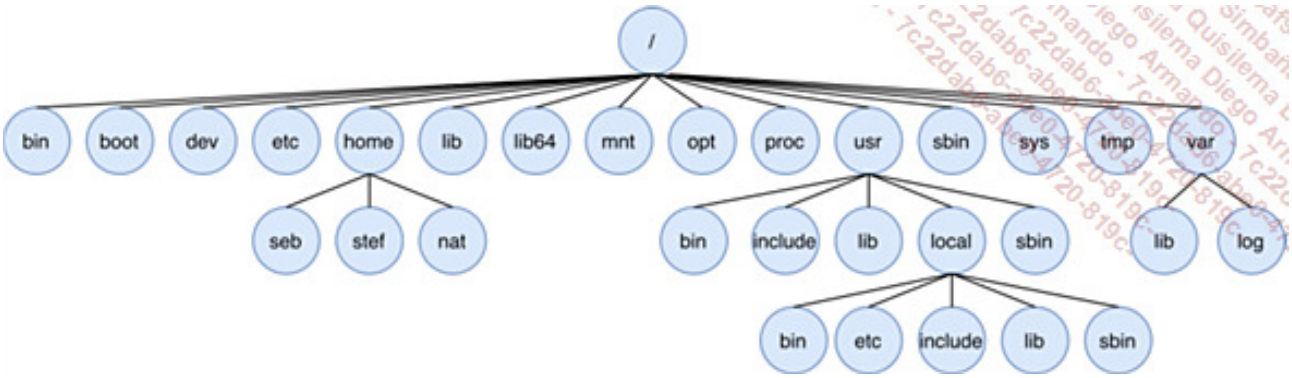
La gestión de los archivos

1. El sistema de archivos

Un sistema de archivos, llamado comúnmente File System o FS, determina la organización de los datos en un soporte de almacenamiento, y por tanto, cómo gestiona y organiza el sistema operativo los archivos.

Linux es, como todo Unix, un sistema operativo completamente orientado a archivos. Se representa todo (o casi todo) con un archivo, tanto los datos (archivos de datos de cualquier tipo, como una imagen o un programa) como los periféricos (terminales, ratones, teclado, tarjeta sonido, etc.) o incluso los medios de comunicación (sockets, tuberías nombradas, etc.). Se puede decir que el sistema de archivos es el corazón de cualquier sistema Unix.



Ejemplo de árbol de directorio Linux

El sistema de archivos de Linux es jerárquico. Describe un árbol de directorios y subdirectorios, a partir de un elemento básico llamado raíz o root directory.

2. Los diferentes tipos de archivos

Distinguimos tres tipos de archivos: ordinarios, catálogo, especiales.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

a. Los archivos ordinarios o regulares

Los archivos ordinarios se llaman también archivos regulares, ordinary files o regular files. Son archivos totalmente clásicos que contienen datos. Por datos se debe entender cualquier contenido:

- texto;
- imagen;
- audio;
- programa binario compilado;
- script;
- base de datos;
- librería de programación;
- etc.

Por defecto, nada permite diferenciar unos de otros, salvo la utilización de algunas opciones de determinados comandos (ls -F por ejemplo) o el comando **file**.

```
$ file /bin/bash
/bin/bash: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=
a6cb40078351e05121d46daa768e271846d5cc54, for GNU/Linux 3.2.0, stripped
```

Linux desconoce la noción de extensión de archivo como componente interno de la estructura del sistema de archivos. Dicho de otro modo, una extensión no es relevante dentro de un sistema de archivos y se la considera simplemente como parte del nombre. Sólo sirve para distinguir visual y rápidamente el posible contenido de un archivo en comparación con otro. No debemos hablar de extensión, sino de sufijo. Sin embargo, el primer término forma parte del lenguaje común, y se puede seguir utilizando: todo el mundo comprenderá lo que quiere decir.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Como Linux no gestiona las extensiones, el nombre de un programa no termina casi nunca por un «.exe»; habrá que encontrar otro método para distinguirlo.

b. Los catálogos

Los archivos catálogo son los directorios o carpetas. Los directorios permiten organizar el disco duro creando una jerarquía. Un directorio puede contener archivos normales, archivos especiales y otros directorios de manera recursiva.

Un directorio no es más que un archivo particular que contiene la lista de los propios archivos presentes en este directorio, como un índice. Esta noción resultará muy útil cuando se trate el tema de los permisos.

c. Los archivos especiales

El tercer tipo de archivos es el especial. Existen varios tipos de archivos especiales. Por ejemplo, los drivers de los periféricos están representados por archivos especiales de la carpeta /dev pero también pueden ser creados en otros directorios, con el comando **mknod**.

Son principalmente, aunque no por fuerza, archivos que sirven de interfaz para los diversos periféricos. Se pueden utilizar, según el caso, como archivos normales. Cuando se accede en modo lectura o escritura a estos archivos se redirigen hacia el periférico (pasando por el driver asociado si existe). Por ejemplo, si dirige un archivo de onda sonora (wave) hacia el archivo que representa la salida de la tarjeta de sonido, hay muchas probabilidades que este sonido sea audible por sus altavoces. He aquí una lista de archivos especiales:

- Los archivos de tipo **bloque** permiten tener acceso a periféricos que funcionan con bloques de datos, como un disco, de manera aleatoria: se puede acceder directamente a cualquier bloque.
- Los archivos de tipo **carácter** permiten tener acceso a un periférico byte por byte, de manera secuencial, empezando por el principio: no se puede acceder al bloque n sin haber accedido primero al bloque n-1. Es el caso en un lector de banda magnética.
- Los archivos de tipo **FIFO**, son pipelines o tuberías que permiten el intercambio de datos entre dos procesos.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

- Los archivos de tipo **socket** permiten también el intercambio de datos entre procesos, pero a través de protocolos clásicos de red.

Finalmente, el enlace simbólico se puede considerar a veces como un archivo especial, aunque no abra ninguna interfaz en particular: solamente hace referencia a otro archivo.

3. Nomenclatura de los archivos

No se puede dar cualquier nombre a un archivo; hay que seguir unas simples reglas, válidas para todos los tipos de archivos.

Un sistema de archivos Unix acepta nombres de hasta 255 caracteres. La posible extensión (el sufijo) está incluida en la longitud del nombre del archivo.

Linux respeta la distinción entre los nombres de archivos en minúsculas y en mayúsculas. Pepito, PEPITO, PePito y pepito son nombres de archivos diferentes, con un contenido diferente. Esta distinción es intrínseca al tipo de sistema de archivos.

Se acepta la mayoría de los caracteres (las cifras, las letras, las mayúsculas, las minúsculas, ciertos signos, los caracteres acentuados), incluido el espacio. Sin embargo, se deben evitar algunos caracteres, ya que tienen un significado particular dentro del shell: & ; ( ) ~ <espacio> \ / | ` ? - (al principio del nombre).

Los nombres siguientes son válidos:

- Archivo1
- Paga.txt
- 123tratamiento.sh
- Paga_junio_2002.xls
- 8

Los nombres siguientes, aunque válidos, pueden crear problemas:

- Archivo*
- Pago(diciembre)

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

- Ben&Nuts
- Paga junio 2002.xls
- -f

4. Las rutas

a. Estructura y nombre de ruta

Las rutas permiten definir una ubicación en el sistema de archivos. Es la lista de los directorios y subdirectorios utilizados para acceder a un sitio determinado de la estructura hasta la posición deseada (directorio, archivo). Se suele completar un nombre de archivo con su ruta de acceso. Por eso el archivo pepito del directorio dir1 es diferente del archivo pepito del directorio dir2. Al ser jerárquico, el sistema de archivos de Unix tiene forma de estructura en árbol.

El esquema presentado en la sección La gestión de los archivos - El sistema de archivos de este capítulo representa una estructura en árbol de un sistema de archivos Linux. La / situada arriba del todo se llama raíz o root directory (no confundir con el directorio del administrador root). El nombre de la ruta o path name de un archivo es la concatenación, desde la raíz, de todos los directorios que se deben cruzar para acceder a él, que están separados cada uno por el carácter /. Es una ruta absoluta, como la siguiente:

> /home/pepito/Docs/Backup/fic.bak

Una ruta absoluta o completa:

- empieza desde la raíz. Por lo tanto, comienza con una /,
- describe todos los directorios que hay que cruzar para acceder al sitio deseado,
- no contiene ni . ni `..`.

b. Directorio personal

Al crear un nuevo usuario, el administrador le asigna un directorio personal llamado home

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

directory. Cuando inicia sesión, el usuario es dirigido directamente a ese directorio, que es el suyo personal, en el que podrá crear sus propios archivos y subdirectorios.

```
Login: seb
Password: xxxxxxxxxx
$ pwd
/home/seb
```

c. Ruta relativa

Un nombre de ruta también puede ser relativo a su posición en el directorio actual. El sistema (o el shell) recuerda la posición actual de un usuario en el sistema de archivos, el directorio activo. Puede acceder a otro directorio de la estructura desde su ubicación actual sin teclear la ruta completa, con sólo precisar la ruta más corta en relación con su posición actual dentro de la estructura.

Para ello, a menudo hace falta utilizar dos entradas particulares de directorios:

- El punto . representa el directorio corriente, activo. Suele estar implícito.
- Los dos puntos .. representan el directorio de nivel superior.

Una ruta relativa:

- describe una ruta relativa a una posición determinada en la estructura, en general (pero no siempre) desde la posición actual;
- describe en principio la ruta más corta para ir de un punto a otro;
- puede contener puntos o dos puntos.

Las tres afirmaciones anteriores no son obligatorias:

- `/usr/local/bin` es una ruta completa o absoluta;
- `Documents/Photos` es una ruta relativa: se considera que existe el directorio Documents en el directorio corriente;
- `./Documents/Photos` es una ruta relativa perfectamente idéntica a la anterior, con la salvedad de que el punto indica el directorio activo (corriente) de manera explícita. «./Documents» indica de manera explícita el directorio

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Documents en el directorio activo;

- `/usr/local/../bin` es una ruta relativa: los .. son relativos a /usr/local y suben un nivel hacia /usr. La ruta final es, por lo tanto /usr/bin.
- `../lib` es una ruta relativa: los .. son relativos al directorio en el que nos encontramos y bajan un nivel para entrar en lib. Si se encuentra en el directorio /usr/bin, esta ruta representará /usr/lib.

d. La virgulilla

El bash interpreta el carácter virgulilla ~ como un alias del directorio personal. Las rutas pueden ser relativas a la virgulilla, pero ésta no debe ir precedida por carácter alguno. Para desplazarse en el directorio tmp de su carpeta personal esté donde esté:

```
$ cd ~/tmp
```

Si introduce esto, obtendrá un error:

```
$ cd /~
```

e. cd

Para desplazarse por los directorios, utilice el comando **cd** (*change directory*). El comando **pwd** (*print working directory*), que ya hemos comentado, muestra la ruta completa del directorio actual.

Si introduce **cd .**, no se mueve. El punto será muy útil cuando tenga que especificar rutas explícitas a comandos ubicados en el directorio donde está ubicado.

Cd .. sube un nivel. Si se encontraba en /home/seb, ahora estará en home.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Observe que « . » et « .. » no son argumentos de cd, pero verdaderos archivos de directorio.

El comando **cd** sin argumento permite volver directamente a su directorio de usuario.

A continuación, presentamos un ejemplo. El usuario seb se encuentra en su directorio personal. Se mueve mediante una ruta relativa hacia /home/public. Con .. sube hacia /home, por lo tanto con ../public se mueve a /home/public. De ahí, vía una ruta completa, se dirige hacia /urs/local/bon, y luego decide, con la ayuda de una ruta relativa, ir a /usr/lib: el primer .. baja hacia usr/local, el segundo hacia /usr, y luego vuelve hacia /usr/lib. Finalmente, seb vuelve a su directorio personal con cd sin argumento. Aquí se da la línea completa para una mejor comprensión.

```
[jolivares@client ~]$ pwd
/home/jolivares
[jolivares@client ~]$ cd ../public
[jolivares@client public]$ cd /usr/local/bin
[jolivares@client bin]$ cd ../../lib
[jolivares@client lib]$ cd
[jolivares@client ~]$ pwd
/home/jolivares
```

5. Los comandos básicos

a. Listar los archivos y los directorios

El comando **ls** permite listar el contenido de un directorio (catálogo) en líneas o columnas. Soporta varios parámetros, de los cuales los más importantes son:

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Parámetro	Significado
`-l`	Para cada archivo o carpeta, facilita información detallada.
`-a`	Se visualizan los archivos escondidos (empiezan por un punto).
`-d`	En un directorio, precisa el propio directorio, y no su contenido.
`-F`	Añade un carácter al final del nombre para especificar el tipo: / para un directorio, * para un ejecutable, @ para un vínculo simbólico, etc.
`-R`	Si el comando detecta subdirectorios, entra en ellos de manera recursiva.
`-t`	Se filtra la salida por fecha de modificación del más reciente al más antiguo. Se visualiza esta fecha.
`-c`	Muestra/ordena (con -t) por fecha de cambio de estado del archivo.
`-u`	Muestra/ordena (con -t) por fecha de acceso del archivo.
`-r`	Se invierte el orden de salida.
`-i`	Muestra el inodo del archivo.
`-C`	La visualización se hace en varias columnas (por defecto).
`-1`	La visualización se hace en una sola columna.
`-Z`	Muestra los atributos del archivo vinculado al tipo de sistema de archivo o al contexto de seguridad (selinux, por ejemplo).

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

El parámetro que le facilita más información es `-l`: proporciona ciertos detalles relativos a los archivos.

```
$ ls -l
total 4568
-rw-r--r-- 1 seb users   69120 sep  3  2006 3i_recuperación_2006.doc
-rw-r--r-- 1 seb users    9632 sep  3  2006 3i_recuperación_2006.odt
-rw-r--r-- 1 seb users    6849 nov 17  2003 control_1l2_mártes.sxw
...
```

La línea de salida indica el tamaño total en bloques de 1024 bytes (o 512 bytes si se define una variable llamada POSIXLY_CORRECT) del contenido del directorio. Este tamaño representa el conjunto de los archivos ordinarios del directorio y no tiene en cuenta los posibles subdirectorios y su contenido (para ello, habrá que utilizar el comando **du**).

Luego viene la lista detallada de todo el contenido.

-rw-r--r--	1	seb	users	69120	sep 3 2006	3i_recuperación_2006.doc
1	2	3	4	5	6	7

- 1: El primer símbolo representa el tipo de archivo (-: ordinario, d: directorio, l: vínculo simbólico...); los otros, por bloques de tres, los permisos para el usuario (rw-), el grupo (r--) y todos (r--). Se explican los permisos en el capítulo Los discos y el sistema de archivos.
- 2: Un contador de vínculos (capítulo Los discos y el sistema de archivos).
- 3: El propietario del archivo, que suele ser su creador.
- 4: El grupo al cual pertenece el archivo.
- 5: El tamaño del archivo en bytes.
- 6: La fecha de la última modificación (a veces con la hora), siguiendo el parámetro (t, c, u).
- 7: El nombre del archivo.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Los permisos van a veces seguidos de un punto «.» o de un signo más «+». El primero significa que el archivo dispone de un contexto de seguridad selinux, el segundo que el archivo dispone de permisos extendidos ACL. Estos conceptos no son abordados en este libro y no tienen ningún impacto en las operaciones propuestas en el libro.

Puede resultar muy útil la posibilidad de listar sus archivos de tal manera que se visualicen al final de la lista los modificados recientemente. Así, en caso de haber un gran número de archivos, tendrá delante estos últimos. El orden por fecha de modificación se hace con `-t`, y en el orden contrario, con `-r`. Añádale los detalles con `-l`.

```
$ ls -lrt
-rw-r--r-- 1 seb users   66107 ene  9 17:24 Parcial_1_1I_2008.pdf
-rw-r--r-- 1 seb users   13777 ene 10 17:58 parcial_3I_ppa_2007.odt
-rw-r--r-- 1 seb users   64095 ene 10 17:58 parcial_3I_ppa_2007.pdf
-rw-r--r-- 1 seb users  100092 feb 22 22:21 curso_shell_unix.odt
```

`ls -l -r -t` es estrictamente idéntico a `ls -lrt`, como ya se ha indicado en la sintaxis general de los comandos.

Un recurso mnemotécnico para recordar esta secuencia de argumentos es utilizarla bajo la forma `-rtl` (el orden de los argumentos no tiene importancia aquí) y pensar en la famosa radio europea RTL.

b. Gestionar los archivos y los directorios

Crear archivos vacíos

Quizás necesite crear archivos vacíos para hacer pruebas. Un comando práctico para ello es **touch**. Utilizado con el nombre de un archivo como argumento únicamente, crea un

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

archivo con un tamaño cero.

```
$ touch fictest
$ ls -l fictest
-rw-r--r-- 1 seb users 0 feb 29 15:13 fictest
```

La creación de archivos vacíos no es el principal uso de touch. Si vuelve a ejecutar el mismo comando en el archivo, observará que la fecha de modificación ha cambiado. El manual de touch le informará de que así es posible modificar completamente la fecha y la hora de un archivo. Esto puede ser útil para forzar las copias de seguridad incrementales de archivos.

Crear directorios

El comando **mkdir** (*make directory*) permite crear uno o varios directorios, o una estructura completa. Por defecto, el comando no crea una estructura. Si pasa como argumentos dir1/dir2 y dir1 no existe, el comando devuelve un error. En este caso, utilice el parámetro `-p`.

```
mkdir [-p] dir1 [dir2] ... [dirn]
```

```
$ mkdir Documentos
$ mkdir Documentos/Fotos
$ mkdir -p Archivos/antiguallas
$ ls -R
.:
Archivos  Documentos  fictest

./Archivos:
antiguallas

./Archivos/antiguallas:

./Documentos:
Fotos
```

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Suprimir directorios

El comando **rmdir** (*remove directory*) suprime uno o varios directorios. No puede suprimir una estructura. Si el directorio que pretende eliminar contiene archivos o directorios, el comando devuelve un error. Por lo tanto, el directorio no debe contener ni archivos ni directorios, y ello aunque los propios subdirectorios estén vacíos.

```
rmdir dir1 [dir2] ... [dirn]
```

No hay parámetro `-r` (recursividad) para el comando **rmdir**. Para suprimir una estructura tendrá que utilizar el comando **rm**.

```
$ rmdir Documentos/
rmdir: Documentos/: El directorio no está vacío.
$ rmdir Documentos/Fotos
$
```

Copiar archivos

El comando **cp** (copy) copia uno o varios archivos en otro archivo o en un directorio.

```
cp fic1 [fic2 ... ficn] Destino
```

En el primer caso, se vuelve a copiar fic1 en Destino. Si Destino existe, se sobreescribe sin aviso según el parámetro pasado y según los permisos. En el segundo caso, se copian de nuevo fic1, fic2 y así sucesivamente en el directorio Destino. Las rutas pueden ser absolutas o relativas.

El comando admite, entre otras, las opciones siguientes:

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Parámetro	Significado
`-i`	Pide confirmación de copia para cada archivo.
`-r`	Recursivo: copia un directorio y todo su contenido.
`-p`	Se preservan los permisos y fechas.
`-f`	Forzar la copia.
`-a`	Copia de archivo: el destino es en la medida de lo posible idéntico al origen. La copia es recursiva.

Preste atención al funcionamiento de cp con las copias de directorios. El funcionamiento es diferente según exista o no el directorio de destino. En el primer caso, dir2 no existe. Se copia el directorio dir1 en dir2. Al final dir2 es una copia exacta de dir1.

```
$ ls -d dir2
ls: no puede acceder a dir2: No existe el archivo o directorio
$ cp -r dir1 dir2
$ ls
dir1  dir2
```

Ahora que dir2 existe, ejecute de nuevo el comando cp. Esta vez, como dir2 existe, no será sobreescrito, como cabía esperar. El comando determina que, al ser dir2 el destino, se debe copiar dir1 en el destino: se copia dir1 en dir2.

```
$ cp -r dir1 dir2
$ ls dir2
dir1
```

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Mover y volver a nombrar un archivo

El comando **mv** (*move*) permite mover, volver a nombrar un archivo o las dos cosas a la vez. Funciona como el comando **cp**. Los parámetros `-f` e `-i` tienen el mismo efecto. Con los tres comandos mv sucesivos siguientes:

- se vuelve a nombrar txt1 como txt1.old;
- se mueve txt2 a dir1;
- se mueve txt3 a dir1 y se vuelve a nombrar como txt3.old.

```
$ touch txt1 txt2 txt3
$ mv txt1 txt1.old
$ mv txt2 dir1/txt2
$ mv txt3 dir1/txt3.old
```

Observe la existencia del parámetro `-u`: si el archivo de destino es más reciente, impide que se sobreescriba.

Suprimir un archivo o una estructura

El comando **rm** (*remove*) suprime uno o varios archivos y, si es preciso, una estructura completa, según las opciones. La supresión es definitiva.

```
rm [Opciones] fic1 [fic2...]
```

Las opciones son las habituales, pero dada la particularidad y la peligrosidad del comando, parece necesario repasarlas.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Parámetro	Significado
`-i`	El comando requerirá una confirmación para cada uno de los archivos que desea suprimir. Según la versión de Unix, el mensaje cambia y la respuesta también: y, Y, O, o, N, n, a veces todas.
`-r`	El parámetro siguiente que se espera es un directorio. En este caso, la supresión es recursiva: se suprimen todos los niveles inferiores, tanto los directorios como los archivos.
`-f`	Fuerza la supresión.

Por orden, los comandos siguientes suprimen un simple archivo, un directorio y una estructura de manera forzada:

```
$ rm fic1
$ rm -r dir1
$ rm -rf /home/public/depots
```

El uso combinado de los parámetros `-r` y `-f`, aunque muy útil y práctico, es muy peligroso, en particular como root. No se le pide ninguna confirmación. Al menos que se utilicen herramientas de recuperación de datos específicos, caras y poco eficaces, se perderán sus datos de manera irremediable. Existe un riesgo adicional: si cree que `rm -rf /` no tocará sus archivos con el pretexto de no contar con permisos en la raíz, ¡comete un error! El comando es recursivo, terminará llegando a su directorio personal…

Veamos un truquito. Suponga que dispone de un archivo cuyo nombre comienza con un

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

guión. ¿Es posible suprimirlo con **rm**?

```
$ >-i # ver las redirecciones
$ rm -i
rm: missing operand
Pruebe: `rm --help' para más información.
```

Es imposible suprimir el archivo «-i» de esta manera, ya que **rm** lo interpreta como un parámetro, y no como un argumento. Por lo tanto hay que actuar con astucia. Existen dos soluciones:

- Utilizar la opción `GNU --` que significa el final de los parámetros y el principio de los argumentos.
- Añadir una ruta, relativa o completa, antes del guión.

Esta última solución tiene la ventaja de ser un estándar. Las dos líneas son equivalentes:

```
$ rm -- -i
$ rm ./-i
```

Los vínculos simbólicos

Puede crear vínculos simbólicos, que son un poco como atajos a un archivo. Un vínculo es un archivo especial que contiene como información la ruta hacia otro archivo. Es un tipo de alias. Existen dos tipos de vínculos: el vínculo duro (hard link), que veremos más adelante, durante el estudio de los sistemas de archivos, y el vínculo simbólico (soft link), que corresponde a la definición dada.

Es posible crear vínculos simbólicos hacia cualquier tipo de archivo, sea cual sea y esté donde esté. El comando de creación de vínculos simbólicos no comprueba la existencia del archivo al que se apunta. Es posible crear vínculos a archivos que no existen con el parámetro `-f`.

```
ln -s archivo vínculo
```

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Si fuera necesario, el vínculo se comportará como el archivo al que se apunta, con los mismos permisos y las mismas propiedades:

- si el archivo al que se apunta es un programa, ejecutar el vínculo lleva a ejecutar el programa;
- si el archivo al que se apunta es un directorio, un cd sobre el vínculo entra en este directorio;
- si el archivo al que se apunta es un archivo especial (periférico), se ve el vínculo como periférico;
- etc.

Sólo en caso de que se elimine, el vínculo simbólico se «separa» del archivo al que apunta. La supresión de un vínculo simbólico implica la supresión de este vínculo únicamente, y no del archivo al que apunta. La supresión del archivo al que se apunta no implica la supresión de los vínculos simbólicos asociados. En este caso, el vínculo apunta al vacío.

```
$ touch fic1
$ ln -s fic1 vínculofic1
$ ls -l
-rw-r--r-- 1 seb users    0 mar  4 19:16 fic1
lrwxrwxrwx 1 seb users    4 mar  4 19:17 vínculofic1 -> fic1
$ ls -F
fic1  vínculofic1@
$ echo titi>fic1
$ cat vínculofic1
titi
```

Este ejemplo muestra que un vínculo simbólico es en realidad un archivo especial de tipo «l» que apunta hacia otro archivo. Observe en la lista detallada la presencia de una flecha que indica sobre qué archivo apunta el vínculo. Se distingue el carácter @ al indicar que se trata de un vínculo simbólico durante la utilización del parámetro `-F`. Si dispone de un terminal de color, es posible que el vínculo simbólico aparezca en azul claro (por convención en Linux). Si aparece en rojo, es que apunta al vacío.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Que un vínculo apunte al vacío no significa que no apunte a nada. Quizá esté hecho a propósito: es posible crear vínculos hacia puertos USB, o CD-ROM, entre otros, detrás de los cuales hay sistemas de archivos removibles. En este caso, el vínculo se vuelve activo cuando se inserta el soporte o se monta el sistema de archivos removido.

Se explicará más adelante el comando **echo** y el signo >. El efecto aquí es la escritura en el archivo fic1 de «titi». El comando **cat** visualiza el contenido de un archivo. Al representar el vínculo fic1, la salida es la esperada.

Cuidado, los permisos indicados son los del archivo especial y no tienen otro significado: no significa que todo el mundo tiene permisos en el archivo al que se apunta. Durante su utilización, son los permisos del archivo o de la carpeta a los que se apuntan los que prevalecen.

c. Comodines: carácter de sustitución

Al utilizar los comandos con el sistema de archivos, puede resultar interesante filtrar la salida de nombres de archivos con ayuda de determinados criterios, por ejemplo con el comando **ls**. En vez de visualizar toda la lista de archivos, se puede filtrar la visualización de varios criterios y caracteres especiales.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Caracteres(s)	Función
`*`	Sustituye una cadena de longitud variable, incluso vacía.
`?`	Sustituye cualquier carácter único.
`[...]`	Una serie o un rango de caracteres.
`[a-b]`	Un carácter entre el rango indicado (de a a b incluida).
`[!...]`	Inversión de la búsqueda.
`[^...]`	Ídem.
`{x,y}`	Una lista arbitraria de elementos, cifras, caracteres o cadenas.
`{x..y}`	Una serie de elementos que se irán incrementando de x a y.

- Supongamos el contenido siguiente:

```
$ ls
afic   afic2  bfic   bfic2  cfic   cfic2  dfic   dfic2
afic1  afic3  bfic1  bfic3  cfic1  cfic3  dfic1  dfic3
```

- Obtiene todos los archivos que empiezan con a:

```
$ ls a*
afic1  afic2  afic3
```

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

- Todos los archivos de cuatro caracteres que empiezan con a:

```
$ ls a???
afic
```

- Todos los archivos de al menos tres caracteres y que empiezan con b:

```
$ ls b??*
bfic  bfic1  bfic2  bfic3
```

- Todos los archivos que terminan con 1 o 2:

```
$ ls *[12]
afic1  afic2  bfic1  bfic2  cfic1  cfic2  dfic1  dfic2
```

- Todos los archivos que empiezan con las letras de a a c, que tienen al menos un segundo carácter antes de la terminación 1 o 2:

```
$ ls [a-c]?*[12]
afic1  afic2  bfic1  bfic2  cfic1  cfic2
```

- Todos los archivos que no terminan por 3:

```
$ ls *[!3]
afic  afic1  afic2  bfic  bfic1  bfic2  cfic  cfic1  cfic2  dfic
dfic1  dfic2
```

Todos los archivos que empiezan con las letras de a a c y que terminan por 1 ó 3:

```
$ ls {a..c}fic{1,3}
afic1  afic3  bfic1  bfic3  cfic1  cfic3
```

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema

Interpretación por el shell

El shell es el encargado de sustituir estos caracteres antes de pasar los parámetros a un comando. Así, en el momento de un `$ cp * Documents` , cp no recibe el carácter `*`, sino la lista de todos los archivos y directorios del directorio activo.

Los comodines pueden utilizarse dentro de todos los argumentos que representan archivos o rutas. Así, el comando siguiente va a volver a copiar todos los archivos README de todos los subdirectorios de Documents en la posición actual:

```
$ cp Documents/*/README
```

Observe el funcionamiento sorprendente de los paréntesis en las listas:

```
$ echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

d. Cierre de caracteres

Se deben cerrar algunos caracteres especiales; por ejemplo, en caso de caracteres poco corrientes en un nombre de archivo.

- La **contrabarra ** permite cerrar un carácter único. `ls paga\ *.xls` va a listar todos los archivos que contienen un espacio después de paga.
- Las **comillas** "..." permiten la interpretación de los caracteres especiales, de las variables, dentro de una cadena.
- Los **apóstrofes** '...' cierran todos los caracteres especiales en una cadena o un archivo.

© Éditions ENI - Todos los derechos reservados - Copia personal de Diego Armando Simbaña Quisilema