r--r-- 1 x1111111 alumno 4 Oct 4 10:42 f
```

La opción **i** de **ls** hace que el listado muestre el nodo i (en este caso 2059) de los enlaces listados. Resulta interesante observar la columna que viene tras los permisos, ésta indica el número de enlaces duros al fichero enlazado. En este caso acaba de ser creado, y sólo posee 1 (*~x1111111/f*).

$ ln f ed *Creamos un enlace duro de nombre ed al fichero que enlaza f*

$ ls –li ed f *Consultamos información sobre los enlaces ed y f*

```
2059 -rw-r--r-- 2 x1111111 alumno 4 Oct 4 10:42 ed

2059 -rw-r--r-- 2 x1111111 alumno 4 Oct 4 10:42 f
```

El nodo i, los permisos, el número de enlaces duros, el propietario y su grupo, el tamaño y la fecha de creación son la misma, pues ambos enlaces hacen referencia al mismo fichero. Note que el contador de enlaces duros es ahora dos.

$ ln –s ~/f es *Creamos un enlace simbólico de nombre es al fichero que enlaza ~/f*

$ ls –li es f *Consultamos información sobre los enlaces es y f*

```
2065 lrwxrwxrwx 1 x1111111 alumno 16 Oct 4 10:50 es - > /home/x1111111/f

2059 -rw-r--r-- 2 x1111111 alumno 4 Oct 4 10:42 f
```

Analizando la información proporcionada por **ls** podemos deducir que es está enlazado con otro fichero, cuyo nodo i es el 2065. Además, este fichero se utiliza para almacenar una trayectoria (el primer carácter l en el listado de permisos nos dice que es un enlace simbólico) y tiene un único enlace duro (*~x11111111/es*). Su fecha de creación es posterior a la del fichero enlazado por *f*. Observando la última columna podemos saber su contenido: */home/x1111111/f* (su tamaño es 16, el número de caracteres de la trayectoria que almacena).

$ cambia f a A *Cambiamos el fichero a través de la entrada f*

$ cambia ed b B *Cambiamos el fichero a través de la entrada ed*

$ cambia es c C *Cambiamos el fichero a través de la entrada es*

$ cat f ed es *Observamos el contenido a través de las tres entradas*

ABC

ABC

ABC

Como puede deducirse de esta secuencia de órdenes a través de los tres enlaces se accede al mismo fichero.

$ rm f *Borramos el primer enlace al fichero*