HACKZTLAN LAB

LINUX DESDE CERO

```
ana@linux:~$
```

EL LIBRO DEL CURSO

Capítulo 0:
Introducción a Linux



Hackztlan Lab



LINUX DESDE CERO

El libro del curso · Capítulo 0: Introducción a Linux

Texto y dibujos: *Hackztlan*

Una edición de Hackztlan Lab.

Este libro acompaña a las clases del curso *Linux desde cero*. Las clases son gratis y lo van a seguir siendo.



El curso completo, gratis, en YouTube

© Hackztlan Lab. Este libro se distribuye bajo la licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 (CC BY-NC-SA 4.0). Puedes copiarlo, compartirlo y adaptarlo sin fines comerciales, dando crédito a Hackztlan Lab y conservando esta misma licencia. Para uso comercial, escríbenos por la página de Facebook de Hackztlan Lab.

Las salidas de terminal son reales. Los dibujos son propios. Compuesto con Georgia, Segoe UI y Consolas.

PRÓLOGO

El futuro llegó a mi casa en una caja color hueso.

Afuera eran los tiempos de los tenis de colores chillones, los copetes altos y tiesos de laca, las discotecques de los sábados. Adentro había una Compaq de torre, con su lector de discos, su unidad de disquete y un monitor de cinescopio que ocupaba medio escritorio y calentaba el cuarto. No teníamos teléfono, así que tampoco internet; yo ni siquiera sabía que eso existía. El mundo entero cabía en cuatro discos: Encarta, Age of Empires, StarCraft y Black & White. Con uno aprendía capitales; con los otros tres, a perder guerras. Ninguno era original, claro. Eran discos del tianguis, o quemados por algún amigo, con el nombre escrito a plumón: los de a de veras no los podíamos pagar.

Para mí aquello no era una computadora. Era una nave espacial, y yo no tenía idea de cómo volaba.

Lo que sí tenía era tiempo. Tardes enteras, vacaciones que no se acababan nunca. Y cuando ya había ganado todo lo que se podía ganar, quedaba lo otro: abrir cada menú, cada carpeta, cada ventana del Panel de control, nada más por ver qué había. Mover algo. Ver qué se rompía. El aburrimiento fue mi primer maestro, y era bastante bueno.

La escuela llegó después y enseñaba menos. En la secundaria las computadoras vivían en un salón aparte, cada una dormida bajo su funda de plástico, y la clase consistía en el procedimiento correcto para encenderlas y apagarlas. Nos enseñaron a no despertarlas de más. A entenderlas, nunca.

Cuando por fin hubo internet, fue una revelación con horario: las salas del Latin Chat, mi primer correo, páginas que tardaban en bajar lo que hoy tarda una película. Y un día llegó a la casa una máquina nueva, una eMachines negra, que en realidad era solo la caja: el monitor venía de un lado, el teclado de otro y el mouse de otro más. Era de todos. La usaba yo.

Fueron los años de LimeWire y de Ares, de bajar una canción y descubrir que era otra, o que no era una canción. Lo que no se encontraba ahí se bajaba de Megaupload, en diez partes, esperando el contador entre una y otra, y rezando para que la parte siete no estuviera caída.

Todavía me acuerdo de nuestro primer quemador de discos. De un día para otro yo era la disquera de la cuadra: les armaba a mis amigos sus discos de canciones, a pedido, con la lista escrita a mano en la carátula.

Instalaba lo que encontraba, de donde fuera, sin preguntar. La máquina se moría, yo la reinstalaba. Se volvía a morir. A la quinta vez uno empieza a hacerse preguntas, y así, sin querer, aprendí qué era un driver y por qué ese teclado ajeno no quería hablar con esa caja.

Intenté hacerlo en serio en una escuela técnica. Duré poco: enseñaban informática para oficinista, y yo ya sabía dónde estaba el botón de negritas. Me fui, y empecé a instalarles

Windows y Office a los vecinos. Lo que había aprendido rompiendo mi computadora se volvió trabajo arreglando las de otros.

A Linux llegué por la puerta de atrás y con malas intenciones: quería la clave del wifi del vecino. No lo recomiendo. Pero buscando cómo, di con un sistema de nombre BackTrack, lo arranqué, y ahí se acabó la travesura.

Pantalla negra. Un cursor parpadeando. Nada más.

Yo llevaba años creyendo que sabía de computadoras. Aquel cursor me informó, con mucha paciencia, que no.

No había videos. Había foros: páginas de fondo oscuro, con firmas animadas, donde alguien había tenido tu mismo problema tres años antes y la única respuesta era «ya lo resolví, gracias». Había Stack Overflow, que recién empezaba y contestaba en inglés y de mal humor. Y había libros técnicos que no entendía, porque había dejado la escuela y me faltaba justo lo que esos libros daban por sabido. Estuve mucho tiempo frente a esa pared.

La rompí regresando a estudiar, y dibujando. En el bachillerato me encontré con C, con C++ y con Ubuntu, y con todos los tropiezos de rigor: el wifi que no funcionaba, las peleas con vi, no saber dónde vivía cada cosa. Entonces empecé a hacer mis propios mapas de lo que me explicaban. Cajas, flechas, garabatos. Y fue hasta que lo vi dibujado que lo entendí.

Lo demás es más corto de contar. Trabajé en una fábrica, donde conocí los servidores, las bases de datos, el cableado y mi primer firewall. Terminé la ingeniería; el trámite del título sigue pendiente, lo cual, si has leído hasta aquí, no debería

sorprenderte. Me certifiqué en seguridad y a eso me dedico. Lo digo solo para que sepas que lo que hay en estas páginas tiene respaldo. Lo importante es lo otro.

Lo otro es que este es el libro que me hubiera servido frente a aquella pantalla negra. No da nada por sabido. Cada cosa viene con su dibujo, porque así fue como yo la entendí. Y varios de los problemas que vas a resolver son los que me tocó sufrir a mí, empezando por el wifi.

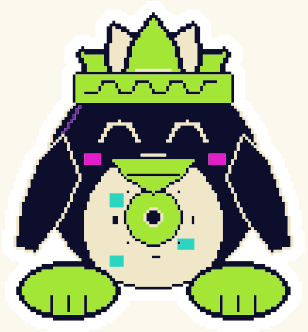
Y por eso las clases son gratis y lo van a seguir siendo.
Yo aprendí con discos del tianguis; tampoco hubiera podido pagarlas.

Si no terminaste la escuela, si crees que ya se te pasó el tiempo, si alguna vez abriste una terminal y la cerraste a los diez segundos: yo también. Aquella nave espacial sí se podía volar. Solo hacía falta que alguien abriera el tablero y explicara para qué era cada botón.

Abre la terminal y hazlo conmigo.

Hackztlan

0



CAPÍTULO 0

Introducción a Linux

> *Acompaña a la Clase #0 del curso Linux desde cero.*

`youtu.be/UuTXOO-yXRk`

Escanéalo para ver la clase en YouTube

EN ESTE CAPÍTULO VAS A APRENDER A:

- ✓ Explicar qué es un sistema operativo y cuáles son los cuatro trabajos del kernel.
- ✓ Contar qué pasa desde que aprietas el botón de encendido hasta que aparece tu sesión.
- ✓ Distinguir terminal, shell y comando, y averiguar con `type` qué clase de comando es cada uno.
- ✓ Moverte por el árbol de carpetas con rutas completas y cortas sin perderte.
- ✓ Leer un renglón de `ls -l` y un mensaje de error, y saber quién te está hablando.
- ✓ Pedir ayuda sin salir de la terminal con `--help`, `help` y `man`.

NECESITAS:

Nada previo. Si ya tienes una terminal de Linux a la mano, tenla junto; si no, lee de corrido y regresa a los ejercicios cuando la tengas. La Clase 0.5 del curso explica cómo instalar Linux sin borrar Windows.

TIEMPO ESTIMADO

Unas dos horas de lectura con la terminal abierta, más media hora para *Tu turno*. No tiene que ser de un jalón.



Ruta rápida. ¿Ya usas `cd`, `ls`, `cat` y `sudo`? Lee las secciones 2 y 3 (lo que hace el kernel), 4 (el arranque y `systemctl`), 10 (terminal, shell y comando), 14 (todo es un archivo) y 16 (pedir ayuda), y ve a *Tu turno*. Si te salen los ejercicios 3, 4 y 11 y los cuatro retos, sáltate el resto.

Antes de empezar

Este libro se lee con la terminal abierta. El video de la Clase #0 te enseña el recorrido; aquí lo vamos a hacer con calma, a ver qué hay debajo de cada cosa y a comprobarlo con tus propios ojos. Nada de lo que hay aquí se echa a perder si lo lees antes de tener Linux: los ejercicios te esperan.

Tres acuerdos entre tú y yo:

1. **Teclea, no copies.** Los dedos aprenden cosas que los ojos no.
2. **Equivócate a gusto.** Yo aprendí rompiendo una computadora que ni era mía del todo. Aquí no vas a romper nada, pero el método es el mismo.
3. **Lee lo que la terminal te contesta.** Casi siempre te está diciendo exactamente qué pasó. Este libro es, sobre todo, un curso de lectura.

Si ya tienes la terminal abierta, no esperes a la sección 12: teclea estos tres comandos, uno por renglón, cada uno seguido de `Enter`. Así vas a ver los ejemplos en todo el libro:

```
ana@linux:~$ pwd
/home/ana
ana@linux:~$ whoami
ana
ana@linux:~$ ls
Descargas
Documentos
Imágenes
```

Ana es nuestra usuaria de ejemplo y `linux` es el nombre de su máquina. Lo que va después del signo `$` es lo que ella teclea; los renglones de abajo son lo que la máquina responde. `pwd` le dice en qué carpeta está, `whoami` («¿quién soy?») le dice con qué usuario entró y `ls` le enseña lo que hay en esa carpeta. Donde dice `ana`, en tu pantalla va a decir tu nombre. Si `ls` no te contesta nada, o te enseña otras carpetas, no pasa nada; en el recuadro que viene te digo cómo dejar tu casa igual que la de Ana.

Todas las salidas de terminal de este libro son reales: se teclearon en un Ubuntu Server 24.04 LTS, en una máquina virtual, y las de este capítulo, todas en una misma sesión (salvo unas pocas que agregué después en la misma máquina, y los ejemplos armados, que van rotulados). En tu máquina van a cambiar las fechas y algunos números; lo demás debe verse igual. Cuando un renglón no cabe en la página, lo parto y marco el corte con ↵; en tu pantalla sale en uno solo. Dos diferencias más: en tu pantalla, `ls` acomoda los nombres en columnas y con colores (aquí salen uno por renglón, que en papel se leen mejor), y si instalaste Ubuntu en español, algunos mensajes te van a salir traducidos. Aquí van en inglés, como los vas a encontrar en casi cualquier foro; no hace falta dominar el idioma, con reconocer media docena de frases alcanza, y te las voy señalando. Si tienes la versión más nueva, la 26.04 LTS, casi todo funciona igual; las diferencias que notarás te las señalo.



Si tu casa no se parece a la de Ana. Si acabas de instalar Linux, sea Ubuntu dentro de Windows (WSL), un servidor o un escritorio, tu casa puede empezar vacía o con otras carpetas, sin `Documentos` ni `nota.txt`, y los ejemplos de Ana no te van a salir igual. Para tener lo mismo que ella, teclea o pega estos tres renglones en tu terminal, uno por uno, cada uno con su `Enter`. Tecléalos así aunque todavía no los entiendas; los explico en el capítulo 1. Si ya tenías alguna de esas carpetas, no pasa nada: no se borra nada de lo que hay adentro.

para copiar tal cual

```
mkdir -p ~/Documentos ~/Descargas ~/Imágenes
printf 'Hola, Linux.\nEsta es mi primera nota.\n' > ~/Documentos/nota.txt
printf 'pendientes de la semana\n' > ~/Documentos/pendientes.txt
```



Si usas WSL. Para pegar: en Windows Terminal pega con `Ctrl+V` o con clic derecho; en una terminal de Linux de escritorio, con `Ctrl+Shift+V`. Y unas cuantas cosas se ven distinto. El arranque de la sección 4 no es tuyo, porque a la máquina la prende Windows, así que `systemd-analyze`, `ls /boot` y `lsmod` pueden darte otra cosa o nada. Tus discos de Windows aparecen montados dentro del árbol, en `/mnt/c` y compañía (qué es «montar», en la sección 11). Y tus archivos nuevos pueden salir como `-rw-r--r--` en vez de `-rw-rw-r--`; los dos están bien. Lo demás del libro funciona igual.

Cuando un dato viene de otro lado, le pongo un número chiquito arriba, así[1]. Al final de cada sección está la nota con el autor y el año. Las citas de fuentes en inglés van traducidas por mí.

1. Así se ven las notas: autor y año. La ficha completa de cada obra está en *Referencias*, al final del capítulo.

1 El tablero de la nave

En el prólogo te conté que mi primera computadora me parecía una nave espacial y que yo no sabía cómo volaba. Empecemos por ahí, porque casi nadie lo explica: ¿qué hay entre tú y la máquina?

Por dentro, tu computadora es fierro (*hardware*): un procesador que hace cuentas, una memoria que recuerda mientras está encendida, un disco que recuerda aunque la apagues, y dispositivos, aparatos, para hablar con el mundo: teclado, pantalla, red. Tú nunca le hablas directo a ese fierro. Cuando abres una foto no le dices al disco «dame estos datos»; se lo pides a un programa, y ese programa se lo pide a otro más grande que manda sobre todo lo demás. Ese programa grande es el **sistema operativo**.

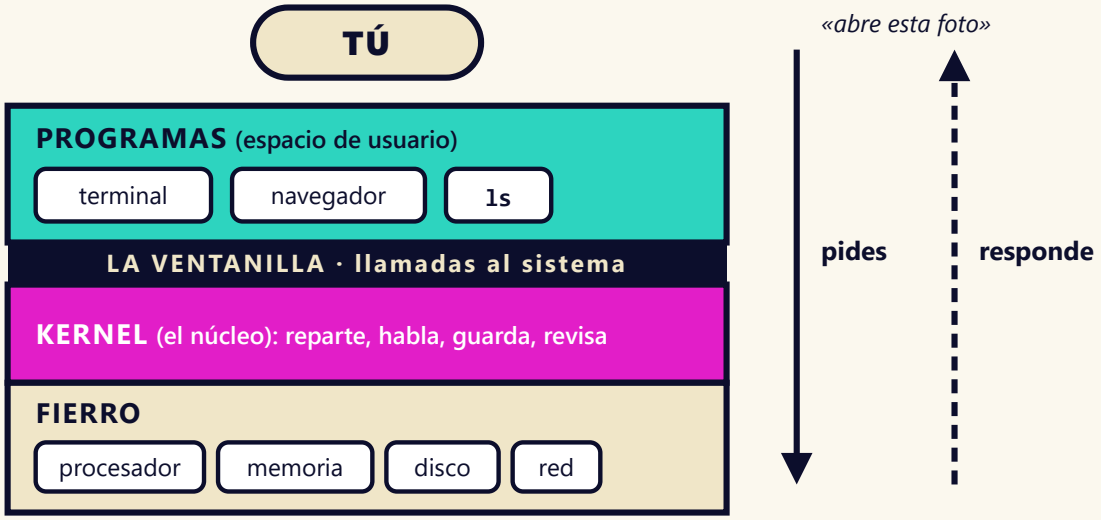


Figura 0.1. *La orden baja por las capas y la respuesta sube. Tú solo ves la de arriba, y entre los programas y el kernel hay una ventanilla que nadie se salta.*

El corazón del sistema operativo se llama **kernel**, que en inglés es «núcleo» o «semilla». Es el que decide qué programa usa el procesador en este instante, cuánta memoria le toca a cada uno y cómo se habla con el teclado, el disco o la tarjeta de red. Stallman lo define en una línea: el núcleo es «el programa del sistema que se encarga de asignar los recursos de la máquina a los demás programas que el usuario ejecuta»[2].

Windows es un sistema operativo. macOS es otro. Android, otro más. Linux es uno más de la lista, con una diferencia que lo cambia todo: es **libre y abierto**. De dónde salió eso te lo cuento en la sección 5; antes quiero que veas qué hace por dentro el kernel.



No te confundas. «Linux», estrictamente, es solo el kernel: la capa de en medio del dibujo. Solo, no sirve para nada; no trae ni un comando para ver tus archivos. El propio Stallman lo dice así: es esencial, pero inútil por sí mismo[2]. El sistema completo que instalas trae además cientos de programas de muchos otros proyectos; el más importante para la terminal es GNU. En la sección 5 te cuento de él.

2. Stallman (s. f.-a).

2 Los cuatro trabajos del kernel

El video dice «el kernel reparte». Vale la pena saber *qué* reparte, porque cuando lleguemos a procesos, a discos y a permisos vas a descubrir que son las cuatro caras del mismo programa.

El kernel es un programa, uno solo y grande, que arranca antes que cualquier otro programa del sistema y no se cierra mientras la máquina esté prendida. Cualquier libro de sistemas operativos dedica sus primeros capítulos a sus cuatro trabajos[3]; para Linux en particular, el libro de Ward es una buena puerta[4].

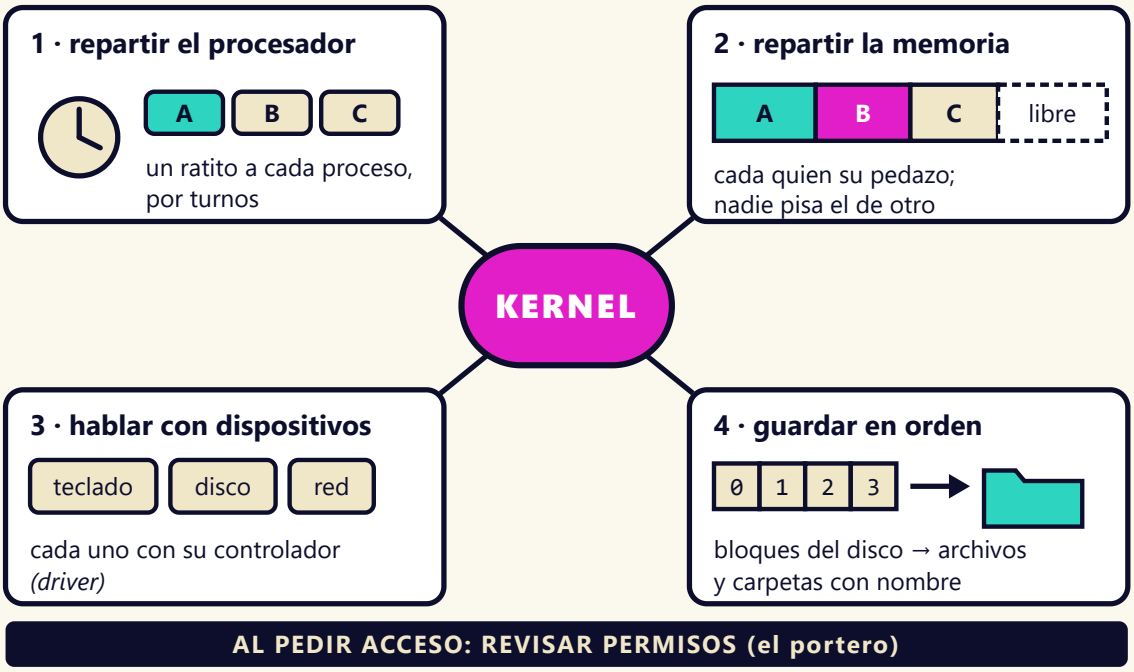


Figura 0.2. Cuatro trabajos y un portero. Todo lo que hacen los demás programas pasa por aquí.

1. Repartir el procesador. En tu máquina hay muchos programas vivos y pocos procesadores. El kernel le da a cada uno un ratito, se lo quita, se lo da a otro, y lo hace tan rápido que parece que todos corren a la vez. A cada programa en marcha le llama **proceso**. Los libros de sistemas operativos le dicen a esto *virtualizar el procesador*: hacerle creer a cada programa que tiene un procesador para él solo[5].

2. Repartir la memoria. Cada proceso cree que tiene su propia memoria privada, entera para él. El que decide qué pedazo de la memoria de verdad le toca, y el que impide que uno lea o pise la de otro, es el kernel[5]. Por eso un programa que se cae no se lleva a los demás.

3. Hablar con los dispositivos. Disco, teclado, pantalla, red. Cada dispositivo necesita un pedazo de programa que sepa hablarle: el **controlador**, o *driver*. Cuando el kernel arranca, inicializa los componentes de la máquina, y a cada porción de software encargada de uno de ellos se le llama así, un *driver*[6]. Sí: es lo mismo que yo aprendí a la mala reinstalando una máquina cinco veces porque un teclado ajeno no quería hablar con ella.

4. Guardar las cosas en orden. Para el disco, todo es una tira enorme de bloques numerados. El kernel la convierte en archivos y carpetas con nombre, dueño y permisos: el **sistema de archivos**. Es la parte encargada de que tus datos sobrevivan al apagado[5].

Y hay un quinto que atraviesa a los otros cuatro: **hacer de portero**. Como todo pasa por él, el kernel es quien revisa si quien pide algo tiene permiso, y lo revisa al pedir acceso, sobre todo al abrir un archivo; no en cada letra que se lee después. Cuando más adelante veas un `Permission denied`, acuérdate de este párrafo: es el portero haciendo su trabajo. Y una rareza: el kernel en sí no es un proceso más de la lista; lo que ves en ella son hilos de trabajo que él mismo arranca (como `kthreadd` y los `kworker`). No es uno más de la fila; es quien hace la fila.

En tu máquina con Linux puedes ver al kernel trabajando. Tres de estos comandos usan la barra `|` y los comandos `wc` y `head`; tecléalos así aunque la `|` te la explico en la sección 12. Así le salió a Ana:

```
ana@linux:~$ uname -r
6.8.0-149-generic
ana@linux:~$ ps -e | wc -l
126
ana@linux:~$ ps -e | head
    PID TTY          TIME CMD
      1 ?        00:00:02 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 pool_workqueue_release
      4 ?        00:00:00 kworker/R-rcu_g
      5 ?        00:00:00 kworker/R-rcu_p
      6 ?        00:00:00 kworker/R-slub_
      7 ?        00:00:00 kworker/R-netns
      8 ?        00:00:00 kworker/0:0-rcu_gp
      9 ?        00:00:00 kworker/0:1-events
```

- `uname -r` dice qué versión de kernel tiene Ana: la 6.8. Es un número distinto del de la distro, con su propio calendario.
- `ps -e | wc -l` cuenta los procesos que se reparten el procesador: 126 renglones, o sea 125 procesos, porque `wc -l` cuenta también el encabezado. En la lista entran hilos del propio kernel: el 2, `kthreadd`, y los `kworker` que le siguen son trabajadores del kernel, no programas de Ana. El 1, `systemd`, te lo presento en la sección 4.

Y los primeros módulos cargados (con `head -n 5` salen el encabezado y cuatro módulos):

```
ana@linux:~$ lsmod | head -n 5
Module                  Size  Used by
qrtr                   53248  2
cfg80211             1368064  0
binfmt_misc            24576  1
intel_rapl_msr         20480  0
```

Muchos módulos son controladores. `cfg80211`, por ejemplo, es el que configura las tarjetas de red inalámbrica. En tu máquina los números y los nombres van a ser otros: ese es justo el chiste.

3. Silberschatz et al. (2018); Tanenbaum y Bos (2023).
4. Ward (2021), cap. 1.
5. Arpaci-Dusseau y Arpaci-Dusseau (2023), cap. 2.
6. Linux man-pages project (s. f.-a).

3 Dos mundos y una ventanilla

Ahora la pregunta obvia: si el kernel hace todo eso, ¿cómo hacen los demás programas para leer un archivo o escribir en la pantalla?

El procesador tiene, por lo menos, dos modos de trabajo. En **modo usuario** el código está restringido: no puede tocar el disco, ni la memoria de otro programa, ni la tarjeta de red. En **modo kernel** se puede todo. Tus programas, sean `ls`, el navegador o la propia terminal, corren siempre en modo usuario. El kernel corre en modo kernel[7]. A esos dos lados se les llama **espacio de usuario** y **espacio del kernel**.

Entonces, ¿cómo lee `cat` un archivo si no puede tocar el disco? **Pidiéndolo.** Hay una lista cerrada de peticiones que un programa le puede hacer al kernel, y cada una se llama **llamada al sistema** (*system call*). La página de manual lo define seco: la llamada al sistema es «la interfaz fundamental entre una aplicación y el kernel de Linux»[8].

Yo me lo imagino como una ventanilla de oficina de gobierno. El programa no entra a la bodega: llena una forma («abre este archivo», «léeme lo que tenga», «escribe esto en la pantalla») y la pasa por la ventanilla. Del otro lado, el kernel revisa si se puede, lo hace él y regresa el resultado.

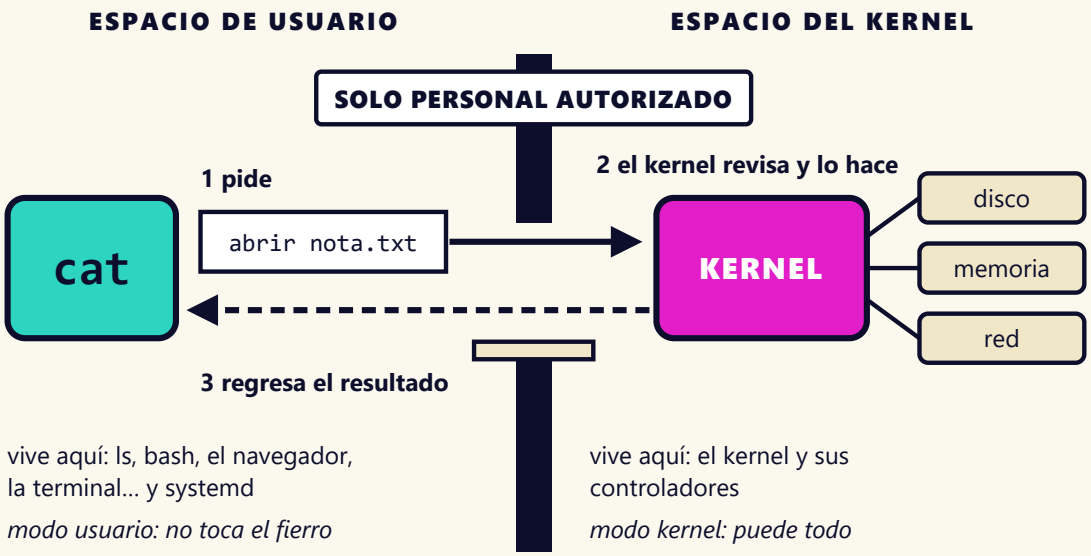


Figura 0.3. *Un programa pide; no toma. Cada forma que pasa por la ventanilla es una llamada al sistema.*

Por dentro, pasar la forma es una instrucción especial del procesador que salta al kernel y sube el nivel de privilegio, y al terminar lo baja otra vez[7]. El programador casi nunca lo ve: llama a una función normal de la biblioteca de C, y esa función toca la ventanilla[9].

Cuando Ana teclea `cat nota.txt`, `cat` pasa cuatro formas: *abrir* `nota.txt`, *leer* su contenido, *escribirlo* en la pantalla y *cerrar*. La revisión de permisos ocurre en la primera, al abrir. Por eso, cuando Ana intenta leer el archivo de contraseñas del sistema, `cat` se queda en la ventanilla sin haber leído una sola letra:

```
ana@linux:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

Fíjate quién firma el mensaje: `cat`. Es él quien te cuenta que la ventanilla le dijo que no. Si te pica la curiosidad de ver las formas una por una, en *Para los curiosos*, al final del capítulo, te enseño cómo espiarlas.



No te confundas. «Con `sudo`, mi programa corre dentro del kernel». No. `sudo`, que veremos en la sección 15, cambia *quién eres* por un momento, solo para ese comando; el programa sigue en modo usuario y sigue pasando por la ventanilla. Lo único que cambia es que al portero ya casi nada le parece mal. «Usuario normal contra administrador» y «modo usuario contra modo kernel» son dos cosas distintas.

7. Arpaci-Dusseau y Arpaci-Dusseau (2023), cap. 6.
8. Linux man-pages project (s. f.-f).
9. Linux man-pages project (s. f.-c).

4 Del botón de encendido al escritorio

Cada vez que prendes la máquina pasan renglones volando, o un logo que da vueltas. Detrás hay una carrera de relevos de cinco corredores, y cada uno hace lo mínimo para pasarle la estafeta al siguiente[10]. Ward le dedica dos capítulos, si quieres el detalle[11].

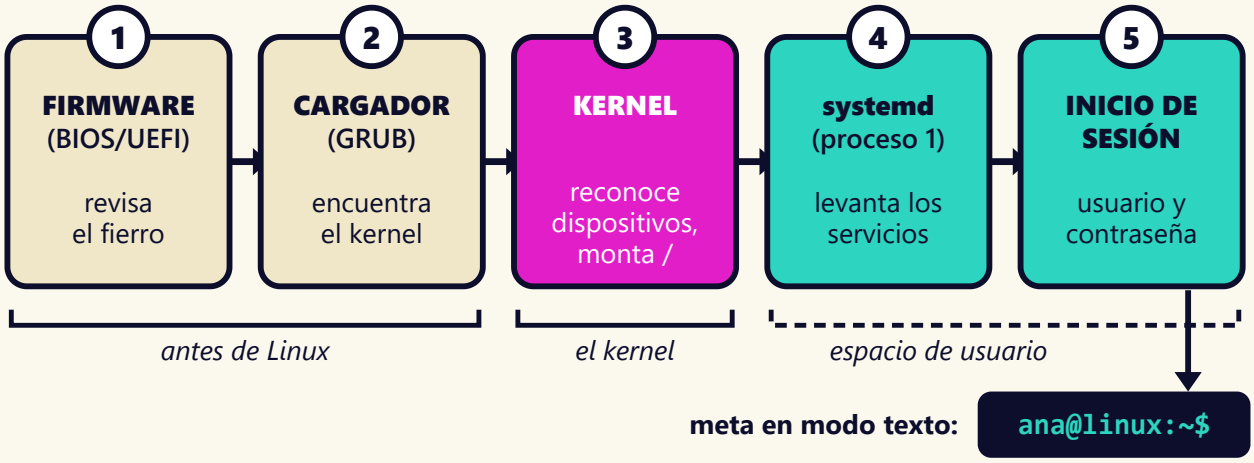


Figura 0.4. Nadie hace todo: cada relevo despierta al siguiente y se hace a un lado.

1. Firmware. Un programa chico grabado en la tarjeta madre. Al encender, el control pasa a él; por razones históricas se le suele llamar *el BIOS*, y en las máquinas actuales su sucesor se llama UEFI[12]. Revisa que el fierro responda y busca en qué disco hay algo que sepa arrancar un sistema.

2. Cargador de arranque. En Ubuntu se llama **GRUB**, y es el menú que a veces alcanzas a ver un segundo al prender. Su único trabajo es localizar el kernel en el disco, subir a la memoria el kernel y su disco provisional (te lo presento en el siguiente relevo) y echarlo a andar[13].

3. Kernel. Se acomoda en la memoria, reconoce el fierro con sus controladores y monta el sistema de archivos raíz, `/`, del que vamos a hablar en la sección 11. **Montar** es colgar un disco, o una parte de él, de una carpeta del árbol para que su contenido se vea ahí; también te lo explico en la sección 11. Para encontrar el disco de verdad suele apoyarse en un disco provisional que vive en la memoria y trae lo justo para eso[14].

4. systemd, el proceso número 1. Ya con el disco de verdad en su lugar, el kernel arranca un programa del lado de los usuarios, el primero que se queda, y todos los demás descienden de él. En Ubuntu ese programa es **systemd**: monta lo que falta y levanta los servicios (la red, los registros, el reloj), muchos de ellos en paralelo[14].

5. Inicio de sesión. systemd llega a su meta. Si la máquina es solo de texto, aparece un renglón que dice `login:`; pones tu usuario y tu contraseña, y si son buenos, el sistema arranca **tu shell**. Ahí aparece, por fin, `ana@linux:~$`. Si la máquina tiene escritorio, la meta es la pantalla de inicio de sesión y, al entrar, tu sesión gráfica; la shell aparece cuando abres la terminal.



No te confundas. Dos confusiones comunes. «GRUB es parte de Linux»: no, es un programa aparte, y de hecho sabe arrancar otros sistemas. «systemd es el kernel»: tampoco; es el primer programa del lado de los usuarios, vive del lado de acá de la ventanilla.

Los relevos se pueden ver en tu máquina. `ps -p 1 -o pid,comm` te dice cómo se llama el proceso número 1; `ls /boot` enseña dónde guarda Ubuntu el kernel y los archivos del cargador, y `systemd-analyze` te dice cuánto tardó el último arranque. Así le salió a Ana:

```
ana@linux:~$ ps -p 1 -o pid,comm
    PID COMMAND
      1 systemd
ana@linux:~$ ls /boot
config-6.8.0-149-generic
grub
initrd.img
initrd.img-6.8.0-149-generic
initrd.img.old
lost+found
System.map-6.8.0-149-generic
vmlinuz
vmlinuz-6.8.0-149-generic
vmlinuz.old
ana@linux:~$ systemd-analyze
Startup finished in 9.259s (kernel) + 8.655s (userspace) = 17.915s
graphical.target reached after 8.551s in userspace.
```

- El proceso número 1 se llama `systemd`, como te dije.
- En `/boot` está el kernel, `vmlinuz-6.8.0-149-generic` (el mismo número que dio `uname -r`), junto a su disco provisional, `initrd.img-…`, y la carpeta `grub`, del cargador. Los nombres sin número son flechas a los de en uso; los `.old` son para tener de dónde agarrarse si un kernel nuevo falla.
- `systemd-analyze` mide los últimos relevos: 9.3 segundos de kernel y 8.7 de espacio de usuario (*userspace*), casi 18 en total. Son los mismos relevos de la Figura 0.4, con cronómetro. Si tu máquina enseña además `(firmware)` y `(loader)`, son los dos primeros; muchas, como la de Ana, que es una máquina virtual, solo miden desde el kernel.

- `graphical.target` es la meta de systemd. En Ubuntu es la meta por defecto aunque no haya escritorio (`systemctl get-default` le contesta `graphical.target` a Ana, en su servidor sin pantalla gráfica), e incluye todo lo de `multi-user.target`, la meta de «servicios arriba y sesiones de texto».



En un servidor real. Un servidor no tiene escritorio: termina en `login:`, y casi nunca te sientas frente a él; entras desde otra máquina por SSH, que vemos en el capítulo 11. Lo primero que revisa un administrador después de un reinicio es si los servicios levantaron. `systemctl status` enseña, hasta arriba, un renglón `State:`: si dice `running`, todo arrancó; si dice `degraded`, algo falló, y `systemctl --failed` te dice qué. Lo vemos con calma en el capítulo 4, el de procesos.

La máquina de Ana es justo un servidor, así que se puede ver:

```
ana@linux:~$ systemctl status | head
● linux
    State: running
    Units: 436 loaded (incl. loaded aliases)
     Jobs: 0 queued
   Failed: 0 units
    Since: Sat 2026-11-14 20:43:26 UTC; 16s ago
  systemd: 255.4-1ubuntu8.18
   CGroup: /
           ├─init.scope
           │ └─1 /sbin/init
ana@linux:~$ systemctl --failed
  UNIT LOAD ACTIVE SUB DESCRIPTION

0 loaded units listed.
```

`State: running` y `Failed: 0 units`: todo arrancó, y por eso `systemctl --failed` sale vacío, con cero unidades en la lista. `Since:` dice desde cuándo está prendida. Abajo empieza el árbol de procesos que cuelgan de systemd; `/sbin/init` es el nombre histórico del proceso 1, y en Ubuntu es una flecha hacia systemd.

---

10. systemd project (s. f.-a); Linux man-pages project (s. f.-a).
11. Ward (2021), caps. 5 y 6.
12. Linux man-pages project (s. f.-a); systemd project (s. f.-a).
13. Linux man-pages project (s. f.-a).
14. systemd project (s. f.-a).

5 Una historia de tres necios

A mí las historias me ayudan a que las cosas se queden. Esta además explica por qué Linux es como es.

1969. Unix. En los laboratorios Bell, en Estados Unidos, Ken Thompson y Dennis Ritchie hacen un sistema operativo y le ponen Unix[15]. En 1973 reescriben su kernel en un lenguaje que habían inventado ahí mismo, el lenguaje C, y eso hace posible llevarlo de una máquina a otra. Unix dejó ideas que seguimos usando: que varias personas usen la misma máquina a la vez, que corran muchas tareas al mismo tiempo, que los programas chicos se puedan conectar unos con otros y que, dentro del sistema, todo se trate como un archivo. De la primera y la última vamos a hablar bastante.

1983. GNU. Con los años, Unix se volvió cada vez más caro y con candados para estudiarlo y compartirlo. A un programador, Richard Stallman, eso le pareció mal, y el 27 de septiembre de 1983 anunció un proyecto con una meta necia: escribir un sistema completo, compatible con Unix, y regalarlo a quien lo quisiera usar[16]. Lo llamó GNU. Para principios de los noventa ya tenía todo el sistema listo… menos el núcleo[17]. Tenía el cuerpo y le faltaba el corazón.

1991. Linux. En Finlandia, un estudiante llamado Linus Torvalds estaba haciendo, por pura curiosidad, un kernel para su computadora. El 25 de agosto de 1991 lo anunció en un grupo de noticias de internet con un mensaje que se volvió famoso:

> *«I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.»*

Traducido: «Estoy haciendo un sistema operativo (gratis), solo un pasatiempo, no será grande ni profesional como GNU, para clones AT 386 (486)»[18]. Se equivocó por completo, y qué bueno.

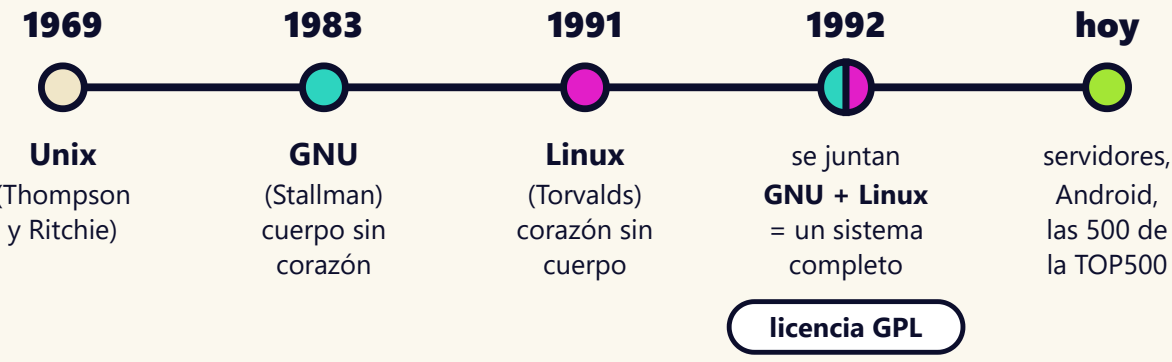


Figura 0.5. *GNU tenía todo menos el núcleo. Linus tenía el núcleo y nada más.*

Las dos piezas encajaron. Ese mismo 1991 Linus publicó la primera versión de su kernel, y en 1992 la puso bajo la licencia de GNU, la GPL, de la que hablamos en la siguiente sección[19]. Con eso se cerró el último hueco importante del sistema GNU[17]. Por eso, para ser justos, el sistema completo se llama **GNU/Linux**: las herramientas de GNU con el corazón de Linus. En el uso diario todos decimos «Linux», y yo también lo voy a hacer.

Años después llegó la mascota: Tux, el pingüino. Lo dibujó Larry Ewing con GIMP, un programa libre de dibujo, cuando en la comunidad se discutía qué logo ponerle a Linux[20]. A Linus le caían bien los pingüinos; no hay una razón más profunda.

Hoy ese pasatiempo mueve buena parte de los servidores de internet, el teléfono Android que quizá traes en la bolsa y, para que veas hasta dónde llegó, todas las supercomputadoras de la lista TOP500, la que junta a las 500 más rápidas del mundo. Desde noviembre de 2017, las 500 corren Linux[21]. En el video lo dije de corrido, «todos los supercomputadores»; el dato exacto es este: todos los de esa lista.

---

15. Ritchie (1984).
16. Stallman (1983).
17. Stallman (s. f.-a).
18. Torvalds (1991).
19. Torvalds (s. f.).
20. Baker (s. f.).
21. TOP500 (s. f.).

---

6 Libre, abierto y la GPL

El video dice «libre y abierto» de corrido, como si fuera una sola palabra. Son dos palabras con dos historias, y la licencia GPL es una de las razones por las que Linux no terminó cerrado como le pasó a Unix.

Un programa es **libre** si quien lo usa tiene cuatro libertades, que el proyecto GNU numera desde el cero[22]: **0**, ejecutarlo como quieras, con cualquier propósito; **1**, estudiar cómo funciona y cambiarlo, para lo que hace falta el código fuente; **2**, redistribuir copias; **3**, distribuir copias de tus versiones modificadas. «Libre» como en «libertad de expresión», no como en «cerveza gratis». A mí la libertad 1 me tomó años apreciarla. Yo venía de discos del tianguis; que alguien te dejara abrir el cofre y mirar el motor era otra cosa.

La GPL es la licencia que cuida esas cuatro libertades: quien reparte un programa con GPL, igual o modificado, tiene que repartir también su código fuente y con la misma licencia. A esa idea se le llama *copyleft*. El kernel Linux va bajo la GPL versión 2.

¿Y «abierto»? «Código abierto» (*open source*) es un término posterior a «software libre», con su propia definición de diez criterios, entre ellos el acceso al código fuente y el permiso para hacer obras derivadas[23]. En la práctica casi siempre señalan a los mismos programas, y Linux cumple las dos. Para este libro basta con esto: *libre* es el porqué, *abierto* es el cómo. Si quieres ver la GPL por dentro, está en *Para los curiosos*.



No te confundas. Tres malentendidos. «Libre es gratis»: no, se puede cobrar. «La GPL me obliga a publicar lo que yo cambie»: solo si lo repartes. «Si mi programa corre en Linux, tiene que ser libre»: no; en *Para los curiosos* te enseño la frontera que lo permite.

22. Free Software Foundation (s. f.-c).
23. Open Source Initiative (2007).

7 Qué clase de sistema es

Cuatro rasgos definen a Linux. Los tres primeros los vas a ir viviendo; el cuarto lo puedes comprobar hoy.

1. **Es libre y de código abierto.** Su código está abierto: cualquiera puede revisar cómo funciona.
2. **Su kernel es monolítico con módulos.** Monolítico quiere decir que es un solo programa grande; con módulos, que puede cargar y soltar piezas, llamadas **módulos**, sin apagar la máquina. Muchos módulos son controladores: así reconoce, por ejemplo, una tarjeta de red que acabas de conectar.
3. **Nació para muchos usuarios y muchas tareas a la vez.** Lo heredó de Unix.
4. **Todo es un archivo.**

Ese cuarto suena a frase de camiseta, así que vamos a verlo. Tus fotos y tus documentos son archivos, eso ya lo sabías. Pero en Linux también lo son el disco, el teclado y otros dispositivos. Mira este:

```
ana@linux:~$ ls -l /dev/null
crw-rw-rw- 1 root root 1, 3 Nov 14 20:43 /dev/null
```

Léelo conmigo. Lo que importa es la primera letra del renglón: una `c`. Un archivo normal empieza con un guion y una carpeta con una `d`. La `c` avisa que esto no es un documento guardado en el disco, sino un **dispositivo** que el sistema te presenta con forma de archivo. Este en particular, `/dev/null`, es el más raro de todos: un hoyo. Todo lo que le mandas, desaparece; y si lo lees, no hay nada.

Otro ejemplo, todavía más curioso. El nombre de la máquina de Ana es `linux`. Ese dato lo tiene el kernel en su memoria, pero te lo presenta como si fuera un archivo de texto que se puede leer con `cat`:

```
ana@linux:~$ cat /proc/sys/kernel/hostname
linux
```

La carpeta `/proc` no está en el disco: el kernel la arma al vuelo cada vez que alguien se asoma. Es la «ventana al kernel» que menciona el video.

No te pido que lo entiendas todo todavía. Quédate con la idea: **una sola regla para todo**. Si sabes leer y escribir archivos, ya sabes hablar con medio sistema. En la sección 14 lo comprobamos.

---

8 ¿Por qué hay cientos de Linux?

Esto confunde a todo el que empieza, y a mí me confundió también. Buscas «descargar Linux» y no hay uno: hay Ubuntu, Debian, Fedora, Mint, Arch y cien más.

La razón es sencilla. Linux, el kernel, es solo el motor. Alguien tiene que armar el coche completo: el motor, las herramientas de GNU, un **gestor de paquetes** (el programa con el que instalas otros programas) y un escritorio. A ese coche ya armado se le llama **distribución**, o **distro**.

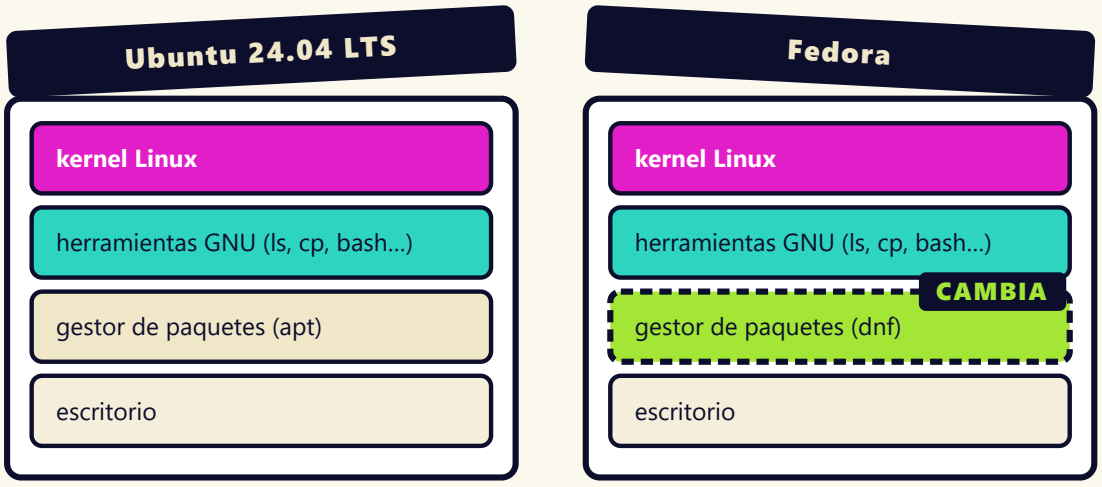


Figura 0.6. *Una distro es un empaque. De una a otra cambia poco, y casi siempre lo que más cambia es el gestor de paquetes.*

Se agrupan en familias, y lo que más cambia de una familia a otra es el comando con que instalas programas:

FAMILIA	ALGUNAS DISTROS	GESTOR DE PAQUETES
Debian	Debian, Ubuntu, Mint	`apt`
Red Hat	Fedora, RHEL, Rocky	`dnf`
Arch	Arch, Manjaro	`pacman`

Todo este libro está hecho en **Ubuntu**, de la familia Debian. Si usas otra, casi todo te va a servir igual; donde haya diferencia, te aviso.

› Cómo leer la versión de tu distro

Lo primero que te van a preguntar cuando pidas ayuda en un foro es «¿qué versión tienes?». La respuesta vive en un archivo de texto:

```
ana@linux:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.5 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
…
```

Corté casi todos los últimos renglones, que son direcciones de internet. Vamos pieza por pieza con «Ubuntu 24.04.5 LTS (Noble Numbat)»:

PEDAZO	QUÉ DICE
`24.04`	**Año y mes** en que salió: abril de 2024. No es «versión 24 punto 4»: Ubuntu numera sus versiones por el año y el mes de entrega[24]
`.5`	Una *point release*: la quinta puesta al día de esa misma versión, con las actualizaciones acumuladas hasta entonces. Para lo que importa, sigue siendo la 24.04
`LTS`	*Long Term Support*, soporte de largo plazo. Sale una cada dos años y recibe 5 años de mantenimiento de seguridad estándar; las versiones de en medio salen cada seis meses y solo reciben 9 meses de actualizaciones[24]
`Noble Numbat`	El apodo. `noble` es la palabra corta que vas a ver en direcciones y repositorios

PEDAZO	QUÉ DICE
`ID_LIKE=debian`	De qué familia viene: los sistemas emparentados de los que deriva[25]. Es la tabla de familias de arriba, escrita en un archivo

Este libro usa la 24.04 LTS, que tiene soporte estándar hasta mayo de 2029. En abril de 2026 salió la 26.04 LTS, que hoy es la LTS vigente, con soporte hasta 2031[24]. Si tienes la 26.04, casi todo funciona igual; las diferencias que notarás te las señalo. Una que puede tocarte desde ya: en la 26.04, las herramientas básicas como `ls` o `cat` vienen de una reescritura en el lenguaje Rust[26], así que algunos mensajes de error pueden variar en la redacción. La forma de leerlos es la misma.

Para aprender y para servidores, usa una **LTS**. Que una versión intermedia como la 25.10 sea más nueva no la hace mejor para empezar: la LTS es la que tiene años de soporte y la que usan los tutoriales. Y no confundas la versión de la **distro** (24.04) con la del **kernel**, que es otro número con otro calendario.



No te confundas. Elegir distro no es elegir «otro sistema». Yo perdí semanas brincando de una a otra creyendo que el problema era la distro, cuando el problema era que todavía no entendía Linux. Agarra Ubuntu o Mint y quédate ahí hasta terminar el libro.

24. Canonical (s. f.-a).
25. systemd project (s. f.-b).
26. Canonical (s. f.-b).

9 Linux visto desde Windows

Yo llegué de Windows, y seguramente tú también. Estas son las diferencias que más me costaron, para que a ti te cuesten menos:

	WINDOWS	LINUX
Los discos	Letras: `C:`, `D:`... cada una su mundo	Un solo árbol que empieza en `/`
La configuración	El registro: una base de datos escondida	Casi siempre, archivos de texto que puedes abrir y leer
Las rutas	Se separan con `\`	Se separan con `/`
Mayúsculas	`Carta` y `carta` son lo mismo	`Carta` y `carta` son **dos archivos distintos**

	WINDOWS	LINUX
La terminación	`.txt` o `.exe` deciden qué es el archivo	Para el kernel y la shell no decide nada (sección 14)
El código	Cerrado	Abierto

La de las mayúsculas es la que más lata da al principio. Ana tiene un archivo que se llama `nota.txt`, todo en minúsculas. Mira lo que pasa si lo pide con mayúscula:

```
ana@linux:~/Documentos$ cat Nota.txt
cat: Nota.txt: No such file or directory
```

«No existe ese archivo o directorio». *Directorio* (*directory*) es como Linux le dice a las carpetas; en este libro uso las dos palabras como sinónimos. Y tiene razón: `Nota.txt` no existe. El que existe es `nota.txt`.

Ese mensaje lo vas a ver cien veces. Casi siempre significa una de dos cosas: escribiste mal el nombre, o estás parado en una carpeta que no es. No significa que rompiste algo. Lo de la terminación lo vamos a comprobar en la sección 14.

10 Terminal, shell y comando

El video dice «abre la terminal» y «teclea el comando» como si fuera una sola cosa. Son tres, y separarlas te va a ahorrar mucha confusión: explica por qué `cd` no tiene página de manual, por qué `Ls` «no existe» y qué es ese `bash:` que aparece al principio de algunos errores.

- **La terminal** es la ventana. Recibe tus teclas y pinta letras; no entiende nada de lo que escribes. Antes era una máquina aparte, con teclado y papel; hoy es un programa que lo imita.
- **La shell** es el programa que corre *dentro* de la ventana: lee el renglón que tecleaste, lo interpreta y manda a ejecutar lo que corresponda. En Ubuntu es **Bash**, un intérprete de comandos que ejecuta lo que lee del teclado o de un archivo[27]. El prompt `ana@linux:~$` lo pinta la shell, no la terminal.
- **El comando** es lo que la shell manda a ejecutar. Y aquí está el detalle fino: hay de varias clases.

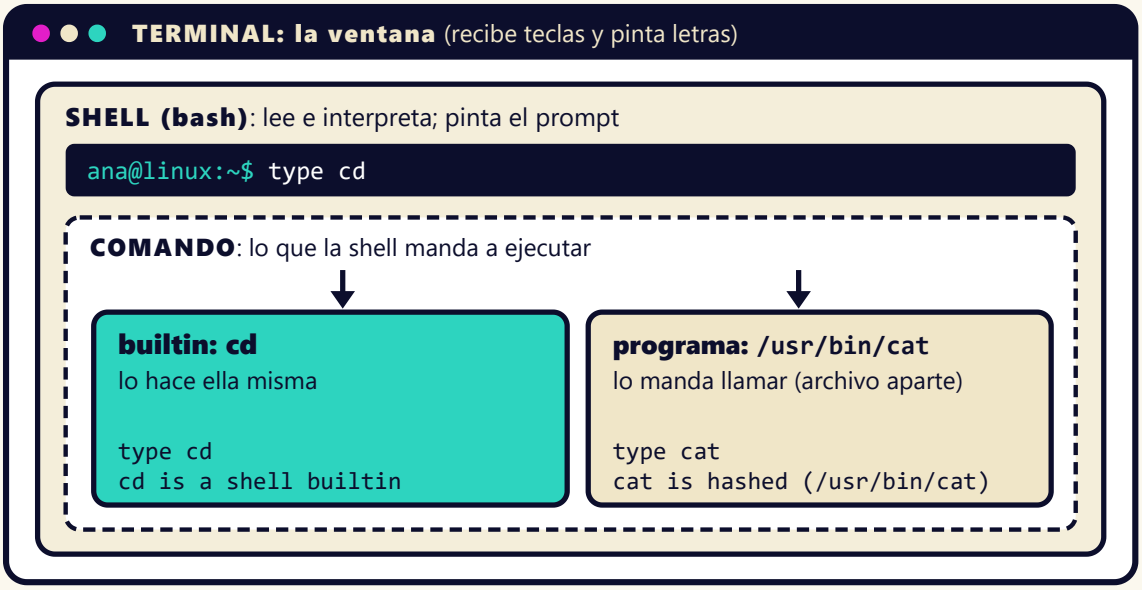


Figura 0.7. Tres cajas, una dentro de otra. La ventana no entiende nada; la que entiende es la shell.

Para saber de qué clase es un comando, se le pregunta a la shell con `type`:

```
ana@linux:~$ echo $SHELL
/bin/bash
ana@linux:~$ type cat
cat is hashed (/usr/bin/cat)
ana@linux:~$ type ls
ls is aliased to `ls --color=auto'
ana@linux:~$ type -a ls
ls is aliased to `ls --color=auto'
ls is /usr/bin/ls
ls is /bin/ls
ana@linux:~$ type cd
cd is a shell builtin
ana@linux:~$ type pwd
pwd is a shell builtin
ana@linux:~$ type -a echo
echo is a shell builtin
echo is /usr/bin/echo
echo is /bin/echo
ana@linux:~$ type ll
ll is aliased to `ls -alF'
ana@linux:~$ alias ll='ls -la'
ana@linux:~$ type ll
ll is aliased to `ls -la'
ana@linux:~$ type lss
bash: type: lss: not found
```

Léelo conmigo, renglón por renglón:

- `echo $SHELL` pregunta cuál es la shell predeterminada de Ana, la que se abre cuando entra: `/bin/bash`. `SHELL` es una **variable**: un nombre con un dato guardado adentro; el `$` de enfrente le pide a la shell que lo cambie por su valor.
- `cat` es **un archivo**, un programa aparte guardado en `/usr/bin/cat`. «Hashed» solo quiere decir que Bash ya lo usó antes y se acordó de dónde está.
- `ls` trae sorpresa: en Ubuntu es un **alias**, un apodo. El Ubuntu de fábrica le pone a `ls` el apodo `ls --color=auto`, para que salga con colores. La opción `-a` de `type` pide *all*: todas las formas, no solo la que gana. Con ella se ve que debajo del apodo está el programa de siempre, en `/usr/bin/ls` (y en `/bin/ls`, que es el mismo, por una flecha que te enseño en la sección 14).
- `cd` es un **builtin**, «de fábrica»: no es un archivo en el disco, es parte de la propia shell. `pwd` también sale como builtin, pero tiene truco, ahorita lo ves.
- `echo` existe de las dos formas, y cuando lo tecleas gana la de la shell, que sale primero en la lista.
- `ll` es otro alias. Ubuntu ya lo trae, como apodo de `ls -alF`; Ana lo cambia por el suyo, `ls -la`, y el cambio le dura hasta que cierre la terminal.
- Y `lss` no es nada. Fíjate en el formato del error: `bash: type: lss: not found`. Habla Bash, sobre su builtin `type`, sobre `lss`, y dice qué pasó.

El truco de `pwd`: con `-a` se ve que, igual que `echo`, también existe como programa aparte. El único de estos que es *solo* builtin es `cd`:

```
ana@linux:~$ type -a pwd
pwd is a shell builtin
pwd is /usr/bin/pwd
pwd is /bin/pwd
ana@linux:~$ type -a cd
cd is a shell builtin
```

Ojo: todo esto pasa en una sesión **interactiva**, la que abres para teclear. Cuando un comando corre sin ella, por ejemplo mandado por SSH en un solo renglón o dentro de un script, la shell no carga los alias ni el ayudante de Ubuntu que sugiere comandos parecidos (lo ves en *Errores típicos*). Si Ana le manda `type ls` a su máquina por SSH desde otra, la respuesta ya no es el alias, sino `ls is /usr/bin/ls`.

¿Por qué `cd` tiene que ser parte de la shell? Porque la carpeta en la que estás parado es un dato *de la shell*. Un programa aparte podría cambiarse de carpeta él solo, pero al terminar, la shell seguiría donde estaba.

¿Y cómo encuentra la shell a `ls`? Si lo que tecleaste no trae diagonales, primero revisa si es un alias, una función (son como programitas guardados dentro de la shell; los verás en el capítulo 7) o un builtin. Si no es nada de eso, lo busca en una lista de carpetas que se llama `PATH`, de izquierda a derecha[27]:

```
ana@linux:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:↵
/usr/local/games:/snap/bin
```

Los dos puntos separan carpetas. Si el nombre no está en ninguna, la respuesta es `command not found`. Por eso `Ls`, con mayúscula, «no existe»: busca un archivo llamado `Ls` en cada una de esas carpetas y no lo encuentra.



No te confundas. «La terminal me dio un error». Casi nunca: mira quién firma. `bash: cd: …` lo dice la shell. `ls: cannot access…` lo dice el programa `ls`. `cat: …` lo dice `cat`. El nombre antes de los primeros dos puntos es quien se queja, y saberlo ya es medio diagnóstico.

27. Fox y Ramey (s. f.).

11 El mapa: un solo árbol

En Windows cada disco es su propio mundo. En Linux hay un solo punto de partida, que se escribe con una diagonal, `/`, y se llama **la raíz**. Todo lo demás cuelga de ahí. Si conectas una memoria USB, no aparece una letra nueva: el sistema la **monta**, es decir, la cuelga de una carpeta del árbol, y por esa carpeta ves lo que tiene.

Esto es lo que hay colgando de la raíz en un Ubuntu:

```
ana@linux:~$ ls /
bin
bin.usr-is-merged
boot
cdrom
dev
etc
home
lib
lib64
lib.usr-is-merged
lost+found
media
mnt
opt
proc
root
run
sbin
sbin.usr-is-merged
snap
srv
swap.img
sys
tmp
usr
var
```

Hoy importan unas pocas: `boot`, `dev`, `etc`, `home`, `proc`, `root`, `tmp`, `usr` y `var`. Tres rarezas de Ubuntu, para que no te asusten: `bin.usr-is-merged`, `lib.usr-is-merged` y `sbin.usr-is-merged` son restos de cuando `/bin` y compañía se fusionaron con `/usr/bin` (enseguida te cuento); `swap.img` es la memoria de intercambio, un pedazo de disco que el sistema usa como memoria de reserva, guardado en un archivo; y `snap` es donde viven los programas que se instalan con `snap`, otro gestor de paquetes de Ubuntu. Este es el mapa que yo hubiera querido tener pegado en la pared. El propósito de cada carpeta no es invento de Ubuntu: lo fija un estándar que comparten las distros, el *Filesystem Hierarchy Standard*[28], y lo resume la página de manual `hier`[29].

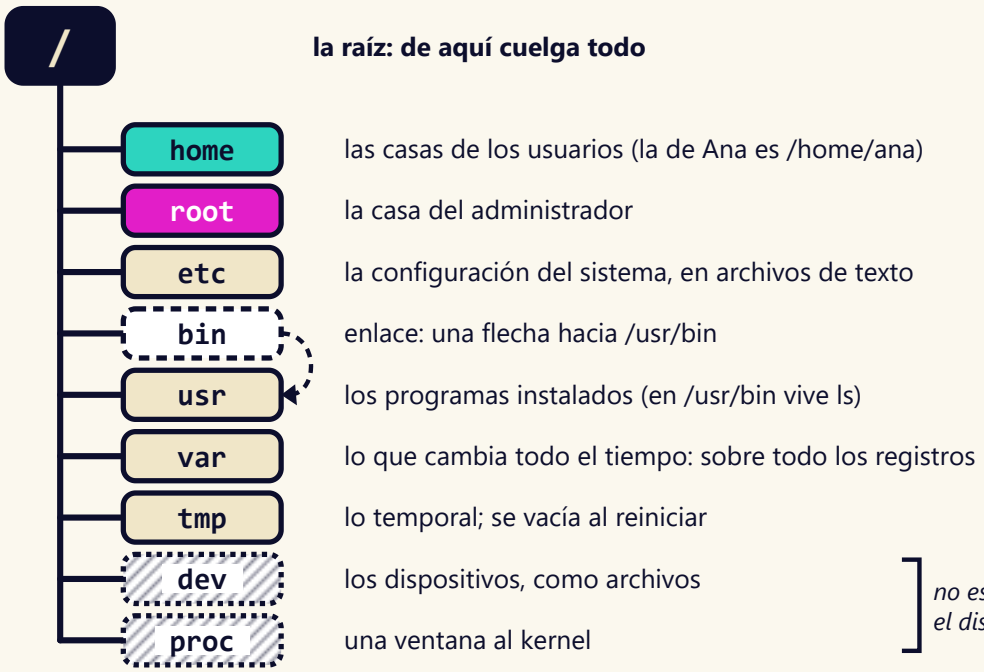


Figura 0.8. *Un árbol al revés: la raíz arriba, las ramas hacia abajo. La línea punteada es una flecha, no una carpeta.*

Tres cosas para que no te hagas bolas:

- `/` **y** `/root` **no son lo mismo.** La diagonal sola es la raíz del árbol. `/root` es una carpeta: la casa del usuario administrador, que se llama **root**, «raíz», porque en los primeros Unix su casa era la raíz misma.
- **Tu casa es** `/home/` **y tu nombre.** Ahí van tus cosas, y es donde aparece la terminal cuando la abres. Para no escribirla completa, la shell la abrevia con una tilde: `~`.
- `/bin` **ya no es una carpeta.** En el video la nombro como «los programas básicos», y es cierto que ahí los buscas; pero en un Ubuntu actual `/bin` es una flecha que apunta a `/usr/bin`, que es donde de verdad viven. Lo comprobamos en la sección 14.



En un servidor real. La carpeta que más vas a visitar es `/var/log`: ahí cada servicio deja su registro, renglón por renglón. `ls /var/log` te enseña cuáles hay; `less` te deja leer uno largo con calma (sales con `q`), y `tail -f` te va enseñando los renglones nuevos conforme llegan, hasta que lo paras con `Ctrl+C`. Cuando algo falla en un servidor, ahí empieza la búsqueda. Una advertencia: con un usuario normal, muchos registros no se dejan leer. Así le sale a Ana:

```
ana@linux:~$ ls -l /var/log/syslog
-rw-r----- 1 syslog adm 697275 Nov 14 21:12 /var/log/syslog
ana@linux:~$ tail -n 3 /var/log/syslog
tail: cannot open '/var/log/syslog' for reading: Permission denied
ana@linux:~$ journalctl -n 3
Hint: You are currently not seeing messages from other users and ↵
the system.
      Users in groups 'adm', 'systemd-journal' can see all messages.
      Pass -q to turn off this notice.
…
```

El archivo es del usuario `syslog` y del grupo `adm`, y los demás no tienen ni la `r` de leer: hace falta `sudo` o estar en el grupo `adm`. Quien se queja es el portero de siempre. `journalctl`, que lee el registro de systemd, no truena: avisa con un `Hint:` que solo te enseña lo tuyo (corté los renglones de abajo). Lo vemos a fondo en el capítulo 9, el de logs.

28. Linux Foundation (2015).
29. Linux man-pages project (s. f.-b).

12 Primer paseo por la terminal

Ya estuvo bueno de teoría. Abre la terminal.

Lo primero que ves es esto, esperándote:

```
ana@linux:~$
```

Hace años, esa línea fue mi pared. Hoy te la traduzco pieza por pieza, porque no es un adorno: te está dando información.

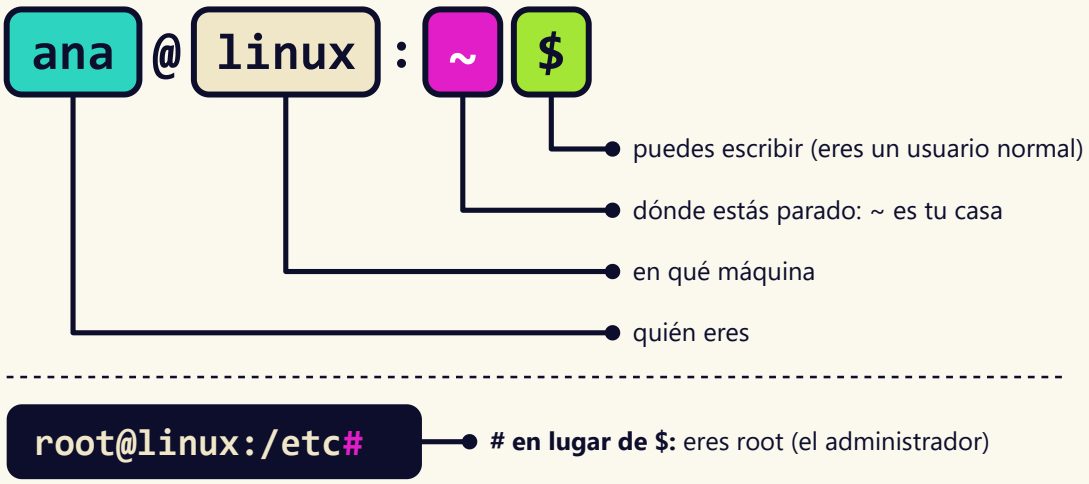


Figura 0.9. Se llama prompt y se lee: «soy Ana, en la máquina linux, parada en mi casa, lista para escribir». Si el último signo es un `#`, cuidado: eres el administrador.

Con cuatro comandos ya te mueves en cualquier Linux. (Si tu casa no se parece a la de Ana, el recuadro de *Antes de empezar* la deja igual.)

› ¿Dónde estoy?

```
ana@linux:~$ pwd
/home/ana
```

`pwd` viene de *print working directory*: «muestra la carpeta de trabajo». Te contesta con la ruta completa, desde la raíz.

› ¿Qué hay aquí?

```
ana@linux:~$ ls
Descargas
Documentos
Imágenes
```

`ls` viene de *list*: «enlista». Pero `ls` a secas no te lo enseña todo. Agrégale `-la` y mira:

```
ana@linux:~$ ls -la
total 40
drwxr-x--- 7 ana  ana  4096 Nov 14 20:43 .
drwxr-xr-x 4 root root 4096 Nov 14 20:35 ..
-rw-r--r-- 1 ana  ana   220 Mar 31  2024 .bash_logout
-rw-r--r-- 1 ana  ana  3771 Mar 31  2024 .bashrc
drwx------ 2 ana  ana  4096 Nov 14 20:36 .cache
drwxrwxr-x 2 ana  ana  4096 Nov 14 20:43 Descargas
drwxrwxr-x 2 ana  ana  4096 Nov 14 20:43 Documentos
drwxrwxr-x 2 ana  ana  4096 Nov 14 20:43 Imágenes
-rw-r--r-- 1 ana  ana   807 Mar 31  2024 .profile
drwx------ 2 ana  ana  4096 Nov 14 20:35 .ssh
```

Aparecieron cosas que antes no estaban, todas con un punto adelante. En Linux, **lo que empieza con punto está oculto**. No hay un atributo «oculto» como en Windows: es solo el punto. La `a` de `-la` significa *all*, «todo», y por eso ahora sí salen. La `l` significa *long*, «largo»: un renglón con detalles por cada cosa.

`.bashrc`, `.profile` y `.bash_logout` son la configuración de la shell, que Ubuntu le copia a cada usuario nuevo. `.cache` y `.ssh` las crean algunos programas (en `.ssh` van las llaves para entrar por SSH, del capítulo 11); en tu máquina puede haber otras, como `.config` o `.local`. Si ya usaste la terminal antes, verás también `.bash_history`, la lista de lo que has tecleado; te la explico en *Trucos y atajos*. Y fíjate en el orden: `ls` acomoda sin hacer caso del punto ni de las mayúsculas, por eso `.profile` sale después de `Imágenes` (con el idioma de Ana; en WSL o con otro idioma puede salir otro orden).

Lee una línea completa, la de `Documentos`:

PEDAZO	QUÉ DICE
`d`	Qué es. `d` es carpeta (*directory*); un guion sería un archivo normal. Hay más letras: sección 14
`rwxrwxr-x`	Quién puede hacer qué. Tiene su capítulo propio, el 2
`2`	Número de enlaces. Lo verás después; por ahora sáltalo
`ana ana`	De quién es, y a qué grupo pertenece
`4096`	En un archivo, cuánto ocupa en bytes. En una carpeta, lo que ocupa su lista de nombres, no lo que pesan los archivos de adentro
`Nov 14 20:43`	Cuándo se modificó por última vez
`Documentos`	Cómo se llama

Hasta arriba hay dos renglones raros: `.` y `..`. Un punto es «esta misma carpeta»; dos puntos, «la carpeta de arriba». Los vas a usar todo el tiempo. Fíjate de quién es cada una: `.`, la casa de Ana, es de `ana`; `..`, que aquí es `/home`, es de `root`. Tu casa es tuya, pero la carpeta que la contiene no.

› Moverme

```
ana@linux:~$ cd Documentos
ana@linux:~/Documentos$ pwd
/home/ana/Documentos
```

`cd` viene de *change directory*: «cambia de carpeta». Fíjate en el prompt: antes decía `~` y ahora dice `~/Documentos`. Te va diciendo dónde estás sin que le preguntes.



Truco. No escribas nombres completos. Teclea `cd Doc` y aprieta la tecla `Tab`: la shell termina la palabra por ti. Y si `Tab` no completa nada, ya te avisó de algo: ese nombre no existe desde donde estás, o hay varios nombres que empiezan igual. Hay más trucos así al final del capítulo.

› Leer un archivo

```
ana@linux:~/Documentos$ ls -l
total 8
-rw-rw-r-- 1 ana ana 38 Nov 14 20:43 nota.txt
-rw-rw-r-- 1 ana ana 24 Nov 14 20:43 pendientes.txt
ana@linux:~/Documentos$ cat nota.txt
Hola, Linux.
Esta es mi primera nota.
```

`cat` viene de *concatenate*, «juntar», porque sirve para pegar varios archivos uno tras otro. En la práctica casi todos lo usamos para lo más simple: ver qué dice un archivo de texto sin abrir ningún programa.

› Anatomía de un comando

Ya tecleaste varios comandos. Todos tienen la misma forma, y conocerla te desbloquea cualquier comando que leas por ahí, aunque nunca lo hayas visto.

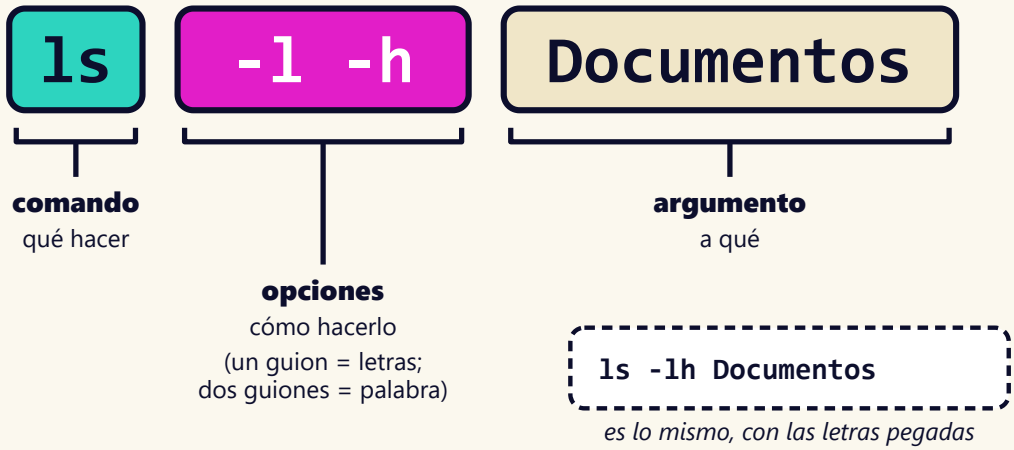


Figura 0.10. *Qué hacer, cómo hacerlo y a qué. En la mayoría de los comandos, las opciones de una letra se pueden pegar.*

- **El comando** va primero y dice *qué hacer*: `ls`, enlistar.
- **Las opciones** dicen *cómo hacerlo*. Con **un guion**, son de una sola letra, y en la mayoría de los comandos se pueden pegar: `-l -h` es lo mismo que `-lh`. No en todos: cuando uno no lo acepte, su ayuda te lo dirá. Con **dos guiones**, son una palabra completa: `--all` es lo mismo que `-a`. La `h` es de *human*: tamaños para humanos, en K y M en lugar de bytes.
- **Los argumentos** dicen *a qué*: aquí, a la carpeta `Documentos`.

Compruébalo: las dos escrituras dan lo mismo.

```
ana@linux:~$ ls -l -h Documentos
total 8.0K
-rw-rw-r-- 1 ana ana 38 Nov 14 20:43 nota.txt
-rw-rw-r-- 1 ana ana 24 Nov 14 20:43 pendientes.txt
ana@linux:~$ ls -lh Documentos
total 8.0K
-rw-rw-r-- 1 ana ana 38 Nov 14 20:43 nota.txt
-rw-rw-r-- 1 ana ana 24 Nov 14 20:43 pendientes.txt
```

Entre un comando y sus piezas va un espacio, y la shell usa los espacios para separar. Por eso un nombre con espacios da problemas, como vas a ver en los errores típicos.

› Juntar comandos

Aquí está lo que, cuando por fin lo entendí, me hizo querer a la terminal. Los comandos de Linux son herramientas chicas que hacen una sola cosa. La gracia está en conectarlas.

La barra vertical `|` se llama **tubería**. Toma lo que escupe el comando de la izquierda y se lo da de comer al de la derecha:

```
ana@linux:~$ ls -l Documentos | grep nota
-rw-rw-r-- 1 ana ana 38 Nov 14 20:43 nota.txt
```

`ls -l` enlista la carpeta; `grep` se queda solo con los renglones que traen la palabra «nota». De dos archivos, salió uno.

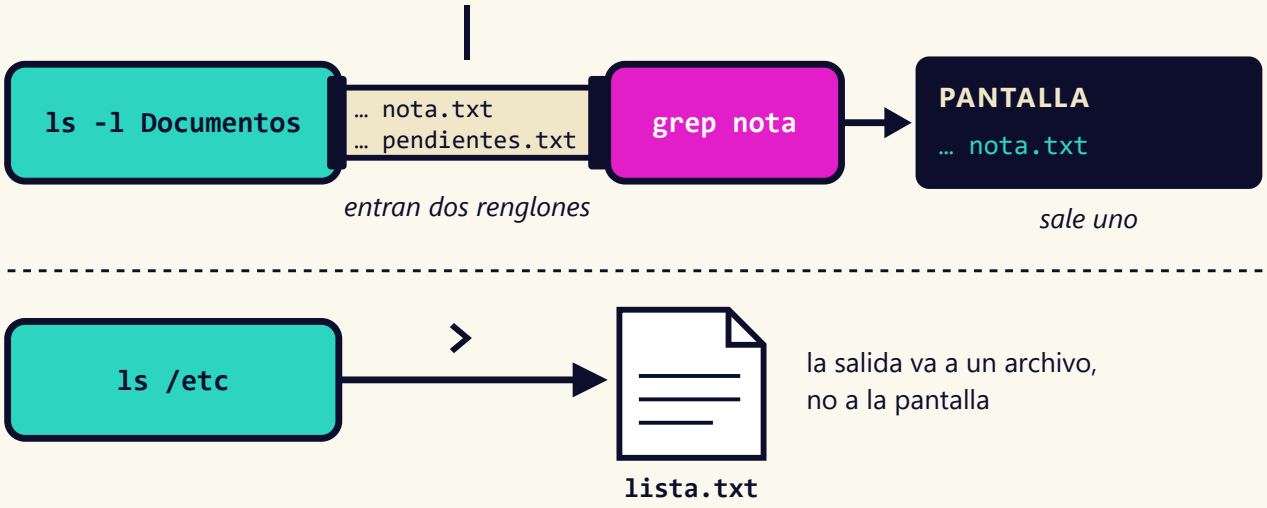


Figura 0.11. *La salida de uno es la entrada del otro. Con `>`, la salida no va a la pantalla: va a un archivo.*

Otra: ¿cuántas cosas hay en la carpeta de configuración del sistema? `ls` las enlista y `wc -l` cuenta renglones.

```
ana@linux:~$ ls /etc | wc -l
195
```

En esta máquina, 195. En la tuya va a salir otro número, y está bien. Un tercero que vas a ver seguido al final de una tubería es `head`: `head -n 5` deja pasar solo los primeros cinco renglones de lo que le llegue, y se calla el resto.

Y el signo `>` manda el resultado a un archivo, en lugar de a la pantalla:

```
ana@linux:~$ ls /etc > lista.txt
ana@linux:~$ wc -l lista.txt
195 lista.txt
```

El primer comando no contestó nada. No es que haya fallado: todo lo que iba a decir se fue al archivo. El segundo lo comprueba: `lista.txt` tiene los mismos 195 renglones. Ojo: si `lista.txt` ya existía, `>` lo reemplaza sin preguntar.



No te confundas. En Linux, que un comando no conteste nada casi siempre es buena señal. El sistema habla cuando hay algo que decir; si se queda callado, es que salió bien.

13 Rutas completas y rutas cortas

Una **ruta** es la lista de carpetas que hay que cruzar para llegar a algo. Hay dos maneras de darla, y la diferencia está en **el primer carácter**:

- Si empieza con `/`, es una ruta **completa** (en los libros y en internet la vas a encontrar como *absoluta*). Se camina desde la raíz, y sirve igual estés donde estés.
- Si no empieza con `/`, es una ruta **corta** (*relativa*). Se camina desde donde estás parado.

Así lo dice el manual: si la ruta empieza con `/`, la búsqueda arranca en la raíz; si no, en la carpeta de trabajo de quien la pide[30].

Mira a Ana probar las dos, primero en su casa y luego parada en otra carpeta, `/usr/share`:

```
ana@linux:~$ cat Documentos/nota.txt
Hola, Linux.
Esta es mi primera nota.
ana@linux:~$ cat /home/ana/Documentos/nota.txt
Hola, Linux.
Esta es mi primera nota.
ana@linux:~$ cd /usr/share
ana@linux:/usr/share$ cat Documentos/nota.txt
cat: Documentos/nota.txt: No such file or directory
ana@linux:/usr/share$ cat ~/Documentos/nota.txt
Hola, Linux.
Esta es mi primera nota.
ana@linux:/usr/share$ cat /home/ana/Documentos/nota.txt
Hola, Linux.
Esta es mi primera nota.
```

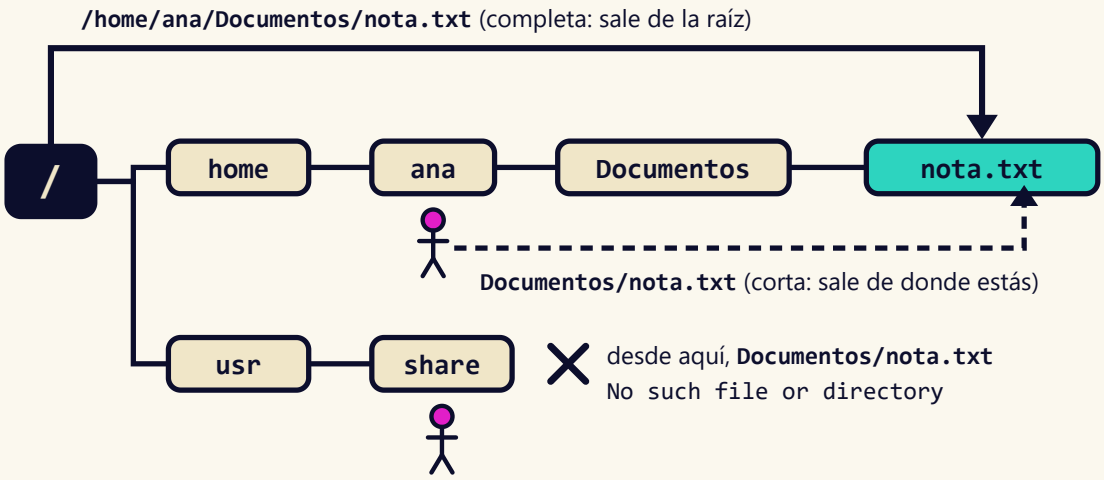


Figura 0.12. *La completa sale de la raíz y llega desde donde sea. La corta sale de donde estás, y desde otro lado no llega.*

El mismo archivo y la misma orden corta: en casa funciona; parada en `/usr/share`, no, porque ahí no hay ninguna carpeta `Documentos`. La completa funciona en los dos lados. Y `~/Documentos/nota.txt` es una completa disfrazada: antes de ejecutar, la shell cambia la tilde por `/home/ana`[31].

Cada carpeta trae además dos nombres de fábrica: `.`, ella misma, y `..`, la de arriba[30]; con ellos se arman rutas cortas que suben, como verás en los trucos. Y si alguna vez dudas a dónde apunta en realidad una ruta corta, `realpath` te la traduce a completa:

```
ana@linux:~$ realpath Documentos/nota.txt
/home/ana/Documentos/nota.txt
```



No te confundas. «El archivo no existe», cuando sí existe pero tu ruta es corta y estás en otro lado. Es la causa número uno de `No such file or directory`. Tres pasos: `pwd` para saber dónde estás; ¿mi ruta empieza con `/`?; si no, ¿existe *desde aquí*?

30. Linux man-pages project (s. f.-e).
31. Fox y Ramey (s. f.).

14 Todo es un archivo, comprobado

En la sección 7 viste la `c` de `/dev/null`. La primera letra de cada renglón de `ls -l` dice qué clase de cosa es. Mira estos dos:

```
ana@linux:~$ ls -l /dev/null /bin
lrwxrwxrwx 1 root root    7 Apr 22  2024 /bin -> usr/bin
crw-rw-rw- 1 root root 1, 3 Nov 14 20:43 /dev/null
```

Ahí está la prueba de lo que te dije de `/bin`: empieza con `l` y tiene una flecha, `/bin -> usr/bin`. Es un **enlace simbólico**: un nombre que apunta a otro. Con lo que ya viste, tienes las cuatro letras que más vas a encontrar:

LETRA	TIPO	QUÉ ES, EN UNA LÍNEA	EJEMPLO
`-`	archivo normal	datos guardados: texto, foto, programa	`nota.txt`
`d`	carpeta (*directory*)	una lista de nombres	`Documentos`
`l`	enlace simbólico (*link*)	una flecha hacia otro nombre	`/bin`
`c`	dispositivo de caracteres	entrega o recibe datos en chorrito, uno tras otro	`/dev/null`

Del lado del kernel hay siete tipos en total[32]. Si te pica la curiosidad, en *Para los curiosos* está la tabla de los siete, con los tres raros creados a mano.

Ahora, la comprobación que vale oro. El comando `file` no se fija en el nombre: mira **adentro** del archivo y dice qué es. Ana copia el programa `ls` a un archivo con nombre de texto, y le pregunta a `file`:

```
ana@linux:~$ cp /usr/bin/ls programa.txt
ana@linux:~$ file programa.txt
programa.txt: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically ↵
linked, interpreter /lib64/ld-linux-x86-64.so.2, ↵
BuildID[sha1]=44900829b39271d878615772bb2a6ffa422f0891, for GNU/Linux 3.2.0, stripped
```

El renglón es largo, pero lo que importa son sus primeras palabras: `ELF 64-bit LSB pie executable`, un programa ejecutable de Linux. Se llama `programa.txt` y sigue siendo un programa. **Para el kernel y la shell, la terminación del nombre no decide nada**: es una cortesía para los humanos. Ojo, no para todos los programas: algunos sí la exigen por convención (los servicios de systemd terminan en `.service`, muchas configuraciones en `.conf`, las listas de `apt` en `.list`), y el escritorio la usa para elegir con qué programa abrir un archivo cuando

le das doble clic. Pero para saber qué *es* un archivo, la terminación no sirve. Para alguien que viene de Windows, donde el `.exe` lo es todo, esto es importante, y para alguien que se dedica a la seguridad, más: un nombre puede mentir; lo de adentro miente mucho menos.



Truco. `ls -F` marca el tipo al final de cada nombre: `/` carpeta, `@` enlace y `*` ejecutable, entre otros[33]. Es una forma rápida de ver qué es cada cosa sin el renglón largo de `-l`.

32. Linux man-pages project (s. f.-d).
33. Free Software Foundation (s. f.-a).

15 El dueño de todo

Queda una pieza, y es de las importantes. En todo Linux hay un usuario que puede absolutamente todo. Se llama **root** y su casa es esa carpeta `/root` del mapa. Ana, que es una usuaria normal, intenta asomarse:

```
ana@linux:~$ ls /root
ls: cannot open directory '/root': Permission denied
```

«Permiso denegado». No es un error del sistema: es el portero de la sección 2 haciendo su trabajo. Y te puedo enseñar por qué, con algo que ya sabes leer:

```
ana@linux:~$ ls -ld /root
drwx------ 3 root root 4096 Sep  9 06:37 /root
```

La `d` dice que es carpeta (la opción `-d` le pide a `ls` que hable de la carpeta misma y no de lo que tiene adentro). El dueño es `root`. Y en la parte de los permisos, después de `rwx` vienen puros guiones: nadie más puede ni mirar.

Como root puede borrar el sistema entero sin que nadie le pregunte, la regla de oro es **no andar como root**. Se vive como usuario normal y, cuando un comando de verdad necesita ese poder, se le pide solo para ese comando, poniéndole adelante la palabra `sudo`. El sistema te pide tu contraseña, ejecuta ese comando como root y vuelves a ser tú: es un cambio de identidad temporal, solo para ese comando. Dos detalles. `sudo` recuerda tu contraseña unos minutos (en la 24.04, quince[34]), así que el segundo `sudo` seguido ya no te la pide. Y solo funciona, normalmente, si tu usuario está en el grupo `sudo`, el de los administradores; el usuario que creaste al instalar Ubuntu ya está. Si alguna vez tu prompt termina en `#` en lugar de `$`, como en la Figura 0.9, estás de root: haz lo que tengas que hacer y sal.

`sudo` lo vas a usar en serio cuando instalemos programas, en el capítulo 5. Por hoy basta con que sepas que existe y cómo funciona. Mientras no lo uses, Ubuntu te lo recuerda cada vez que abres la terminal, con un aviso en inglés que empieza `To run a command as administrator`; deja de salir en cuanto usas `sudo` por primera vez con éxito.



No te confundas. Cuando `sudo` te pide la contraseña, la que escribes es **la tuya**, no una «contraseña de root». En la 24.04, mientras la escribes, la pantalla no enseña nada: ni letras ni asteriscos. No está trabada; es a propósito. En la 26.04 cambió: su `sudo` nuevo, llamado sudo-rs, enseña un asterisco por cada letra. Las notas de la 26.04 confirman el cambio a sudo-rs[35], y el registro de cambios de sudo-rs dice que desde su versión 0.2.13 los asteriscos vienen activados de fábrica[36].



En un servidor real. En un Ubuntu, el usuario root normalmente no tiene contraseña con la que se pueda entrar: todo lo de administrador se hace con `sudo` desde tu propio usuario. Eso tiene una ventaja: cada uso de `sudo` queda anotado en los registros del sistema, con quién, cuándo y qué comando. Cuando en un servidor algo cambió y nadie sabe quién fue, ahí se busca. Lo vemos en los capítulos 9 y 10, el de logs y el de usuarios.

34. Sudo project (s. f.).
35. Canonical (s. f.-b).
36. Trifecta Tech Foundation (2026).

16 Cómo pedir ayuda sin salir de la terminal

Esta sección es la que te vuelve independiente de este libro. Tiene una trampa que desanima a todo principiante, y vamos a desactivarla.

Primero se pregunta **qué clase de comando es**, con el `type` de la sección 10, porque de eso depende a quién pedirle ayuda[37]:

SI `TYPE` DICE…	LA AYUDA ESTÁ EN…
`… is /usr/bin/…` o `… is hashed …` (un programa)	`comando --help` (resumen) y `man comando` (manual completo)
`… is a shell builtin`	`help comando`, aunque también exista un programa con ese nombre
`… is aliased to …`	Es un apodo: pregunta por el comando de verdad

Así se ve en la práctica:

```
ana@linux:~$ ls --help | head -n 6
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
ana@linux:~$ man cd
No manual entry for cd
ana@linux:~$ help cd | head -n 5
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable. If DIR is "-", it is converted to $OLDPWD.
```

La ayuda de `ls` es larga, así que Ana se queda con los primeros seis renglones con una tubería y `head -n 6`. Ahí está la trampa: `man cd` dice que no hay manual. Y es verdad, porque `cd` no es un programa, es parte de la shell. Su ayuda se le pide a la shell, con `help`.

Y una trampa más fina. `echo` y `pwd` son builtins *y además* programas, así que `man echo` sí abre algo, pero no lo que crees: el manual del programa `/usr/bin/echo`, no el del builtin que de verdad usas al teclear `echo`. Las opciones pueden no coincidir. `man -w` te dice qué archivo abriría, y es el del programa:

```
ana@linux:~$ man -w echo
/usr/share/man/man1/echo.1.gz
```

La regla corta: si `type` dice *builtin*, manda `help`.

Cómo leer el renglón `Usage:`, que vale para todos los comandos: lo que va entre corchetes `[ ]` es opcional; los tres puntos `...` quieren decir «puedes poner varios»; lo que va en MAYÚSCULAS es «aquí va lo tuyo». `ls [OPTION]... [FILE]...` se lee: «`ls`, si quieres con opciones, si quieres con nombres de archivo».

› El manual, por secciones

`man` abre el manual completo de un programa en un lector de texto. Ahí las teclas no escriben, mandan: con las flechas o la barra espaciadora avanzas, con `/palabra` buscas, con `n` vas a la siguiente aparición y con `q` **sales**. El manual está partido en **secciones** numeradas, como los tomos de una enciclopedia[38]. Las tres que más vas a usar:

- **Sección 1:** programas y comandos de la shell. Es la de `ls`.

- **Sección 5:** formatos de archivo y convenciones, como el de `/etc/passwd`.
- **Sección 8:** comandos de administración, casi siempre para root.

Las nueve están en *Para los curiosos*. Por eso a veces verás escrito `ls(1)` o `passwd(5)`: el número es la sección. Un mismo nombre puede estar en varias, y `man -f` te lo dice:

```
ana@linux:~$ man -f passwd
passwd (1)           - change user password
passwd (1ssl)        - OpenSSL application commands
passwd (5)           - the password file
ana@linux:~$ man -k "list directory"
dir (1)              - list directory contents
ls (1)               - list directory contents
ntfsls (8)           - list directory contents on an NTFS filesystem
vdir (1)             - list directory contents
```

`passwd` es un programa, el que cambia tu contraseña (sección 1), y también es un archivo de configuración (sección 5). `man passwd` abre el primero; `man 5 passwd` abre el otro. Y `man -k` busca en los resúmenes de todos los manuales una palabra: es lo que usas cuando **no sabes cómo se llama** el comando que necesitas[38].

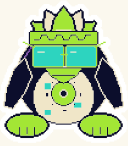


Truco. Si la pantalla «se quedó trabada» mostrando un texto largo con dos puntos o `(END)` hasta abajo, no está trabada: estás dentro del lector del manual. Sal con `q`.



No te confundas. «No hay manual de `cd`, entonces no hay ayuda». Sí la hay: `cd` es un builtin, y se le pide a `help`.

37. Fox y Ramey (s. f.).
38. man-db project (s. f.).



Trucos y atajos

Lo que sigue no viene en el video. Son las mañas para moverte rápido y no cansarte. Pruébalos todos; en una semana los vas a usar sin pensar.

› Teclas que ahorran tiempo

- `Tab` termina la palabra: `cd Doc` y `Tab` se vuelve `cd Documentos/`. Según cómo empiece lo que escribiste, Bash prueba con variables, usuarios, comandos o nombres de archivo[39]. Además es un detector de errores: si no completa, ese nombre no existe desde donde estás, o hay varios

que empiezan igual; un segundo `Tab` te enseña la lista.

- `↑` y `↓` caminan por lo que ya tecleaste. Sirven para repetir y, sobre todo, para corregir un error de dedo sin escribir todo de nuevo: `↑`, arreglas la letra, `Enter`.
- `Ctrl+L` limpia la pantalla y vuelve a pintar arriba el renglón que estabas escribiendo, sin perderlo[39]. `clear` se parece, pero es un programa aparte (`type clear` te dice `clear is /usr/bin/clear`) y además borra el historial de desplazamiento, lo que podías ver subiendo con la rueda del ratón; `Ctrl+L` no lo borra. Ninguno toca tus archivos: solo despejan la vista.
- `Ctrl+C` abandona el renglón que estás escribiendo, o detiene un comando que no termina. `Ctrl+D`, en un renglón vacío, cierra la shell, igual que teclear `exit`: le avisa que ya no hay nada más que leer.

› cd -, cd a secas y la tilde

Tres maneras de no perderse. El guion solo, como destino de `cd`, quiere decir «la carpeta donde estaba antes»: Bash la guarda en una variable llamada `OLDPWD` y te imprime a dónde llegaste. `cd` sin destino te lleva a la carpeta guardada en la variable `HOME`, que es tu casa. Y la tilde `~` la **reemplaza** la shell por tu casa antes de ejecutar; el comando nunca la ve, como se comprueba con `echo`, que solo repite lo que le llega[39]:

```
ana@linux:~$ cd Documentos
ana@linux:~/Documentos$ cd /etc
ana@linux:/etc$ cd -
/home/ana/Documentos
ana@linux:~/Documentos$ cd -
/etc
ana@linux:/etc$ cd
ana@linux:~$ pwd
/home/ana
```

La tilde, vista con `echo`:

```
ana@linux:~$ echo ~
/home/ana
ana@linux:~$ echo ~/Documentos
/home/ana/Documentos
ana@linux:~$ cd /var/log
ana@linux:/var/log$ ls ~
Descargas
Documentos
Imágenes
lista.txt
programa.txt
```

Dos `cd -` seguidos te columpian entre dos carpetas: es el botón «atrás», pero de un solo paso. `cd` a secas es el «llévame a casa» para cuando te perdiste; `cd`, `cd ~` y `cd /home/ana` hacen lo mismo. Y con la tilde nombras algo de tu casa estando en cualquier otro lado, como hizo Ana desde `/var/log`. (`lista.txt` y `programa.txt` son los que creó en las secciones 12 y 14.)

› Punto y dos puntos: aquí y arriba

`.` es la carpeta actual y `..` la que la contiene. No son atajos de la shell: son dos entradas que vienen dentro de toda carpeta, y por eso `ls -a` las enseña. Con ellas se sube y se baja:

```
ana@linux:~$ cd Documentos
ana@linux:~/Documentos$ cd ../Descargas
ana@linux:~/Descargas$ pwd
/home/ana/Descargas
ana@linux:~/Descargas$ cd ../..
ana@linux:/home$ pwd
/home
```

`cd ..` sube uno, `cd ../..` sube dos, y `cd ../Descargas` sube uno y baja a la carpeta hermana. El punto solo parece inútil hasta que hace falta decir «aquí mismo»: `cp /etc/hostname .` copia un archivo *aquí*.

› history y !número: lo que ya tecleaste

`history` imprime la lista numerada de lo que has tecleado. El signo `!` seguido de un número quiere decir «el comando con ese número»; Bash primero te enseña cuál va a repetir y luego lo ejecuta[39]. Ana empezó con el historial vacío, por eso su lista arranca en 1:

```
ana@linux:~$ history
    1  pwd
    2  ls
    3  cd Documentos
    4  cat nota.txt
    5  cd
    6  history
ana@linux:~$ history 3
    5  cd
    6  history
    7  history 3
ana@linux:~$ !2
ls
Descargas
Documentos
Imágenes
lista.txt
programa.txt
```

En tu máquina los números van a ser otros: dependen de todo lo que hayas tecleado antes, incluso en sesiones anteriores. Por eso primero se mira el número con `history` y luego se usa; nunca copies un `!N` a ciegas, ni de este libro ni de internet, porque en tu lista ese número puede ser cualquier otro comando.

Al cerrar la terminal, la lista se guarda en un archivo oculto de tu casa, `.bash_history`, y la siguiente sesión sigue la cuenta desde ahí. Por eso, nunca escribas una contraseña como parte de un comando: el historial guarda **todo**.

› Un guion, dos guiones y dos guiones solos

Ya viste que **un guion** son opciones de una letra y **dos guiones**, una opción con nombre completo. Falta el caso raro: **dos guiones solos**, `--`, quieren decir «aquí se acabaron las opciones; lo que sigue son nombres». Ana crea un archivo cuyo nombre empieza con guion y luego intenta borrarlo. Usa dos comandos que vemos en el capítulo 1: `touch`, que crea un archivo vacío, y `rm`, que borra. Ojo con `rm`: borra sin papelera y sin preguntar. Aquí el primer intento falla sin borrar nada:

```
ana@linux:~$ touch -- -raro.txt
ana@linux:~$ rm -raro.txt
rm: invalid option -- 'a'
Try 'rm ./-raro.txt' to remove the file '-raro.txt'.
Try 'rm --help' for more information.
ana@linux:~$ rm -- -raro.txt
```

`rm` creyó que `-raro.txt` era un montón de opciones de una letra (`-r`, `-a`, `-r`, `-o`...) y tronó en la `a`, que no conoce. El propio mensaje te da otra salida: `./-raro.txt`, una ruta corta que ya no empieza con guion. Y con `--` delante, `rm` entendió que era un nombre y lo borró sin decir nada.

---

39. Fox y Ramey (s. f.).



Errores típicos

Los primeros nueve mensajes salieron de verdad en la máquina de Ana; los últimos cinco son de los que te vas a topar más adelante, y te digo en qué capítulo los vemos. (El de `sudo` también lo saqué en su máquina, con una shell sin el ayudante de Ubuntu y sin la carpeta de `sudo` en el `PATH`.) Fíjate en que casi todos tienen la misma forma: *quién se queja* `:` *sobre qué* `:` *qué pasó*. `cat: nota.txt: No such file or directory` se lee «habla `cat`, sobre `nota.txt`: no existe».

ME SALIÓ ESTE MENSAJE	QUÉ SIGNIFICA	QUÉ REVISAR
`Command 'lss' not found, but there are 16 similar ones.`	No hay comando con ese nombre; Ubuntu además avisa que hay parecidos	Error de dedo: `↑`, corrige, `Enter`

ME SALIÓ ESTE MENSAJE	QUÉ SIGNIFICA	QUÉ REVISAR
`Ls: command not found`	Lo mismo: `Ls` y `ls` son nombres distintos. Sale sin `bash:` porque lo imprime el ayudante de Ubuntu que busca comandos parecidos, no Bash	Las mayúsculas
`cat: nota.txt: No such file or directory`	Desde donde estás no hay nada con ese nombre	`pwd` y `ls`; ¿ruta corta desde la carpeta equivocada? (sección 13)
`cat: Documentos: Is a directory`	Le pediste leer una carpeta como si fuera archivo	Para ver una carpeta es `ls`, no `cat`
`bash: cd: Documentos/nota.txt: Not a directory`	Al revés: quisiste entrar a un archivo	Para leer un archivo es `cat`, no `cd`
`ls: cannot open directory '/root': Permission denied`	Existe, pero no tienes permiso. El portero hace su trabajo	No se «arregla», se entiende: `ls -ld /root` te dice por qué
`ls: invalid option -- 'z'`	Esa opción no existe para ese comando	El renglón siguiente te dice dónde mirar: `ls --help`
`bash: cd: too many arguments`	El espacio separa palabras: con `cd Mis Documentos`, `cd` recibió dos destinos	Los nombres con espacios van entre comillas (capítulo 1)
`No manual entry for cd`	Ese comando no tiene manual porque no es un programa	`type cd`; si es builtin, `help cd`
`E: Could not get lock …`	Otro programa está instalando o actualizando, a veces la actualización automática, y tiene apartado el candado	Espera a que termine; no borres el candado a mano. Capítulo 5
`No space left on device`	El disco, o esa parte del árbol, se llenó	Cuánto espacio queda y quién se lo comió. Capítulo 8
`Read-only file system`	Esa parte del árbol está montada solo para leer; a veces el sistema la pone así solo al detectar errores en el disco	Si fue de repente, sospecha del disco. Capítulo 8
`Permission denied (publickey)`	Al entrar por SSH, el servidor solo acepta llaves y no reconoció la tuya	El usuario con el que entras y tu llave. Capítulo 11
`bash: sudo: command not found`	En esa máquina no está instalado `sudo`; pasa en contenedores e instalaciones mínimas. Ahí no está el ayudante de Ubuntu, así que el mensaje lo firma Bash	Pregunta a quien la administra cómo se hace ahí lo de root. Capítulo 10

El primero es propio de Ubuntu: en otras distros sale solo `bash: lss: command not found`. Y si tu sistema está en español, varios te van a salir traducidos; la forma, *quién* `:` *sobre qué* `:` *qué pasó*, es la misma.



TU TURNO

Los ejercicios con **(leer)** no son de teclear: son de interpretar una salida, que es lo que más vas a hacer en la vida real. Los **retos** son casos como los que te van a llegar en un trabajo. Hazlos sin ver las respuestas, que están en la página siguiente.

1. Abre la terminal y averigua en qué carpeta estás parado.
2. Cuenta cuántas cosas hay colgando de la raíz de tu sistema, con una tubería.
3. **(Leer)** ¿Cuál de estos tres es un programa guardado en el disco? ¿Cuál tiene página de `man`? ¿Cómo pides ayuda de los otros dos?

```
ana@linux:~$ type cd
cd is a shell builtin
ana@linux:~$ type cat
cat is hashed (/usr/bin/cat)
ana@linux:~$ type ll
ll is aliased to `ls -la'
```

4. **(Leer)** Fíjate solo en la primera letra de cada renglón. ¿Qué es cada cosa?

```
ana@linux:~$ ls -l /dev/null /bin
lrwxrwxrwx 1 root root    7 Apr 22  2024 /bin -> usr/bin
crw-rw-rw- 1 root root 1, 3 Nov 14 20:43 /dev/null
```

5. **(Leer)** Ana está en `/usr/share` y le sale esto. ¿Qué pasó? Da dos formas de arreglarlo sin cambiarse de carpeta.

```
ana@linux:/usr/share$ cat Documentos/nota.txt
cat: Documentos/nota.txt: No such file or directory
```

6. **(Leer)** ¿Qué versión de Ubuntu es? ¿De qué mes y año? ¿Es de soporte largo?

```
ana@linux:~$ lsb_release -a
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.5 LTS
Release:        24.04
Codename:       noble
```

7. **(Leer)** Ana tecleó esto. Sin ejecutarlo: ¿en qué carpeta termina?, ¿y si teclea `cd -` una vez más?

```
ana@linux:~$ cd Documentos
ana@linux:~/Documentos$ cd /etc
ana@linux:/etc$ cd -
```

8. **(Leer)** Tres mensajes. Para cada uno: ¿quién se queja y qué hizo mal Ana?

```
cat: Documentos: Is a directory
bash: cd: Documentos/nota.txt: Not a directory
bash: cd: too many arguments
```

9. **(Leer)** Este prompt: `luis@oficina:/etc$`. ¿Quién es, en qué máquina está y en qué carpeta? ¿Es administrador?

10. Sin escribir `/home/` ni tu nombre: parado en `/etc`, enlista lo que hay en tu casa; luego ve a tu casa con el comando más corto posible.

11. Averigua sin internet: (a) qué hace la opción `-A` de `ls` (pista: en la ayuda, busca la palabra *almost*); (b) en qué sección del manual está `passwd`, el archivo; (c) qué clase de comando es `history` y dónde está su ayuda.

12. **Reto (leer).** Te llega este ticket de Luis: «No me deja entrar a mis reportes y luego me dice que es un directorio. ¿Se dañó la carpeta?». Pegó esto de su terminal (es un ejemplo armado para el ejercicio). ¿Quién se queja en cada error, qué pasó de verdad y qué le dices que revise primero?

```
luis@oficina:~$ cd Reportes
bash: cd: Reportes: No such file or directory
luis@oficina:~$ ls
reportes
luis@oficina:~$ cat reportes
cat: reportes: Is a directory
```

13. **Reto (leer).** Ana escribió su primer programa propio, `saludo.sh`, en su carpeta `trabajo`, y ya le dio permiso de ejecutarse. Con `./saludo.sh` funciona, pero con el puro nombre no. ¿Por qué? ¿De qué dos formas lo puede correr?

```
ana@linux:~/trabajo$ ./saludo.sh
Hola Linux
ana@linux:~/trabajo$ saludo.sh
saludo.sh: command not found
ana@linux:~/trabajo$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:↵
/usr/games:/usr/local/games:/snap/bin
```

14. **Reto (leer).** A Ana le llegaron por correo dos archivos, una factura y un recibo. Antes de abrirlos, les pregunta a `file`. ¿Cuál es de verdad un PDF? ¿Qué harías con el otro?

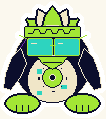
```
ana@linux:~/Descargas$ file factura.pdf
factura.pdf: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), ↵
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, ↵
BuildID[sha1]=44900829b39271d878615772bb2a6ffa422f0891, for GNU/Linux ↵
3.2.0, stripped
ana@linux:~/Descargas$ file recibo.pdf
recibo.pdf: PDF document, version 1.4, 1 page(s)
```

15. **Reto (leer).** Te toca revisar el servidor de la oficina después de un reinicio. Esto es lo que ves (es un ejemplo armado, con el formato real de `systemctl` de la 24.04). ¿Arrancó todo? Si no, ¿qué falló, y con qué dos comandos lo revisas?

```
luis@oficina:~$ systemctl status | head -n 5
● oficina
    State: degraded
    Units: 432 loaded (incl. loaded aliases)
     Jobs: 0 queued
   Failed: 1 units
luis@oficina:~$ systemctl --failed
  UNIT          LOAD   ACTIVE SUB    DESCRIPTION
● nginx.service loaded failed failed A high performance web server and ↵
a reverse proxy server

Legend: LOAD   → Reflects whether the unit definition was properly ↵
loaded.
        ACTIVE → The high-level unit activation state, i.e. ↵
generalization of SUB.
        SUB    → The low-level unit activation state, values depend on ↵
unit type.

1 loaded units listed.
```



Respuestas

1. `pwd`. Lo normal es que conteste `/home/` y tu nombre.
2. `ls / | wc -l`. La tubería le pasa la lista a `wc -l`, que cuenta renglones. A Ana le salen 26; el tuyo puede variar un poco según tu instalación.
3. Solo `cat` es un archivo en el disco (`/usr/bin/cat`), y es el que tiene `man cat`. `cd` es parte de la shell: su ayuda es `help cd`. `ll` es un apodo de `ls -la`: no tiene ayuda propia; se la pides a `ls`.
4. `l`: `/bin` es un enlace, una flecha hacia `usr/bin`. `c`: `/dev/null` es un dispositivo de caracteres, el hoyo.
5. La ruta es corta (no empieza con `/`), así que se busca desde `/usr/share`, y ahí no hay ninguna carpeta `Documentos`. Arreglos: la ruta completa, `cat /home/ana/Documentos/nota.txt`, o con tilde, `cat ~/Documentos/nota.txt`.
6. Es Ubuntu 24.04, de abril de 2024, en su quinta puesta al día (`.5`). Sí es de soporte largo: lo dice `LTS`. Su apodo corto es `noble`.
7. En `/home/ana/Documentos`, porque `cd -` regresa a la anterior; y al llegar, lo imprime. Un segundo `cd -` la devuelve a `/etc`.
8. (a) Se queja `cat`: le dieron una carpeta; para verla era `ls Documentos`. (b) Se queja Bash, por su builtin `cd`: Ana quiso entrar a un archivo; para leerlo era `cat`. (c) Bash, por `cd` otra vez: le dio dos palabras como destino, seguramente un nombre con espacio sin comillas.
9. Es **luis**, en la máquina **oficina**, parado en `/etc`, la carpeta de la configuración. Como termina en `$`, es un usuario normal; si fuera administrador, terminaría en `#`.
10. `ls ~` y luego `cd`, a secas.
11. (a) `ls --help` (o `man ls`): `-A, --almost-all`, lo mismo que `-a` pero sin `.` ni `..`. (b) `man -f passwd`: el archivo está en la sección 5, «the password file». (c) `type history` dice que es un *shell builtin*; su ayuda es `help history`.
12. Nada se dañó. El primer error lo firma Bash, por su builtin `cd`: no existe `Reportes` con mayúscula. El `ls` enseña que la carpeta se llama `reportes`, en minúsculas. El segundo lo firma `cat`: le pidió leer una carpeta como si fuera archivo. Lo primero que le dices que revise son las mayúsculas: `cd reportes` y luego `ls` para ver qué hay adentro.
13. Sin diagonal, la shell busca el nombre en las carpetas del `PATH`, y `/home/ana/trabajo` no está en esa lista; la carpeta donde estás parado no se revisa sola. Con `./saludo.sh` el nombre ya trae diagonal, así que la shell no busca: va directo a ese archivo, aquí mismo. Las dos formas: `./saludo.sh` estando en `trabajo`, o la ruta completa, `~/trabajo/saludo.sh`, desde donde sea. (El mensaje sale sin `bash:` por el ayudante de Ubuntu, como el de `Ls`.)

14. `recibo.pdf` sí es un PDF: `file` vio adentro un documento PDF, versión 1.4, de una página. `factura.pdf` no: por dentro es un programa ejecutable de Linux (`ELF … executable`) con nombre de PDF. No se abre ni se ejecuta; se borra o se le pasa a quien atienda la seguridad. El nombre miente; lo de adentro, mucho menos.

15. No arrancó todo: `State: degraded` y `Failed: 1 units` dicen que algo falló, y `systemctl --failed` dice qué: `nginx.service`, el servidor web, con `ACTIVE` y `SUB` en `failed`. (La `Legend:` de abajo solo explica las columnas.) Lo primero es `systemctl status nginx.service`, que enseña su estado y sus últimos renglones de registro; para leer más atrás, `journalctl -u nginx.service`. Con un usuario normal, el registro puede salir incompleto, como viste en el recuadro de `/var/log`: ahí se usa `sudo`. Los dos comandos los vemos con calma en los capítulos 4 y 9.

Lo que te llevas

COMANDO	VIENE DE	PARA QUÉ
`pwd`	*print working directory*	Saber dónde estás
`whoami`	*who am I*	Saber con qué usuario entraste
`ls`	*list*	Ver qué hay
`ls -la`	*long*, *all*	Verlo con detalle y con lo oculto
`ls -lh`	*long*, *human*	Con tamaños para humanos
`cd`	*change directory*	Moverte de carpeta (a secas: a casa; con `-`: a la anterior)
`cat`	*concatenate*	Leer un archivo de texto
`file`		Saber qué es un archivo, mirando adentro
`realpath`	*real path*	Traducir una ruta corta a completa
`grep`		Quedarte con los renglones que traen una palabra
`wc -l`	*word count*, *lines*	Contar renglones
`head -n 5`		Quedarte con los primeros renglones
`type`		Saber qué clase de comando es
`help`, `--help`, `man`	*manual*	Pedir ayuda, según la clase de comando
`history`		Ver lo que ya tecleaste
`|`		Pasarle el resultado a otro comando
`>`		Guardar el resultado en un archivo
`sudo`	*superuser do*	Ejecutar un solo comando como administrador

Y cinco ideas, que valen más que los comandos:

- Entre tú y el fierro hay un sistema operativo; su corazón es el kernel, y los programas le piden todo por una ventanilla.
- Linux es libre: se puede usar, estudiar, modificar y compartir, y la GPL cuida que siga así.
- Todo cuelga de una sola raíz, `/`, y una ruta que empieza con `/` llega desde cualquier lado.
- Todo es un archivo, hasta los dispositivos, y lo que dice qué es un archivo es su contenido, no su nombre.
- La terminal casi siempre te dice qué pasó, y quién lo dice. Hay que leerla.

Para los curiosos

Aquí dejé lo que no hace falta para seguir, pero que vale la pena si te quedaste con ganas. Nada de esto se pregunta en *Tu turno*.

› La GPL por dentro

Las cuatro libertades de la sección 6 son permisos. La GPL les agrega **una condición** para que no se pierdan en el camino; eso es el *copyleft*:

- **Si solo usas o modificas el programa para ti, no te obliga a nada:** puedes hacer versiones modificadas y usarlas en privado[40].
- **Si lo repartes, igual o modificado, tienes que repartir también el código fuente**, o una oferta por escrito de entregarlo[41].
- **Y lo que repartas va con la misma licencia:** la obra que contenga o derive del programa se licencia completa bajo los mismos términos[42]. No se puede tomar Linux, mejorarlo y vender el resultado cerrado.
- **Sí se puede vender**[40]; quien compra recibe las mismas cuatro libertades.

El kernel aclara una frontera en su propio archivo de licencia: los programas que solo *usan* al kernel a través de las llamadas al sistema no se consideran obras derivadas de él[43]. Dicho en limpio: un programa que pasa sus formas por la ventanilla de la sección 3 **no** queda obligado a ser libre. Por eso sobre Linux corren también programas cerrados.

Una línea honesta sobre este mismo libro: su licencia Creative Commons te deja copiarlo, compartirlo y adaptarlo, dando crédito y compartiéndolo igual. Se parece al copyleft en ese «compartir igual», pero no es libre en el sentido de la sección 6, porque no permite el uso comercial. Los detalles, en la página de créditos.

› Los siete tipos de archivo

En la sección 14 viste cuatro letras. Para ver las otras, Ana crea a propósito un tubo con nombre (`mkfifo`) y un enlace simbólico (`ln -s`), y enlista. `lista.txt` y `programa.txt` siguen ahí desde las secciones 12 y 14; son los archivos normales, los del guion:

```
ana@linux:~$ mkfifo tubo
ana@linux:~$ ln -s Documentos/nota.txt acceso
ana@linux:~$ ls -l
total 156
lrwxrwxrwx 1 ana ana     19 Nov 14 20:43 acceso -> Documentos/nota.txt
drwxrwxr-x 2 ana ana   4096 Nov 14 20:43 Descargas
drwxrwxr-x 2 ana ana   4096 Nov 14 20:43 Documentos
drwxrwxr-x 2 ana ana   4096 Nov 14 20:43 Imágenes
-rw-rw-r-- 1 ana ana   1910 Nov 14 20:43 lista.txt
-rwxr-xr-x 1 ana ana 142312 Nov 14 20:43 programa.txt
prw-rw-r-- 1 ana ana      0 Nov 14 20:43 tubo
```

Y en `/dev` busca los dispositivos de bloques, los que empiezan con `b`:

```
ana@linux:~$ ls -l /dev | grep '^b' | head -n 2
brw-rw---- 1 root disk    252,   0 Nov 14 20:43 dm-0
brw-rw---- 1 root disk      7,   0 Nov 14 20:43 loop0
```

LETRA	TIPO	QUÉ ES, EN UNA LÍNEA	EJEMPLO
`-`	archivo normal	datos guardados: texto, foto, programa	`nota.txt`
`d`	carpeta (*directory*)	una lista de nombres	`Documentos`
`l`	enlace simbólico (*link*)	una flecha hacia otro nombre	`acceso`, `/bin`
`c`	dispositivo de caracteres	entrega o recibe datos en chorrito, uno tras otro	`/dev/null`
`b`	dispositivo de bloques	se lee por pedazos y en cualquier orden: los discos	`dm-0`, `loop0`
`p`	tubería con nombre (FIFO)	la tubería `|`, pero con nombre en una carpeta	`tubo`
`s`	*socket*	un enchufe para que dos programas platiquen	aparecen en `/run`

Del lado del kernel son exactamente esos siete tipos[44]. Y `file` los reconoce todos, mirando adentro:

```
ana@linux:~$ file Documentos/nota.txt Documentos acceso tubo /dev/null /usr/bin/ls
Documentos/nota.txt: ASCII text
Documentos:          directory
acceso:              symbolic link to Documentos/nota.txt
tubo:                fifo (named pipe)
/dev/null:           character special (1/3)
/usr/bin/ls:         ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), ↵
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, ↵
BuildID[sha1]=44900829b39271d878615772bb2a6ffa422f0891, for GNU/Linux 3.2.0, stripped
```

Con `ls -F`, el tubo sale marcado como `tubo|` y un socket con `=`[45].

› Las nueve secciones del manual

SECCIÓN	QUÉ TRAE
1	Programas y comandos de la shell
2	Llamadas al sistema (las formas de la ventanilla de la sección 3)
3	Funciones de bibliotecas de programación
4	Archivos especiales, casi siempre de `/dev`
5	Formatos de archivo y convenciones, como `/etc/passwd`
6	Juegos
7	Varios: convenciones, estándares, explicaciones generales
8	Comandos de administración (casi siempre para root)
9	Rutinas del kernel (no es estándar)

La lista sale del propio manual de `man`[46].

› Espiar la ventanilla con strace

El programa `strace` le pone un espía a otro programa y anota cada forma que pasa por la ventanilla. Con `-c` no las enseña una por una: al final hace la cuenta. En el Ubuntu Server de Ana ya venía instalado:

```
ana@linux:~$ strace -c cat Documentos/nota.txt
Hola, Linux.
Esta es mi primera nota.
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 19.83    0.001033        1033         1           execve
 19.75    0.001029         102        10           mmap
 11.55    0.000602         200         3           mprotect
  8.45    0.000440          88         5           fstat
  7.18    0.000374          93         4           openat
  7.16    0.000373          62         6           close
  4.18    0.000218          72         3           brk
  3.51    0.000183          61         3           read
…
  1.07    0.000056          56         1           write
…
------ ----------- ----------- --------- --------- ----------------
100.00    0.005210         108        48         1 total
```

Primero sale lo de `cat`, la nota, y abajo la cuenta de `strace`; corté los renglones marcados con `…`. Lee las dos columnas de la derecha: `syscall` es el nombre de la forma y `calls`, cuántas veces la pasó. En total, 48 formas para leer un archivo de dos renglones, porque cualquier programa, solo para arrancar, pide muchas cosas (`execve`, `mmap`…). Las cuatro que tocan a `nota.txt` están ahí: `openat` (abrir), `read` (leer), `write` (escribir en la pantalla; una sola vez, los dos renglones juntos) y `close` (cerrar). `openat`, `read` y `close` salen más de una vez porque `cat` también abre, lee y cierra otros archivos que necesita para arrancar (la biblioteca de C, por ejemplo). Y entre los `close` de más están dos que no son de archivos del disco: al terminar, `cat` cierra su salida estándar y su salida de errores, lo que escribe en la pantalla. Si en tu máquina no está `strace`, no lo instales todavía: lo vas a poder hacer con calma en el capítulo de instalar programas.

---

40. Free Software Foundation (s. f.-b).
41. Free Software Foundation (1991), sección 3.
42. Free Software Foundation (1991), sección 2b.
43. Linux kernel developers (s. f.-a); Linux kernel developers (s. f.-b); The Linux Kernel documentation (s. f.).
44. Linux man-pages project (s. f.-d).
45. Free Software Foundation (s. f.-a).
46. man-db project (s. f.).

Referencias

Arpaci-Dusseau, R. H. y Arpaci-Dusseau, A. C. (2023). *Operating systems: Three easy pieces* (versión 1.10). Arpaci-Dusseau Books. https://pages.cs.wisc.edu/~remzi/OSTEP/

Baker, S. (s. f.). *A complete history of Tux (so far)*. Recuperado el 14 de noviembre de 2026, de https://landley.net/history/mirror/linux/tux.html

Canonical. (s. f.-a). *Ubuntu release cycle*. Ubuntu. Recuperado el 14 de noviembre de 2026, de https://ubuntu.com/about/release-cycle

Canonical. (s. f.-b). *Ubuntu 26.04 LTS release notes: Summary for LTS users*. Ubuntu Documentation. Recuperado el 14 de noviembre de 2026, de https://documentation.ubuntu.com/release-notes/26.04/summary-for-lts-users/

Fox, B. y Ramey, C. (s. f.). *bash(1): GNU Bourne-Again SHell* (paquete bash 5.2.21-2ubuntu4) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man1/bash.1.html

Free Software Foundation. (s. f.-a). *ls(1): List directory contents* (paquete coreutils 9.4-3ubuntu6.3) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man1/ls.1.html

Free Software Foundation. (s. f.-b). *Preguntas frecuentes acerca de las licencias de GNU*. Proyecto GNU. Recuperado el 14 de noviembre de 2026, de https://www.gnu.org/licenses/gpl-faq.html

Free Software Foundation. (1991). *GNU General Public License, version 2*. https://www.gnu.org/licenses/old-licenses/gpl-2.0.html

Free Software Foundation. (s. f.-c). *¿Qué es el software libre?* Proyecto GNU. Recuperado el 14 de noviembre de 2026, de https://www.gnu.org/philosophy/free-sw.es.html

Linux Foundation, LSB Workgroup. (2015). *Filesystem Hierarchy Standard* (versión 3.0). https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html

Linux kernel developers. (s. f.-a). *COPYING* [Archivo de licencia del código fuente de Linux]. GitHub. Recuperado el 14 de noviembre de 2026, de https://github.com/torvalds/linux/blob/master/COPYING

Linux kernel developers. (s. f.-b). *Linux-syscall-note* [Excepción de licencia del código fuente de Linux]. GitHub. Recuperado el 14 de noviembre de 2026, de https://github.com/torvalds/linux/blob/master/LICENSES/exceptions/Linux-syscall-note

Linux man-pages project. (s. f.-a). *boot(7): System bootup process based on UNIX System V Release 4* (paquete manpages 6.7-2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man7/boot.7.html

Linux man-pages project. (s. f.-b). *hier(7): Description of the filesystem hierarchy* (paquete manpages 6.7-2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man7/hier.7.html

Linux man-pages project. (s. f.-c). *intro(2): Introduction to system calls* (paquete manpages 6.7-2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man2/intro.2.html

Linux man-pages project. (s. f.-d). *inode(7): File inode information* (paquete manpages 6.7-2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man7/inode.7.html

Linux man-pages project. (s. f.-e). *path_resolution(7): How a pathname is resolved to a file* (paquete manpages 6.7-2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man7/path_resolution.7.html

Linux man-pages project. (s. f.-f). *syscalls(2): Linux system calls* (paquete manpages-dev 6.7-2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man2/syscalls.2.html

man-db project. (s. f.). *man(1): An interface to the system reference manuals* (paquete man-db 2.12.0-4build2) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man1/man.1.html

Open Source Initiative. (2007). *The Open Source Definition* (versión 1.9). https://opensource.org/osd

Ritchie, D. M. (1984). The evolution of the Unix time-sharing system. *AT&T Bell Laboratories Technical Journal, 63*(6), 1577–1593. https://www.nokia.com/bell-labs/about/dennis-m-ritchie/hist.html

Silberschatz, A., Galvin, P. B. y Gagne, G. (2018). *Operating system concepts* (10.ª ed.). Wiley.

Stallman, R. (1983, 27 de septiembre). *Anuncio inicial*. Proyecto GNU. https://www.gnu.org/gnu/initial-announcement.es.html

Stallman, R. (s. f.-a). *Linux y el sistema GNU*. Proyecto GNU. Recuperado el 14 de noviembre de 2026, de https://www.gnu.org/gnu/linux-and-gnu.es.html

Stallman, R. (s. f.-b). *Por qué el «código abierto» pierde de vista lo esencial del software libre*. Proyecto GNU. Recuperado el 14 de noviembre de 2026, de https://www.gnu.org/philosophy/open-source-misses-the-point.es.html

Sudo project. (s. f.). *sudo(8): Execute a command as another user* (paquete sudo 1.9.15p5-3ubuntu5.24.04.4) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man8/sudo.8.html

systemd project. (s. f.-a). *bootup(7): System bootup process* (paquete systemd 255.4-1ubuntu8.18) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man7/bootup.7.html

systemd project. (s. f.-b). *os-release(5): Operating system identification* (paquete systemd 255.4-1ubuntu8.18) [Página de manual]. Ubuntu Manpages. Recuperado el 14 de noviembre de 2026, de https://manpages.ubuntu.com/manpages/noble/en/man5/os-release.5.html

Tanenbaum, A. S. y Bos, H. (2023). *Modern operating systems* (5.ª ed.). Pearson.

The Linux Kernel documentation. (s. f.). *Linux kernel licensing rules*. Recuperado el 14 de noviembre de 2026, de https://www.kernel.org/doc/html/latest/process/license-rules.html

TOP500. (s. f.). *Operating system family / Linux*. Recuperado el 14 de noviembre de 2026, de https://www.top500.org/statistics/details/osfam/1/

Torvalds, L. (1991, 25 de agosto). *What would you like to see most in minix?* [Mensaje en el grupo de noticias comp.os.minix]. Google Groups. https://groups.google.com/g/comp.os.minix/c/dlNtH7RRrGA/m/SwRavCzVE7gJ

Torvalds, L. (s. f.). *RELNOTES-0.12* [Notas de la versión 0.12 de Linux]. The Linux Kernel Archives. Recuperado el 14 de noviembre de 2026, de https://www.kernel.org/pub/linux/kernel/Historic/old-versions/RELNOTES-0.12

Trifecta Tech Foundation. (2026, 10 de marzo). *sudo-rs: Changelog* (versión 0.2.13). GitHub. Recuperado el 14 de noviembre de 2026, de https://github.com/trifectatechfoundation/sudo-rs/blob/main/CHANGELOG.md

Ward, B. (2021). *How Linux works: What every superuser should know* (3.ª ed.). No Starch Press.

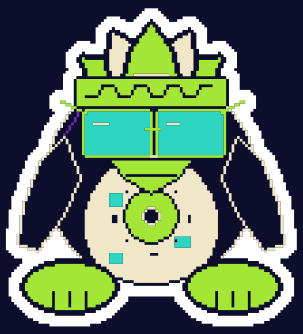
Para saber más

- **Arpaci-Dusseau y Arpaci-Dusseau (2023), capítulos 2 y 6.** Gratis, en línea y en un inglés llano. Es el mejor siguiente paso si te quedaste con ganas de entender qué hace el kernel con el procesador y la memoria, y de dónde sale la ventanilla.
- **Shotts, W. (s. f.). *The Linux command line*. https://linuxcommand.org/tlcl.php** Libro gratuito, en inglés (la edición de internet, versión 25.12; en papel lo publica No Starch Press, en su 3.ª edición), que cubre con calma todo lo que este libro toca de la terminal. Buen compañero para leer en paralelo.
- **Stallman, R. (s. f.-b). *Por qué el «código abierto» pierde de vista lo esencial del software libre*. https://www.gnu.org/philosophy/open-source-misses-the-point.es.html** La discusión entre «libre» y «abierto» contada por uno de sus protagonistas, en español.

Lo que sigue

Ya sabes qué hay debajo de la pantalla, cómo arranca la máquina, cómo moverte y cómo mirar. En el siguiente capítulo vas a empezar a hacer: crear tus propias carpetas y archivos, copiarlos, moverlos y borrarlos, y armar la carpeta `trabajo` que nos va a acompañar todo el libro. Ahí también vas a ver por qué los nombres con espacios necesitan comillas, que hoy quedó pendiente en los errores típicos.

Aquella pantalla negra ya no es una pared. Es una puerta, y ya la abriste.



LINUX DESDE CERO

El libro del curso · Hackztlan Lab