# UD04.- Administración de un Sistema Operativo Libre I

## Caso práctico

Nuria Barroso (Elaboración propia)

Iván y Marta, en su empresa de prácticas, han tenido que instalar un servidor Linux, para optimizar el trabajo del sistema operativo y facilitar el trabajo a los usuarios y usuarias. Una vez instalado van a necesitar realizar ciertas tareas administrativas adicionales, las de administración del sistema y otras útiles para cualquier usuario o usuaria como la instalación de nuevo software, la actualización del sistema operativo, la instalación o uso de los servicios del sistema, mejorar el rendimiento, etc.

Shell que en castellano significa concha, es el intérprete de comandos del sistema.
Es una interfaz de texto de altas prestaciones, que te sirve fundamentalmente para tres cosas:

1. Administra el sistema operativo.
2. Lanza aplicaciones.
3. Interactúa con ellas y como entorno de programación.

El shell es tanto un intérprete de comandos u órdenes, como un lenguaje de programación. Sobre él vas a poder ejecutar órdenes con las que puedes crear programas, que se llaman script y ejecutar **comandos** para administrar el sistema operativo.

# 0.- Conceptos fundamentales

En este capítulo se presentan conceptos fundamentales para administrar un sistema operativo libre.

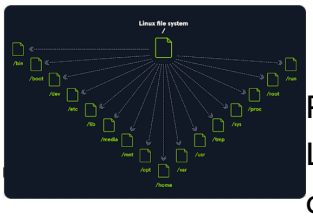
La administración de un sistema incluye una amplia gama de tareas tales como las de instalar una impresora o un escáner, configurar y compartir el acceso a Internet, instalar programas, configurar cortafuegos, añadir nuevos usuarios, etc... en definitiva, crear un entorno de trabajo seguro, cómodo y productivo.

Además garantizará la disponibilidad, integridad y confiabilidad de la información.

Junto a todo esto, las operaciones de administración de sistemas también están encaminadas a proporcionar servicio de soporte técnico, solucionar problemas y la actualización de la documentación del propio sistema.

La administración de cuentas de usuario y grupos es una parte esencial de la administración de sistemas dentro de una organización. Pero para hacer esto efectivamente, un buen administrador de sistemas primero debe entender lo que son las cuentas de usuario y los grupos y cómo funcionan.

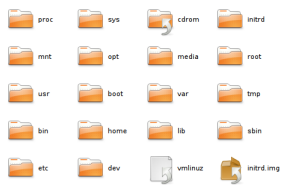
Ya vimos en la unidad anterior como realizar esta gestión de forma gráfica. En ésta, detallaremos los tipos de usuarios, para realizar su administración en la siguiente unidad.



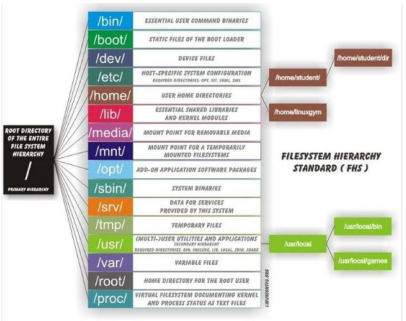

Por otro lado, conviene conocer el esquema del árbol de directorios de Linux, antes de que empecemos a trabajar con la interfaz de línea de comandos.

# 0.1.- Jerarquía del sistema de archivos

## Estándar FHS - Filesystem Hierarchy Standard




El estándar de jerarquía del sistema de archivos (en inglés Filesystem Hierarchy Standard, también conocido por sus siglas FHS) es una norma que define los directorios principales y sus contenidos en el sistema operativo GNU/Linux y otros sistemas de la familia Unix. Se diseñó originalmente en 1994 para estandarizar el sistema de archivos de las distribuciones de Linux, basándose en la tradicional organización de directorios de los sistemas Unix. En 1995 se amplió el ámbito del estándar a cualquier Unix que se adhiera voluntariamente.




FHS define algunos directorios con mucha precisión. Los más comunes definidos por FHS o utilizados por convención, son los siguientes:

**/**: Los sistemas de ficheros Linux tienen su raíz en un mismo directorio conocido como sistema de ficheros raíz o directorio raíz. Todos los demás directorios se ramifican desde éste. Linux no utiliza letras de unidad, sino que las particiones o discos extraíbles se montan en un punto dentro del sistema de ficheros raíz. Algunos directorios críticos deben residir siempre en la partición raíz, pero otros pueden encontrarse en particiones independientes. No se debe confundir el directorio /root con el directorio raíz.

**/boot:** Contiene ficheros estáticos y no compartibles relacionados con el arranque del ordenador. Algunos sistemas imponen límites particulares a /boot , por ejemplo, en BIOS antiguas y versiones antiguas del LILO pueden requerir que /boot se encuentre por debajo del cilindro 1024 del disco duro. También puede que se requiera que /boot sea una partición independiente.

**/bin:** Contiene algunos ficheros ejecutables que son accesibles para todos los usuarios y constituyen los comandos más importantes que pueden ejecutar los usuarios normales. Contiene ficheros estáticos. Sus ficheros son compartibles, pero son tan importantes para el funcionamiento básico del ordenador que este directorio casi nunca se comparte. Cada cliente debe tener su directorio /bin en local.

**/sbin:** Es similar a /bin, pero contiene programas que sólo ejecuta el administrador. Es estático y en teoría compartible. En la práctica, sin embargo, no tiene sentido compartirlo.

**/lib:** Contiene bibliotecas de programa que son código compartido por muchos programas y que se almacenan en ficheros independientes, para ahorrar RAM y espacio en disco. /lib/modules contiene módulos o drivers que se pueden cargar y descargar según necesitemos. Es estático y teóricamente compartible, aunque en la práctica no se comparten.

**/usr:** Aloja el grueso de los programas de un ordenador Linux. Tiene un contenido compartible y estático, lo que permite montarlo en modo solo lectura. Se puede compartir con otros sistemas Linux; muchos administradores separan /usr en una partición independiente, aunque no es necesario. Contiene algunos subdirectorios similares a los del directorio raíz como /usr/bin y /usr/lib , que contienen programas y bibliotecas que no son totalmente críticos para el funcionamiento del ordenador.

**/usr/local:** Contiene subdirectorios que reflejan la organización de /usr. Aloja los ficheros que instala localmente el administrador y es un área a salvo de las actualizaciones automáticas de todo el SO. Después de la instalación de Linux debería estar vacío, excepto para determinados subdirectorios stub. Se suele separar en una partición para protegerlo de las reinstalaciones del SO.

**/usr/X11R6:** Alberga los ficheros relacionados con el sistema X Window (entorno GUI). Contiene subdirectorios similares a los de /usr, como /usr/X11R6/bin y /usr/X11R6/lib.

**/opt:** Es similar a /usr/local, pero está pensado para los paquetes que no vienen con el SO como los procesadores de texto o juegos comerciales, que se guardan es sus propios subdirectorios. El contenido de /opt es estático y compartible. Se suele separar en su propia partición para convertirlo en un enlace simbólico a un subdirectorio de /usr/local.

**/home:** contiene los datos de los usuarios y es compartible y variable. Se considera opcional en FHS, pero, en la práctica, lo opcional es el nombre. El directorio /home con mucha frecuencia reside en su propia partición.

**/etc:** Contiene archivos de configuración del sistema específicos del Host de todo el sistema.

**/root:** Es el directorio home del usuario root. Como la cuenta de root es tan crítica y específica del sistema, este directorio variable no es realmente compartible.

**/var:** Contiene ficheros efímeros de varios tipos, de registro del sistema, de cola de impresión,

de correo y news, etc. El contenido del directorio es variable, pues algunos subdirectorios son compartibles y otros no. Se suele colocar /var en su propia partición, sobre todo si el sistema registra una gran actividad en /var.

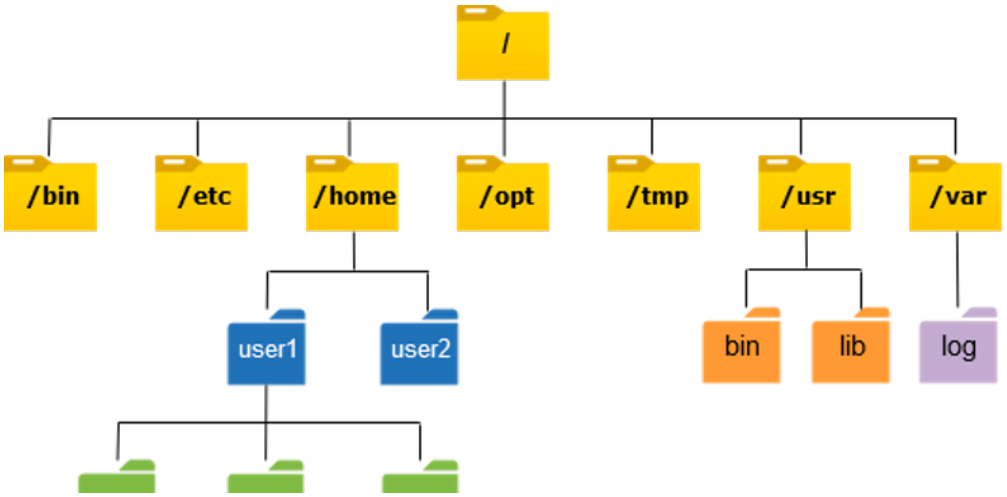
**/tmp:** Es donde se crean los archivos temporales y variables que necesitan los programas. La mayoría de las distribuciones limpian este directorio periódicamente en el inicio. Este directorio raramente se comparte, pero se suele poner en una partición independiente para que los procesos no controlados no provoquen problemas en el sistema de ficheros al ocupar demasiado.

**/mnt:** La finalidad de este directorio es albergar el montaje de los dispositivos. En la estructura de directorios, algunas distribuciones crean subdirectorios dentro de /mnt para que hagan de puntos de montado; otras utilizan directamente /mnt o incluso puntos de montado independientes de /mnt, como /floppy o /cdrom. FHS sólo menciona /mnt y no especifica cómo se ha de utilizar. Los medios montados en esta partición pueden ser estáticos o variables y, por norma general, son compartibles.

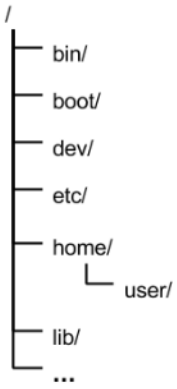
**/media**: Es una parte opcional del FHS como /mnt, pero que podría contener subdirectorios para tipos de medio específicos. Muchas distribuciones modernas utilizan subdirectorios /media como punto de montado para los discos extraíbles.

**/dev:** Linux trata la mayoría de los dispositivos de hardware como si fueran ficheros y el SO debe tener un lugar para estos en su sistema de ficheros. Ese lugar es el directorio /dev que contiene un gran número de ficheros que hacen de interfaces de hardware. Con los permisos apropiados se accede al hardware del dispositivo leyendo y escribiendo en el fichero de dispositivo asociado. El kernel permite que /dev sea un sistema de ficheros virtual creado automáticamente. El kernel y las herramientas de soporte crean sobre la marcha entradas en /dev para adaptarse a las necesidades de los drivers específicos. La mayoría de las distribuciones emplean este recurso.

**/proc:** Es un directorio inusual, ya que no corresponde a un directorio o partición normal, sino que se trata de un sistema de ficheros virtual que proporciona acceso a ciertos tipos de información del hardware dinámicamente. Esta información no se encuentra accesible a través de /dev.



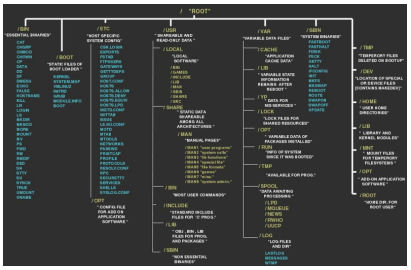

## Debes conocer



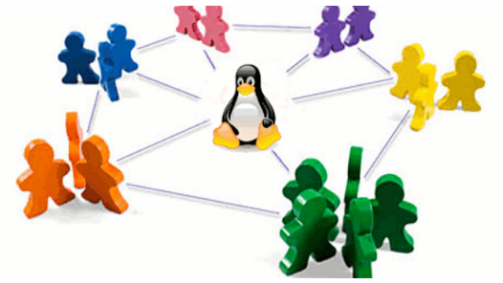

En la administración de Linux conocer la finalidad de los directorios resulta tremendamente útil ya que si instalamos por ejemplo un programa en una ubicación equivocada, un binario colocado en /bin cuando debería estar en /usr/local/bin puede que se sobrescriba o elimine al realizar una actualización del sistema.

## Para saber más

Puedes consultar el árbol de directorios ampliado pinchando en esta imagen.



## 0.2. Tipos de usuarios



La seguridad informática se basa en la administración efectiva de los permisos de acceso a los recursos informáticos. La administración de cuentas de usuario y grupos es una parte esencial de la administración de sistemas dentro de una organización.

La razón principal para la existencia de cuentas de usuario es verificar la identidad de cada individuo utilizando un ordenador. Y otra razón, también importante, es la de permitir la utilización personalizada de recursos y privilegios de acceso.

Por tanto, las cuentas de usuario permiten la autenticación y la asignación de permisos.

### Debes conocer

En Linux hay tres tipos de cuentas de usuario: root, estándar o locales, y las de usuarios asociados a servicios, llamadas también cuentas del sistema.

- La cuenta del **usuario root**: es la del dueño y administrador del sistema Linux. Tiene acceso total a todos los archivos del sistema, puede crear otros usuarios, instalar software, compilar el Kernel, etc. Por todo ello es aconsejable utilizarla únicamente para la administración de nuestro sistema. También podemos identificarlo como **superusuario**.

- Las cuentas de los **usuarios estándar o locales**: tienen limitaciones en cuanto a las acciones que pueden iniciar así como a los archivos y carpetas a los que pueden acceder, salvo en su directorio personal en este último caso. Es por ello que son los usuarios recomendados para el uso diario del sistema.

- Las cuentas del sistema o cuentas de **usuarios asociados a servicios**: las cuales no pueden iniciar sesiones en el sistema, pero por medio de ellas se pueden establecer los permisos asociados a dichos servicios, ejemplos son las cuentas de los usuarios apache, bin... Es decir, son cuentas que existen para ser usadas con aplicaciones específicas. Los usuarios de servicios no existen físicamente, se emplean por

razones de administración.

# 1.- La shell y sus comandos.

## Caso práctico

Nuria Barroso (Elaboración propia)

En el aula Iván y Marta tuvieron su primer encuentro con la shell, en un principio pensaron que no les iba a hacer falta en el mundo laboral.

En las empresas de prácticas han visto que pueden llegar a trabajar con Red Hat, Ubuntu y Knoppix. Por ello, para hacer una labor administrativa básica se ven en la dificultad de que en cada distribución tienen que estudiar el entorno gráfico, con los cambios que cada distribución tiene.

Esto, lo han solucionado fácilmente, pues se han acostumbrado a trabajar con comandos de texto, y así no necesitan trabajar con el entorno gráfico, van a tiro hecho usando comandos.

Vamos a entrar en el mundo de la línea de comandos de Linux, verás que es muy flexible y potente, y no tan complicado como pudiera parecer en un principio.

## 1.1.- Antecedentes históricos.

La **shell** te permite interactuar con el **kernel** a través de la interpretación de los comandos que escribes en la línea de comandos, o a través de los scripts.

Cada shell cuenta con sus propias características de uso, contando con atajos de teclado, visualización en vivo de ficheros y atajos durante la navegación entre directorios.

- Hay varias shell, la primera shell fue programada por Steven Bourne, a ésta se le llama **sh o bsh**, es una shell limitada y usa una sintaxis de **comandos** usada en los primeros **sistemas UNIX**.
- Cronológicamente, la siguiente shell fue la **c-shell o csh**, como su nombre indica usa comandos muy parecidos al lenguaje de **programación C.**
- En 1986, David Korn juntó lo mejor de la Bourne shell y la c-shell y programó la **korn-shell o ksh.**
- En la mayoría de los sistemas Linux, viene incorporada por defecto la shell bash de Bourne Again Shell, En referencia al inventor de la primera shell. Esta shell posee toda la funcionalidad de sh con características avanzadas de C.
- Si creamos cualquier script .sh correrá perfectamente en la shell bash.

La shell bash fue programada por desarrolladores del **proyecto GNU** (Proyecto que potencia el software libre, iniciado por Richard Stallman).

Para acceder al modo texto o al **terminal**, a partir de Ubuntu 22.04, puedes situarte sobre el escritorio y pulsar el botón derecho para seleccionar la opción de *Terminal*.

Si quieres saber las shells que tienes en tu **distribución GNU/Linux** sólo tienes que ver el contenido del fichero shells del **directorio /etc**. Para ello, escribe en la terminal: **cat /etc/shells**

### Autoevaluación

**La shell bash:**

- ○ Debe su nombre a Steven Bourne.
- ○ Se debe a David Korn.
- ○ No existe.

Muy bien. Has captado la idea.

Incorrecta, David Korn desarrollo la ksh.

No es la respuesta correcta, la shell bash viene por defecto en la mayoría de las distribuciones de Linux.

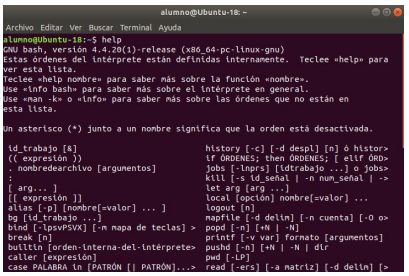
**Solución**

1. **Opción correcta**
2. **Incorrecto**
3. **Incorrecto**

# 1.2.- Generalidades sobre introducción de comandos.

La sintaxis de las órdenes es la siguiente: **orden** [ **-modificadores**] [**argumentos**]

- La orden indica al intérprete de comandos la acción a realizar.
- Los modificadores u opciones se introducen precedidos del carácter – guion.
- Los argumentos son caracteres que se utilizan como entrada del **comando**. El argumento puede ser un archivo o un **directorio**.




Agustín Nieto Espino

(Elaboración propia)

  Antes de ver los comandos básicos,existen distintos modos de poder ejecutar varios comandos. Ahora veremos sólo la ejecución en secuencia.

- **En secuencia**. Separados por punto y coma. Se ejecuta un comando detrás de otro en **primer plano** o foreground:

```
comando1 ; comando2 ; comando3
```

Para ver el listado de comandos, puedes escribir en la **terminal** el comando **`help`**. La ayuda de un comando la puedes obtener de distintas maneras con **`info`**, **`help`** o **`man`** (acrónimo de manual):

```
comando1 ; comando2 & comando3 ; comando4 & comando5...
```

Otra ayuda con la que cuentas es la opción de **autocompletar**, para ello, empiezas a escribir en la terminal y a continuación pulsas la tecla de tabulación, con la que se completará el nombre.

## Autoevaluación

**Para introducir comandos, ¿Sólo se puede introducir uno por línea?**

- ○ Verdadero.
- ○ Falso.

Incorrecto. En una misma línea se pueden escribir varios comandos separados por punto y coma.

Efectivamente, hay varias maneras usando & o punto y coma.

**Solución**

1. **Incorrecto**
2. **Opción correcta**

## 1.3.- Directorios.

En el sistema operativo Windows, existe el concepto de carpetas que es la forma que existe de organizar los ficheros. En Linux, las carpetas reciben el nombre de **directorios**. Por lo tanto, carpeta o directorio es lo mismo. Depende del sistema operativo en el cual nos encontremos será nombrada de una manera u otra.

Con los directorios puedes almacenar información de forma ordenada. Es una forma de estructurar la información almacenada en nuestro soporte de almacenamiento. Por lo tanto, los directorios son un tipo especial de ficheros que almacenan el listado de los ficheros o subdirectorios que contienen.

Vas a ver una lista de **comandos** que te serán útiles cuando quieras ver el contenido de un fichero, copiar ficheros, eliminaros, moverles y movernos por la estructura jerárquica o árbol de directorios. Los comandos son:

- **`pwd`** (Acrónimo de Print Working Directory). Muestra la trayectoria absoluta del directorio de trabajo.
- **`ls`** (Acrónimo de list): Enumera al contenido de un directorio, mostrando el nombre y extensión de cada archivo, el tamaño en bytes, la fecha y hora en que se creó o modificó por última vez, y los subdirectorios que cuelgan de él.
- **`chdir`** o **`cd`** (Acrónimo de Change Directory). Con este comando podemos movernos por la estructura de directorios creada. Sirve para subir o bajar por la estructura jerárquica. Si ejecutamos el comando cd sin **parámetros**, nos situaremos automáticamente en el directorio de trabajo con el cual nos encontremos identificados en el sistema.
- **`mkdir`** (Acrónimo de Make directory). Crea directorios por debajo del actual. Por ejemplo, si queremos crear el directorio pruebas, escribiríamos la orden mkdir pruebas. Si quisiéramos crear una ruta de directorios de una sola vez. Por ejemplo, queremos crear los directorios som/ejemplos/tareas, utilizaríamos el comando `mkdir` pasándole el parámetro -p para que se creen todos directorios intermedios que aparecen en la ruta ya que no existen. Escribiríamos en el terminal: mkdir -p som/ejemplos/tareas

- **`rmdir`** (Acrónimo de Remove Directory). Con este comando podemos eliminar directorios en cualquier parte de la estructura jerárquica siempre y cuando tengamos los privilegios suficientes.

  Para borrar directorios hemos de tener en cuenta que sólo se puede borrar un directorio si está vacío con este comando

- Si quisiéramos eliminar un directorio que no se encontrara vacío debemos de utilizar el comando **`rm`** pasándole el **parámetro** **`-r`**. Por ejemplo, queremos eliminar el directorio

pruebas que no se encuentra vacío. Escribiríamos la orden: **`rm -r pruebas.`**

Por ejemplo, vas a crear dos directos: directorio1 y directorio2 en tu directorio de trabajo. El directorio de trabajo de los usuarios del sistema suele estar situado dentro del directorio /home. Al acceder a un terminal nos situamos directamente en el directorio de trabajo del usuario. Supón que el directorio de trabajo es /home/usuario. Para ello, escribe en la terminal:

```
mkdir directorio1 directorio2
```

Comprueba que las has creado haciendo un listado de tu directorio de trabajo. Para ello escribes: **`ls`**

Si quieres ir a la carpeta **directorio2**, escribe en la terminal: **`cd directorio2`**
Verás que te ha cambiado el **prompt**.

Para volver al directorio anterior, escribe en la terminal: **`cd ..`**

---

Debes de observar que para separar los directorios al indicar una ruta se utiliza el símbolo **/** en el sistema operativo Linux mientras que en los sistemas operativos Windows se utiliza, para separar las carpetas al indicar una ruta, el carácter ****

El sistema operativo Linux distingue entre mayúsculas y minúsculas a la hora de ejecutar una orden. Por lo tanto, debemos de tener cuidado al escribirlas. Para Linux, una orden escrita en minúsculas no es lo mismo que una orden escrita en mayúsculas. Al contrario de lo que ocurría en el sistema operativo Windows.

---

## Debes conocer

---

En el siguiente vídeo, puedes ver cómo utilizar los comandos esenciales para poder utilizar correctamente los directorios:

**Comandos para manejar directorios**


Agustín Nieto Espino. *Descripción textual alternativa para el vídeo "Comandos para manejar directorios".*

(Elaboración propia)

## Autoevaluación

**Relaciona los comandos con lo que muestran, escribiendo el número asociado a lo que hace el comando en el hueco correspondiente.**

Ejercicio de relacionar

| Comando | Relación | Resultado |
|---|---|---|
| ls | | 1.- Crea directorios. |
| cd | | 2.- Lista ficheros de un directorio. |
| mkdir | | 3.- Borra carpetas y subcarpetas. |
| rm -r | | 4.- Cambia de directorio. |

Son algunos de los comandos más usados sobre directorios.

# 1.4.- Ficheros.

Vas a ver una lista de comandos que te serán útiles cuando quieras trabajar con ficheros y obtener listados de los mismos, realizar copias, reubicarles, etc.

**COMANDO LISTAR (ls)**

**ls** (acrónimo de list). Lista de ficheros del **directorio** (por defecto el actual).

- **-l** Con detalles, lista los atributos. Muestra por cada fichero o directorio toda la información posible.


Agustín Nieto Espino (Elaboración propia)

Entre la información que nos muestra destacan:

- El tipo de fichero y los **permisos** asociados al objeto.

- El propietario o dueño del objeto

- El grupo de usuarios a los cuales pertenece el objeto

- La fecha de la última vez que el objeto ha sido modificado.

- **-a** Incluye ficheros ocultos (Los que comienzan con el carácter punto ".").
- **-R** Lista recursivamente los contenidos de los directorios. Es decir, muestra el contenido de todos los subdirectorios que existieran.
- **-al** También puedes combinar los **parámetros**.

Si ejecutamos el comando `ls -l`, se nos mostrará por cada objeto que exista, en el lugar que nos encontremos, la información indicada anteriormente. Si solo queremos ver la información posible de un fichero o de un directorio deberemos de indicar su nombre a continuación de la orden.

---

Ejemplo: visualizar la información del directorio Descargas

Escribiríamos en el terminal la orden: **`ls -l Descargas`**

---

**OTROS COMANDOS**

**cat** (Acrónimo de concatenate). Muestra el contenido de los ficheros.

**more** fich1 [fich2] … Muestra el contenido página a página. Un uso muy típico del comando more es como terminación de una **tubería**, para mostrar los resultados de otro comando página a página. Por ejemplo: ps -a -x -l | more

**cp** fichOrigen fichDestino. cp es acrónimo de copy. Copia el fichero origen con el nuevo nombre destino sobrescribiendo el ya existía. Si queremos copiar un fichero en el interior de un directorio, solo es necesario indicar el nombre del fichero que queremos copiar y la ruta o path de destino.

**mv** fichOrigen fichDestino. Mueve el fichero. Cambia el nombre y/o la localización.

**file** fich. Indica de que tipo es el fichero (ejecutable, texto, …).

**sort** [fich] Ordena el fichero, o la entrada por defecto. Consulta sus opciones en la página de manual. El nombre de fichero es opcional. Si no hay fichero, busca en la entrada por defecto. Por eso, se puede utilizar como parte de una tubería (filtro).

**rm** se utiliza para borrar ficheros. Se le debe de indicar como parámetro el nombre del fichero que deseamos eliminar. No hace falta desplazarnos al lugar en donde se encuentre el fichero a eliminar, le podemos indicar la ruta absoluta o relativa junto con el nombre del fichero que se desea eliminar.

---

Ejemplo: crea un fichero con la salida del comando ls

Para ello, escribe en la terminal: **`ls > fichero1`**

Para ver el contenido del fichero fichero1 que acabas de crear, escribe en la terminal: **`cat fichero1`**

Ahora si quieres copiar el fichero fichero1 en directorio1, suponiendo que directorio 1 ya existe, escribe en el terminal: **`cp fichero1 /home/usuario/directorio1`**

También sería correcto la orden: `cp fichero1 /home/usuario/directorio1/fichero1`

---

## Debes conocer

---

En el siguiente vídeo, puedes ver como se utilizan algunos comandos sobre los ficheros:

**Comandos para manejar los ficheros**


0:00 / 6:17

Agustín Nieto Espino. Descripción textual alternativa para el vídeo "Comandos para manejar los ficheros". (Elaboración propia)

## 1.4.1.- Comodines. Carácter de sustitución.

Durante la utilización de comandos en relación con el sistema de ficheros, puede resultar interesante filtrar la salida de nombres de ficheros con la ayuda de determinados criterios, por ejemplo con el comando **`ls`**. En vez de visualizar toda la lista de ficheros, se puede filtrar la visualización de varios criterios y caracteres especiales.

| Caracteres | Significado |
|---|---|
| `*` | 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 la letra **a** a la letra **b** incluida). |
| `[!...]` | Inversión de la búsqueda. |
| `[^...]` | Inversión de la búsqueda. |

Supongamos que tenemos la siguiente salida por consola:

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

Veamos algunos ejemplos aplicando los caracteres comodines que acabamos de ver:

1. Obtiene todos los ficheros que empiezan con a:

   ```
   $ ls a*
   afic1 afic2 afic3
   ```
2. Todos los ficheros de cuatro caracteres que empiezan con a:

   ```
   $ ls a???
   afic
   ```
3. Todos los ficheros de al menos tres caracteres y que empiezan con b:

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

4. Todos los ficheros que terminan con 1 o 2:

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

5. Todos los ficheros 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
```

6. Todos los ficheros que no terminan por 3:

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

---

## Debes conocer

---

## ¿Cómo interpreta la Shell los caracteres comodín?

El shell es el encargado de efectuar la sustitución de estos caracteres antes de pasar los parámetros a un comando. Así, en el momento de ejecutar el comando:

```
$ cp * Documents
```

cp no recibe el carácter *, sino la lista de todos los ficheros y directorios del directorio activo.

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

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

---

## Para saber más

---

## Caracteres especiales

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

- La **contrabarra** \ permite cerrar un carácter único. ls paga\ *.xls va a listar todos los ficheros 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 fichero.

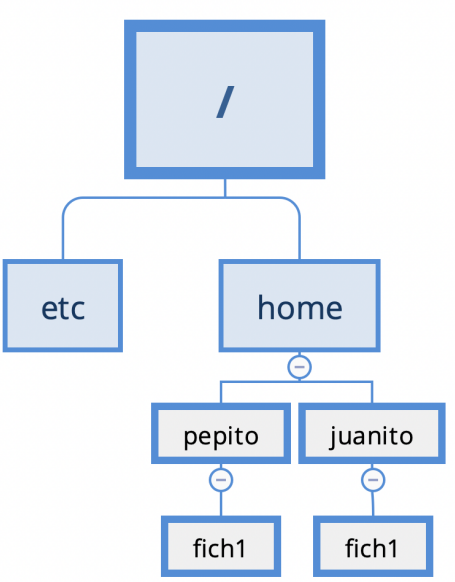
---

# 1.4.2.- Rutas

## 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 o archivo). Se suele completar un nombre de archivo con su ruta de acceso.

Por eso, el archivo **fich1** del directorio **pepito** es diferente del archivo **fich1** del directorio **Juanito**. Al ser jerárquico, el sistema de archivos de Linux tiene forma de estructura en árbol.




Esta imagen 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/fich1`**

**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 **. (punto)** ni **..(doble punto)**

**Una 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 actual o activo. Suele estar implícito. Los dos puntos .. representan el directorio de nivel superior o directorio padre.

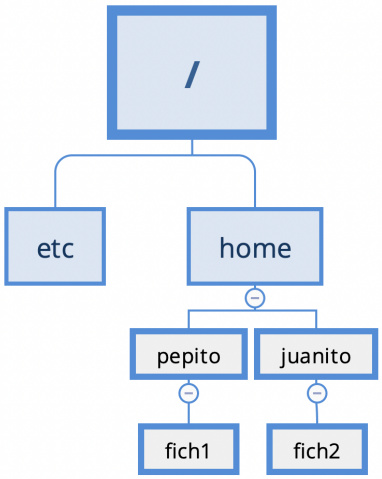
Una ruta relativa:

- 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 el punto .  o los 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 Documento en el directorio actual.
- `./Documents/Photos` es una ruta relativa perfectamente idéntica a la anterior, con la salvedad de que el punto indica el directorio activo (actual) de manera explícita. "`./Documents`" indica de manera explícita el directorio `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`.

Ejemplo con la estructura de la imagen siguiente:




Suponiendo que estamos en el directorio `/home/juanito` podemos acceder a `fich1` así:

- Con ruta absoluta --> `/home/pepito/fich1`
- Con ruta relativa --> `../pepito/fich1`

Siguiendo con la misma imagen y en el mismo directorio (`/home/juanito`), para acceder a `fich2` podemos acceder de dos maneras:

- `./fich1`
- `fich1`

**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:

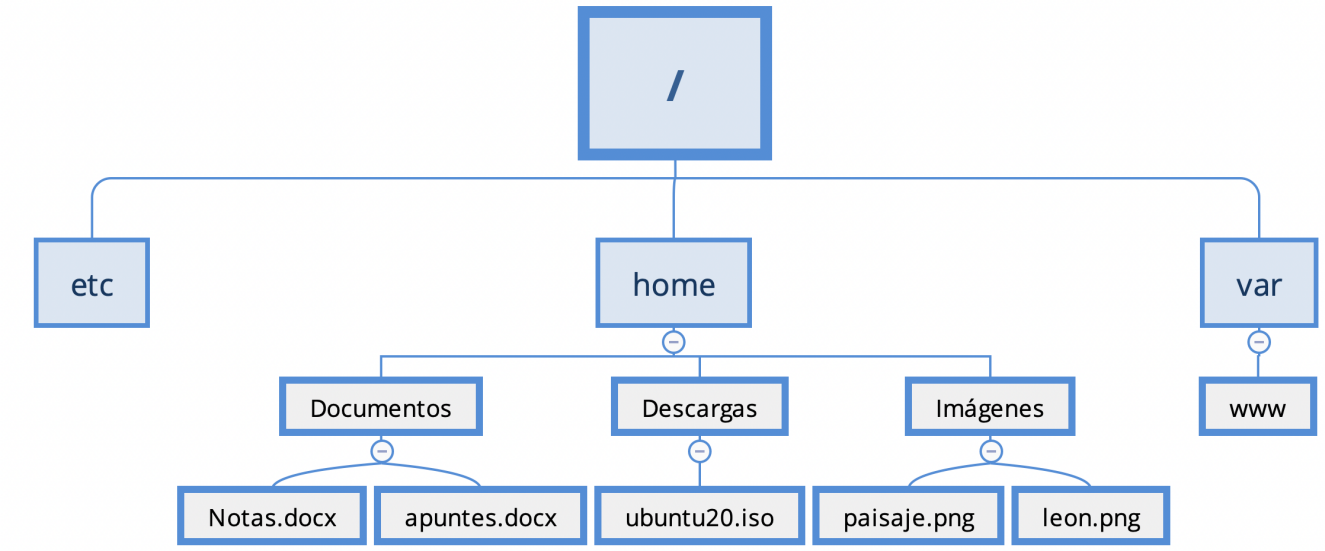
```
$ cd /~
```

---

## Autoevaluación

---

Teniendo en cuenta la imagen con la siguiente estructura de directorios y archivos:




y suponiendo que nuestro directorio actual es `Imágenes`, ¿Cuál de las siguientes opciones para acceder al fichero `apuntes.docx` es correcta?

- Documentos/apuntes.docx
- home/Documentos/apuntes.docx
- ./apuntes.docx
- ../Documentos/apuntes.docx

Incorrecto

Incorrecto

Incorrecto

Opción correcta

**Solución**

1. **Incorrecto**
2. **Incorrecto**
3. **Incorrecto**
4. **Opción correcta**

# 1.5.- Visualización de texto

## Visualización de texto

Con los comandos que verás en cada una de las pestañas puedes mostrar por pantalla parte del contenido de un archivo y darle un aspecto de tabla si lo necesitas. Pulsa en cada una de ellas para averiguar cómo:

### El comando head

Sirve para mostrar por la salida estándar (por pantalla) el comienzo del **contenido de un archivo**. Su sintaxis general es la siguiente:

**head [-c número_bytes] [-n número_líneas] [fichero] ...**

- **-c** se refiere a bytes (o caracteres) y número_bytes a la cantidad que se muestran.
- **-n** se refiere a líneas y número_líneas a la cantidad de líneas que se muestran.
- **fichero** es el fichero en cuestión.
- ... indica que podemos poner más de un fichero.

**Ejemplo**: con el comando cat, que ya conocemos, mostramos el contenido de lista.txt y, posteriormente, vemos como solo mostramos las tres primeras líneas del mismo fichero

```
santos@santos-vm:~$ cat lista.txt
comprimido_1-4_all.deb
cuenta.txt
Descargas
Documentos
errores.txt
Escritorio
Imágenes
santos@santos-vm:~$ head -n 3 lista.txt
comprimido_1-4_all.deb
cuenta.txt
Descargas
```

### El comando tail

Sirve para mostrar por la salida estándar (por pantalla) el final del contenido de un archivo. Su sintaxis general es la siguiente:

**tail [-valor[b/c]] [-f] [fichero] ...**

- **valor** es la cantidad de b - bloques de 512 bytes se muestran o c - caracteres. Si no indican **b** o **c,** se mostrarán líneas.
- **-f** indica que el fichero se queda abierto y mostrará en tiempo real la modificaciones que sufra. La apertura del fichero se termina pulsando CTRL+C
- **fichero** es el fichero en cuestión.
- ... indica que podemos poner más de un fichero.

Ejemplo:

```
santos@santos-vm:~$ cat lista.txt
comprimido_1-4_all.deb
cuenta.txt
Descargas
Documentos
errores.txt
Escritorio
Imágenes
santos@santos-vm:~$ tail -5 lista.txt
sorpresa-1-3.noarch.rpm
Vídeos
xaa
xab
xac
```

Permite mostrar información en forma de tabla y, entre otras opciones, admite:

- `-t` para indicar que muestre la información en forma tabular.
- `-s` para indicar el caracter delimitador para formar las columnas de las tabla.

**Ejemplo:**

```
santos@santos-vm:~$ column -t -s: /etc/group
root              x  0
daemon            x  1
bin               x  2
sys               x  3
```

## 1.6.- Comando alias

Habitualmente, necesitamos usar un comando una y otra vez. Escribir o copiar el mismo comando una y otra vez reduce nuestra productividad y nos distrae de lo que realmente estamos haciendo. Para ahorrar algo de tiempo tenemos a nuestra disposición los alias.

Los alias son como accesos directos personalizados utilizados para representar un comando (o conjunto de comandos) ejecutados con o sin opciones personalizadas. Es muy probable que estemos utilizándolos sin darnos cuenta.

### El comando `alias` en Linux

Antes de nada, veamos si en nuestro sistema hay definidos alias o no. Para ello ejecutamos el comando alias sin opciones, tal cual:

```
$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -lF'
alias ls='ls --color=auto'
```

Como podemos observar, **nuestro sistema sí tiene alias diferentes** listos para utilizarse. Estos alias que vemos, los consideramos permanentes porque existen en cada reinicio del sistema. Para conseguir esto, tenemos que guardar la definición de los alias que queramos que sean permanentes en un **archivo específico del sistema**.

La sintaxis general para definir un alias es la siguiente:

```
alias nombre="comando personalizado"
```

Cuando utilizamos el comando alias directamente en el terminal, estamos definiendo **alias temporales**. Veamos un ejemplo de creación y uso de alias:

Creamos el alias:

```
$ alias lo='ls -lrt'
```

Comprobamos que se ha añadido:

```
$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo
terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]
\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -lF'
alias lo='ls -lrt'  <-- aquí está
alias ls='ls --color=auto'
```

Ejecutamos el alias para comprobar que funciona correctamente:

```
$ lo
total 136
drwxr-xr-x 2 santos santos 4096 ene  1 13:54 Vídeos
drwxr-xr-x 2 santos santos 4096 ene  1 13:54 Público
drwxr-xr-x 2 santos santos 4096 ene  1 13:54 Plantillas
drwxr-xr-x 2 santos santos 4096 ene  1 13:54 Escritorio
drwxr-xr-x 2 santos santos 4096 ene  1 13:54 Documentos
drwxr-xr-x 2 santos santos 4096 ene  1 13:54 Descargas
-rw-rw-r-- 1 santos santos   32 ene  9 12:11 lista
-rw-rw-r-- 1 santos santos  555 ene 10 20:20 listado.txt
-rw-rw-r-- 1 santos santos   12 ene 10 20:57 cuenta.txt
...
```

Por otro lado, para eliminar un alias definido, como el que acabamos de crear, escribimos **unalias** y el nombre del alias que queremos eliminar:

```
$ unalias lo
```

Comprobamos que ya no está:

```
$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo
terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]
\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
```

```
alias ll='ls -lF'
alias lo='ls -lrt'
alias ls='ls --color=auto'
```

Sólo queda añadir que para eliminar todos los alias creados a la vez, ejecutaremos:

```
$ unalias -a
```

Para establecer un alias de forma permanente en una **shell bash**, tenemos que editar el archivo .bashrc ubicado en el directorio personal del usuario. Por tanto, tendremos que editar este archivo ya añadir al final las líneas de los comando alias que queramos tener activos cada vez que abramos una terminal del comandos.

Una vez añadidos los alias, podemos reiniciar el sistema para que se carguen automáticamente o forzar su carga con el comando:

```
$ source ~/.bashrc
```

## Para saber más

Para ampliar información sobre el comando alias visita el enlace:

Comando alias <http://manpages.ubuntu.com/manpages/focal/en/man1/alias.1posix.html>

# 1.7.- Enlaces

## ¿Qué son los enlaces?

Un enlace es un puente a un archivo o directorio y representa una referencia que podemos poner en cualquier lugar que nos interese y que actúa como un acceso directo a cualquier otro.

Disponemos de dos tipos de enlaces:

- Enlaces blandos o simbólicos.
- Enlaces duros.

### Enlaces blandos o simbólicos

Un **enlace simbólico** indica un acceso a un directorio o fichero que se encuentra en un lugar distinto dentro de la estructura de directorios. Los enlaces simbólicos actúan como una cadena que crea rutas para diferentes archivos o carpetas en el sistema.

Los enlaces simbólicos no solo son útiles para crear accesos directos y para la gestión de archivos en sistemas operativos como Linux, también sirven como una forma de crear varias ubicaciones para las carpetas de usuario principal, por ejemplo, documentos, imágenes, descargas y mucho más.

Para crear enlaces simbólicos, utilizamos el comando **ln** y la opción **-s**. La sintaxis del comando es:

```
ln -s [fichero|directorio] [nombre simbólico]
```

- **`-s`:** el comando para enlaces simbólicos.
- **`[fichero]`:** nombre del archivo existente para el cual creamos el enlace.
- **`[directorio]`:** nombre del directorio existente para el cual creamos el enlace.
- **`[nombre simbólico]`:** nombre del enlace simbólico.

Veamos un ejemplo trabajando con enlaces simbólicos:

1. Comprobamos el contenido del **directorio actual**:

```
$ ls -l
total 12
drwxrwxr-x 2 santos santos 4096 feb 9 17:50 Documentos
-rw-rw-r-- 1 santos santos 8 feb 9 17:51 documento.txt
-rw-rw-r-- 1 santos santos 8 feb 9 17:50 lista.txt
```

2. Creamos un enlace simbólico al archivo **documento.txt**:
   ```
   $ ln -s ~/Descargas/documento.txt doc
   ```

3. Volvemos a comprobar el contenido del directorio:
   ```
   $ ls -l
   total 12
   drwxrwxr-x 2 santos santos 4096 feb 9 17:50 Documentos/
   lrwxrwxrwx 1 santos santos 33 feb 9 17:52 doc -> /home/santos/Descargas/documento.txt
   -rw-rw-r-- 1 santos santos 8 feb 9 17:51 imagen.jpg
   -rw-rw-r-- 1 santos santos 8 feb 9 17:50 lista.txt
   ```
   En la columna de permisos podemos ver que el primer carácter, una letra `l`, indica que doc es un enlace simbólico.

4. Si editamos doc o copiamos doc, realmente estaremos editando o copiando el archivo documento.txt.

## Enlaces duros

Un enlace duro solo está permitido para recursos en la misma partición y, si borramos el archivo, el enlace queda activo hasta que no haya más enlaces apuntando a este archivo (momento en el cual se borrará el archivo destino). Crear un enlace duro a un directorio solo está permitido al usuario root.

Para crear un enlace duro, veamos el siguiente ejemplo:

1. Mostramos el contenido del directorio:
   ```
   $ ls -l
   total 16
   drwxrwxr-x 2 santos santos 4096 feb 9 17:50 Documentos/
   -rw-rw-r-- 1 santos santos 8 feb 9 17:59 foto2
   -rw-rw-r-- 1 santos santos 8 feb 9 17:51 imagen.jpg
   -rw-rw-r-- 1 santos santos 8 feb 9 17:50 lista.txt
   ```

2. Creamos el enlace duro al archivo lista.txt:
   ```
   $ ls lista.txt enlace
   ```

3. Comprobamos que ha creado un archivo nuevo de nombre enlace pero, no se ve que apunte al destino lista.txt:
   ```
   $ ls -l
   total 20
   ```

```
drwxrwxr-x 2 santos santos 4096 feb 9 17:50 Documentos
-rw-rw-r-- 2 santos santos 8 feb 9 17:50 enlace
-rw-rw-r-- 1 santos santos 8 feb 9 17:59 foto2
-rw-rw-r-- 1 santos santos 8 feb 9 17:51 imagen.jpg
-rw-rw-r-- 2 santos santos 8 feb 9 17:50 lista.txt
```

4. Lo que sí podemos apreciar visualmente es que tanto el archivo enlace como el archivo lista.txt tienen dos enlaces que apuntan a cada uno:

   ```
   $ ls -l
   total 20
   drwxrwxr-x 2 santos santos 4096 feb 9 17:50 Documentos
   -rw-rw-r-- 2 santos santos 8 feb 9 17:50 enlace
   -rw-rw-r-- 1 santos santos 8 feb 9 17:59 foto2
   -rw-rw-r-- 1 santos santos 8 feb 9 17:51 imagen.jpg
   -rw-rw-r-- 2 santos santos 8 feb 9 17:50 lista.txt
   ```

   De hecho, esta columna de números indica precisamente eso, cuántos enlaces duros apuntan a cada uno de los ficheros o directorios.

5. Si ejecutamos el comando ls con la opción -i vemos una primera columna con el número de i-nodo de cada archivo o directorio:

   ```
   $ ls -li
   total 20
   539624 drwxrwxr-x 2 santos santos 4096 feb 9 17:50 Documentos
   543768 -rw-rw-r-- 2 santos santos 8 feb 9 17:50 enlace
   543780 -rw-rw-r-- 1 santos santos 8 feb 9 17:59 foto2
   543735 -rw-rw-r-- 1 santos santos 8 feb 9 17:51 imagen.jpg
   543768 -rw-rw-r-- 2 santos santos 8 feb 9 17:50 lista.txt
   ```

   Y podemos comprobar que los archivos y sus enlaces duros tienen el mismo i-inodo. Esta es la característica que los relaciona.

Un enlace duro no muestra a qué archivo o directorio apunta como en un enlace blando. Lo que sí podemos averiguar fácilmente es cuántos enlaces duro apuntan a un archivo o directorio consultándoos sus i-nodos.

## Eliminar un enlace

Puedes eliminar un enlace a un archivo o directorio mediante el comando `unlink` o el comando `rm`.

Así es como puedes hacerlo con el comando `unlink`:

```
unlink [nombre del enlace]
```

Y así con el comando `rm`:

```
rm [nombre del enlace]
```

## Diferencias entre enlaces duros y blandos

- Todas las acciones realizadas sobre un enlace simbólico se realizan realmente sobre el archivo original, salvo los comandos `ls` y `rm`.

- Los bits de permisos en un enlace simbólico no se usan (siempre aparecen como rwxrwxrwx). En su lugar, los permisos del enlace simbólico son determinados por los permisos del archivo "apuntado" por el enlace.

- Si el archivo original se borrara, el enlace simbólico apuntará a un archivo que ya no existe. En cambio, si se borra un enlace duro a un archivo, lo único que se modifica es el atributo número de enlace, pero el archivo original sigue existiendo. Sólo desaparece físicamente del disco el archivo original, cuando se borra el último enlace.

- Los enlaces duros tienen la desventaja que registran como propietario (en el i-nodo) al usuario creador del archivo. La creación de un enlace a ese archivo no modifica la propiedad.

  Si el propietario desea eliminar el archivo, no puede hacerlo hasta que el segundo usuario decida borrarlo y el contador de enlaces lo permita.

  Los enlaces simbólicos son otro archivo conteniendo un camino o pathname. Cuando el propietario lo elimina el archivo se destruye.

- Los enlaces simbólicos son de ayuda puesto que identifican al archivo al que apuntan; con enlaces duros no hay forma fácil de saber qué archivo está enlazado al mismo número de i- nodo. Se puede averiguar con el comando find y la opción inum.

Para saber más

Pulsa en el enlace siguiente para ampliar información sobre los enlaces en Linux:

Enlaces en Linux <http://manpages.ubuntu.com/manpages/focal/en/man1/ln.1posix.html>

## 1.8.- Compresión y descompresión en Linux

El comando **tar** es una herramienta de fácil manejo disponible en todas las versiones de UNIX/Linux que permite copiar ficheros individuales o directorios completos en un único fichero. Con el comando tar también se puede extraer los archivos resultantes.

Las ventajas del comando Tar de Linux:

Jean-loup Gailly and Mark Adler (Dominio público) <https://commons.wikimedia.org/wiki/File:Gzip-Logo.svg>

- Tar, cuando se trata de compresión, tiene una relación de compresión del 50%, lo que significa que comprime eficientemente.
- Reduce drásticamente el tamaño de las carpetas y archivos comprimidos.
- Tar en Linux no altera las características de los archivos y directorios. Los permisos y otras particularidades permanecen intactos mientras se comprime.
- El comando Tar está disponible en las versiones más comunes de Linux. También se encuentra disponible en el firmware de Android, así como en versiones compatibles de Linux más antiguas.
- Comprime y descomprime rápidamente.
- Es fácil de usar.

Escenarios donde utilizar tar es más útil:

- Si estás trabajando en sistemas basados en Linux y necesitas compresión de archivos.
- Para transferir una gran cantidad de archivos y carpetas de un servidor a otro.
- Para hacer una copia de seguridad de tu sitio web, de datos o de cualquier otra cosa.
- Para reducir el uso de espacio en tu sistema, ya que la compresión ocupará menos espacio
- Para cargar y descargar carpetas.

La sintaxis básica de este comando es:

```
tar <opciones> <archivoSalida> <archivo1> <archivo2> ... <archivoN>
```

El apartado **<opciones>** se detalla en la tabla siguiente:

| Opción | Significado |
|---|---|
| `-c` | Crea un nuevo archivo .tar |

| | |
|---|---|
| `-v` | Muestra información detallada del progreso de la ejecución del comando. |
| `-f` | Nombre del archivo que se crea. |
| `-z` | Compresión en gzip |
| `-j` | Compresión en bzip2 |
| `-r` | Actualizar o agregar un archivo o directorio en un archivo .tar existente |
| `-x` | Extraer contenido del archivo |
| `-C` | Extraer archivos en un directorio diferente |

## Para saber más

Para ampliar información, visita el enlace siguiente:

Comando tar <http://manpages.ubuntu.com/manpages/focal/en/man1/tar.1.html>

## 1.8.1.- Crear archivos con tar

### Crear un archivo .tar en Linux

Puedes crear archivos **.tar** tanto para un archivo como para directorios. Un ejemplo de este tipo de archivo es:

```
tar -cvf carpeta.tar /home/usuario/Documentos
```

Aquí `/home/usuario/Documentos` es el directorio que necesita ser empaquetado creando `carpeta.tar`.

### Crear un archivo .tar.gz en Linux

Si deseas una con compresión, también puedes usar **.tar.gz**. Un ejemplo de esto es:

```
tar -cvzf carpeta.tar.gz /home/usuario/Documentos
```

La opción adicional **`-z`** representa la compresión gzip. Alternativamente, puedes crear un archivo **`.tgz`** que sea similar a **`tar.gz`**. Veamos un ejemplo de esto último a continuación:

```
tar -cvzf OtraCarpeta.tgz /home/usuario/Documentos
```

### Crear un archivo .tar.bz2 en Linux

El archivo **`.bz2`** proporciona más compresión en comparación con gzip. Sin embargo, esta alternativa tomará más tiempo para comprimir y descomprimir. Para usarla, debes usar la opción **`-j`**. Un ejemplo de cómo se vería la operación es el siguiente:

```
tar -cvjf documentos.tar.bz2 /home/usuario/Documentos
```

Dicha operación es similar a **`.tar.tbz`** o **`.tar.tb2`**. Veamos un ejemplo a continuación:

```
tar -cvjf documentos.tar.tbz /home/usuario/Documentos
```

```
tar -cvjf documentos.tar.tb2 /home/usuario/Documentos
```

### Crear un archivo .tar.gz excluyendo archivos

Utilizamos la opción **`--exclude`** de **`tar`** para excluir una extensión específica o una lista de extensiones de archivos. La sintaxis es similar a la siguiente:

```
tar -czvf /ruta/archivo.tgz --exclude=\*.
{jpg,gif,png,wmv,flv,tar.gz,zip} /ruta/archivo
```

Veamos un ejemplo en el cual se crea un archivo comprimido con los archivos del directorio `/home/usuario/imagenes` excluyendo los ficheros de imagen con extensión **`jpg`** o **`png`** :

```
tar -czvf fotos.tgz --exclude=\*.{jpg,png} /home/usuario/imagenes
```

## 1.8.2.- Extraer archivos con tar

### Cómo descomprimir archivos .tar en Linux

El comando Tar de Linux también se puede utilizar para extraer un archivo. El siguiente comando extraerá los archivos en el directorio actual:

```
$ tar -xvf archivo.tar
```

Si deseas extraer tus archivos a un directorio diferente, puedes usar la opción **-C**:

```
$ tar -xvf archivo.tar -C /home/usuario/mi_carpeta
```

Puedes usar un comando similar para descomprimir archivos **.tar.gz**, tal como se muestra a continuación:

```
$ tar -xvf archivo.tar.gz
```

```
$ tar -xvf archivo.tar.gz -C /home/usuario/mi_carpeta
```

Los archivos **.tar.bz2** o **.tar.tbz** o **.tar.tb2** pueden descomprimirse de manera similar. Para esto deberás teclear el siguiente comando en la línea de comando:

```
$ tar -xvf archivo.tar.bz2
```

Al descomprimir no es necesario incluir la opción con la que se comprimió tal y como se ha mostrado en los ejemplos. Se puede indicar **-z** o **-j** pero, no es necesario. Eso sí, si ponemos la opción de compresión equivocada, devolverá un error y no extraerá el contenido del archivo comprimido.

### Cómo extraer un archivo concreto de un archivo .tar

El sintaxis comando para realizar esta operación es:

```
tar --extract -vv --occurrence --file=./archivo.tar archivo_a_extraer
```

o

```
tar -xvf Archivo.tar archivo_a_extraer
```

Si queremos extraer el archivo `doc.txt` del tar comprimido `documentos.tar.gz`, ejecutaremos el comando:

```
$ tar -xvf documentos.tar.gz doc.txt
```

Sería equivalente ejecutar:

```
$ tar --extract -vv --occurrence --file=documentos.tar.gz doc.txt
```

## Extraer un grupo de archivos usando comodines

Si queremos extraer los archivos de extensión txt de un archivo tar, haremos lo siguiente:

```
$ tar -xvf archivo.tar --wildcards '*.txt'
```

## 1.8.3.- Otras operaciones con tar

### Agregar un archivo a un archivo tar existente

Para realizar esta operación, podemos utilizar dos opciones. Veamos con un ejemplo las dos maneras de conseguir que al archivo **lista.tar** se le añada un nuevo archivo:

1. `tar -rf lista.tar nuevo_documento.txt`
2. `tar --append --file=lista.tar nuevo_documento.txt`

No se puede añadir un archivo a un archivo tar comprimido.

### Concatenar archivos comprimidos con tar

Para añadir un archivo tar a otro, podemos hacerlo de varias maneras:

1. `tar -Af uno.tar dos.tar`
   En este ejemplo se añade el archivo dos.tar a uno.tar

2. `tar --concatenate --file=uno.tar dos.tar`
   Igualmente se consigue, en este ejemplo, lo mismo que en el anterior.

Los archivos comprimidos no puede se concatenados.

### Listar o verificar el contenido de un archivo con tar

Para mostrar el contenido de un fichero tar, bien sea comprimido o no comprimido es:

```
$ tar -tvf fichero.tar
```

### Mantener los enlaces simbólicos con tar

Para preservar los enlaces simbólicos en un archivo tar y recuperarlos después correctamente, debemos utilizar la opción `-h`:

```
$ tar -cvhf archivo.tar .
```

# 1.9.- Redireccionamiento y tuberías.


Nuria Barroso (Elaboración propia)

**Redirección de la salida por defecto.**
Los resultados de un comando en pantalla pueden redireccionarse a un fichero de dos formas distintas:

- Borrando el fichero si ya existía, en caso contrario, lo crea nuevo: comando > fichero
- Añadiendo la información al fichero si ya existía, en caso contrario, lo crea nuevo: comando >> fichero

El redireccionamiento en Linux, funciona de la misma forma que el redireccionamiento en Windows.

**Tuberías o pipelines**
Los resultados de un comando en pantalla pueden redireccionarse a la entrada de otro comando separándolos con el carácter de tubería o pipe: |. Sólo el último comando de una tubería muestra sus resultados en pantalla (que también podrían redireccionarse a un fichero) : comando1 | comando2 | ... | comandoN

Situaciones en las que la salida de un comando se utiliza como la entrada de otro son muy frecuentes en la práctica. Por ejemplo, supón que deseas obtener un listado ordenado alfabéticamente con información acerca de todos los usuarios conectados actualmente en la máquina. El comando **who** permite obtener la información acerca de los usuarios conectados (si estamos trabajando con una distribución de Linux con entorno gráfico, el comando who nos devuelve los usuarios que han iniciado sesión a través del entorno gráfico) y el comando **sort** puede ordenarla alfabéticamente. Una posible solución es:

```
who > tmp
```

```
sort < tmp
```

```
usuario1 :0 Sep 19 19:53
usuario2 :1 Sep 19 15:45
```

```
rm tmp
```

En este caso el fichero tmp guarda de forma temporal la información proporcionada por who antes de usar el comando sort.

Una forma simplificada de hacer esto mismo es usando tuberías. Tal y como lo ves a continuación: **who | sort**

```
usuario1 :0 Sep 19 19:53
usuario2 :1 Sep 19 15:45
```

El efecto es similar a la creación de un archivo temporal, con la única diferencia de que el uso de las tuberías es mucho más eficiente.

## Autoevaluación

**Cuando usas el comando > fichero:**

- ○ Crea un fichero nuevo o sobrescribe lo que tenía.
- ○ Estas añadiendo el resultado del comando a un fichero ya creado.
- ○ La salida de un comando es la entrada de otro.

Muy bien. Has captado la idea.

Incorrecta, para añadir información hay que usar >>.

No es la respuesta correcta, para eso usas tuberías.

**Solución**

1. **Opción correcta**
2. **Incorrecto**
3. **Incorrecto**

## 1.10.-Localización en ficheros

# 1.10.1.- grep - Buscar en el contenido de los ficheros

El comando que permite realizar búsquedas de palabras y patrones dentro de un fichero o grupo de ficheros es el comando grep. Por defecto, grep busca las líneas que coinciden y las imprime en la salida estándar.

La sintaxis general con la que vamos a trabajar este comando se resume de la siguiente manera:

**$ grep [opciones] patrón [fichero]**
**$ grep [opciones] `-f` fichero-que-contiene-los-patrones-de-búsqueda fichero**

Las opciones básicas que vamos trabajar:

- **`-c`**: Cuenta el número de coincidencias.
- **`-E`**: Expresión regular extendida.
- **`-f`**: Obtiene el patrón o los patrones de búsqueda de un fichero (Uno por cada línea).
- **`-i`**: Insensible a mayúsculas y minúsculas.
- **`-l`**: Imprime el nombre de cada fichero de entrada donde se encuentren coincidencias.
- **`-n`**: Imprime el número de línea en donde se encuentren coincidencias.
- **`-o`**: Imprime sólo la parte que coincide.
- **`-v`**: Invierte el sentido de la búsqueda, es decir, se muestran las que no coinciden.

Vamos a ver con algunos ejemplo el funcionamiento del comando:

Supongamos que tenemos el archivo **listado_de_frutas.txt** con el contenido siguiente:

La Manzana es una fruta pomácea comestible obtenida del
manzano doméstico
La ManZana es una de las frutas más cultivadas del mundo
La manzana puede comerse fresca pelada o con piel
MANZANA Cameo

Mango
Mamey
Güayaba
Pera

**1. Sensible a mayúsculas y minúsculas**

```
$ grep manzana frutas.txt
```

```
La manzana puede comerse fresca pelada o con piel
```

**2. Insensible a mayúsculas y minúsculas**

```
$ grep -i manzana frutas.txt
La Manzana es una fruta pomácea comestible obtenida del
manzano doméstico
La ManZana es una de las frutas más cultivadas del mundo
La manzana puede comerse fresca pelada o con piel
MANZANA Cameo
```

**3. Contar el número de coincidencias (Sensible a mayúsculas y minúsculas)**

```
$ grep -c manzana frutas.txt
1
```

**4. Contar el número de coincidencias (Insensible a mayúsculas y minúsculas)**

```
$ grep -ci manzana frutas.txt
4
```

**5. Prefijar número de línea**

```
$ grep -n manzana frutas.txt
4:La manzana puede comerse fresca pelada o con piel
```

**6. Sólo la parte que coincide (Sensible a mayúsculas y minúsculas)**

```
$ grep -o manzana frutas.txt
manzana
```

**7. Sólo la parte que coincide (Insensible a mayúsculas y minúsculas)**

```
$ grep -io manzana frutas.txt
Manzana
ManZana
manzana
MANZANA
```

**8. Encontrar las líneas que contienen manzana o Mamey**

```
$ grep -E 'manzana|Mamey' frutas.txt
La manzana puede comerse fresca pelada o con piel
Mamey
```

**9. Invertir la búsqueda (Sensible a mayúsculas y minúsculas)**

```
$ grep -v manzana frutas.txt
La Manzana es una fruta pomácea comestible obtenida del
manzano doméstico
La ManZana es una de las frutas más cultivadas del mundo
MANZANA Cameo

Mango
Mamey
Güayaba
Pera
```

**10. Invertir la búsqueda (Insensible a mayúsculas y minúsculas)**

```
$ grep -iv manzana frutas.txt
manzano doméstico

Mango
Mamey
Güayaba
Pera
```

**11. Búquedas recursiva**

```
$ grep -lR estudiante proyectos/
```

El ejemplo anterior realiza una búsqueda recursiva dentro del DIR proyectos y encuentra/lista todos los todos los ficheros que contienen la palabra estudiante.

## 1.10.2.- find - Buscar ficheros

Para buscar fichero o directorios en nuestro sistema de archivos, tenemos un comando muy versátil llamado **find**, que nos permite buscarlos y localizarlos muy fácilmente. Su sintaxis general es:

**find [ruta] [expresión_de_búsqueda] [acción]**

Para utilizar correctamente el comando find, primero debemos conocer sus diferentes opciones:

- **-name**
  Permite buscar por nombre de fichero y, si se utilizan caracteres comodines, se escriben entre comillas.

  Mostrar los ficheros que comienzan por **fic** en el directorio actual:

```
$ find . -name "fic*"
./fic1
./fic2
./fic3
./fic4
```

- **-type**
  Permite buscar por tipo de fichero y aquí puedes comprobar la tabla de valores que se puede utilizar:

| Código | Tipo de fichero |
|---|---|
| b | Fichero especial en modo bloque |
| c | Fichero especial en modo carácter |
| d | Directorio |
| f | Fichero ordinario |
| l | Vínculo simbólico |
| p | Tubería con nombre (pipe) |
| s | Socket (Conexión de red) |

  Visualizar todos los directorios que comienzan por **D**:

```
$ find . -type d -name "D*"
```

```
./Descargas
./Documentos
```

- **-user** y **-group**
  Estas opciones del comando permiten buscar archivos basándose en el usuario propietario (-user) o el grupo propietario (-group). También podemos indicar sus UID o GID respectivamente para realizar la búsqueda en lugar del nombre.

  Visualizar los archivos que pertenezcan al usuario **seb** y al grupo **users**.
  ```
  $ find . -tipo f -user seb -group users
  ./fic1
  ./fic3
  ```

- **-size**
  En este caso, podremos buscar ficheros en función de su tamaño teniendo en cuenta que será muy importante indicar la unidad de medida en función de la tabla siguiente:

| Carácter | Significado |
|---|---|
| B | Es la opción por defecto si no se indica nada y equivale a un bloque de 512 bytes. |
| C | Equivale a un carácter ASCII y, por tanto, 1 byte. |
| W | Equivale a una palabra, es decir, de 2 bytes. |
| K | Equivale a 1 KB (1024 bytes). |

  El valor de -size indica que debe tener ese tamaño exactamente pero, si indicamos además, el símbolo:
  **+**, indica que el tamaño será más de...
  **-**, indica que el tamaño será menos de...

  **Ejemplos de valores posibles:**
  -size 10: busca los ficheros de un tamaño de 10 bloques (512 bytes por bloque, es decir, 5120 bytes).
  -size 100c: busca los ficheros de un tamaño de 100 caracteres (bytes).
  -size 10k: busca los ficheros de un tamaño de 10 KB (10*1024 bytes = 10240 bytes).
  -size +2000k: los ficheros de más de 2000 KB.
  -size -300k: los ficheros de menos de 300 KB.)

  Mostrar los archivos que tenga un tamaño de más de 100k:
  ```
  $ find -size +100k
  ./zypper.log-20080227.bz2
  ./lastlog
  ```

```
./zypper.log-20080302.bz2
```

- **-atime, -mtime y -ctime**
  **atime**: permite buscar por la fecha del último acceso al archivo (**access time**).
  **mtime**: busca en la fecha de la última modificación (**modification time**).
  **ctime**: busca en la fecha de modificación (**change time**).

  Estos tres criterios sólo trabajan con días (**periodos de 24 horas**).

  Los signos + o - significan «más de...» y «menos de...». Veamos algunos ejemplos para entenderlo mejor:
  -mtime 1: correspondería a ficheros modificados ayer (entre 24 y 48 horas).
  -mtime -3: correspondería a ficheros modificados hace menos de tres días (72 horas).
  -atime +4: correspondería a ficheros modificados hace más de cuatro días (más de 96 horas).

  Visualizar aquellos ficheros modificados hace menos de un día (las últimas 24 horas):

```
$ find . -mtime -1
./vmware./vmware/vmware-serverd-0.log
./vmware/vmware-serverd.log
./mail.info
./Xorg.0.log
```

- **-perm**
  Permite seleccionar archivos en función de sus permisos. Estos permisos se expresan en octal de forma completa. El carácter - colocado antes del valor octal significa que los ficheros buscados deben tener al menos los permisos establecidos.

  Mostrar los directorios sobre los cuales todo el mundo (user, group, others) tiene permiso de ejecución (permiso x, o sea 1):

```
$ find -type d -perm -111
```

- **-links y -inum**
  Permite buscar cuántos enlaces duros (hard links) existen sobre un archivo. El número de **vínculos**, lo indicamos con el símbolo:
  **+**, aquellos archivos de más de **n** vínculos.
  **-**, aquellos archivos de menos de **n** vínculos.

  Mostrar archivos que tengan más de 2 enlaces duros:

```
$ find . -type f -links +2 -print
./fic2
./hardlink3_fic2
```

```
./hardlink_fic2
./hardlink2_fic2
```

`inum` permite una búsqueda por número de inodo. Es útil en el caso de una búsqueda de todos los vínculos que llevan un mismo **número de inodo**. Observa el siguiente ejemplo:

```
$ ls -i
491891 acpid 491793 mail.info 491860 Xorg.0.log
491791 boot.log 491794 mail.warn 490686 Xorg.0.log.old
491729 boot.msg 492046 mcelog 492060 Xorg.1.log

$ find . -inum 491791 -print
./boot.log
```

- **-and, -or, -not (!)**(Concatenar condiciones)
  Es posible combinar las opciones de criterio de selección mostrada en la tabla siguiente:

| Condición | Significado |
|---|---|
| -a, -and | AND, Y lógico, por defecto |
| -o, -or | OR, O lógico |
| -not, ! | Negación del criterio |

Mostrar todos los ficheros que no contienen **fic** en su nombre y todos los ficheros que no son ni normales ni directorios:

```
$ find . ! -name "*fic*" -print
.
./dir1
./lista
./mypass
./users
./lista2
./ls.txt
./pepito.tar.gz
./nohup.out
./lista_ls
./dir2
./seb1
./seb2
$ find . ! \( -type f -o -type d \) -ls
```

```
409 0 lrwxrwxrwx 1 oracle system 4 Ago 14 15:21
./vínculo_fic1 -> fic1
634 0 lrwxrwxrwx 1 oracle system 4 Ago 14 15:21
./vínculo_fic2 -> fic2
```

## 1.10.3.- Ejecutar después de buscar

Todo lo que acabamos de ver sobre el comando find puede ser enlazado para que, en cada línea de resultado del comando, ejecute uno de los siguientes comandos:

- **-ls**
  Este comando consigue que para cada fichero encontrado se ejecute el comando ls como si lo escribiéramos nosotros mismos, teniendo en cuenta que se ejecuta con las opciones `-d, -i, -l` y `-s` (en bloque de 1KB).

  Listar los archivos mayores de 500000k en formato extendido:
  ```
  $ find -size +500000k -ls
  2342935 584388 -rw-r--r-- 1 seb users 597817344 feb 24
  11:52 ./eeexubuntu-7.10.3-desktop-i386.iso
  ```

- **-exec**
  En este caso, para cada coincidencia encontrada por find se pasa a este valor para formar parte de la ejecución del comando que se indique a continuación.

  Para escribirlo correctamente, hay que seguir las siguientes consideraciones:
  - exec debe ser obligatoriamente la última opción de find.
  - El comando ejecutado por exec debe terminar con **\;**

  El criterio **exec** va a ejecutar el comando colocado justo después con cada coincidencia encontrada y para indicar esta coincidencia ya que escribir los caracteres **{}**.

  Eliminar todos los archivos que terminan por «.txt»:
  ```
  $ find . -type f -name "*.txt" -exec rm -f {} \;
  ```

- **-ok**
  Se comporta exactamente igual que -exec con la particularidad añadida que, para cada coincidencia pregunta al usuario confirmación de la ejecución del comando.

  Borrar los archivos que terminen en «.jpg»:
  ```
  $ find . -type f -name "*.jpg" -ok rm -f {} \;
  < rm ... ./foto.jpg> (yes)? n
  < rm ... ./paisaje.jpg > (yes)? y
  ```

# 1.10.4.- Buscar ejecutables

**whereis**

Este comando muestra los **binarios** que queramos buscar en el sistema pero, no solo esto, sino que además realiza la búsqueda en el manual o en el código fuente. Algunas de las opciones más interesantes del comando se resumen en la tabla siguiente:

| Opción | Explicación |
|---|---|
| `-b` | Buscar solo binarios. |
| `-m` | Buscar solo manuales. |
| `-s` | Buscar solo por fuentes. |
| `-l` | Lista los caminos de búsqueda del comando. |

Ejemplos whereis:

Buscar firefox y muestra todo lo que encuentres:

```
$ whereis firefox
firefox: /usr/bin/firefox /usr/lib/firefox /etc/firefox /usr/share
/man/man1/firefox.1.gz
```

Buscar solo el binario o programa cp:

```
$ whereis -b cp
cp: /usr/bin/cp
```

Buscar solo en el manual cp:

```
$ whereis -m cp
cp: /usr/share/man/man1/cp.1.gz
```

---

**which**

Busca y muestra por pantalla la primera ubicación que encuentra en el **PATH** el comando o programa que indiquemos.

Observar los ejemplos siguientes:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr
/games:/usr/local/games:/snap/bin
$ which date
/usr/bin/date
$ which firefox
```

```
/usr/bin/firefox
```

---

**locate**

El comando **locate** es una alternativa útil, ya que es más rápido que **find** para realizar búsquedas. Eso se debe a que sólo escanea tu base de datos de Linux en lugar de todo el sistema. Además, la sintaxis es más fácil de escribir.

*--> Cómo instalar el paquete locate*

Por defecto, Linux no viene con el comando **locate** preinstalado. Para obtener el paquete, ejecuta los siguientes comandos en tu terminal:

```
$ sudo apt-get update
$ sudo apt-get install mlocate
```

*--> La sintaxis básica*

Ahora puedes usar el comando para buscar archivos usando esta sintaxis:

**locate [my-file]**

El comando **locate** estándar a veces puede mostrar archivos que han sido eliminados, si la base de datos no fue actualizada. La mejor solución es *actualizar manualmente la base de datos* ejecutando lo siguiente:

```
sudo updatedb
```

*--> Cómo usar el comando locate de Linux*

Buscar por el nombre exacto del archivo

La sintaxis básica sólo te permite buscar archivos que contengan el término de búsqueda. Si quieres obtener el archivo con el nombre exacto, puedes utilizar la opción **-r** y añadir el símbolo de dólar (**$**) al final del término de búsqueda, por ejemplo:

```
locate -r my-file$
```

Contar el número de archivos

Para saber cuántos archivos aparecen en el resultado de la búsqueda, introduce **-c** después del comando locate.

```
locate -c my-file
```

En lugar de listar todos los archivos, te dará el número total de ellos.

Ignorar distinción entre mayúsculas y minúsculas

Usa **-i** en tu comando **locate** para ignorar la distinción entre mayúsculas y minúsculas. Por ejemplo:

```
locate -i my-file
```

Se mostrarán todos los archivos con este nombre, independientemente de las mayúsculas o minúsculas que se encuentren.

Mostrar archivos existentes

Como hemos mencionado, el comando **locate** de Linux puede incluso mostrarte un archivo eliminado si no has actualizado la base de datos. Afortunadamente, puedes resolver esto usando la opción **-e**, así:

```
locate -e my-file
```

Al hacer esto, sólo obtendrás los archivos que existen en el momento de ejecutar el comando **locate**.

Desactiva los errores durante la búsqueda

La opción **-q** evitará que cualquier error aparezca cuando se procese la búsqueda. Para hacer esto, simplemente introduce:

```
locate -q my-file
```

Limitar el número de resultados de búsqueda

Si quieres limitar el número de resultados de la búsqueda, **-n <number>** funcionará. Sin embargo, recuerda que debes poner la opción al final de la línea de comandos. Echa un vistazo a este ejemplo:

```
locate my-file -n 10
```

El comando sólo mostrará los primeros 10 archivos que encuentre, incluso aunque haya más.

## 2.- Comandos de administración básica.

### Caso práctico

Nuria Barroso (Elaboración propia)

Desde el servidor de Linux que han instalado en la empresa, Iván va a tener que hacer labores administrativas, por ejemplo, va a tener que dar de alta a un grupo de usuarios y, además, tiene que hacer que pertenezcan al mismo grupo, para luego facilitar el poner o quitar permisos a todo el grupo. Como el servidor sólo tiene entorno de texto, esta tarea la tiene que realizar mediante comandos.

## 2.1.- Comandos de utilidades.

Hay muchos más comandos, ahora vas a ver una serie de comandos que se usan a menudo en el testeo del equipo:

- **echo** mensaje: muestra el mensaje en la pantalla. Es similar al comando echo del sistema operativo Windows.
- **df** (acrónimo de display free): Muestra la ocupación del disco.
- **du** (acrónimo de disk usage): muestra la ocupación de disco de cada uno de los ficheros.
- **date**: muestra la fecha del sistema.
- **cal** (acrónimo de calendary): muestra el calendario.
- **grep**: busca las líneas de un fichero donde aparezca la cadena especificada. Nos permite localizar un texto dentro de fichero. La sintaxis es: `grep "texto a buscar" fichero.`

  Si le pasamos el **parámetro** -i al hacer la búsqueda no distingue entre mayúsculas y minúsculas.

- **lpr**: manda a imprimir.
- **shutdown:** apaga el sistema.

  Por ejemplo, si escribes en la **terminal**: `shutdown`

  El sistema se apagará después de un minuto.

  Si escribes la orden: `shutdown -h now`, apaga el ordenador en este momento.

- **reboot**: reinicia el sistema
- **uname**: muestra información del sistema.
- **logname**: muestra el usuario que se ha conectado.
- **apt-get**: manipula **paquetes**. Se denomina paquetes a los ficheros de instalación de las aplicaciones en el sistema operativo Linux.
- **at:** programa tareas.

### Para saber más

En este enlace, puedes ver una página muy completa con los comandos de la shell. Nos pueden servir cuando tengamos que acceder al equipo en modo seguro.

Comandos del shell bash. <http://www.osmosislatina.com/linux/comandos.htm>

## Autoevaluación

**Relaciona los comandos con lo que muestran, escribiendo el número asociado a lo que hace el comando en el hueco correspondiente.**

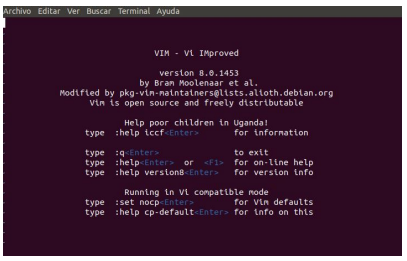
Comandos de utilidades

| Comando | Relación | Resultado |
|---|---|---|
| echo | | 1.- Imprime un fichero. |
| du | | 2.- Muestra un mensaje por pantalla. |
| df | | 3.- Muestra la ocupación de los ficheros en disco. |
| lpr | | 4.- Muestra el espacio disponible. |

Son algunos de los comandos más usados.

## 2.2.- Editores de texto.

Para ver el contenido de los ficheros puedes usar el comando **cat**, pero si, además, vas a querer modificarlos y cambiar algún valor en algún fichero de configuración, debes usar un editor de textos.




Agustín Nieto Espino

(Elaboración propia)

Algunos de los editores de texto más usados en Linux son:

- **gedit:** es el editor por defecto en modo gráfico en distribuciones GNOME.
- **nano**: es el editor por defecto en modo texto.
- **pico:** editor básico.
- **vi** (Acrónimo de visual): editor de texto característico de Linux, tiene dos modos de trabajo:
    - Modo comando o dos puntos.
    - Modo introducción de texto.

Cuando abres el **editor vi** desde la **terminal**, escribiendo vi, estás en modo comando, para poder introducir texto tienes que pasar a **modo de insertar texto**, para ello pulsa **:i (dos puntos i).** Ahora escribe el texto que quieras que contenga el fichero y para salir pulsas **control + c** y tecleas **:wq** y el nombre del archivo.

Para insertar `:i` .Cuando estás en **modo comando o dos puntos** tienes una serie de **parámetros** que puedes usar, como son:

- Para borrar, pulsaremos las teclas `:x`
- Para buscar `:fg`
- Para salir sin grabar `:q!`
- Para salir y guardar los cambios `:wq`

Para crear un fichero de texto, también puedes usar el comando cat, para ello, escribes en la consola **`cat > nombre_del_fichero`** y después escribes el texto que quieras que contenga el fichero y para salir pulsas **CTRL** más la tecla **d.**

### Debes conocer

Los **scripts** son ficheros que contienen órdenes. Son muy útiles para automatizar tareas en Linux. Para poder crearlos o modificarlos, necesitarás saber cómo crear ficheros desde el terminal. En el siguiente vídeo, se muestran los editores de textos que más se utilizan:

**Editores de texto**


Agustín Nieto Espino. *Descripción textual para el vídeo "Editores de texto".* (Elaboración propia)

## Autoevaluación

**El editor vi:**

- Es un editor gráfico.
- Es equivalente a gedit.
- Trabaja en modo dos puntos y modo insertar texto.

Incorrecta, es un editor en modo texto.

Incorrecta.

Correcto, es un editor de texto básico.

**Solución**

1. **Incorrecto**
2. **Incorrecto**
3. **Opción correcta**

## Para saber más

En este enlace, puedes ver una página sobre los comandos y manejo del editor vi: **Editor vi. <https://www.unirioja.es/cu/enriquez/docencia/Quimica/vi.pdf>**