

LINUX SIN VENTANUCOS.

Encendemos el ordenador, apretando a ese botón que pone Power, el sistema operativo arranca, y al cabo de un rato veremos la pantalla de color negro y un mensaje similar a:

```
[zx80]$ _
```

Ese mensajito, es el llamado "prompt", traducido como simbolo del sistema, y quiere decir, que el sistema operativo, está preparado y listo para aceptar órdenes, o sea, escribimos algo, pulsamos retorno de carro y el sistema operativo, nos responde, y vuelve a aparecer el shell para que podamos decir más cosas.

Si pulsamos varias veces la tecla de retorno de carro sin decir nada, se escribe más veces el símbolo del sistema.

```
[zx80]$  
[zx80]$  
[zx80]$  
[zx80]$  
[zx80]$ _
```

Comenzamos a trabajar.

Siempre pulsamos retorno de carro para acabar una orden:

```
[zx80]$ naz, que pasa tronco?  
bash: naz,: command not found
```

Parece que no nos ha hecho mucho caso, lo mismo hay que hablarle en giri.

```
[zx80]$ hello, what's the matter?  
bash: hello: command not found
```

¿Qué es lo que pasa?, que el sistema operativo no es un sistema experto, no podemos hablar con el como si fuera otra persona, tenemos que introducirle órdenes, el se las traga, hace lo que tenga que hacer y escribe en pantalla los resultados. Vamos a comenzar con una primera orden. "clear".

```
[zx80]$ CLEAR  
bash: CLEAR: command not found
```

Ops, parece que no funciona, ¿por qué?, ¿tiene que ser en minúsculas?, pues si.

```
[zx80]$ clear
```

Y la pantalla se nos limpia, de todas formas si pulsamos la combinación de teclas **Control + L**, se nos limpia la pantalla también.

TIP 1: Linux distingue las mayúsculas de las minúsculas

La siguiente orden, "ls", lista el contenido de un directorio, un directorio es el equivalente a una carpeta de trabajo, como las que tenemos en Windows, solo que no la vemos en una ventana. "ls" viene de la contracción de la palabra inglesa "list", en castellano "LiStar".

```
[zx80]$ ls  
[zx80]$ _
```

¿Qué ha pasado?, ¿no ha funcionado?, por lo visto no hay ningún fichero en la carpeta en la que estamos y por lo tanto no ha devuelto nada.

TIP 2: Cuando Linux no tiene nada que decir, no dice nada.

Si no tenemos ficheros, pues nos lo hacemos, vamos a empezar por otra carpeta.

```
[zx80]$ mkdir mi carpeta
[zx80]$
```

Y vamos a ver si la ha creado.

```
[zx80]$ ls
carpeta mi
[zx80]$
```

Opssss, ¿ha puesto el nombre del directorio al revés?, no, resulta que ha creado dos directorios, un directorio "mi" y un directorio "carpeta", vamos a intentarlo de otra forma:

```
[zx80]$ mkdir "mi carpeta"
[zx80]$ ls mi carpeta
carpeta mi
[zx80]_
```

Se pueden remplazar tambien las dos comillas " por las comillas simples ' Bien, si queremos cambiar de directorio, usamos la órden "cd".

```
[zx80]$ cd carpeta
[carpeta]$ _
```

Observamos, que el prompt ha cambiado, indicando el directorio en el que estamos, en cualquier momento podemos hallar el directorio en el cual estamos situados usando la órden pwd.

```
[carpeta]$ pwd
/home/zx80/carpeta
[carpeta]$ _
```

El mensaje que nos ha respondido quiere decir, que estamos en el directorio carpeta, que está dentro del directorio zx80 que a su vez está en el directorio home.

Linux, al igual que todos los Unix, utiliza para separar los directorios el símbolo correcto, o sea el "/", 'backslack', de hecho es el que se utiliza en los URL de internet.

El directorio más alto de todos, es el directorio "/", llamado directorio raiz, un directorio, puede contener varios directorios, llamados subdirectorios, o directorios hijos, pero un directorio hijo, o subdirectorio, no puede contener varios directorios padres.

Para subir en la jerarquía de directorios, podemos teclear.

```
[carpeta] cd ..
[zx80] _
```

Los dos puntitos separados del cd, pues de lo contrario, no se reconocerá la órden como si pasa en MS-DOS. También podíamos haber dado el camino completo, veamos como:

```
[zx80] cd -
[carpeta] cd /home/zx80
[zx80] _
```

La primera órden cd -, vuelve al anterior directorio en el que hayamos estado, y con la segunda órden, indicamos el camino completo. También se puede indicar un camino relativo, veamos como.

```
[zx80] cd -
[carpeta] cd ../mi
[mi] cd ../../zx80
[zx80] cd mi
[mi] _
```

El directorio "mi" está a la misma altura que el directorio "carpeta", ambos cuelgan del directorio "zx80", así que vamos al directorio "mi" dando como referencia el directorio padre, (los dos puntos), "../mi".

Estando en el directorio "mi", la orden "cd ../../zx80" es equivalente a "cd ..", simplemente hemos dado un rodeo.

Voy a introducir en este punto una orden, la orden es "touch", sirve para crear un fichero si no existe.

```
[mi]$ touch fichero1
[mi]$ ls
fichero1
```

Hemos creado un fichero llamado "fichero1", este fichero estará vacío, no contiene nada. Vamos al directorio padre, y vamos con lo último que nos queda de los directorios, el borrado de estos. El borrado de directorios se hace con "rmdir".

```
[mi]$ cd ..
[zx80]$ ls
carpeta    mi        mi carpeta
[zx80]$ rmdir carpeta
[zx80]$ ls
mi        mi carpeta
[zx80]$ rmdir "mi carpeta"
[zx80]$ ls
mi
[zx80]$ rmdir mi
rmdir: mi: Directory not empty
[zx80] _
```

No podemos borrar el directorio "mi", por que hemos creado antes el fichero "fichero1". tenemos que entrar en ese directorio, borrar el fichero, volver y borrar el directorio. Para borrar ficheros, usamos la orden "rm".

```
[zx80]$ cd mi
[mi]$ rm fichero1
[mi]$ ls
[mi]$ cd ..
[zx80]$ rmdir mi
[zx80]$ ls
[zx80]$ _
```

Ya está, hay otra forma de borrar directorios que no están vacíos, es con la orden "rm -r", la r de recursivo, es decir, para cuando hay directorios dentro de los directorios.

Para salir de una sesión abierta de Linux, se usa la orden exit o logout.

```
[zx80]$ exit
Login:
```

Algunas órdenes de la Shell.

Volvemos a enfrentarnos una vez más con el dragón.

```
Login: zx80
password:
[zx80]$ _
```

Vamos a introducir algunas órdenes, para familiarizarnos un poco con el shell que estamos utilizando,

```
[zx80]$ ls
[zx80]$ pwd
/home/zx80
[zx80]$ _
```

Estas órdenes ya las vimos en la primera parte, si queremos repetir en cualquier otro momento algunas de las órdenes, no hace falta teclearlas de nuevo, usando las flecha de cursor arriba, podemos repetir órdenes que ya hemos tecleado anteriormente, e incluso editarlas.

Sigamos metiendo órdenes, crear directorios, borrarlos, crear ficheros, etc. Llegará un momento, en que el texto desaparezca por la parte superior de la pantalla.

La consola, sólo tiene 80 columnas de ancho y 25 líneas de alto, para ver una línea que haya desaparecido, podemos usar la combinación de teclas **Mayúsculas + Re pag** para ir arriba y **Mayúsculas + Av pag** para ir hacia abajo.

Si lo deseamos, podemos ejecutar dos o más órdenes en una misma línea separandolas con el punto y coma, ";"

```
[zx80]$ mkdir directorio ; ls ; cd directorio ; pwd
directorio
/home/zx80/directorio
[directorio]$ _
```

A diferencia de lo que ocurre con otros sistemas operativos orientados a caracteres, casi todos los comandos básicos que acepta el shell, son externos, o sea, programas aparte, muchos de esos programas están en el directorio `/bin` y otros en el directorio `/usr/bin`. Para verlos puede hacer un `ls`:

```
[directorio]$ ls /bin
arch          date          grep          mknod
rmdir         true          ash           dd
gtar          mktemp        rpm           umount
.....
[directorio]$ _
```

En cualquier momento, cuando se tenga curiosidad acerca de qué hace un programa determinado, podemos consultar la orden "man".

```
[directorio]$ man ls
```

Con las teclas de cursor, nos desplazamos hacia arriba y abajo, para salir pulsar la tecla **q**.

La mayoría de las órdenes aceptan parámetros, los parámetros especifican que cosas debe hacer un programa o como debe funcionar.

```
[directorio]$ cd .. ; ls -l
drwxr-xr-x  5 zx80  users          1024 Jan 13 00:22 directorio
[zx80]$ _
```

TIP 3: Los parámetros en Unix se pasan con el simbolo "-", a diferencia del DOS que usa el símbolo "/".

Con el parámetro `-l`, hemos dicho a `ls` que liste los ficheros en formato largo, un fichero por línea, dando más información sobre ese fichero. Probar a hacer:

```
[zx80]$ ls -l /usr/bin
-rwxr-xr-x  1 root    root      20648 Jul 30  1999 zipsplit
-rwxr-xr-x  1 root    root         41 Mar 25  1999 zless
-rwxr-xr-x  1 root    root     1068 Mar 25  1999 zmore
-rwxr-xr-x  1 root    root     3502 Mar 25  1999 znew
[zx80]$ _
```

Veremos desplazarse los ficheros rápidamente y se perderán los primeros, este es el mejor momento para practicar aquello de las combinaciones de teclas **Mays + Re Pag** y **Mays + Av Pag**.

Otro parámetro de `ls`, es el parámetro `-a`, con `-a` listamos también los ficheros ocultos:

```
[zx80]$ ls -a
.                .gftp           .twmX8Fmri
..               .gimp           .user.rdb
Directorio
[zx80]$ _
```

Hay más ficheros ocultos, pero se han omitido por comodidad, los ficheros ocultos en Unix son los que empiezan con un punto, normalmente suelen ser ficheros de configuración.

De esos ficheros, hay dos, el fichero `"."` y el fichero `".."` que son directorios, `"."` es el directorio en el que estamos y `".."` es el nivel superior, o directorio padre.

TIP 4: Cuando usamos más de un parámetro, normalmente se pueden agrupar con un único símbolo `"-"`

```
[zx80]$ ls -alF
```

Siguiendo con el tema de los parámetros, el shell del sistema operativo, no mete a los programas los parámetros tal y como nosotros se los damos, tiene que mirar si existen unos caracteres especiales llamados metacaracteres.

Un metacaracter o comodín, es el símbolo `"*"`, cuando pasamos un asterisco como parámetro a un programa, el sistema operativo lo sustituye por los nombres de todos los ficheros que existen en el directorio.

```
[zx80]$ cd directorio
[directorio]$ touch fichero1
[directorio]$ touch fichero2
[directorio]$ cd ..
[zx80]$ mkdir dir2
[zx80]$ touch fichero3
[zx80]$ touch fichero4
[zx80]$ touch fichero5
[zx80]$ ls *
fichero3 fichero4 fichero5
directorio:
fichero1  fichero2
dir2:
[zx80]$ _
```

La orden `ls *` en nuestro caso, equivalía a `"ls directorio dir2 fichero3 fichero4 fichero5"`

El caracter asterisco, naturalmente también se puede usar como comodín de varios caracteres.

```
[zx80]$ ls *.txt
```

Otro metacaracter es el símbolo "?" que sustituye a un caracter.

```
[zx80]$ ls fichero?  
fichero3 fichero4 fichero5
```

Para más información se puede consultar "man bash"

Permisos.

Al principio de cada línea, hay una información muy rara, `-rwxr-xr-x`. La primera letra, indica que tipo de fichero es, si es una "d" es un directorio, y si es el caracter "-" es un fichero ordinario, o sea, un programa, un fichero de texto, un fichero de sonido, un AVI, etc.

Si volvemos a hacer un `ls -l`, vemos que efectivamente, el fichero "directorio" que hemos creado con anterioridad, es un directorio.

```
[zx80]$ ls -l  
drwxr-xr-x  5 zx80  users          1024 Jan 13 00:22 directorio  
[zx80]$ _
```

Vemos en la tercera columna, la palabra "zx80", o sea, dice que lo hemos creado nosotros. La palabra "users" indica el grupo al que pertenece y como pertenecemos al grupo de trabajo "users", el fichero también pertenece al grupo "users".

La siguiente columna, 1024, indica el tamaño del fichero, observa el listado de `/usr/bin`.

Luego viene la fecha y la hora, y por último el nombre del fichero. Pero volvamos a la primera columna de otro caso.

```
-rwxr-xr-x  1 root  root          3502 Mar 25 1999 znew
```

Sobre el fichero ordinario znew, hay unos permisos de lectura y escritura, vemos que el fichero, pertenece a "root" y es de un grupo de usuarios llamado "root".

El primer carácter, indica tipo de fichero:

- Fichero ordinario, como programas, mpeg, mp3, txt, ...
- d Directorio, se accede a el con "cd", "ls", etc.
- c Fichero de control de dispositivo en modo caracter.
- b Fichero de control de dispositivo en modo bloque.
- s Socket de tipo Unix.
- l Enlace simbólico.

Quitamos el primer caracter, que nos dice que tipo de fichero es, y separamos los demás caracteres en bloques de tres.

```
rwX   r-x   r-x
```

El primer bloque, se aplica sobre el usuario propietario, en este caso "root", el segundo bloque sobre el grupo propietario, también "root" y el último bloque, a todos los demas.

- r: Permite escribir en el archivo.
- w: Permite escritura, incluido el borrado.
- x: Permite la ejecución.

En nuestro caso, como no somos "root" y somos de otro grupo, "users", se aplica

el tercer bloque, podemos leer el fichero, y podemos ejecutarlo, pero no podemos sobrescribirlo, ni borrarlo, ni hacer ninguna modificación. Esto es una seguridad básica del sistema operativo, en una máquina en la que existan varios usuarios trabajando al mismo tiempo, ninguno podrá modificar los programas ni los ficheros para meter troyanos ni bromas.

Supongamos que alguien escribiera un virus para Linux, y lo ejecutamos trabajando como "zx80", el virus tendría nuestros permisos y no podría borrar los ficheros importantes.

```
[zx80]$ ls -l /etc/shadow
-r----- 1 root root 745 Dec 22 02:01 /etc/shadow
```

El fichero `/etc/shadow`, contiene los passwords de los usuarios, y sólo lo puede leer el usuario "root".

Mario bross.

No, por ahora no vamos a jugar, vamos a tratar los pipes o tuberías, y los filtros.

```
Login: zx80
password:
[zx80]$ _
```

Para empezar, necesitamos crear algunos ficheros con datos, para ello podemos utilizar un editor de textos, hay varios, el "joe", el "vi", el "jed", y el "emacs" son cuatro de los más populares, el único del que hay garantía de que se puede encontrar en cualquier Unix es el vi, pero es un poco complejo de utilizar, así que vamos a crear un fichero a pelo.

```
[zx80]$ echo hola
hola
[zx80]$ _
```

La órden `echo`, envía a "salida estandar" un mensaje, ¿Y qué eso de salida estandar?

Veamos, cualquier aplicación de Unix/Linux, e incluso las aplicaciones de DOS, tienen tres ficheros abiertos, que utilizan para poder funcionar.

1- Entrada estandar: De solo lectura representa normalmente al teclado, los programas leen a traves de este fichero todo lo que escribimos. Normalmente se conoce a este fichero como **stdin**.

2- Salida estandar: El segundo fichero es de sólo escritura, y representa la salida de nuestro programa, cuando un programa quiere escribir algo en la pantalla, lo que hace es escribir en este fichero como **stdout**.

3- Error estandar: Es también de sólo escritura, es similar al anterior, y se usa para imprimir mensajes de error como **stderr**.

He dicho, que la salida estandar representa la pantalla y la entrada estandar, representa el teclado, pero eso no tiene por que ser siempre así.

Podemos cambiar la salida estandar de un programa con el operador ">"

```
[zx80]$ echo hola > fichero.txt
[zx80]$ _
```

No hemos visto la salida de `echo` por la pantalla, la salida ha sido redirigida al fichero "fichero.txt".

```
[zx80] ls
fichero.txt
```

```
[zx80] _
```

Si queremos ver el contenido de ese fichero usamos la orden "cat"

```
[zx80] cat fichero.txt
hola
[zx80] _
```

¿Y como podemos añadir más líneas al fichero?, vamos a intentarlo.

```
[zx80]$ echo linea2 > fichero.txt ; cat fichero.txt
linea2
[zx80]$ _
```

Recordemos que se pueden meter dos órdenes en una sólo línea si las separamos con un punto y coma.

¿Qué ha pasado?, se ha machacado el archivo y contiene lo último que hemos escrito. Para que no se sobrescriba utilizamos el símbolo ">>".

```
[zx80]$ echo linea1 fichero.txt
[zx80]$ echo linea2 fichero.txt
[zx80]$ echo linea3 fichero.txt
[zx80]$ cat fichero.txt
linea1
linea2
linea3
[zx80]$ _
```

Cualquier cosa que envíe cosas a pantalla usando la salida estandar puede ser redirigido, por ejemplo la orden "ls", o incluso la orden "cat".

```
[zx80]$ cat fichero1.txt >> fichero2.txt
[zx80]$ ls
fichero1.txt fichero2.txt
[zx80]$ _
```

En este caso como fichero2.txt no existía, lo crea nuevo, y en este caso hemos copiado el fichero, ambos tienen el mismo contenido.

Tick 5: Cuidado con el operador ">", puede machacar documentos, pues siempre los crea nuevos.

¿Qué utilidades podemos encontrar redirigiendo la salida estandar?, veamos

```
[zx80]$ ls -l /usr/bin
```

Salen un montón de ficheros y no dá tiempo a verlos.

```
[zx80]$ ls -l /usr/bin fichero3.txt
[zx80]$ _
```

TIP 6: En lugar de teclear de nuevo "ls -l /usr/bin" podemos recuperar las órdenes anteriores con los cursores y editarlas. En este caso con pulsar la flecha arriba una vez, aparece "ls -l /usr/bin" y sólo tenemos que añadir fichero3.txt

Si usamos la orden cat, para visualizar fichero3.txt, estamos en las mismas, no nos da tiempo a ver nada, usaremos mejor la orden "more"

```
[zx80]$ more fichero3.txt
muchas lineas
--more--(12%)
```

Vemos como en esta ocasión. cuando se acaba la pantalla, la salida se para, si pulsamos el retorno de carro, avanzamos una línea, si pulsamos la barra espaciadora, avanzamos una página entera.

Salimos del paginador "more" cuando se acaba el documento a visualizar, o bien cuando pulsamos la tecla "q". En linux hay otro paginador mejor que "more", se llama "less", permite avanzar y retroceder por el texto usando las tecla del cursor.

Para volver a nuestro directorio, usamos:

```
[txt] cd ~  
[zx80] _
```

Hasta ahora, hemos visto que podemos guardar la salida estandar de un programa en un fichero, y luego, ver cómodamente esa salida estandar con un paginador, como puede ser "less".

Pero eso es un poco incómodo, pues podemos llenar el disco duro de ficheros innecesarios, ¿como podemos evitarnos ese paso?

Con ayuda de una "tubería", ya era hora que comenzara a hablar de lo que trata este capítulo.

Para utilizar una tubería, se utiliza el operador pipe y es este "|", la barra vertical que se consigue pulsando tecla "1" con la tecla "Alt Gr".

```
[zx80]$ ls -l /etc/bin | less
```

Ya podemos ver tranquilamente en la pantalla qué ficheros tenemos en el directorio /etc/bin.

Para hacerlo más divertido, con la órden tree, podemos obtener un listado de los directorios existentes a partir de un directorio en forma de árbol.

```
[zx80]$ tree -d /usr | less
```

Si lo usamos con la opción -a, muestra todo, incluidos los ficheros. Existen otros filtros que se pueden usar con una tubería, uno de los más conocidos es el filtro "sort", que ordena alfabéticamente la entrada estandar.

Cuando redireccionamos la "salida estandar" a un fichero, si se produce un error en el programa, el mensaje de error no se imprime en el fichero, si no que se vuelca a la pantalla. Esto es debido a que los mensajes de error se imprimen usando "error estandar", "stderr". Para redireccionar el error estandar.

```
[zx80]$ cc programa.c -o programa 2errores.txt
```

La órden cc, sirve para compilar programas que están escritos en lenguaje "C" y "C++", en el ejemplo, los errores de compilación se enviarán al fichero "errores.txt".

Que tipos de ficheros tenemos.

```
Login: zx80  
password:  
[zx80]$_
```

Vimos en un capítulo anterior, que con la órden "cat", podemos ver el contenido de un fichero.

```
[zx80]$ cat fichero.txt  
Esto es el contenido de mi fichero  
[zx80]$_
```

Pero vamos a cotillear otros ficheros, (si estas accediendo con un telnet no

hagas la siguiente orden), por ejemplo, vamos a ver que tiene el fichero "less" por dentro.

```
[zx80]$ cat /usr/bin/less
dsfgvrtyv456v3ty 45 tg fgf f wf werv345v
v 4345v2345v45v3 (pitido)sdf df sdfsd(pitido)
df f wef wefwe(requetepitido) sdf sdf df sdfsd
```

Hemos visto aparecer en la pantalla una sucesión de símbolos raros, pitidos, e incluso, se nos ha desconfigurado la consola. Si tecleamos algo, las letras han sido cambiadas por símbolos semigráficos. Para dejar la consola como estaba, se usa la orden "reset", también se restaura solo cada vez que se reanuda el ordenador, aunque de todas formas, cada vez es más raro que se desconfigure el terminal.

Vamos a entrar en otra sesión. Linux, puede disponer en su consola de varias sesiones virtuales, por defecto hasta siete, pulsando la combinación de teclas **Alt + F2**, vamos a la sesión 2, para volver pues **Alt + F1**, podemos combinar **Alt** con las teclas **F1** hasta **F6**. **F7** normalmente está reservada para XWindow, y para conmutar de X Window a modo texto, es necesario pulsar además la tecla **Control**.

```
Login: zx80
password:
[zx80]$ _
```

Cuando hemos volcado el fichero "less", nos ha salido los símbolos raros, por que "less" no es un fichero para ser leído, no es un fichero de texto plano, y puede contener, y contiene caracteres que no son imprimibles en la salida estandar.

Para saber que tipo de fichero es un fichero determinado, tenemos la orden "file".

```
[zx80]$ file /usr/doc/FAQ/txt/Linux-FAQ
/usr/doc/FAQ/txt/Linux-FAQ: English Text
[zx80]$ file /usr/bin/less
/usr/bin/less: ELF 32-bit LSB executable, Intel 80386, version 1,
dynamically linked (uses shared libs), stripped
[zx80] _
```

Del primer documento nos dice que es de texto, o sea, se puede volcar con "cat" y del tercero, nos dice que es un programa.

Con este programa, podemos ver los otros dos ficheros. El comando file, reconoce entre muchos tipos de ficheros, gif, jpeg, midi, y incluso ejecutables de otros sistemas operativos, y no se deja engañar por la extensión del fichero.

Los ficheros, además de crearlos con un editor de textos, se pueden copiar, mover, y borrar.

```
[zx80]$ cp /usr/bin/less .
[zx80]$ ls -l less
-rwxr-xr-x 1 zx80 users 81236 Mar 21 22:54 less
[zx80]$ _
```

Con la orden anterior, "cp", hemos copiado el fichero less a nuestro directorio de trabajo, y como esta copia es nuestra, el propietario del fichero es "zx80", y el grupo propietario es "users". Con el puntito, hemos indicado el directorio al que queremos copiar el fichero, puesto que "cp" no permite omitir el segundo parámetro, y que se tome el directorio actual como destino.

TIP 7: El usuario "root", también puede acceder y escribir en nuestros ficheros aunque quitemos los permisos.

```
[zx80]$ mkdir OtroDir
[zx80]$ cd OtroDir
[OtroDir]$ _
```

Creamos una carpeta nueva, y entramos en ella. Vamos a mover el fichero "less"

```
[OtroDir]$ mv ../less .
[OtroDir]$ ls -l
-rwxr-xr-x 1 zx80 users 81236 Mar 21 22:54 less
[OtroDir]$ _
```

Si queremos borrarlo, usamos la órden rm

```
[OtroDir]$ rm less
[OtroDir]$ _
```

Nuestra copia de "less" ya no existe.

Un poco de teoría:

¿Sabeis lo que es un inode?.

Supongamos que tenemos una partición vacía de cualquier sistema operativo tipo Unix, SCO, BSD, ext2fs ...

En esa partición vacía, hay una o más zonas, en la que no podremos guardar nuestros datos, pues son para uso administrativo del sistema operativo, esa zona, se divide en partes o "cachitos", llamados inodes, normalmente no suelen tener más de 100 octetos, aunque pueden ser de mayor tamaño, dependiendo del sistema de ficheros.

Cuando creamos un fichero vacío, el sistema operativo, rellena de datos un inode, ¿y qué datos son esos?, el nombre del fichero, en qué sectores del disco duro están, su tamaño o longitud, la fecha de última modificación, etc.

Cuando escribimos datos en el fichero, el inode se actualiza, para reflejar los cambios.

Y ahora pregunto, ¿Qué impide que un fichero tenga dos inodes?, nada. El hecho de que un fichero tenga dos inodes, implica, el que está al mismo tiempo en dos sitios con dos nombres diferentes, o en un mismo directorio con dos nombres distintos.

Eso se llama hacer un enlace.

```
[zx80]$ ls -l /usr/bin/less
-rwxr-xr-x 1 root root 81236 Aug 28 1999 /usr/bin/less
[zx80]$ ln /usr/bin/less ./paginador
[zx80]$ ls -l paginador
-rwxr-xr-x 2 root root 81236 Aug 28 1999 paginador
[zx80]$ ls -l /usr/bin/less
-rwxr-xr-x 2 root root 81236 Aug 28 1999 /usr/bin/less
[zx80]$ _
```

Fijaros en dos detalles, después de los permisos, "-rwxr-xr-x" hay un número 2, donde antes había un "1". Eso quiere decir, que ese número, es el número de inodes que tiene ese fichero.

¿Cuál es el segundo detalle?, que el fichero "paginador" sigue perteneciendo a "root", si no fuese así, sería un fallo de seguridad, pues para acceder a cualquier fichero, bastaría con hacer un enlace y podremos modificarlo a nuestro antojo.

Cuando usamos la orden rm para borrar un fichero, lo que hace es eliminar un inode, pero los demás inodes siguen existiendo, el fichero, bloque de datos del fichero, sólo se borra cuando no quedan más inodes.

Es posible que al intentar hacer el enlace, aparezca un mensaje como el siguiente:

```
ln: no se puede crear un enlace duro de './less' a '/usr/bin/less': Enlace cruzado entre dispositivos no permitido.
```

Eso es debido a que se está intentando hacer un enlace a un sistema de ficheros diferente. Por ejemplo:

```
[zx80]$ df
Filesystem lk-blocks Used Available Use% Mounted on
/dev/hda7 606405 188388 386692 33% /
/dev/hda1 23300 2655 19442 12% /boot
/dev/hda9 1756260 820291 845204 49% /home
/dev/hda8 1554262 655666 818273 44% /usr
/dev/hda6 85520 16686 64418 21% /var
[zx80]$ _
```

Con la orden "df" vemos que particiones tenemos en el disco duro, nosotros estamos en el directorio `/home/zx80`, que está en la partición `/dev/hda9` y la orden "less" está en el directorio `/usr/bin` que está en la partición `/dev/hda8`. A este tipo de enlace se le llama enlace duro.

También se pueden hacer enlaces duros, aunque muchos unices no lo permiten, y en muchos sistemas operativos, sólo está permitido a "root" y está fuertemente desaconsejado, eso es debido a que muchas aplicaciones como las utilidades de backup, y todas aquellas que exploren el disco duro para hacer o buscar algo, podrían entrar en un bucle infinito, y terminarían fallando.

Otro tipo de enlaces, son los enlaces simbólicos, estos enlaces son de alguna forma similares a los accesos directos de Windows, o mejor dicho, los accesos directos de Windows son como los enlaces simbólicos de Unix.

Si haceis analizado alguna vez un acceso directo de Windows, en realidad es un fichero PIF, o sea, un fichero que contiene el nombre de otro fichero. Los enlaces simbólicos son un fichero con el nombre de otro fichero, y tratado de forma especial por el sistema operativo para que lo trate como si fuera el fichero al que apunta. Para crear un enlace simbólico

```
[zx80]$ ln -s /usr/bin/less ./pag-simbolico
[zx80]$ ls -l pag-simbolico
lrwxrwxrwx 1 zx80 users 13 Mar 22 00:24 pag-simbolico-/usr/bin/less
[zx80]$ _
```

Observe como el fichero tiene ahora la letra "l" como primer caracter, la "l" indica enlace simbólico.

Por último nos queda por ver los dispositivos controladores. Están todos en el directorio `/dev`

Existen dos tipos, de carácter y de bloque. Los dispositivos de bloque, tienen soporte de buffer y se accede mediante grandes bloques de datos al mismo tiempo, un ejemplo, son las particiones del disco duro, y el propio disco duro.

```
/dev/hda Primer disco duro IDE
/dev/hda1 Primera partición del primer disco duro
/dev/sdb Segundo disco SCSI
```

Si volcáramos con la orden `cat` un fichero a uno de estos ficheros, escribiríamos directamente a estos ficheros, y por lo tanto, machacaríamos algo, como la tabla de particiones, o el superbloque de una partición linux, o el arranque de un Windows. Por ese motivo, sólo "root" o programas con permisos de root, puede acceder a estos ficheros.

Los ficheros de caracter, como su nombre indica, se accede a ellos caracter a caracter.

Por ejemplo `/dev/modem` suele ser un enlace simbólico a un dispositivo de carácter.

```
[zx80]$ ls -l /dev/modem
lrwxrwxrwx 1 root root 10 Dec 22 01:05 /dev/modem ->/dev/ttyS1
[zx80]$ ls -l /dev/ttyS1
crw----- 1 root tty 4, 65 Mar 21 22:11 /dev/ttyS1
[zx80]$_
```

Los programas que acceden al modem, lo hacen como si fuera un fichero convencional, (para los que conozcan el lenguaje "C" `fopen()`, `fread()`, `fwrite()`, `fclose()`, `fprintf()`, etc.).

```
[zx80]$ tty
/dev/tty2
[zx80]$_
```

La órden anterior, "tty", nos dice cual es el controlador de dispositivo de nuestra sesión. `/dev/tty`

Si el usuario "root" estuviera conectado mediante un telnet, podría hacer una gamberrada tal y como

```
[root] # echo "Tonto el que lo lea" /dev/tty2
```

Y en mi pantalla aparecería ese mensaje.

Tick 8: En las academias, donde todo el mundo consigue por ingeniería social el password de root, es muy posible que cuando la gente se aburre, no se pueda trabajar debido a que todo el mundo envíe mensajitos usando este método.

También es común gamberradas más gordas volcando ficheros de todo tipo, como por ejemplo, los que tengais tarjeta de sonido y micrófono, grabar fichero de sonido desde el micrófono.

```
cat /dev/audio grabación.raw
```

Y posteriormente lo envieis a los altavoces

```
cat /grabacion.raw /dev/audio
```

No me acuerdo como se cortaba la grabación, ¿control + "c"?

٪