

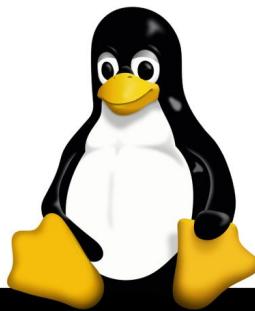
28 DE NOVIEMBRE DE 2020 / #LINUX

El Manual de Comandos de Linux



Juan Carrillo

THE LINUX COMMANDS HANDBOOK



Este manual de comandos de Linux cubrirá 60 comandos básicos de Bash que necesitarás como desarrollador. Cada comando incluye código de ejemplo y consejos sobre cuándo usarlo.

Este Manual de Comandos de Linux sigue la regla 80/20: aprenderás el 80% de un tema en alrededor del 20% del tiempo que pases estudiándolo.

Encuentro que este enfoque te da una buena visión general sobre el tema.

pequeños comandos clave que usarás el 80% o 90% del tiempo, e intenta simplificar el uso de los más complejos.

Todos estos comandos funcionan en Linux, macOS, WSL, y en cualquier lugar donde tengas un entorno UNIX.

Espero que el contenido de este manual te ayude a conseguir lo que quieras: **sentirte cómodo con Linux.**

Puedes guardar esta página en favoritos de tu navegador para poder consultar este manual en el futuro.

Y puedes [descargar este manual en formato PDF / ePUB / Mobi de forma gratuita \(en inglés\).](#)

¡Disfrútalo!

Índice

- [Introducción a Linux y los shells](#)
- [El comando man de Linux](#)
- [El comando ls de Linux](#)
- [El comando cd de Linux](#)
- [El comando pwd de Linux](#)
- [El comando mkdir de Linux](#)
- [El comando rmdir de Linux](#)
- [El comando mv de Linux](#)
- [El comando cp de Linux](#)
- [El comando open de Linux](#)
- [El comando touch de Linux](#)
- [El comando find de Linux](#)
- [El comando ln de Linux](#)
- [El comando gzip de Linux](#)
- [El comando gunzip de Linux](#)
- [El comando tar de Linux](#)
- [El comando alias de Linux](#)
- [El comando cat de Linux](#)

- [El comando tail de Linux](#)
- [El comando wc de Linux](#)
- [El comando grep de Linux](#)
- [El comando sort de Linux](#)
- [El comando uniq de Linux](#)
- [El comando diff de Linux](#)
- [El comando echo de Linux](#)
- [El comando chown de Linux](#)
- [El comando chmod de Linux](#)
- [El comando umask de Linux](#)
- [El comando du de Linux](#)
- [El comando df de Linux](#)
- [El comando basename de Linux](#)
- [El comando dirname de Linux](#)
- [El comando ps de Linux](#)
- [El comando top de Linux](#)
- [El comando kill de Linux](#)
- [El comando killall de Linux](#)
- [El comando jobs de Linux](#)
- [El comando bg de Linux](#)
- [El comando fg de Linux](#)
- [El comando type de Linux](#)
- [El comando which de Linux](#)
- [El comando nohup de Linux](#)
- [El comando xargs de Linux](#)
- [El comando vim del editor de Linux](#)
- [El comando emacs del editor de Linux](#)
- [El comando nano del editor de Linux](#)
- [El comando whoami de Linux](#)
- [El comando who de Linux](#)
- [El comando su de Linux](#)

- [El comando passwd de Linux](#)
- [El comando ping de Linux](#)
- [El comando traceroute de Linux](#)
- [El comando clear de Linux](#)
- [El comando history de Linux](#)
- [El comando export de Linux](#)
- [El comando crontab de Linux](#)
- [El comando uname de Linux](#)
- [El comando env de Linux](#)
- [El comando printenv de Linux](#)
- Conclusión

Introducción a Linux y los shells

¿Qué es Linux?

Linux es un sistema operativo, como macOS o Windows.

También es el sistema operativo de código abierto más popular, y te da mucha libertad.

Alimenta a la gran mayoría de los servidores que componen Internet. Es la base sobre la que se construye todo. Pero no sólo eso. Android está basado en (una versión modificada de) Linux.

El "núcleo" de Linux (llamado kernel) nació en 1991 en Finlandia, y ha recorrido un largo camino desde sus humildes comienzos. Se convirtió en el núcleo del sistema operativo GNU, creando el dúo GNU/Linux.

Hay una cosa acerca de Linux que las corporaciones como Microsoft, Apple y Google nunca podrán ofrecer: la libertad de hacer lo que uno quiera con su computadora.

En realidad van en dirección contraria, construyendo jardines amurallados, especialmente en el lado móvil.

Linux es la libertad definitiva.

Es desarrollado por voluntarios, algunos pagados por empresas que dependen de él, algunos de forma independiente. Pero no hay una sola compañía comercial que pueda dictar lo que

También puedes usar Linux como tu computadora del día a día. Uso macOS porque realmente disfruto de las aplicaciones y el diseño (y también solía ser un desarrollador de aplicaciones para iOS y Mac). Pero antes de usar macOS usaba Linux como sistema operativo de mi computadora principal.

Nadie puede dictar qué aplicaciones puedes ejecutar, o "llamar a casa" con aplicaciones que te rastrean, tu posición y más.

Linux también es especial porque no hay "un solo Linux", como es el caso de Windows o macOS. En su lugar, tenemos **distribuciones**.

Una "distro" es hecha por una compañía u organización y empaqueta el núcleo de Linux con programas y herramientas adicionales.

Por ejemplo, tenemos Debian, Red Hat y Ubuntu, probablemente las distribuciones más populares.

Pero existen muchas, muchas más. También puedes crear tu propia distribución. Pero lo más probable es que uses una popular que tenga muchos usuarios y una comunidad a su alrededor. Esto te permite hacer lo que necesitas hacer sin perder demasiado tiempo reinventando la rueda y encontrando respuestas a problemas comunes.

Algunas computadoras de escritorio y laptops vienen con Linux preinstalado. O puedes instalarlo en tu computadora con Windows o en Mac.

Pero no es necesario interrumpir tu computadora sólo para tener una idea de cómo funciona Linux.

No tengo una computadora con Linux.

Si usas una Mac, sólo necesitas saber que debajo de macOS hay un sistema operativo UNIX. Comparte muchas de las mismas ideas y software que usa un sistema GNU/Linux, porque GNU/Linux es una alternativa libre a UNIX.

Unix es un término paraguas que agrupa muchos sistemas operativos utilizados en las grandes corporaciones e instituciones, a partir de los años 70.

La terminal de macOS te da acceso a los mismos comandos exactos que describiré en el resto de este manual.

Linux) que puedes (¡y deberías!) instalar en Windows. Esto te dará la posibilidad de ejecutar Linux de una manera muy fácil en tu PC.

Pero la gran mayoría de las veces ejecutarás una computadora Linux en la nube a través de un VPS (Virtual Private Server, o Servidor Privado Virtual) como DigitalOcean.

¿Qué es un shell de Linux?

Un shell es un intérprete de comandos que expone una interfaz para que el usuario trabaje con el sistema operativo subyacente.

Permite ejecutar operaciones usando texto y comandos, y proporciona a los usuarios características avanzadas como la posibilidad de crear scripts.

Esto es importante: los shells te permiten realizar cosas de una manera más optimizada de lo que una GUI (Graphical User Interface o Interfaz Gráfica de Usuario) podría permitirte hacer. Las herramientas de línea de comandos pueden ofrecer muchas opciones de configuración diferentes sin ser demasiado complejas de usar.

Hay muchos tipos diferentes de shells. Este post se centra en los shells de Unix, los que se encuentran comúnmente en computadoras Linux y macOS.

Muchos tipos diferentes de shells fueron creados para estos sistemas a lo largo del tiempo, y algunos de ellos dominan el espacio: Bash, Csh, Zsh, Fish, ¡y muchos más!

Todos los shells se originan del Bourne Shell, llamado `sh`. "Bourne" porque su creador fue Steve Bourne.

Bash significa *Bourne-again shell*. `sh` era propietario y no de código abierto, y Bash fue creado en 1989 para crear una alternativa libre para el proyecto GNU y la Free Software Foundation. Como los proyectos tenían que pagar para usar el Bourne shell, Bash se hizo muy popular.

Si usas una Mac, intenta abrir tu terminal de Mac. Por defecto funciona con ZSH (o, pre-Catalina, Bash).

Puedes configurar tu sistema para ejecutar cualquier tipo de shell – por ejemplo, yo uso el Fish shell.

Cada shell tiene sus propias características y uso avanzado, pero todos comparten una funcionalidad común: pueden permitirte ejecutar programas, y pueden ser programadas.

El comando man de Linux

El primer comando que presentaré te ayudará a entender todos los demás comandos.

Cada vez que no sé cómo usar un comando, escribo `man <command>` para obtener el manual:

```
LS(1) BSD General Commands Manual LS(1)

NAME
    ls -- list directory contents

SYNOPSIS
    ls [-ABCDEFGHJKLMNOPRSTUW@abcdefghijklmnopqrstuvwxyz] [file ...]

DESCRIPTION
    For each operand that names a file of a type other than directory,
    ls displays its name as well as any requested, associated informa-
    tion. For each operand that names a file of type directory, ls dis-
    plays the names of files contained within that directory, as well as
    any requested, associated information.

    If no operands are given, the contents of the current directory are
    displayed. If more than one operand is given, non-directory oper-
    ands are displayed first; directory and non-directory operands are
    sorted separately and in lexicographical order.

    The following options are available:

    -@      Display extended attribute keys and sizes in long (-l) out-
            put.

    -1      (The numeric digit ``one''). Force output to be one entry
:
```

Este es un página man (`del_manual_`). Las páginas de man son una herramienta esencial para aprender como desarrollador. Contienen tanta información que a veces es casi demasiado.

La captura de pantalla anterior es sólo 1 de las 14 pantallas de explicación del comando `ls`.

La mayoría de las veces cuando necesito aprender un comando rápidamente utilizo este sitio llamado **páginas tldr**: <https://tldr.sh>. Es un comando que puedes instalar, que luego puedes ejecutar así: `tldr <command>`. Te da una visión general muy rápida de un comando, con algunos ejemplos prácticos de escenarios de uso común:

```
[+ ~ tldr ls

ls

List directory contents.

- List files one per line:
  ls -1

- List all files, including hidden files:
  ls -a

- Long format list (permissions, ownership, size and modification date) of all
  files:
  ls -la

- Long format list with size displayed using human readable units (KB, MB, GB)
  :
  ls -lh

- Long format list sorted by size (descending):
  ls -ls

- Long format list of all files, sorted by modification date (oldest first):
  ls -ltr

→ ~
```

Esto no es un sustituto de `man`, sino una herramienta útil para evitar perderse en la enorme cantidad de información presente en una página de `man`. Entonces puedes usar la página de `man` para explorar todas las diferentes opciones y parámetros que puedes usar en un comando.

El comando `ls` de Linux

Dentro de una carpeta puedes enlistar todos los archivos que contiene la carpeta usando el comando `ls`:

```
ls
```

Si agregas un nombre de una carpeta o una ruta, se imprimirá el contenido de esa carpeta:

```
ls /bin
```

```
flaviocopes — fish — /Users/flaviocopes — -fish — 72x9
[ ~ ls /bin
[      csh      ed      launchctl mv      rmdir      tcsh
bash    date    expr      link      pax      sh      test
cat      dd      hostname ln      ps      sleep      unlink
chmod   df      kill      ls      pwd      stty      wait4path
cp      echo    ksh      mkdir      rm      sync      zsh
[ ~ ]
```

`ls` acepta muchas opciones. Una de mis combinaciones favoritas es `-al`. Pruébalo:

```
ls -al /bin
```

```
flaviocopes — fish — /Users/flaviocopes — -fish — 72x20
[ ~ ls -al /bin
total 5120
drwxr-xr-x@ 37 root  wheel  1184 Feb  4 10:05 .
drwxr-xr-x  30 root  wheel   960 Feb  8 15:32 ..
-rwxr-xr-x  1 root  wheel  22704 Jan 16 02:21 [
-rwxr-xr-x  1 root  wheel 618416 Jan 16 02:21 bash
-rwxr-xr-x  1 root  wheel  23648 Jan 16 02:21 cat
-rwxr-xr-x  1 root  wheel  34144 Jan 16 02:21 chmod
-rwxr-xr-x  1 root  wheel  29024 Jan 16 02:21 cp
-rwxr-xr-x  1 root  wheel 379952 Jan 16 02:21 csh
-rwxr-xr-x  1 root  wheel  28608 Jan 16 02:21 date
-rwxr-xr-x  1 root  wheel  32000 Jan 16 02:21 dd
-rwxr-xr-x  1 root  wheel  23392 Jan 16 02:21 df
-rwxr-xr-x  1 root  wheel  18128 Jan 16 02:21 echo
-rwxr-xr-x  1 root  wheel  54080 Jan 16 02:21 ed
-rwxr-xr-x  1 root  wheel  23104 Jan 16 02:21 expr
-rwxr-xr-x  1 root  wheel  18288 Jan 16 02:21 hostname
-rwxr-xr-x  1 root  wheel  18688 Jan 16 02:21 kill
-rwxr-xr-x  1 root  wheel 1282864 Jan 16 02:21 ksh
-rwxr-xr-x  1 root  wheel 121296 Jan 16 02:21 launchctl
```

Comparado con el comando de `ls` puro, esto devuelve mucha más información:

Tienes, de izquierda a derecha:

- los permisos de los archivos (y si tu sistema soporta ACLs, también obtienes una bandera ACL)
- el número de enlaces a ese archivo

- el grupo del archivo
- el tamaño del archivo en bytes
- la fecha y hora de la última modificación del archivo
- el nombre del archivo

Este conjunto de datos es generado por la opción `l`. La opción `a`, en cambio, también muestra los archivos ocultos.

Los archivos ocultos con archivos que comienzan con un punto (`.`).

El comando `cd` de Linux

Una vez que tienes una carpeta, te puedes mover a ella usando el comando `cd`. `cd` significa **c**hange **d**irectory (o cambio de directorio). Se invoca especificando la carpeta a la que te vas a mover. Puedes especificar un nombre de carpeta o una ruta de acceso completa.

Por ejemplo:

```
mkdir fruits  
cd fruits
```

Ahora estás en la carpeta `fruits`.

Puedes usar la ruta especial `..` para indicar la carpeta padre:

```
cd .. #devuelta a la carpeta home
```

El carácter `#` indica el comienzo de un comentario, que dura toda la línea después de ser encontrado.

Puedes usarlo para formar una ruta:

```
mkdir fruits  
mkdir cars
```

Hay otro indicador de ruta espacial que es `..`, e indica la carpeta **actual**.

También puedes utilizar rutas absolutas, que empiezan en la carpeta raíz `/`:

```
cd /etc
```

El comando `pwd` de Linux

Cuando te sientas perdido en el sistema de archivos, llama al comando `pwd` para saber dónde estás:

```
pwd
```

Imprimirá la ruta de la carpeta actual.

El comando `mkdir` de Linux

Puedes crear carpetas usando el comando `mkdir`:

```
mkdir fruits
```

Puedes crear múltiples carpetas con un solo comando:

```
mkdir dogs cars
```

También puedes crear múltiples carpetas anidadas al agregar la opción `-p`:

```
mkdir -p fruits/apples
```

del nombre del comando y cambian el comportamiento del mismo. A menudo también puedes combinar varias opciones.

Puedes encontrar qué opciones soporta un comando escribiendo `man <command>`. Inténtalo ahora con `man mkdir` por ejemplo (presiona la tecla `q` para salir de la página de `man`). Las páginas de manual son la increíble ayuda incorporada para UNIX.

El comando `rmdir` de Linux

Así como puedes crear una carpeta usando `mkdir`, puedes eliminar una carpeta usando `rmdir`:

```
mkdir fruits  
rmdir fruits
```

Puedes borrar múltiples carpetas a la vez:

```
mkdir fruits cars  
rmdir fruits cars
```

La carpeta que borres debe de estar vacía.

Para borrar carpetas con archivos dentro, usaremos el comando genérico `rm` el cual borrar archivos y carpetas, usando la opción `-rf`:

```
rm -rf fruits cars
```

Ten cuidado, ya que este comando no pide confirmación e inmediatamente eliminará todo lo que le pidas que elimine.

No hay papelera cuando se eliminan archivos desde la línea de comandos, y recuperar archivos perdidos puede ser difícil.

El comando `mv` de Linux

actual del archivo, y su nueva ruta:

```
touch pear  
mv pear new_pear
```

El archivo `pear` se ha movido a `new_pear`. Así es como se renombran los archivos y las carpetas.

Si el último parámetro es una carpeta, el archivo ubicado en la ruta del primer parámetro se moverá a esa carpeta. En este caso, puedes especificar una lista de archivos y todos ellos se moverán en la ruta de la carpeta identificada por el último parámetro:

```
touch pear  
touch apple  
mkdir fruits  
mv pear apple fruits # pear y apple se movieron a la carpeta fruits
```

El comando `cp` de Linux

Puedes copiar un archivo usando el comando `cp`:

```
touch apple  
cp apple another_apple
```

Para copiar carpetas hay que agregar la opción `-r` para copiar recursivamente todo el contenido de la carpeta:

```
mkdir fruits  
cp -r fruits cars
```

El comando `open` de Linux

El comando `open` te permite abrir un archivo usan esta sintaxis:

También puedes abrir un directorio, el cual en macOS abre la aplicación Finder con el directorio actual abierto:

```
open <directory name>
```

Lo uso todo el tiempo para abrir el directorio actual:

```
open .
```

El símbolo especial . apunta al directorio actual, así como .. apunta al directorio padre

El mismo comando puede ser usado para correr una aplicación:

```
open <application name>
```

El comando touch de Linux

Puedes crear un archivo vacío usando el comando `touch`:

```
touch apple
```

Si el archivo ya existe, es abierto en modo de escritura, y se actualiza la marca de tiempo (timestamp) del archivo.

El comando find de Linux

El comando `find` puede utilizarse para encontrar archivos o carpetas que coincidan con un patrón de búsqueda determinado. Busca de forma recursiva.

Aprendamos a usarla con un ejemplo.

relativa de cada archivo que coincida:

```
find . -name '*.js'
```

Es importante usar comillas alrededor de los caracteres especiales como * para evitar que el shell los interprete.

Encuentra directorios bajo el árbol actual que coincidan con el nombre "src":

```
find . -type d -name src
```

Usa -type f para buscar sólo archivos, o -type l para buscar sólo enlaces simbólicos.

-name es sensible a mayúsculas y minúsculas. Usar -iname para realizar una búsqueda insensible a mayúsculas y minúsculas.

Puedes buscar en múltiples árboles raíz:

```
find folder1 folder2 -name filename.txt
```

Encuentra directorio bajo el árbol actual que coincide con el nombre "node_modules" o "public":

```
find . -type d -name node_modules -or -name public
```

También puedes excluir la ruta usando -not -path :

```
find . -type d -name '*.md' -not -path 'node_modules/*'
```

Puedes buscar archivos con más de 100 caracteres (bytes) en ellos:

Buscar archivos mayores a 100kB pero menores a 1MB:

```
find . -type f -size +100k -size -1M
```

Buscar archivos editados no hace más de 3 días:

```
find . -type f -mtime +3
```

Buscar archivos editados en las últimas 24 horas:

```
find . -type f -mtime -1
```

Puedes borrar todos los archivos que coinciden con una búsqueda al agregar la opción `-delete`. Esto borra todos los archivos editados en las últimas 24 horas:

```
find . -type f -mtime -1 -delete
```

Puede ejecutar un comando en cada resultado de la búsqueda. En este ejemplo ejecutamos `cat` para imprimir el contenido del archivo:

```
find . -type f -exec cat {} \;
```

Fíjate en la terminación `\;`. `{}` se llena con el nombre del archivo en el momento de la ejecución.

El comando `ln` de Linux

El comando `ln` es parte de los comandos del sistema de archivos de Linux.

archivo que apunta a otro archivo. Puede que estés familiarizado con los atajos de Windows. Son similares.

Tenemos dos tipos de enlaces: **hard links** y **soft links**.

Hard links

Los hard links se usan raramente. Tienen algunas limitaciones: no se pueden enlazar a directorios, y no se pueden enlazar a sistemas de archivos externos (discos).

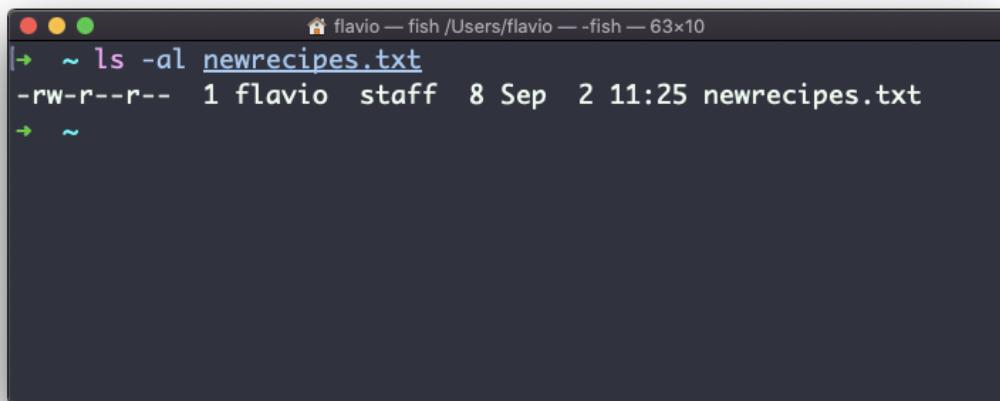
Un hard link se crea usando la siguiente sintaxis:

```
ln <original> <link>
```

Por ejemplo, digamos que tienes un archivo llamado `recipes.txt`. Puedes crear un hard link hacia él usando:

```
ln recipes.txt newrecipes.txt
```

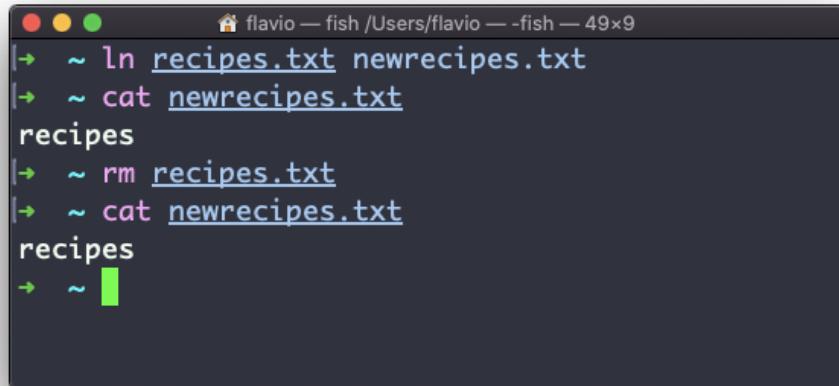
El nuevo hard link que creaste es indistinguible de un archivo regular:



```
flavio — fish /Users/flavio — -fish — 63x10
|→ ~ ls -al newrecipes.txt
|-rw-r--r-- 1 flavio staff 8 Sep 2 11:25 newrecipes.txt
|→ ~
```

Ahora cada vez que edites cualquiera de esos archivos, el contenido se actualizará para ambos.

Si eliminas el archivo original, el enlace seguirá almacenando el contenido del archivo original, ya que no se eliminará hasta que haya un enlace duro que apunte a él.



A screenshot of a terminal window titled "flavio — fish /Users/flavio — -fish — 49x9". The terminal shows the following sequence of commands:

```
|→ ~ ln recipes.txt newrecipes.txt
|→ ~ cat newrecipes.txt
recipes
|→ ~ rm recipes.txt
|→ ~ cat newrecipes.txt
recipes
|→ ~ |
```

The terminal uses color-coded syntax highlighting where underlined file names like "recipes.txt" and "newrecipes.txt" are displayed in blue. The terminal window has a dark background with light-colored text and a green cursor bar.

Soft links

Los soft links son diferentes. Son más poderosos ya que pueden enlazar a otros sistemas de archivos y a directorios. Pero ten en cuenta que cuando se elimina el original, el enlace se rompe.

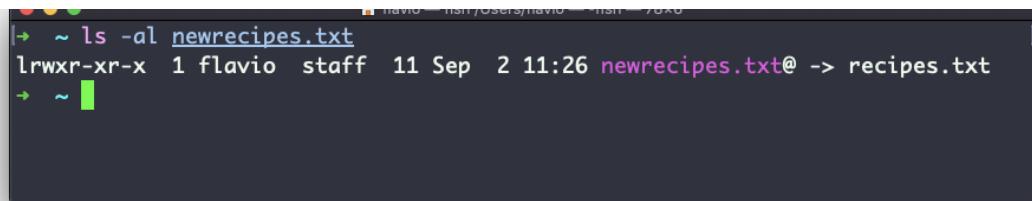
Puedes crear soft links usando la opción `-s` de `ln`:

```
ln -s <original> <link>
```

Por ejemplo, digamos que tienes un archivo llamado `recipes.txt`. Puedes crear un soft link hacia él usando:

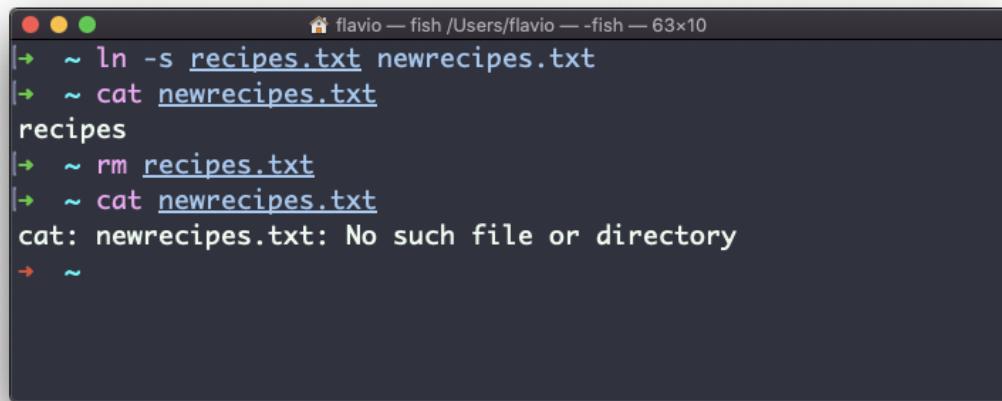
```
ln -s recipes.txt newrecipes.txt
```

En este caso, puedes ver que hay una bandera especial `l` cuando enumeras el archivo usando `ls -al`. El nombre del archivo tiene un `@` al final, y también tiene un color diferente si tienes los colores activados:



```
flavio ~ ls -al newrecipes.txt
lrwxr-xr-x 1 flavio staff 11 Sep 2 11:26 newrecipes.txt@ -> recipes.txt
~
```

Ahora bien, si borrar el archivo original, los enlaces se romperán, y el shell te dirá "No existe tal archivo o directorio" si intentas acceder a él:



```
flavio ~ ln -s recipes.txt newrecipes.txt
~ cat newrecipes.txt
recipes
~ rm recipes.txt
~ cat newrecipes.txt
cat: newrecipes.txt: No such file or directory
~
```

El comando gzip de Linux

Puedes comprimir un archivo usando el protocolo de compresión llamado LZ77 usando el comando `gzip`.

Este es el uso más simple:

```
gzip filename
```

Esto comprimirá el archivo, y le añadirá una extensión `.gz`. El archivo original es eliminado.

Para evitar esto último, puedes utilizar la opción `-c` y usar la redirección de salida para escribir el archivo `filename.gz`:

```
gzip -c filename > filename.gz
```

La opción `-c` especifica que la salida irá al flujo de salida estándar, dejando el archivo original intacto.

O puedes usar la opción `-k` :

```
gzip -k filename
```

Hay varios niveles de compresión. Cuanto mayor sea la compresión, más tiempo llevará comprimir (y descomprimir). Los niveles van de 1 (la compresión más rápida y peor) a 9 (la compresión más lenta y mejor), el valor predeterminado es 6.

Puedes elegir un nivel específico con la opción `-<NUMBER>` :

```
gzip -1 filename
```

Puedes comprimir múltiples archivos al enlistarlos:

```
gzip filename1 filename2
```

Puedes comprimir todos los archivos en un directorio, recursivamente, usando la opción `-r` :

```
gzip -r a_folder
```

La opción `-v` imprime la información del porcentaje de compresión. Aquí está un ejemplo siendo usado con la opción `-k` (keep o mantener):



```
flavio — fish /Users/flavio — -fish — 58x5
~ gzip -kv wget-log
wget-log: 49.7% -- replaced with wget-log.gz
~
```

`gzip` también puede ser usado para descomprimir un archivo, usando la opción `-d` :

```
gzip -d filename.gz
```

El comando `gunzip` de Linux

El comando `gunzip` es básicamente equivalente al comando `gzip`, excepto que la opción `-d` siempre está activado por defecto.

El comando puede ser invocado de la siguiente manera:

```
gunzip filename.gz
```

Esto hará que se ejecute `gunzip` y eliminará la extensión `.gz`, poniendo el resultado en el archivo "filename". Si ese archivo existe, lo sobreescibirá.

Puedes extraer a un archivo diferente usando la redirección de salida utilizando la opción `-c`:

```
gunzip -c filename.gz > anotherfilename
```

El comando `tar` de Linux

El comando `tar` es usado para crear un archivo, agrupando múltiples archivos en uno solo.

archivos se almacenaban en cintas).

Este comando crea un archivo llamado `archive.tar` con el contenido del `file1` y `file2`:

```
tar -cf archive.tar file1 file2
```

La opción `c` significa crear. La opción `f` es usada para escribir el archivo y archivar al mismo.

Para extraer archivos de otro de la carpeta actual, usa:

```
tar -xf archive.tar
```

la opción `x` representa extracción

Y para extraerlos a un directorio en específico, usa:

```
tar -xf archive.tar -C directory
```

También puedes solo enlistar los archivos que se contienen en un archivo `tar`:



```
flavio — fish /Users/flavio — -fish — 55x5
|> ~ tar -tf archive.tar
file1
file2
|> ~
```

`tar` se usa a menudo para crear un archivo comprimido.

Esto se hace usando la opción `z`:

```
tar -czf archive.tar.gz file1 file2
```

Esto es como crear un archivo tar, y después aplicar gzip en el mismo.

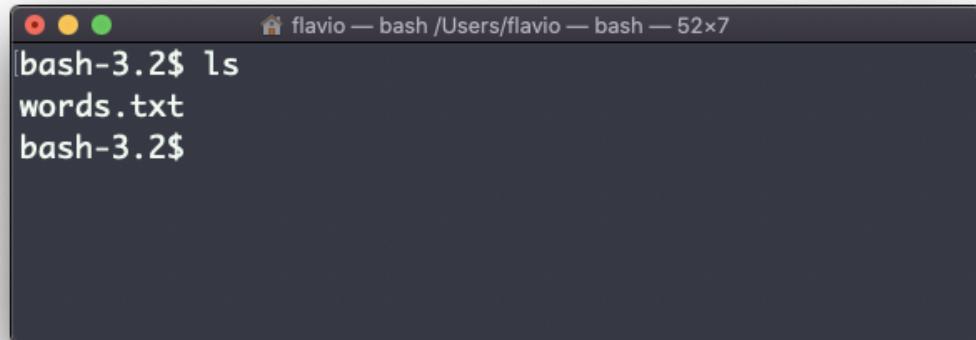
Para desarchivar un archivo comprimido, puedes usar gunzip , o gzip -d , y después desarchivarlo. Pero tar -xf lo reconocerá como un archivo comprimido y lo hará por ti.

```
tar -xf archive.tar.gz
```

El comando alias de Linux

Es común que siempre se ejecute un programa con un conjunto de opciones que te gusta usar.

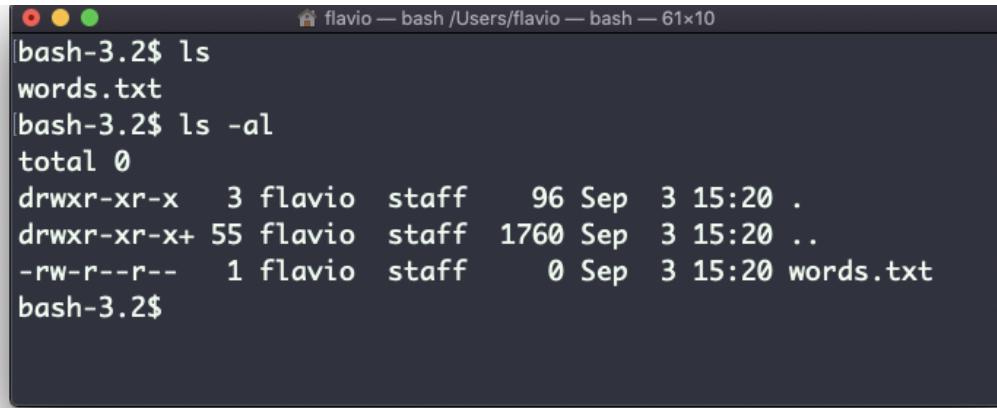
Por ejemplo, toma el comando ls . Por defecto, imprime muy poca información:



A screenshot of a macOS terminal window titled "flavio — bash /Users/flavio — bash — 52x7". The window shows the command "ls" being run, which outputs "words.txt". The prompt "bash-3.2\$" appears twice, once before and once after the output.

```
flavio — bash /Users/flavio — bash — 52x7
|bash-3.2$ ls
words.txt
bash-3.2$
```

Pero si usas las opción -al imprimirá algo más útil, incluyendo la fecha de modificación del archivo, el tamaño, el propietario y los permisos. También enlistará los archivos ocultos (archivos que empiezan con un .):



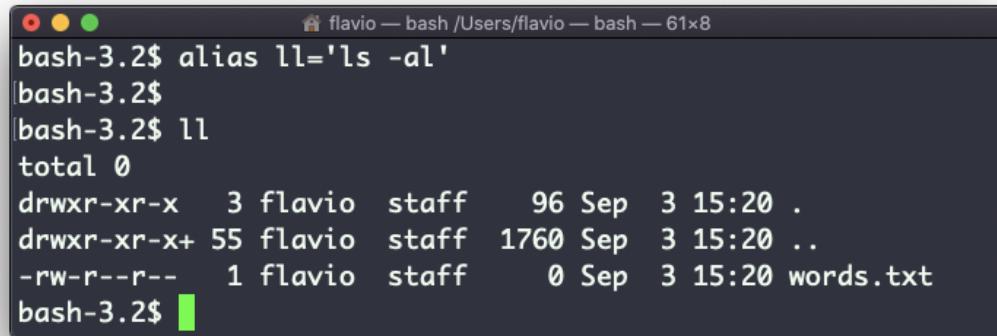
```
flavio — bash /Users/flavio — bash — 61x10
|bash-3.2$ ls
|words.txt
|bash-3.2$ ls -al
|total 0
|drwxr-xr-x  3 flavio  staff   96 Sep  3 15:20 .
|drwxr-xr-x+ 55 flavio  staff  1760 Sep  3 15:20 ..
|-rw-r--r--  1 flavio  staff    0 Sep  3 15:20 words.txt
bash-3.2$
```

Puedes crear un nuevo comando, por ejemplo, me gusta llamarlo `ll`, que es un alias de `ls -al`.

Lo haces así:

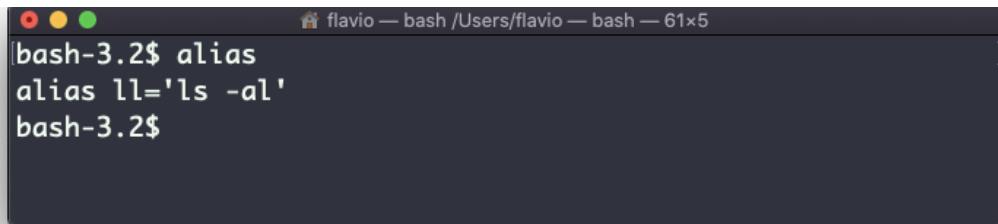
```
alias ll='ls -al'
```

Una vez que lo hagas, puedes llamar a `ll` como si fuera un comando regular de UNIX:



```
flavio — bash /Users/flavio — bash — 61x8
|bash-3.2$ alias ll='ls -al'
|bash-3.2$
|bash-3.2$ ll
|total 0
|drwxr-xr-x  3 flavio  staff   96 Sep  3 15:20 .
|drwxr-xr-x+ 55 flavio  staff  1760 Sep  3 15:20 ..
|-rw-r--r--  1 flavio  staff    0 Sep  3 15:20 words.txt
bash-3.2$
```

Ahora llamando `alias` sin ninguna opción enlistará los aliases definidos:



A screenshot of a macOS terminal window titled "flavio — bash /Users/flavio — bash — 61x5". The window shows the command "alias ll='ls -al'" being typed at the prompt "bash-3.2\$". The terminal has a dark theme with orange window controls.

```
flavio — bash /Users/flavio — bash — 61x5
|bash-3.2$ alias
alias ll='ls -al'
bash-3.2$
```

El alias funcionará hasta que se cierre la sesión de la terminal.

Para hacerlo permanente,, necesitas agregarlo a la configuración del shell. Esto podría ser `~/.bashrc` o `~/.profile` o `~/.bash_profile` si usas un shell Bash, dependiendo del caso de uso.

Ten cuidado con las comillas si tienes variables en el comando: si usas comillas dobles, la variable se resuelve en el momento de definición. Si usas comillas simples, se resuelve en el momento de la invocación. Esas 2 son diferentes:

```
alias lsthis="ls $PWD"
alias lscurrent='ls $PWD'
```

`$PWD` se refiere a la carpeta actual en la que se encuentra el shell. Si ahora navegas a una nueva carpeta, `lscurrent` enlista los archivos en la nueva carpeta, mientras que `lsthis` todavía enlista los archivos en la carpeta donde estaba cuando definió el alias.

El comando cat de Linux

Similar a [tail](#) en algunos aspectos, tenemos `cat`. Excepto que `cat` también puedes agregar contenido a un archivo, y esto lo hacer súper poderoso.

En su uso más simple, `cat` imprime el contenido de un archivo a la salida estándar:

```
cat file
```

Puedes imprimir el contenido de múltiples archivos:

y usando el operador de redirección de salida `>` puedes concatenar el contenido de múltiples archivo a un nuevo archivo:

```
cat file1 file2 > file3
```

Usando `>>` puedes anexar el contenido de múltiples archivo en un nuevo archivo, creándolo si este no existiera:

```
cat file1 file2 >> file3
```

Cuando miras archivos con código fuente, es útil poder ver el número de líneas. Puedes hacer que `cat` los imprima usando la opción `-n`:

```
cat -n file1
```

Sólo puedes agregar un número a las líneas no vacías usando `-b`, o también puedes eliminar todas la múltiples líneas vacías usando `-s`.

`cat` se usa a menudo en combinación con el operador pipe `|` para alimentar el contenido de un archivo como entrada de otro comando: `cat file1 | otro comando`.

El comando `less` de Linux

El comando `less` es uno que uso mucho. Te muestra el contenido almacenado dentro de un archivo, en una agradable e interactiva interfaz de usuario.

Uso: `less <filename>`

```

title: "Introduction to Bash Shell Scripting"
date: 2019-01-15T07:00:00+02:00
description: "A detailed overview to scripting the Bash Shell"
tags: cli
---

Shell scripting is an powerful way to automate tasks that you regularly execute on your computer.

In this tutorial I give an extensive overview of shell scripting, and will be the base reference for more in-depth and advanced tutorials on creating practical shell scripts.

> Check out my [introduction to Bash](/bash/) post.

Bash gives you a set of commands that put together can be used to create little programs, that by convention we call scripts.

Note the difference. We don't say Bash programming but Bash scripting, and we don't call Bash scripts "Bash programs". This is because you can g
:
```

Una vez que estás dentro de una sesión `less`, puedes salir presionando `q`.

Puedes navegar el contenido del archivo usando las teclas de `arriba` u `abajo`, o usando la barra espaciadora y `b` para navegar página por página. También puedes saltar al final de un archivo presionando `G` y saltar devuelta al inicio presionando `g`.

Puedes buscar contenido dentro del archivo presionando `/` y escribiendo la palabra a buscar. Esto busca *hacia delante*. Puedes buscar hacia atrás usando el signo `?` y escribiendo una palabra.

Este comando sólo visualiza el contenido de un archivo. Puedes abrir directamente un editor de texto presionando `v`. Esto usará el editor del sistema, que en la mayoría de casos es `vim`.

Presionando la tecla `F` entramos a *modo de seguimiento*, o *modo de observación* (follow mode, o watch mode en inglés, respectivamente). Cuando el archivo es cambiado por alguien más, como desde otro programa, puedes ver los cambios *en vivo*.

Esto no pasa por defecto, y sólo ves la versión del archivo en el momento de abrirlo. Necesitar presionar `ctrl-C` para salir de este modo. En este caso el comportamiento es similar a ejecutar el comando `tail -f <filename>`.

Puedes abrir múltiples archivos y navegar a través de ellos usando `:n` (para ir al siguiente archivo) y `:p` (para ir al anterior).

El comando tail de Linux

El mejor caso de uso de `tail` en mi opinión es cuando se llama con la opción `-f`. Abre el archivo al final, y observa los cambios de archivo.

Cada vez que hay un nuevo contenido en el archivo, se imprime en la ventana. Esto es genial para ver los archivos de registro, por ejemplo:

```
tail -f /var/log/system.log
```

Para salir, presiona `ctrl-C`.

Puedes imprimir las últimas 10 líneas de un archivo:

```
tail -n 10 <filename>
```

Puedes imprimir todo el contenido del archivo a partir de una línea específica usando `+` antes del número de línea:

```
tail -n +10 <filename>
```

`tail` puede hacer mucho más y como siempre mi consejo es revisar `man tail`.

El comando wc de Linux

El comando `wc` nos brinda información útil sobre un archivo o entrada que se recibe vía "pipes".

```
echo test >> test.txt
wc test.txt
1 1 5 test.txt
```

Ejemplo usando pipes, contamos la salida de la corrida del comando `ls -al`:

```
ls -al | wc  
6 47 284
```

La primera columna regresa el número de líneas. La segunda es el número de palabras. La tercera es el número de bytes.

Podemos decirle que sólo cuente las líneas:

```
wc -l test.txt
```

o sólo las palabras:

```
wc -w test.txt
```

o sólo los bytes:

```
wc -c test.txt
```

Los bytes en los juegos de caracteres (Charsets) ASCII equivalen a caracteres. Pero con los charsets que no son ASCII, el numero de caracteres pueden diferir porque algunos caracteres pueden tomar múltiples bytes (por ejemplo, esto sucede en Unicode).

En este caso la bandera `-m` te ayudará a obtener el valor correcto:

```
wc -m test.txt
```

El comando grep de Linux

El comando `grep` es una herramienta muy útil. Cuando lo domines, te ayudará enormemente en tu trabajo diario como programador.

expression *print* en inglés).

Puedes usar `grep` para buscar en archivos, o combinarlo con pipes para filtrar la salida de otro comando.

Por ejemplo, así es como podemos encontrar las ocurrencias de la línea `document.getElementById` en el archivo `index.md`:

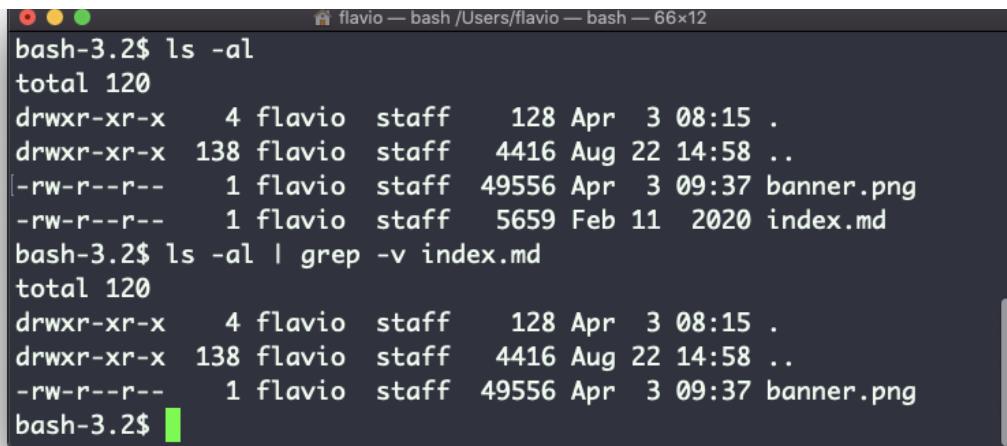
```
grep document.getElementById index.md
```



A screenshot of a macOS terminal window titled "flavio — bash /Users/flavio — bash — 77x7". The window shows the command "bash-3.2\$ grep document.getElementById index.md" followed by two lines of code: "document.getElementById('button').addEventListener('click', () => {" and "document.getElementById('button').addEventListener('click', () => {". The prompt "bash-3.2\$" is at the bottom.

Usando la opción `-n` mostrará el número de líneas:

```
grep -n document.getElementById index.md
```



```
flavio — bash /Users/flavio — bash — 66x12
bash-3.2$ ls -al
total 120
drwxr-xr-x  4 flavio  staff   128 Apr  3 08:15 .
drwxr-xr-x  138 flavio  staff  4416 Aug 22 14:58 ..
[ -rw-r--r--  1 flavio  staff  49556 Apr  3 09:37 banner.png ]
-rw-r--r--  1 flavio  staff  5659 Feb 11  2020 index.md
bash-3.2$ ls -al | grep -v index.md
total 120
drwxr-xr-x  4 flavio  staff   128 Apr  3 08:15 .
drwxr-xr-x  138 flavio  staff  4416 Aug 22 14:58 ..
[ -rw-r--r--  1 flavio  staff  49556 Apr  3 09:37 banner.png ]
bash-3.2$
```

Una cosa muy útil es decirle a grep que imprima 2 líneas antes y 2 líneas después de la línea emparejada para darle más contexto. Eso se hace usando la opción `-C`, que acepta un número de líneas:

```
grep -nC 2 document.getElementById index.md
```

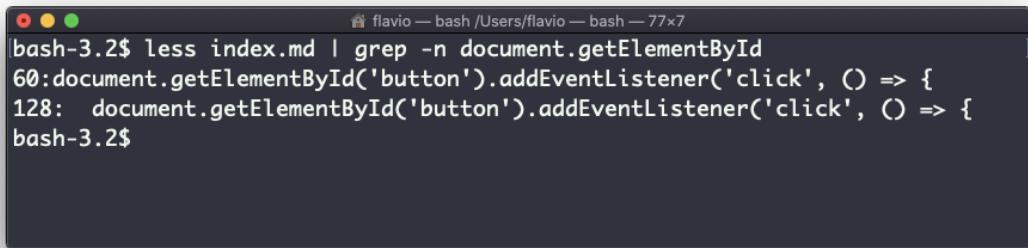


```
flavio — bash /Users/flavio — bash — 76x16
bash-3.2$ grep -nC 2 document.getElementById index.md
58-
59-```js
60:document.getElementById('button').addEventListener('click', () => {
61-   //item clicked
62-})
--
--
126-```js
127>window.addEventListener('load', () => {
128:   document.getElementById('button').addEventListener('click', () => {
129-     setTimeout(() => {
130-       items.forEach(item => {
bash-3.2$
```

La búsqueda es sensible a mayúsculas y minúsculas por defecto. Usa la bandera `-i` para hacerla insensible.

Podemos replicar la misma funcionalidad de arriba usando:

```
less index.md | grep -n document.getElementById
```



```
flavio — bash /Users/flavio — bash — 77x7
bash-3.2$ less index.md | grep -n document.getElementById
60:document.getElementById('button').addEventListener('click', () => {
128:  document.getElementById('button').addEventListener('click', () => {
bash-3.2$
```

La cadena de búsqueda puede ser una expresión regular, y esto hace que `grep` sea muy poderoso.

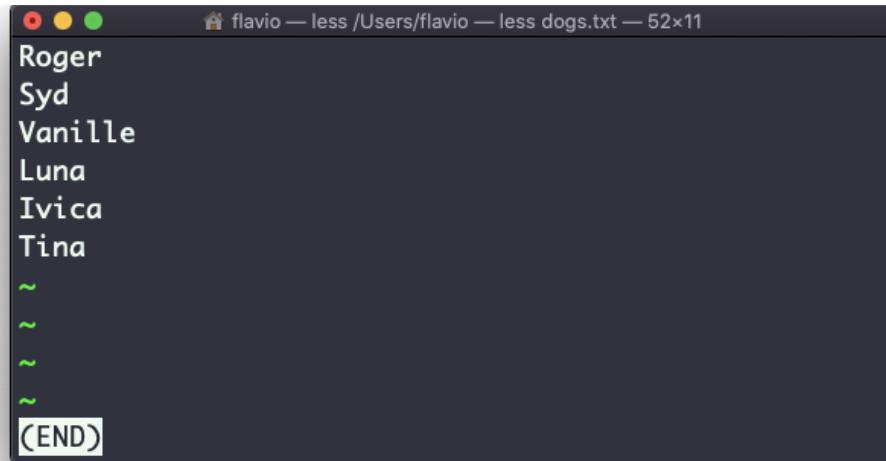
Otra cosa que puede ser muy útil es invertir el resultado, excluyendo las líneas que coinciden con una cadena particular, usando la opción `-v`:



```
flavio — bash /Users/flavio — bash — 77x7
bash-3.2$ grep -n document.getElementById index.md | grep -v document.getElementById
^60:document.getElementById('button').addEventListener('click', () => {
^128:  document.getElementById('button').addEventListener('click', () => {
bash-3.2$
```

El comando `sort` de Linux

Supongamos que tienes un archivo de texto que contiene nombres de perros:



```
flavio — less /Users/flavio — less dogs.txt — 52x11
Roger
Syd
Vanille
Luna
Ivica
Tina
~
~
~
~
(END)
```

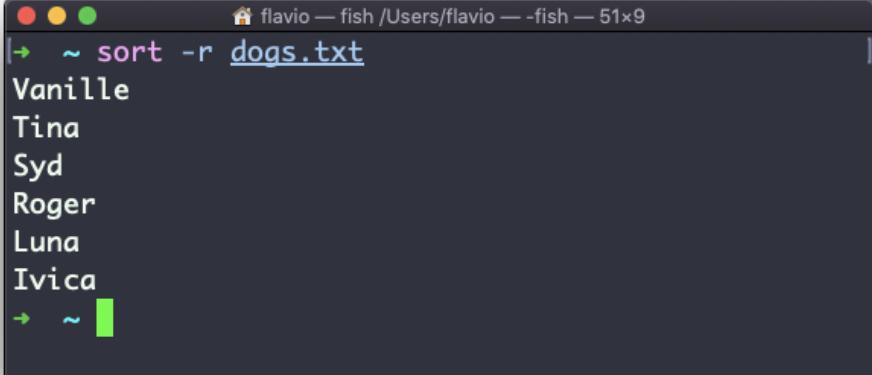
Esta lista está desordenada.

El comando `sort` ayudar a ordenarlos por nombre:



```
flavio — fish /Users/flavio — -fish — 52x11
|~ sort dogs.txt
Ivica
Luna
Roger
Syd
Tina
Vanille
→ ~
```

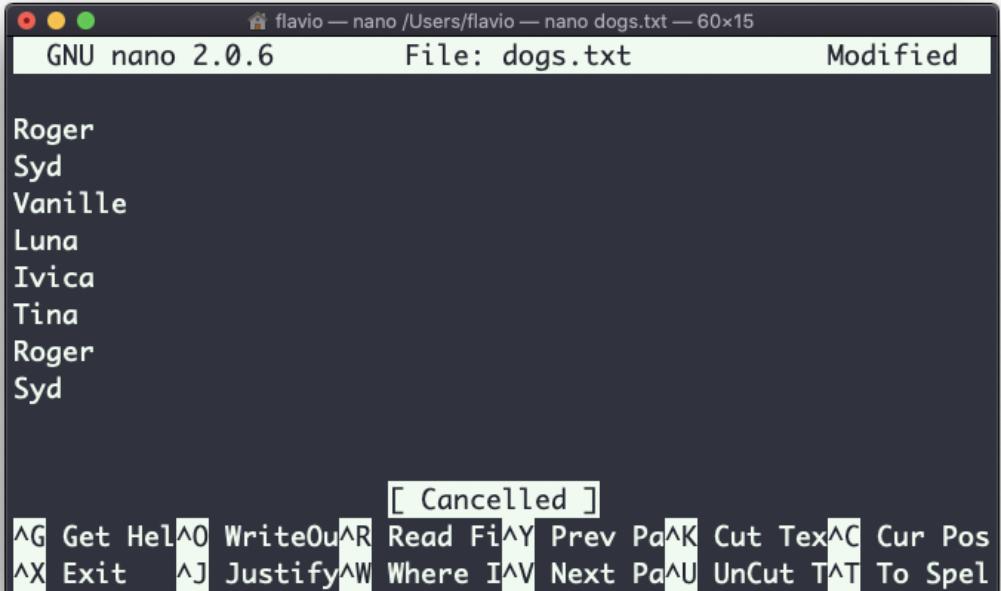
Puedes usar la opción `-r` para invertir el orden:



```
flavio — fish /Users/flavio — -fish — 51x9
~ sort -r dogs.txt
Vanille
Tina
Syd
Roger
Luna
Ivica
~
```

El ordenamiento por defecto es sensible a mayúsculas y minúsculas, y alfabético. Usa la opción `--ignore-case` para ordenar ignorando la sensibilidad a mayúsculas y minúsculas, y la opción `-n` para ordenarlas de manera numérica.

Si el archivo contiene líneas duplicadas:



```
flavio — nano /Users/flavio — nano dogs.txt — 60x15
GNU nano 2.0.6           File: dogs.txt           Modified

Roger
Syd
Vanille
Luna
Ivica
Tina
Roger
Syd

[ Cancelled ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit      ^J Justify ^W Where I ^V Next Page ^U Uncut T ^T To Spell
```

Puedes usar la opción `-u` para eliminarlos:



A screenshot of a terminal window titled "flavio — fish /Users/flavio — -fish — 40x8". The command entered is "sort -u dogs.txt". The output shows the names Ivica, Luna, Roger, Syd, Tina, and Vanille, each on a new line. The terminal has a dark background with light-colored text and a green cursor.

```
sort -u dogs.txt
Ivica
Luna
Roger
Syd
Tina
Vanille
```

`sort` no sólo funciona en los archivos, como muchos comandos de UNIX – sino que también funciona con pipes. Así que puedes usarlo en la salida de otro comando. Por ejemplo, puedes ordenar los archivos devueltos por `ls` con:

```
ls | sort
```

`sort` es muy poderoso y tiene muchas opciones, los cuáles puedes explorar al llamar `man sort`.

```

SORT(1)           BSD General Commands Manual          SORT(1)

NAME
    sort -- sort or merge records (lines) of text and binary files

SYNOPSIS
    sort [-bcCdfghiRMmnrsuVz] [-k field1[,field2]] [-S memsize] [-T dir] [-t char] [-o output]
          [file ...]
    sort --help
    sort --version

DESCRIPTION
    The sort utility sorts text and binary files by lines. A line is a record separated from the subsequent record by a newline (default) or NUL '\0' character (-z option). A record can contain any printable or unprintable characters. Comparisons are based on one or more sort keys extracted from each line of input, and are performed lexicographically, according to the current locale's collating rules and the specified command-line options that can tune the actual sorting behavior. By default, if keys are not given, sort uses entire lines for comparison.

    The command line options are as follows:

    -c, --check, -C, --check-silent|quiet
        Check that the single input file is sorted. If the file is not sorted, sort produces the appropriate error messages and exits with code 1, otherwise returns 0. If -C or --check=silent is specified, sort produces no output. This is a "silent" version of -c.

    -m, --merge
        Merge only. The input files are assumed to be pre-sorted. If they are not sorted the output order is undefined.

    -o output, --output=output
        Print the output to the output file instead of the standard output.

    -S size, --buffer-size=size
        Use size for the maximum size of the memory buffer. Size modifiers %,b,K,M,G,T,P,E,Z,Y can be used. If a memory limit is not explicitly specified, sort takes up to about 90% of available memory. If the file size is too big to fit into the memory buffer, the temporary disk files are used to perform the sorting.

    -T dir, --temporary-directory=dir
        Store temporary files in the directory dir. The default path is the value of the environment variable TMPDIR or /var/tmp if TMPDIR is not defined.

    -u, --unique
        Unique keys. Suppress all lines that have a key that is equal to an already processed one. This option, similarly to -s, implies a stable sort. If used with -c or -C, sort also checks that there are no lines with duplicate keys.

    -s
        Stable sort. This option maintains the original record order of records that have an equal key. This is a non-standard feature, but it is widely accepted and used.

    --version
        Print the version and silently exits.

:

```

El comando uniq de Linux

uniq es un comando que te ayuda a ordenar las líneas de texto.

Puedes obtener esas líneas de un archivo, o usando pipes de la salida de otro comando:

```
uniq dogs.txt
```

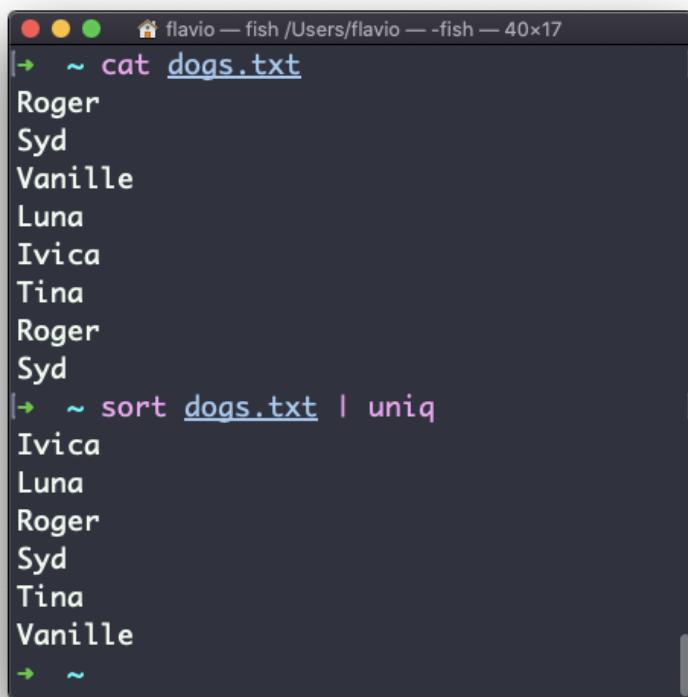
```
ls | uniq
```

Debes considerar esta clave: **uniq** sólo detectará las líneas duplicadas adyacentes.

```
sort dogs.txt | uniq
```

El comando `sort` tiene su propia manera de remover duplicar con la opción `-u` (*unique*). Pero `uniq` tiene más poder.

Por defecto remueve líneas duplicadas:



The screenshot shows a terminal window with the following content:

```
flavio — fish /Users/flavio — -fish — 40x17
|~ cat dogs.txt
Roger
Syd
Vanille
Luna
Ivica
Tina
Roger
Syd
|~ sort dogs.txt | uniq
Ivica
Luna
Roger
Syd
Tina
Vanille
|~
```

The terminal window has a dark background and light-colored text. It shows the command `cat dogs.txt` followed by its output, which includes multiple instances of names like Roger, Syd, and Vanille. Below that, the command `sort dogs.txt | uniq` is run, and its output shows that the duplicates have been removed, leaving only one instance of each name.

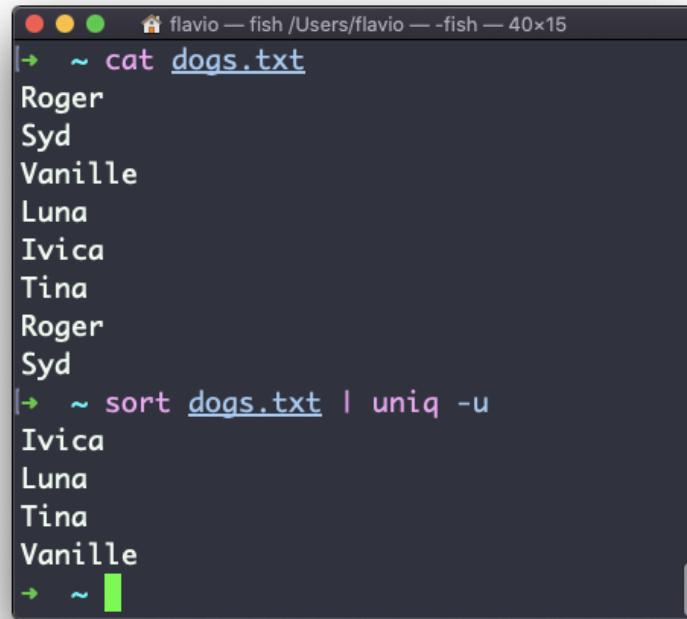
Puedes decirle que sólo muestre líneas duplicadas, por ejemplo, con la opción `-d`:

```
sort dogs.txt | uniq -d
```



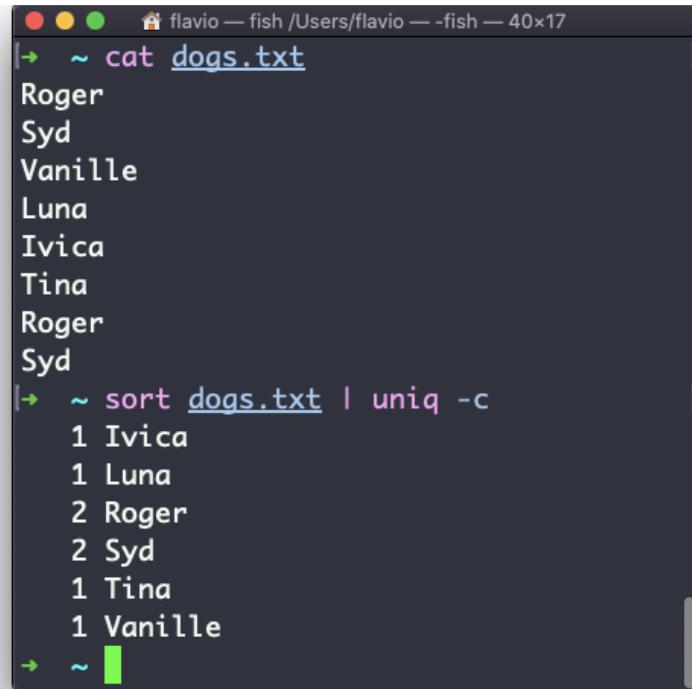
```
flavio — fish /Users/flavio — -fish — 40x13
|~ cat dogs.txt
Roger
Syd
Vanille
Luna
Ivica
Tina
Roger
Syd
|~ sort dogs.txt | uniq -d
Roger
Syd
|~
```

Puedes usar la opción `-u` para sólo mostrar líneas no duplicadas:



```
flavio — fish /Users/flavio — -fish — 40x15
|~ cat dogs.txt
Roger
Syd
Vanille
Luna
Ivica
Tina
Roger
Syd
|~ sort dogs.txt | uniq -u
Ivica
Luna
Tina
Vanille
|~
```

Puedes contar las ocurrencias de cada línea con la opción `-c` :

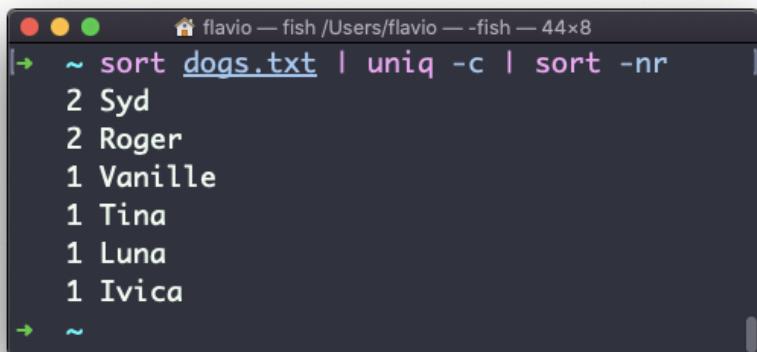


```
flavio — fish /Users/flavio — -fish — 40x17
|~ cat dogs.txt
Roger
Syd
Vanille
Luna
Ivica
Tina
Roger
Syd
|~ sort dogs.txt | uniq -c
1 Ivica
1 Luna
2 Roger
2 Syd
1 Tina
1 Vanille
|~
```

Puedes usar la combinación especial:

```
sort dogs.txt | uniq -c | sort -nr
```

para luego ordenar esas líneas por su mayor frecuencia:



```
flavio — fish /Users/flavio — -fish — 44x8
|~ sort dogs.txt | uniq -c | sort -nr
2 Syd
2 Roger
1 Vanille
1 Tina
1 Luna
1 Ivica
|~
```

EL COMANDO UTIL DE LINUX

`diff` es un comando útil. Supongamos que tienes 2 archivos, que contienen casi la misma información, pero que no puedes encontrar la diferencia entre ellos.

`diff` procesará los archivo y te dirá la diferencia.

Supongamos que tienes 2 archivos: `dogs.txt` y `moredogs.txt`. La diferencia es que `moredogs.txt` contiene un nombre de perro más:



```
flavio — fish /Users/flavio — -fish — 50x8
|→ ~ cat dogs.txt
Roger
Syd
|→ ~ cat moredogs.txt
Roger
Syd
Vanille
→ ~
```

`diff dogs.txt moredogs.txt` te dirá que el segundo archivo tiene una línea más, línea 3 con la línea `Vanille`:



```
flavio — fish /Users/flavio — -fish — 50x5
|→ ~ diff dogs.txt moredogs.txt
2a3
> Vanille
→ ~
```

Si inviertes el orden de los archivos, te dirá que el segundo archivo le falta la línea 3, cuyo contenido es `Vanille`:

```
flavio — fish /Users/flavio — -fish — 50x5
~ diff moredogs.txt dogs.txt
3d2
< Vanille
→ ~
```

Usa la opción `-y` que comparará los 2 archivos línea por línea:

```
flavio — fish /Users/flavio — -fish — 86x9
~ diff -y dogs.txt moredogs.txt
Roger          Roger
Syd           Syd
> Vanille
```

La opción `-u`, sin embargo, te resultará familiar, porque es la misma que utiliza el sistema de control de versiones de Git para mostrar las diferencias entre versiones:

```
flavio — fish /Users/flavio — -fish — 67x8
~ diff -u dogs.txt moredogs.txt
--- dogs.txt    2020-09-07 08:54:56.000000000 +0200
+++ moredogs.txt      2020-09-07 08:55:09.000000000 +0200
@@ -1,2 +1,3 @@
Roger
Syd
+Vanille
→ ~
```

Comparar directorios funciona de la misma manera. Debes usar la opción `-r` para comparar de forma recursiva (yendo a subdirectorios):



```
testing — fish /Users/flavio/testing — -fish — 68x13
|→ testing ls dir1
dogs.txt
|→ testing ls dir2
dogs.txt
|→ testing diff -u dir1 dir2
diff -u dir1/dogs.txt dir2/dogs.txt
--- dir1/dogs.txt      2020-09-07 08:54:56.000000000 +0200
+++ dir2/dogs.txt      2020-09-07 08:55:09.000000000 +0200
@@ -1,2 +1,3 @@
    Roger
    Syd
+Vanille
→ testing
```

En caso de que estés interesado en saber qué archivos difieren, más que el contenido, utiliza las opciones `r` y `q`:



```
testing — fish /Users/flavio/testing — -fish — 68x5
|→ testing diff -rq dir1 dir2
Files dir1/dogs.txt and dir2/dogs.txt differ
→ testing
```

Hay muchas más opciones que puedes explorar en la página de manual al correr `man diff`:

```

diff(1)                               User Commands                            diff(1)

NAME
    diff - compare files line by line

SYNOPSIS
    diff [OPTION]... FILES

DESCRIPTION
    Compare files line by line.

    -i  --ignore-case
        Ignore case differences in file contents.

    --ignore-file-name-case
        Ignore case when comparing file names.

    --no-ignore-file-name-case
        Consider case when comparing file names.

    -E  --ignore-tab-expansion
        Ignore changes due to tab expansion.

    -b  --ignore-space-change
        Ignore changes in the amount of white space.

    -w  --ignore-all-space
        Ignore all white space.

    -B  --ignore-blank-lines
        Ignore changes whose lines are all blank.

    -I RE  --ignore-matching-lines=RE
        Ignore changes whose lines all match RE.

    --strip-trailing-cr
        Strip trailing carriage return on input.

    -a  --text
        Treat all files as text.

    -c  -C NUM  --context[=NUM]
        Output NUM (default 3) lines of copied context.

    -u  -U NUM  --unified[=NUM]
        Output NUM (default 3) lines of unified context.

    --label LABEL
        Use LABEL instead of file name.

    -p  --show-c-function
        Show which C function each change is in.

    -F RE  --show-function-line=RE
:
```

El comando echo de Linux

El comando `echo` hace un simple trabajo: imprime en la salida el argumento que se le ha pasado.

Este ejemplo:

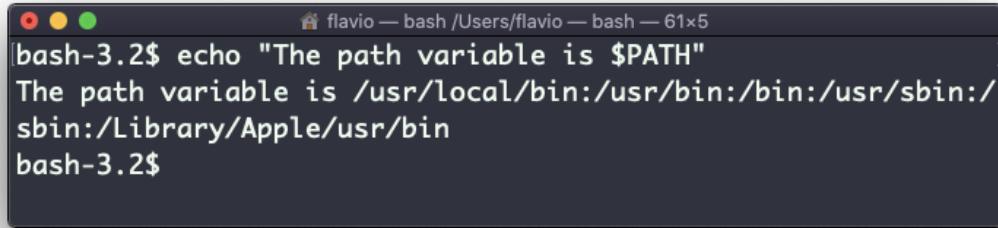
```
echo "hello"
```

imprimirá `hello` en la terminal.

```
echo "hello" >> output.txt
```

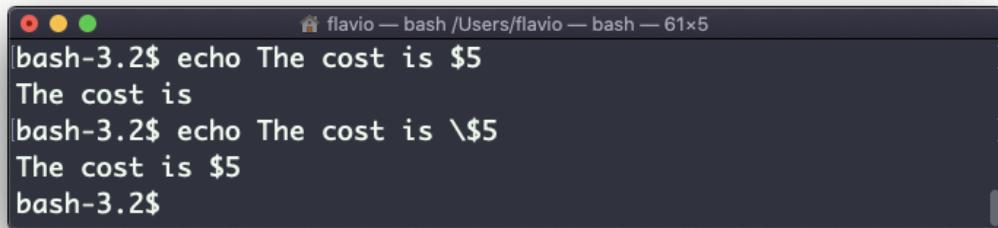
Podemos interpolar variable de ambiente:

```
echo "The path variable is $PATH"
```



```
flavio — bash /Users/flavio — bash — 61x5
bash-3.2$ echo "The path variable is $PATH"
The path variable is /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/Apple/usr/bin
bash-3.2$
```

Tenga cuidado con los caracteres especiales que deben ser escapados con una barra invertida \ . \\$ por ejemplo:



```
flavio — bash /Users/flavio — bash — 61x5
bash-3.2$ echo The cost is $5
The cost is
bash-3.2$ echo The cost is \$5
The cost is $5
bash-3.2$
```

Esto es solo el comienzo. Podemos hacer buenas cosas cuando se trata de interactuar con las características del shell.

Podemos realizar echo a los archivos en la carpeta actual:

```
echo *
```

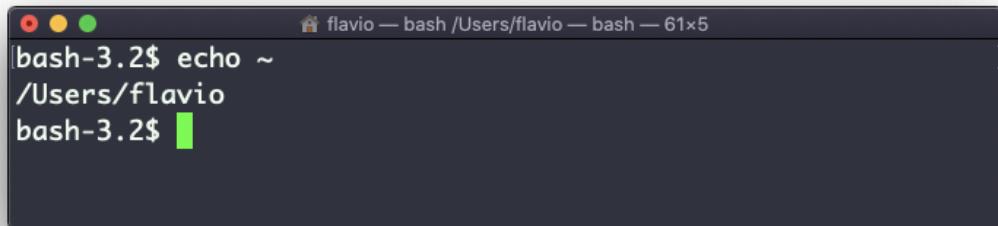
Podemos realizar `echo` a los archivos en la carpeta actual que inician con la letra `o`:

```
echo o*
```

Cualquier comando Bash (o de cualquier shell que estés usando) y característica pueden ser usados aquí.

Puedes imprimir la ruta de la carpeta `home`:

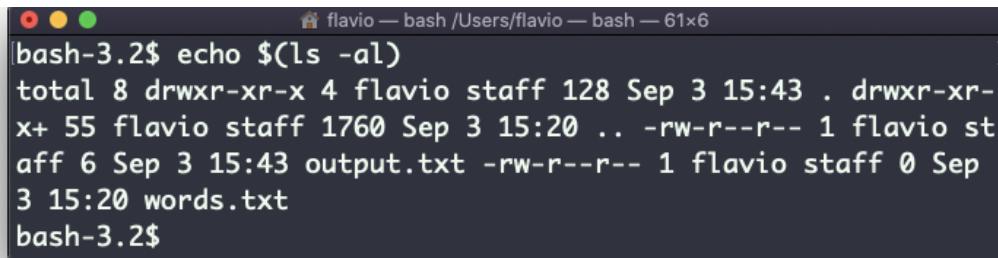
```
echo ~
```



A screenshot of a macOS terminal window titled "flavio — bash /Users/flavio — bash — 61x5". The window shows the command "bash-3.2\$ echo ~" followed by the output "/Users/flavio". The cursor is at the end of the line.

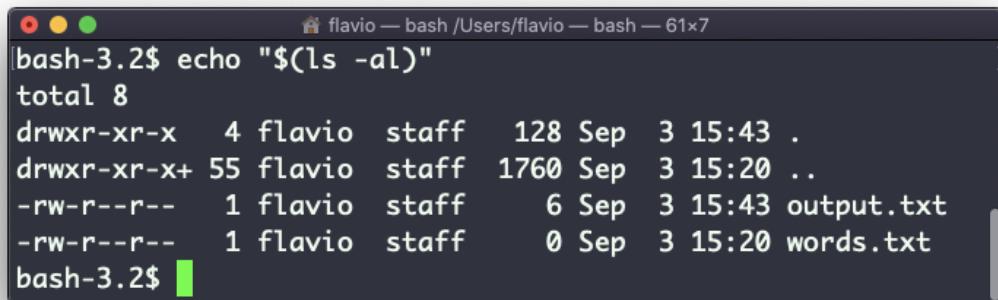
También puedes ejecutar comandos, e imprimir el resultado como salida estándar (o a un archivo, como veas):

```
echo $(ls -al)
```



```
flavio — bash /Users/flavio — bash — 61x6
|bash-3.2$ echo $(ls -al)
total 8 drwxr-xr-x 4 flavio staff 128 Sep 3 15:43 .
drwxr-xr-x+ 55 flavio staff 1760 Sep 3 15:20 ..
-rw-r--r-- 1 flavio staff 6 Sep 3 15:43 output.txt
-rw-r--r-- 1 flavio staff 0 Sep 3 15:20 words.txt
bash-3.2$
```

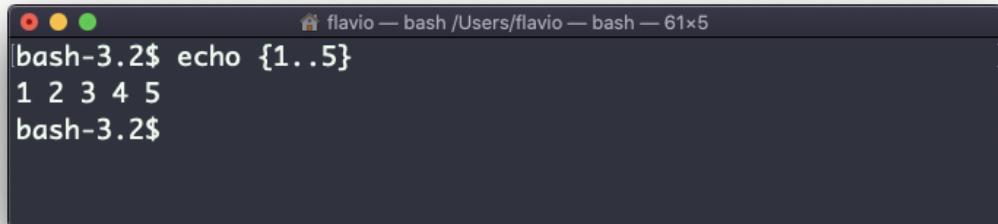
Ten en cuenta que los espacios en blanco no se conservan por defecto. Necesitas envolver el comando entre comillas para hacerlo:



```
flavio — bash /Users/flavio — bash — 61x7
|bash-3.2$ echo "$(ls -al)"
total 8
drwxr-xr-x 4 flavio staff 128 Sep 3 15:43 .
drwxr-xr-x+ 55 flavio staff 1760 Sep 3 15:20 ..
-rw-r--r-- 1 flavio staff 6 Sep 3 15:43 output.txt
-rw-r--r-- 1 flavio staff 0 Sep 3 15:20 words.txt
bash-3.2$
```

Puedes generar una lista de cadenas, por ejemplo rangos:

```
echo {1..5}
```



```
flavio — bash /Users/flavio — bash — 61x5
|bash-3.2$ echo {1..5}
1 2 3 4 5
bash-3.2$
```

El comando chown de Linux

Cada archivo/directorio en un sistema operativo como Linux o macOS (y cada sistema UNIX en general) tiene un **propietario**.

El propietario de un archivo puede hacer todo con él. Puede decidir el destino de ese archivo.

El propietario (y el usuario `root`) puede cambiar el propietario a otro usuario, también, usando el comando `chown`:

```
chown <owner> <file>
```

Así:

```
chown flavio test.txt
```

Por ejemplo, si tienes un archivo que es propiedad de `root`, no puedes escribir en él como otro usuario:



A screenshot of a macOS terminal window titled "flavio — fish /Users/flavio — -fish — 65x7". The window shows the following command history:

```
|~ sudo touch test.txt
|~ echo test >> test.txt
<W> fish: An error occurred while redirecting file 'test.txt'
open: Permission denied
|~
```

Puedes usar `chown` para transferirte la propiedad:



A screenshot of a terminal window titled "flavio — fish /Users/flavio — -fish — 65x7". The terminal shows the following session:

```
~ sudo touch test.txt
~ echo test >> test.txt
<W> fish: An error occurred while redirecting file 'test.txt'
open: Permission denied
~ sudo chown flavio test.txt
~ echo test >> test.txt
~
```

Es bastante común la necesidad de cambiar la propiedad de un directorio, y recursivamente todos los archivos dentro, además de todos los subdirectorios y los archivos contenidos en ellos también.

Puedes hacer esto usando la bandera `-R`:

```
chown -R <owner> <file>
```

Los archivos/directorios no sólo tienen un propietario, también tienen un grupo. A través de este comando puedes cambiar eso simultáneamente mientras cambias el propietario:

```
chown <owner>:<group> <file>
```

Por ejemplo:

```
chown flavio:users test.txt
```

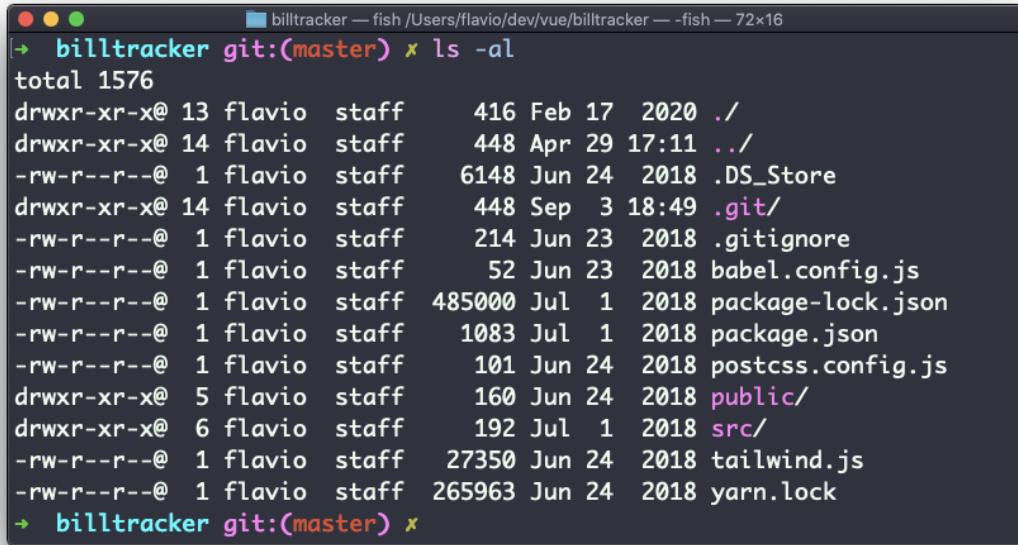
También puedes cambiar el grupo de un archivo usando el comando `chgrp`:

```
chgrp <group> <filename>
```

El comando chmod de Linux

tiene 3 permisos: leer, escribir y ejecutar.

Ve a una carpeta, y corre el comando `ls -al`:



```
billtracker — fish /Users/flavio/dev/vue/billtracker — -fish — 72x16
[+ billtracker git:(master) ✘ ls -al
total 1576
drwxr-xr-x@ 13 flavio  staff   416 Feb 17  2020 .
drwxr-xr-x@ 14 flavio  staff   448 Apr 29 17:11 ..
-rw-r--r--@  1 flavio  staff  6148 Jun 24  2018 .DS_Store
drwxr-xr-x@ 14 flavio  staff   448 Sep  3 18:49 .git/
-rw-r--r--@  1 flavio  staff   214 Jun 23  2018 .gitignore
-rw-r--r--@  1 flavio  staff   52 Jun 23  2018 babel.config.js
-rw-r--r--@  1 flavio  staff  485000 Jul  1  2018 package-lock.json
-rw-r--r--@  1 flavio  staff   1083 Jul  1  2018 package.json
-rw-r--r--@  1 flavio  staff   101 Jun 24  2018 postcss.config.js
drwxr-xr-x@  5 flavio  staff   160 Jun 24  2018 public/
drwxr-xr-x@  6 flavio  staff   192 Jul  1  2018 src/
-rw-r--r--@  1 flavio  staff  27350 Jun 24  2018 tailwind.js
-rw-r--r--@  1 flavio  staff  265963 Jun 24  2018 yarn.lock
→ billtracker git:(master) ✘
```

Las extrañas cadenas que ves en cada línea del archivo, como `drwxr-xr-x`, definen los permisos del archivo o carpeta.

Vamos a diseccionarlo.

La primera letra indica el tipo de archivo:

- `-` significa que es un archivo normal
- `d` significa que es un directorio
- `l` significa que es un enlace

Luego tienes 3 conjuntos de valores:

- El primer conjunto representa los permisos del **propietario** del archivo
- El segundo conjunto representa los permisos de los miembros del **grupo** al que el archivo está asociado
- El tercer conjunto representa los permisos de **todos los demás**

acceso de lectura, escritura y ejecución. Todo lo que se elimina se intercambia con un `-`, que permite formar varias combinaciones de valores y permisos relativos: `rw-` , `r--` , `r-x` , y así sucesivamente.

Puedes cambiar los permisos dados a un archivo usando el comando `chmod` .

`chmod` puede ser usado de dos maneras. La primera es usando argumento simbólicos, la segunda es usando argumentos numéricos. Empecemos con los símbolos primero, lo cual es más intuitivo.

Escribes `chmod` seguido de un espacio y una letra:

- `a` significa *todos*
- `u` significar *usuario*
- `g` significa *grupo*
- `o` significa *otros*

Luego escribes `+ o -` para agregar permiso, o para eliminarlo. Después ingresas uno o más símbolos de permiso (`r` , `w` , `x`).

Todo seguido por el nombre del archivo o la carpeta.

Aquí hay algunos ejemplos:

```
chmod a+r filename # todos pueden leerlo  
chmod a+rw filename # todos puende leerlo y escribirlo  
chmod o-rwx filename # otros (no el propietario, no en el mismo grupo del archivo) no pueden leerlo
```

Puedes aplicar los mismos permisos a múltiples personas al agregar múltiples letras antes de `+ / - :`

```
chmod og-r filename # otros y el grupo no pueden leer el archivo
```

En caso de que estés editando una carpeta, puedes aplicar los permisos a cada archivo dentro de esa carpeta usando la bandera `-r` (recursiva).

a día. Usas un dígito que representa los permisos de la persona. Este valor numérico puede ser un máximo de 7, y se calcula de esta manera:

- 1 si tiene permiso de ejecución
- 2 si tiene permiso de escritura
- 4 si tiene permiso de lectura

Esto nos da 4 combinaciones:

- 0 no tiene permisos
- 1 puede ejecutar
- 2 puede leer
- 3 puede escribir, ejecutar
- 4 puede leer
- 5 puede leer, ejecutar
- 6 puede leer, escribir
- 7 puede leer, escribir y ejecutar

Los usamos en pares de 3, para establecer los permisos de los 3 grupos en conjunto:

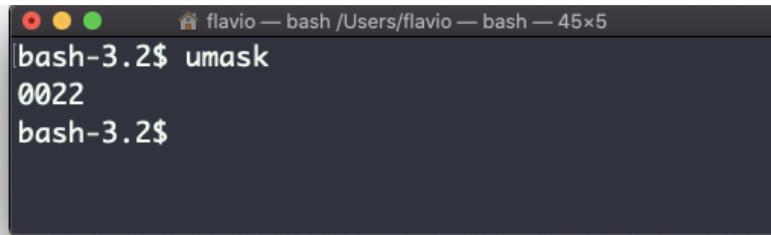
```
chmod 777 filename  
chmod 755 filename  
chmod 644 filename
```

El comando umask de Linux

Cuando creas un archivo, no tienes que decidir los permisos por adelantado. Los permisos tienen valores por defecto.

Esos valores por defecto pueden ser controlados y modificados usando el comando `umask`.

Escribir `umask` sin argumentos te mostrará el umask actual, en este caso `0022`:

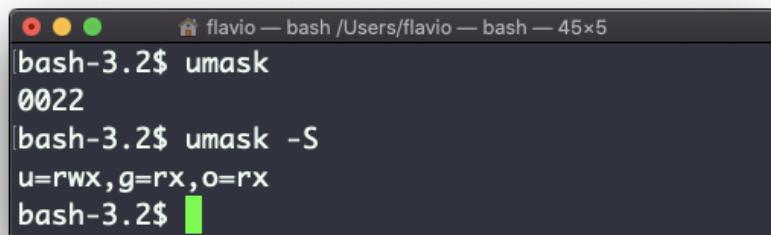


```
flavio — bash /Users/flavio — bash — 45x5
|bash-3.2$ umask
0022
bash-3.2$
```

¿Qué significa 0222 ? Es un valor octal que representa los permisos.

Otro valor común es 0002 .

Usa `umask -S` para ver una notación legible por los humanos:



```
flavio — bash /Users/flavio — bash — 45x5
|bash-3.2$ umask
0022
|bash-3.2$ umask -S
u=rwx,g=rx,o=rx
bash-3.2$ █
```

En este caso, el usuario (u), propietario del archivo, tiene permisos de lectura, escritura y ejecución en los archivos.

Otros usuarios pertenecientes al mismo grupo (g) tienen permiso de lectura y ejecución, al igual que todos los demás usuarios (o).

En la notación numérica, normalmente cambiamos los últimos 3 dígitos.

Aquí hay una lista que da un significado al número:

- 0 lectura, escritura y ejecución
- 1 lectura y escritura
- 2 lectura y ejecución

- 4 escritura y ejecución
- 5 sólo escritura
- 6 sólo ejecución
- 7 sin permisos

Nota que esta notación numérica difiere de la que usamos en `chmod`.

Podemos establecer un nuevo valor para la máscara fijando el valor en formato numérico:

```
umask 002
```

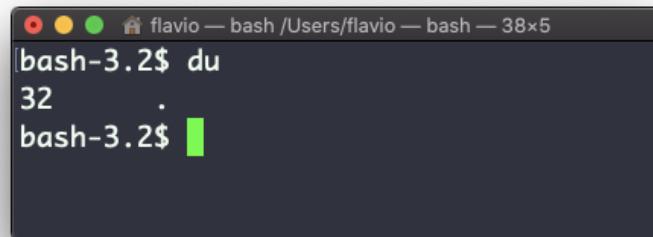
o puedes cambiar el permiso de un rol específico:

```
umask g+r
```

El comando `du` de Linux

El comando `du` calculará el tamaño de un directorio en su conjunto:

```
du
```

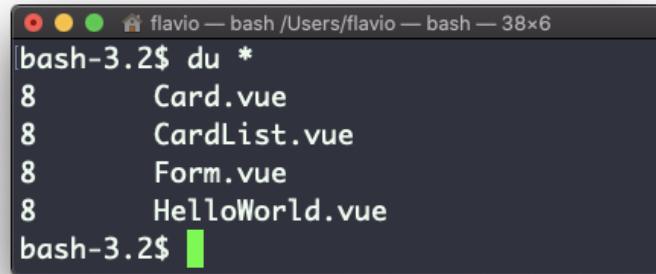


A screenshot of a terminal window titled "flavio — bash /Users/flavio — bash — 38x5". The window shows the command `du` being run, followed by the output "32 .". The terminal has a dark background with white text.

```
flavio — bash /Users/flavio — bash — 38x5
[bash-3.2$ du
32 .
bash-3.2$ ]
```

El número 32 aquí es un valor expresado en bytes.

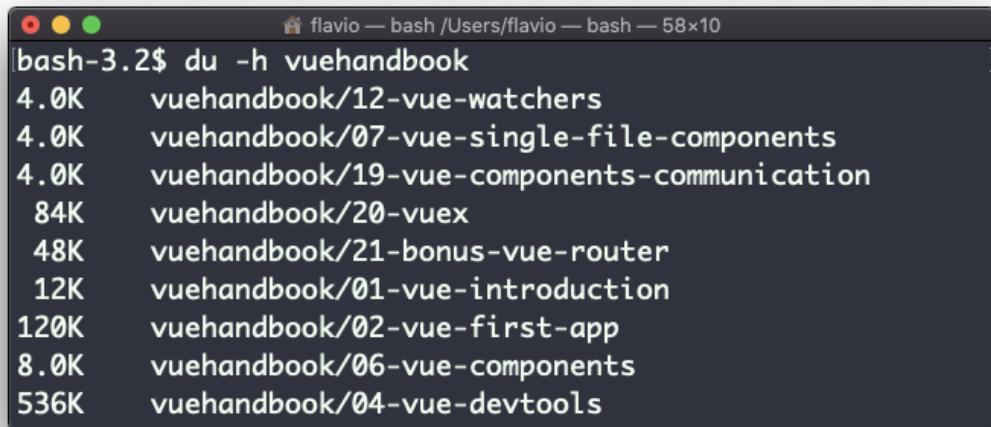
A continuación se calculará el tamaño de cada archivo individualmente.



```
flavio — bash /Users/flavio — bash — 38x6
bash-3.2$ du *
8      Card.vue
8      CardList.vue
8      Form.vue
8      HelloWorld.vue
bash-3.2$
```

Puedes configurar `du` para que muestre los valores en MegaBytes usando `du -m`, y GigaBytes usando `du -g`.

La opción `-h` mostrará una notación legible por los humanos para los tamaños, adaptándose al tamaño:



```
flavio — bash /Users/flavio — bash — 58x10
bash-3.2$ du -h vuehandbook
4.0K    vuehandbook/12-vue-watchers
4.0K    vuehandbook/07-vue-single-file-components
4.0K    vuehandbook/19-vue-components-communication
84K     vuehandbook/20-vuex
48K     vuehandbook/21-bonus-vue-router
12K     vuehandbook/01-vue-introduction
120K    vuehandbook/02-vue-first-app
8.0K    vuehandbook/06-vue-components
536K    vuehandbook/04-vue-devtools
```

Agregando la opción `-a` también imprimirá el tamaño de cada uno de los archivos en los directorios:

```
flavio — bash /Users/flavio — bash — 65x14
[bash-3.2$ du -ah vuehandbook
4.0K    vuehandbook/12-vue-watchers/index.md
4.0K    vuehandbook/12-vue-watchers
4.0K    vuehandbook/07-vue-single-file-components/index.md
4.0K    vuehandbook/07-vue-single-file-components
4.0K    vuehandbook/19-vue-components-communication/index.md
4.0K    vuehandbook/19-vue-components-communication
36K     vuehandbook/20-vuex/vuex-store.png
12K     vuehandbook/20-vuex/index.md
36K     vuehandbook/20-vuex/codesandbox.png
84K     vuehandbook/20-vuex
12K     vuehandbook/.DS_Store
32K     vuehandbook/21-bonus-vue-router/banner.jpg
16K     vuehandbook/21-bonus-vue-router/index.md
```

Una cosa útil es clasificar los directorios por tamaño:

```
du -h <directory> | sort -nr
```

y luego agregar un pipe a `head` para sólo obtener los primeros 10 resultados:

```
flavio — bash /Users/flavio — bash — 63x13
[bash-3.2$ du -h vuehandbook | sort -nr | head
932K    vuehandbook/05-vue-vscode
636K    vuehandbook/.git/objects/75
544K    vuehandbook/03-vue-cli
536K    vuehandbook/04-vue-devtools
120K    vuehandbook/02-vue-first-app
88K     vuehandbook/.git/objects/pack
88K     vuehandbook/.git/objects/b0
84K     vuehandbook/20-vuex
76K     vuehandbook/.git/objects/6f
64K     vuehandbook/.git/objects/41
bash-3.2$
```

El comando `df` se usa para obtener información sobre el uso del disco.

Su forma básica imprimirá información sobre los volúmenes montados:

```
flavio — fish /Users/flavio — -fish — 106x9
~ df
Filesystem 512-blocks Used Available Capacity iused ifree %iused Mounted on
/dev/disk1s1 976490576 21974760 487735816 5% 488418 4881964462 0% /
devfs 375 375 0 100% 649 0 100% /dev
/dev/disk1s2 976490576 454785000 487735816 49% 4287975 4878164905 0% /System/Volumes/Data
/dev/disk1s5 976490576 10485848 487735816 3% 5 4882452875 0% /private/var/vm
map auto_home 0 0 0 100% 0 0 100% /System/Volumes/Data/home
~
```

Usando la opción `-h` (`df -h`) mostrará los valores en un formato legible para los humanos:

```
flavio — fish /Users/flavio — -fish — 100x9
~ df -h
Filesystem Size Used Avail Capacity iused ifree %iused Mounted on
/dev/disk1s1 466Gi 10Gi 233Gi 5% 488418 4881964462 0% /
devfs 188Ki 188Ki 0Bi 100% 649 0 100% /dev
/dev/disk1s2 466Gi 217Gi 233Gi 49% 4287984 4878164896 0% /System/Volumes/Data
/dev/disk1s5 466Gi 5.0Gi 233Gi 3% 5 4882452875 0% /private/var/vm
map auto_home 0Bi 0Bi 0Bi 100% 0 0 100% /System/Volumes/Data/home
~
```

También puedes especificar el nombre de un archivo o directorio para obtener información sobre el volumen específico del que vive:

```
flavio — fish /Users/flavio — -fish — 109x6
~ df dev
Filesystem 512-blocks Used Available Capacity iused ifree %iused Mounted on
/dev/disk1s2 976490576 454788176 487732640 49% 4287987 4878164893 0% /System/Volumes/Data
~
```

El comando basename de Linux

Supongamos que tienes una ruta a un archivo, por ejemplo `/Users/flavio/test.txt`.

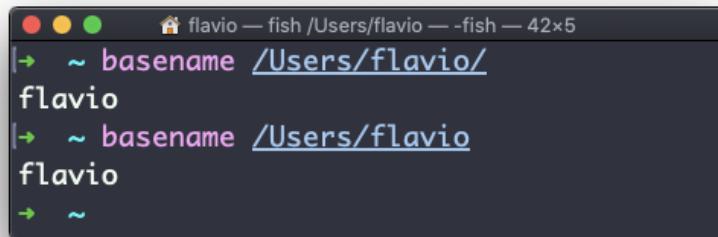
```
basename /Users/flavio/test.txt
```

retornará la cadena `text.txt`:



A screenshot of a macOS terminal window titled "flavio — fish /Users/flavio — -fish — 42x5". The window shows the command `~ basename /Users/flavio/test.txt` and its output `test.txt`. The terminal has a dark theme with red, yellow, and green window controls.

Si ejecutas `basename` en una cadena de rutas que apunta a un directorio, obtendrás el último segmento de la ruta. En este ejemplo, `Users/flavio` es un directorio:



A screenshot of a macOS terminal window titled "flavio — fish /Users/flavio — -fish — 42x5". The window shows two commands: `~ basename /Users/flavio/` followed by `flavio`, and `~ basename /Users/flavio` followed by `flavio`. Both commands return the directory name `flavio`. The terminal has a dark theme with red, yellow, and green window controls.

El comando `dirname` de Linux

Supongamos que tienes una ruta a un archivo, por ejemplo `Users/flavio/test.txt`.

Al ejecutar

retornará la cadena `/Users/flavio` :

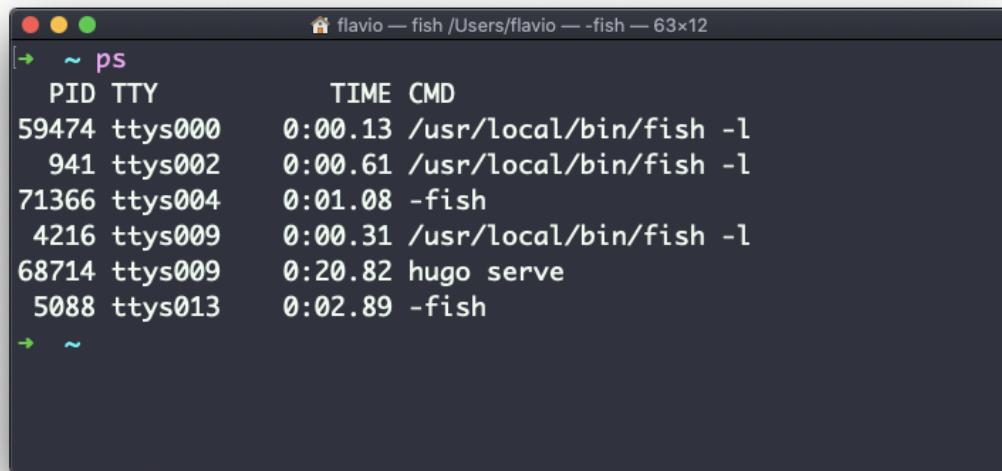


```
flavio — fish /Users/flavio — -fish — 44x5
[~] ~ dirname /Users/flavio/test.txt
[~/Users/flavio]
[~]
```

El comando ps de Linux

Tu computadora ejecuta un montón de diferentes procesos todo el tiempo.

Puedes inspeccionarlos a todos usando el comando `ps` :



```
flavio — fish /Users/flavio — -fish — 63x12
[~] ~ ps
  PID TTY          TIME CMD
 59474 ttys000  0:00.13 /usr/local/bin/fish -l
    941 ttys002  0:00.61 /usr/local/bin/fish -l
 71366 ttys004  0:01.08 -fish
 4216 ttys009  0:00.31 /usr/local/bin/fish -l
 68714 ttys009  0:20.82 hugo serve
 5088 ttys013  0:02.89 -fish
[~]
```

Esta es la lista de procesos iniciados por el usuario que se están ejecutando actualmente en la sesión en curso.

editor, y una instancia de Hugo ejecutando la vista previa de desarrollo de un sitio.

Esos son sólo los comandos asignados al usuario actual. Para enlistar todos los procesos necesitamos pasarle algunas opciones a `ps`.

La más común que uso es `ps ax`:

PID	TT	STAT	TIME	COMMAND
1	??	Ss	43:24.80	/sbin/launchd
92	??	Ss	2:03.80	/usr/sbin/syslogd
93	??	Ss	4:56.03	/usr/libexec/UserEventAgent (System)
96	??	Ss	0:18.74	/System/Library/PrivateFrameworks/Uninstall.framework/Resources
97	??	Ss	1:36.94	/usr/libexec/kextd
98	??	Ss	12:31.61	/System/Library/Frameworks/CoreServices.framework/Versions
99	??	Ss	0:21.48	/System/Library/PrivateFrameworks/MediaRemote.framework/Su
102	??	Ss	22:56.23	/usr/sbin/systemstats --daemon
103	??	Ss	2:25.78	/usr/libexec/configd
105	??	Ss	5:32.22	/System/Library/CoreServices/powerd.bundle/powerd
109	??	Rs	9:48.23	/usr/libexec/logd
110	??	Ss	0:01.46	/usr/libexec/keybagd -t 15
113	??	Ss	0:31.41	/usr/libexec/watchdogd
117	??	Ss	44:38.55	/System/Library/Frameworks/CoreServices.framework/Frameworks
118	??	Ss	0:00.55	/System/Library/CoreServices/iconservicesd
119	??	Ss	0:49.70	/usr/libexec/diskarbitrationd
123	??	Ss	1:09.30	/usr/libexec/coreduetd
126	??	Ss	7:31.22	/usr/libexec/opendirectoryd
127	??	Ss	0:51.05	/System/Library/PrivateFrameworks/ApplePushService.framework
128	??	Ss	0:00.54	/Library/PrivilegedHelperTools/com.docker.vmnetd
129	??	Ss	19:37.63	/System/Library/CoreServices/launchservicesd
130	??	Ss	0:14.36	/usr/libexec/timed

La opción `a` se utiliza para enumerar también los procesos de otros usuarios, no sólo los tuyos propios. `x` muestra los procesos no vinculados a una terminal (no iniciados por los usuarios a través de una terminal).

Como puedes ver, los comandos más largos se cortan. Usa el comando `ps axww` para continuar el listado de comando en una nueva línea en lugar de cortarlo:

```
[~] ~ ps axww
 PID  TT  STAT      TIME COMMAND
   1  ??  Ss    43:25.81 /sbin/launchd
   92  ??  Ss    2:03.82 /usr/sbin/syslogd
   93  ??  Ss    4:56.05 /usr/libexec/UserEventAgent (System)
   96  ??  Ss    0:18.74 /System/Library/PrivateFrameworks/Uninstall.framework/Resources/uninstallld
   97  ??  Ss    1:36.94 /usr/libexec/kextd
   98  ??  Ss    12:31.92 /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/FSEvents.framework/Versions/A/Support/fsevents
   99  ??  Ss    0:21.48 /System/Library/PrivateFrameworks/MediaRemote.framework/Support/mediaremoved
  102  ??  Ss    22:56.23 /usr/sbin/systemstats --daemon
  103  ??  Ss    2:25.80 /usr/libexec/configd
  105  ??  Ss    5:32.38 /System/Library/CoreServices/powerd.bundle/powerd
  109  ??  Ss    9:48.48 /usr/libexec/logd
  110  ??  Ss    0:01.46 /usr/libexec/keybagd -t 15
  113  ??  Ss    0:31.42 /usr/libexec/watchdogd
  117  ??  Ss    44:38.88 /System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/Support/mds
  118  ??  Ss    0:00.55 /System/Library/CoreServices/iconservicesd
```

Necesitamos especificar `w` 2 veces para aplicar este ajuste (no es un error tipográfico).

Puedes buscar un proceso específico que combine el comando `grep` con un pipe, como este:

```
ps axww | grep "Visual Studio Code"
```

```
[~] ~ ps axww | grep "Visual Studio Code"
367 ?? S 40:32.47 /Applications/Visual Studio Code.app/Contents/MacOS/Electron -psn_0_77843
566 ?? S 93:02.21 /Applications/Visual Studio Code.app/Contents/Frameworks/Code Helper (GPU).app/Contents/MacOS/Code Helper (GPU) --type=gpu-process --field-trial-handle=1718379636,8422946071360466993,8181735644596197527,131072 --disable-features=LayoutNG,PictureInPicture,SpareRendererForSitePerProcess --disable-color-correct-rendering --gpu-preferences=KAAAAAAAqAaaaaaaaaaaaaAYaaaaAAEaaaaaaaaaaaaAAAgBAAAqAAAAAAEaaaaaaaaaaIAQaaaaAAABAAAAAAAGAEaaaaAAAgAQAAAAAAcGBAaaaaAAmaAAAAAA4AQaaaaAAEABAaaaaAAASAEaaaaAAABQAQAAAAAAfGbaaaaaAAAYAEaaaaAAboAQAAAAAAAHABAaaaaAAeAEaaaaAAACAAQaaaaAAAtIgBAAAAAAkAEaaaaAAcyAQaaaaAAkABAaaaaAAqAEaaaaAAcWqAaaaaAAALgBAAAAAAAwAEaaaaAAADIAQaaaaAAAnABAaaaaAA2AEaaaaAAAdgAQaaaaAAOgBAAAAAA8AEaaaaAAAD4AQaaaaAAABAAAAAAaaaaaaaaaaaaAAQaaaaAAACAAAAEaaaaAAABgAAABAAAAAAaaaaaaaaaaaaAcAAAAQAAAAAIAAAAEEaaaaAAACGbaABAAAAAAAsAAAAQAAAaaaaAAAnAAAAEaaaaAAABAAAAAAABAAAAAAABAAAAAAQAAAAQAAAIAAAAQAAAAAaaaaAAEAAAAGAAAAAAEAAAAABAAAABwAAAABAAAAAAQAAAqAAAAQAAAAAEEAAAakAAAEEAAAABAAAACWqAAAABAAAAAAQAAAQAAA0AAAQAAAAAAQAAAAAAEAAAAEAAAAAgAAABAAAAAAABAAAAAYAAAQAAAAAAQAAAHAaaaEAAAEEAAAAACAAAABAAAAAAABAAAAAABAAAoAAAQAAAAAAQAAAALAAAEEAAAAAAEAAAAdQAAABAAAAAAABgAAAAAAQAAAAAAQAAAAYAAAACAAAEEAAAAAAAGAAAAbgAAABAAAABAAAABgAAAAcAAAQAAAAAAAYAAAIAAAAEEAAAAAAAGAAAACGqAAAABAAAAAAABgqAAAAsAAAAQAAAAAAQAAAAYAAAANAAA --service-request-channel-token=4754214311225613582
```

Las columnas devueltas por `ps` representan alguna información clave.

La primera información es `PID`, la identificación del proceso (*process ID* en inglés). Esta es la clave cuando se quiere hacer referencia a este proceso en otro comando, por ejemplo, matarlo.

Luego tenemos `TT` que nos dice el ID de la terminal utilizada.

Después STAT nos dice el estado del proceso:

I Un proceso que está inactivo (dormido durante más de 20 segundos)

R un proceso ejecutable

S un proceso que duerme por menos de 20 segundos

T un proceso detenido

U un proceso en espera ininterrumpida

Z un proceso muerto (un zombi)

Si tienes más de una letra, la segunda representa más información, que puede ser muy técnica.

Es común tener el + que indica que el proceso está en primer plano en su terminal. La s significa que el proceso es un líder de sesión.

TIME nos dice cuánto tiempo ha estado funcionando el proceso.

El comando top

El comando `top` se utiliza para mostrar información dinámica en tiempo real sobre los procesos en ejecución en el sistema.

Es muy útil para entender lo que está pasando.

Su uso es simple — sólo tienes que escribir `top`, y la terminal estará completamente inmersa en esta nueva vista:

```

Processes: 574 total, 2 running, 572 sleeping, 3807 threads          11:39:53
Load Avg: 1.24, 1.93, 2.23 CPU usage: 4.61% user, 3.68% sys, 91.69% idle
SharedLibs: 234M resident, 62M data, 16M linkedit.
MemRegions: 369828 total, 6215M resident, 163M private, 1603M shared.
PhysMem: 14G used (3684M wired), 1742M unused.
VM: 3080G vsiz, 1991M framework vsiz, 316459873(128) swapins, 322451178(0) swa
Networks: packets: 41309179/63G in, 44958830/24G out.
Disks: 68270763/1950G read, 43422129/1680G written.

      PID   COMMAND    %CPU TIME #TH #WQ #PORT MEM PURG CMPRS PGRP
229 WindowServer 16.1 13:11:15 10    4 6607+ 845M+ 1216K- 173M 229
1185 com.docker.h 7.6 15:21:25 18    0 42 2374M 0B 316M 1162
82661 top         7.4 00:01.00 1/1   0 27+ 8352K+ 0B 0B 82661
381 Terminal     6.0 04:31.43 11    4 436 105M 14M 28M- 381
1152 1Password    5.6 02:00:08 8     2 2473- 530M- 244K 349M 1152
0 kernel_task   3.9 06:15:45 262/12 0     0 510M+ 0B 0B 0
82622 Google Chrom 2.8 00:07.20 16    1 209 81M 4096B 0B 38992
377 Bear         2.1 41:08.16 12    5 1760- 557M- 13M- 460M- 377
98108 Books       2.1 38:16.24 6     1 822 302M 0B 286M 98108
347 cloudd       2.0 10:10.00 16    5 785+ 31M+ 384K+ 12M- 347
149 hidd         1.9 45:27.12 7     2 560 7208K 0B 2496K 149
50060 Music       1.5 45:57.57 22    3 684 201M+ 460K 77M 50060
148 bluetoothd   1.5 44:49.50 3     1 801 18M 0B 7520K 148
1154 Rectangle   1.3 07:31.75 3     1 210- 52M- 0B 45M 1154

```

El proceso está corriendo continuamente. Para salir, puedes escribir la letra `q` o `ctrl-C`.

Se nos da mucha información: el número de procesos, cuántos están corriendo o durmiendo, la carga del sistema, el uso de la CPU, y mucho más.

A continuación, la lista de los procesos que más memoria y CPU consumen se actualiza constantemente.

Por defecto, como se puede ver en la columna `%CPU` resaltada, están ordenados por la CPU utilizada.

```
top -o mem
```

El comando kill de Linux

Los procesos de Linux pueden recibir **señales** y reaccionar a ellas.

Esa es una forma en la que podemos interactuar con los programas en ejecución.

El programa `kill` puede enviar una variedad de señales a un programa.

No sólo se usa para terminar un programa, como el nombre lo sugiere, sino que es su principal trabajo. Lo usamos de esta manera:

```
kill <PID>
```

Por defecto, este manda una señal `TERM` al identificador de proceso indicado.

Podemos usar banderas para mandar otras señales, incluyendo:

```
kill -HUP <PID>
kill -INT <PID>
kill -KILL <PID>
kill -TERM <PID>
kill -CONT <PID>
kill -STOP <PID>
```

`HUP` significa **colgar** (**hang up** en inglés). Se envía automáticamente cuando una ventana de terminal que inició un proceso se cierra antes de terminar el proceso.

`INT` significa **interrumpir** (**interrupt** en inglés), y envía la misma señal que se usa cuando pulsamos `ctrl-C` en la terminal, que normalmente termina el proceso.

`KILL` no se envía al proceso, sino al núcleo del sistema operativo, que inmediatamente se detiene y termina el proceso.

mismo. Es la señal por defecto enviada por `kill`.

`CONT` significa **continuar** (`continue` en inglés). Se puede usar para reanudar un proceso detenido.

`STOP` no se envía al proceso, sino al núcleo del sistema operativo, que inmediatamente detiene (pero no termina) el proceso.

Podrías ver números usados en su lugar, como `kill -1 <PID>`. En este caso,

- 1 corresponde a `HUP`.
- 2 corresponde a `INT`.
- 9 corresponde a `KILL`.
- 15 corresponde a `TERM`.
- 18 corresponde a `CONT`.
- 15 corresponde a `STOP`.

El comando `killall` de Linux

Similar al comando `kill`, `killall` enviará la señal a múltiples procesos a la vez en lugar de enviar una señal a un identificador de proceso específico.

Su sintaxis es:

```
killall <name>
```

donde `name` es el nombre del programa. Por ejemplo, puedes tener múltiples instancias del programa `top` en ejecución, y `killall top` terminará con todos ellos.

Puedes especificar la señal, como con `kill` (y consulta el tutorial de `kill` para leer más sobre los tipos específicos de señales que podemos enviar), por ejemplo:

```
killall -HUP top
```

El comando `jobs` de Linux

ejecute en segundo plano, usando el símbolo & después del comando.

Por ejemplo, podemos ejecutar `top` en el fondo:

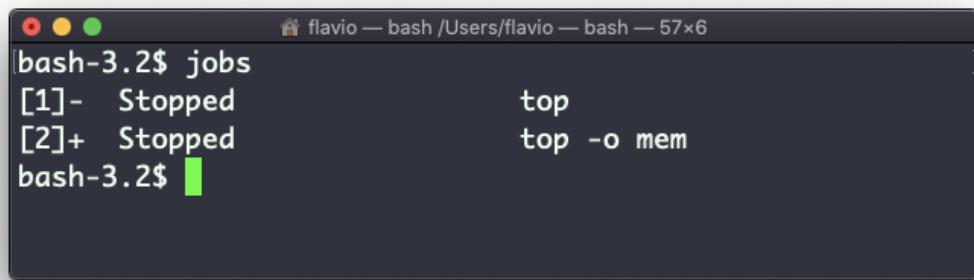
```
top &
```

Esto es muy útil para los programas de larga duración.

Podemos volver a ese programa usando el comando `fg`. Esto funciona bien si sólo tenemos un trabajo en el fondo, de lo contrario necesitamos usar el número de trabajo: `fg 1`, `fg 2` y así sucesivamente.

Para obtener el número de trabajo, usamos el comando `jobs`.

Digamos que ejecutamos `top &` y luego `top -o mem &`, entonces tenemos 2 instancias `top` en ejecución. `jobs` nos dirá esto:



```
flavio — bash /Users/flavio — bash — 57x6
[bash-3.2$ jobs
[1]-  Stopped                  top
[2]+  Stopped                  top -o mem
bash-3.2$ ]
```

Ahora podemos volver a uno de esos usando `fg <jobid>`. Para detener el programa de nuevo, podemos pulsar `ctrl-Z`.

Ejecutando `jobs -l` también imprimirá el identificador de proceso de cada trabajo.

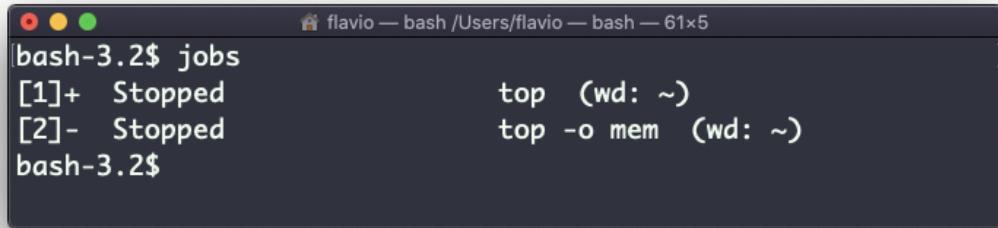
El comando `bg` de Linux

Cuando un comando se está ejecutando puedes suspenderlo usando `ctrl-Z`.

El comando se detendrá inmediatamente, y volverás al shell de la terminal.

pero no te impedirá hacer otro trabajo en la terminal.

En este ejemplo tengo 2 comandos detenidos:



A screenshot of a macOS terminal window titled "flavio — bash /Users/flavio — bash — 61x5". The window contains the following text:

```
bash-3.2$ jobs
[1]+  Stopped                  top  (wd: ~)
[2]-  Stopped                  top -o mem  (wd: ~)
bash-3.2$
```

Puedo ejecutar `bg 1` para reanudar en segundo plano la ejecución del trabajo #1.

También podría haber dicho `bg` sin ninguna opción, ya que por defecto es elegir el trabajo #1 de la lista.

El comando `fg` de Linux

Cuando un comando se ejecuta en segundo plano, porque lo iniciaste con `&` al final (ejemplo: `top &` o porque lo pusiste en segundo plano con el comando `bg`), puedes ponerlo en primer plano usando `fg`.

Ejecutando

`fg`

reanudará en primer plano el último trabajo que fue suspendido.

También puedes especificar qué trabajo quieres retomar en primer plano pasando el número de trabajo, que puedes obtener usando el comando `jobs`.

```
flavio — bash /Users/flavio — bash — 61x5
bash-3.2$ jobs
[1]+  Stopped                  top (wd: ~)
[2]-  Stopped                  top -o mem (wd: ~)
bash-3.2$
```

Ejecutando `fg 2` reanudará el trabajo #2:

```
flavio — bash /Users/flavio — top — 61x23
Processes: 574 total, 2 running, 572 sleeping, 3823 threads
16:12:54 Load Avg: 1.44, 1.60, 1.74
CPU usage: 3.76% user, 2.99% sys, 93.23% idle
SharedLibs: 235M resident, 62M data, 16M linkedit.
MemRegions: 365072 total, 6752M resident, 166M private, 1835M
PhysMem: 15G used (3692M wired), 738M unused.
VM: 3092G vsize, 1991M framework vsize, 318057086(0) swapins,
Networks: packets: 43489522/66G in, 47248873/25G out.
Disks: 69434725/1968G read, 44158267/1695G written.

PID      COMMAND      %CPU TIME      #TH      #WQ      #PORT      MEM
1185     com.docker.h 9.1  15:40:34 18      0        42      2374M
566      Code Helper  0.0  93:50.08 8       1        267      1606M
85440    hugo          0.0  09:19.41 20      0        29      1029M
229      WindowServer 16.0 13:29:14 10      4        6193+    851M
1152     1Password 7   6.9  02:10:02 8       2        2400      581M
377      Bear           0.0  43:56.57 13      6        1763      570M+
605      Code Helper  0.1  63:35.69 27      1        226      557M
42433    Photos          0.0  02:37.69 7       1        535      524M
588      Figma Helper  0.0  26:13.88 8       1        219      522M
0        kernel_task   3.2  06:24:14 262/12 0       0        514M
38992    Google Chrom  0.6  02:52:53 35      2        1238      506M
594      Code Helper  0.3  54:59.85 27      1        228      471M
```

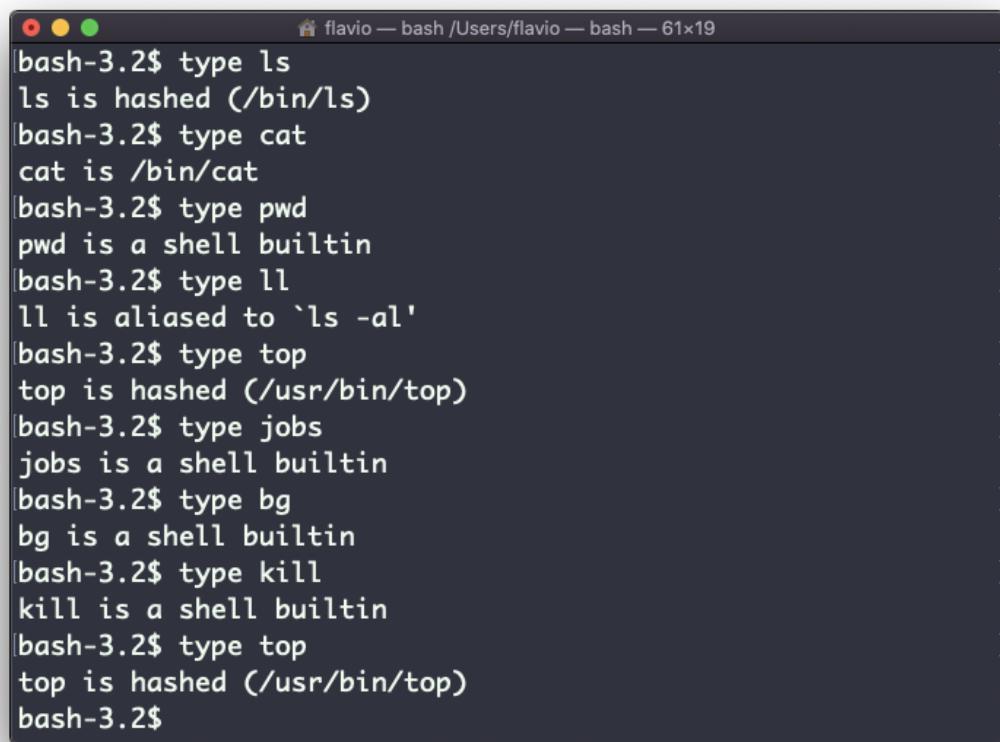
El comando type de Linux

Un comando puede ser uno de 4 tipos:

- un programa incorporado en shell
- una función shell
- un alias

El comando `type` puede ayudar a resolver esto, en caso de que queramos saberlo o simplemente tengamos curiosidad. Te dirá cómo será interpretado el comando.

La salida dependerá del shell utilizado. Este es Bash:



```
flavio — bash /Users/flavio — bash — 61x19
|bash-3.2$ type ls
ls is hashed (/bin/ls)
|bash-3.2$ type cat
cat is /bin/cat
|bash-3.2$ type pwd
pwd is a shell builtin
|bash-3.2$ type ll
ll is aliased to `ls -al'
|bash-3.2$ type top
top is hashed (/usr/bin/top)
|bash-3.2$ type jobs
jobs is a shell builtin
|bash-3.2$ type bg
bg is a shell builtin
|bash-3.2$ type kill
kill is a shell builtin
|bash-3.2$ type top
top is hashed (/usr/bin/top)
bash-3.2$
```

Este es Zsh:

```
flavio@mbp ~ % type ls
ls is /bin/ls
flavio@mbp ~ % type cat
cat is /bin/cat
flavio@mbp ~ % type pwd
pwd is a shell builtin
flavio@mbp ~ % type ll
ll not found
flavio@mbp ~ % alias ll='ls -al'
flavio@mbp ~ % type ll
ll is an alias for ls -al
flavio@mbp ~ % type top
top is /usr/bin/top
flavio@mbp ~ % type jobs
jobs is a shell builtin
flavio@mbp ~ % type bg
bg is a shell builtin
flavio@mbp ~ % type kill
kill is a shell builtin
flavio@mbp ~ % type top
top is /usr/bin/top
flavio@mbp ~ %
```

Este es Fish:

```

[~] ~ type ls
ls is a function with definition
# Defined in /usr/local/Cellar/fish/3.1.0/share/fish/functions/ls.fish @ line 13
function ls --description 'List contents of directory'
    set -l opt -G
        isatty stdout
        and set -a opt -F
        command ls $opt $argv
end
[~] ~ type cat
cat is /bin/cat
[~] ~ type pwd
pwd is a builtin
[~] ~ type ll
ll is a function with definition
# Defined in /usr/local/Cellar/fish/3.1.0/share/fish/functions/ll.fish @ line 4
function ll --description 'List contents of directory using long format'
    ls -lh $argv
end
[~] ~ type top
top is /usr/bin/top
[~] ~ type jobs
jobs is a builtin
[~] ~ type bg
bg is a function with definition
# Defined in /usr/local/Cellar/fish/3.1.0/share/fish/config.fish @ line 274
function bg
    set -l jobbltn bg
        builtin $__fish_expand_pid_args $argv
end
[~] ~ type kill
kill is a function with definition
# Defined in /usr/local/Cellar/fish/3.1.0/share/fish/config.fish @ line 279
function kill
    command kill($__fish_expand_pid_args $argv)
end
[~] ~ type top
top is /usr/bin/top
[~]

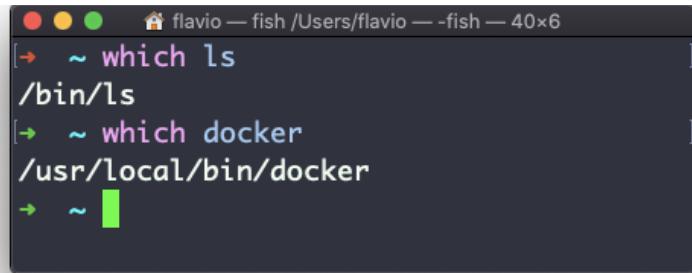
```

Una de las cosas más interesantes aquí es que para los aliases te dirá a qué se está aliando. Puedes ver el alias `ll`, en el caso de Bash y Zsh, pero Fish lo proporciona por defecto, así que te dirá que es una función de shell incorporada.

El comando `which` de Linux

Supongamos que tienes un comando que puedes ejecutar, porque está en la ruta del shell, pero quieres saber dónde se encuentra.

Puedes hacerlo usando `which`. El comando devolverá la ruta al comando especificado:



```
flavio — fish /Users/flavio — -fish — 40x6
[~] ~ which ls
/bin/ls
[~] ~ which docker
/usr/local/bin/docker
[~]
```

`which` sólo funcionará con ejecutables almacenados en disco, no alias o funciones de shell incorporadas.

El comando `nohup` de Linux

A veces hay que ejecutar un proceso de larga duración en una máquina remota, y luego hay que desconectarse.

O simplemente quieres evitar que el comando se detenga si hay algún problema de red entre tú y el servidor.

La forma de hacer que un comando se ejecute incluso después de cerrar la sesión o la sesión con un servidor es usar el comando `nohup`.

Usa `nohup <command>` para que el proceso continúe funcionando incluso después de cerrar la sesión.

El comando `xargs` de Linux

El comando `xargs` se utiliza en un shell de UNIX para convertir la entrada de una entrada estándar en argumentos para un comando.

En otras palabras, mediante el uso de `xargs` la salida de un comando se utiliza como la entrada de otro comando.

Aquí está la sintaxis que utilizarás:

```
command1 | xargs command2
```

command2 , usando la salida del command1 como su(s) argumento(s).

Hagamos un ejemplo sencillo. Quieres remover algunos archivos específicos de un directorio. Esos archivos están enlistados dentro de un archivo de texto.

Tenemos 3 archivos: file1 , file2 , file3 .

En todelete.txt tenemos una lista de archivos que queremos borrar, en este ejemplo, file1 y file3 :



```
testing — fish /Users/flavio/testing — -fish — 62x7
[+ testing ls
file1      file2      file3      todelete.txt
[+ testing cat todelete.txt
file1
file3
+ testing
```

Canalizaremos la salida de cat todelete.txt al comando rm , a través de xargs .

De este manera:

```
cat todelete.txt | xargs rm
```

Ese es el resultado, los archivos que hemos enumerado se han eliminado:



```
testing — fish /Users/flavio/testing — -fish — 62x10
|+ testing ls
file1          file2          file3          todelete.txt
|+ testing cat todelete.txt
file1
file3
|+ testing cat todelete.txt | xargs rm
|
|+ testing ls
file2          todelete.txt
+ testing
```

La forma en que funciona es que `xargs` ejecutará `rm` dos veces, una por cada línea devuelta por `cat`.

Este es el uso más simple de `xargs`. Hay varias opciones que podemos usar.

Una de las más útiles, en mi opinión (especialmente cuando se empieza a aprender `xargs`), es `-p`. Usar esta opción hará que `xargs` imprima un aviso de confirmación con la acción que va a realizar:



```
testing — cat /Users/flavio/testing — xargs -p rm — 44x5
|+ testing cat todelete.txt | xargs -p rm
rm file1 file3?...
```

La opción `-n` te permite decirle a `xargs` que realice una iteración a la vez, para que puedas confirmarlos individualmente con `-p`. Aquí le decimos a `xargs` que realice una iteración a la vez con `-n1`:



```
testing — cat /Users/flavio/testing — xargs -p -n1 rm — 55x5
|→ testing cat todelete.txt | xargs -p -n1 rm
rm file1?...
```

La opción `-I` es otra muy utilizada. Te permite obtener la salida de un marcador, y entonces puedes hacer varias cosas.

Una de ellas es ejecutar múltiples comandos:

```
command1 | xargs -I % /bin/bash -c 'command2 %; command3 %'
```



```
testing — cat /Users/flavio/testing — xargs -p -I % sh -c 'ls %; rm %' — 67x5
|→ testing cat todelete.txt | xargs -p -I % sh -c 'ls %; rm %'
sh -c ls file1; rm file1?...
```

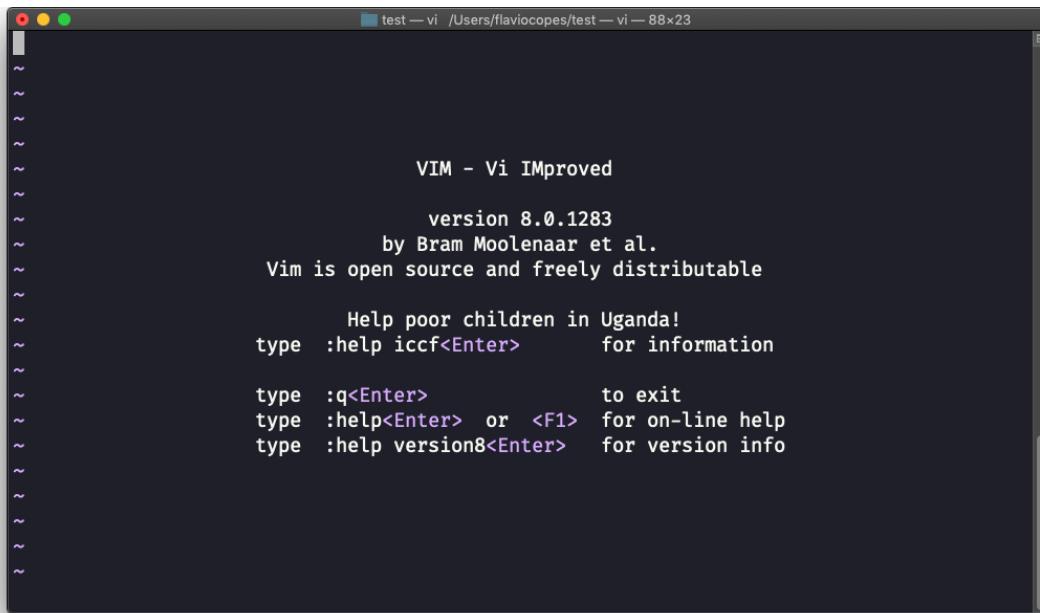
Puedes cambiar el símbolo `%` que usé arriba por cualquier otra cosa – es una variable.

El comando `vim` del editor de Linux

`vim` es un editor de archivos **muy** popular, especialmente entre los programadores. Es desarrollado activamente y se actualiza con frecuencia, y hay una gran comunidad a su alrededor. ¡Incluso hay una [conferencia de Vim!](#)

`vi` en los sistemas modernos es sólo un alias para `vim`, lo que significa `vi improved`.

Se inicia ejecutando `vi` en la línea de comandos.



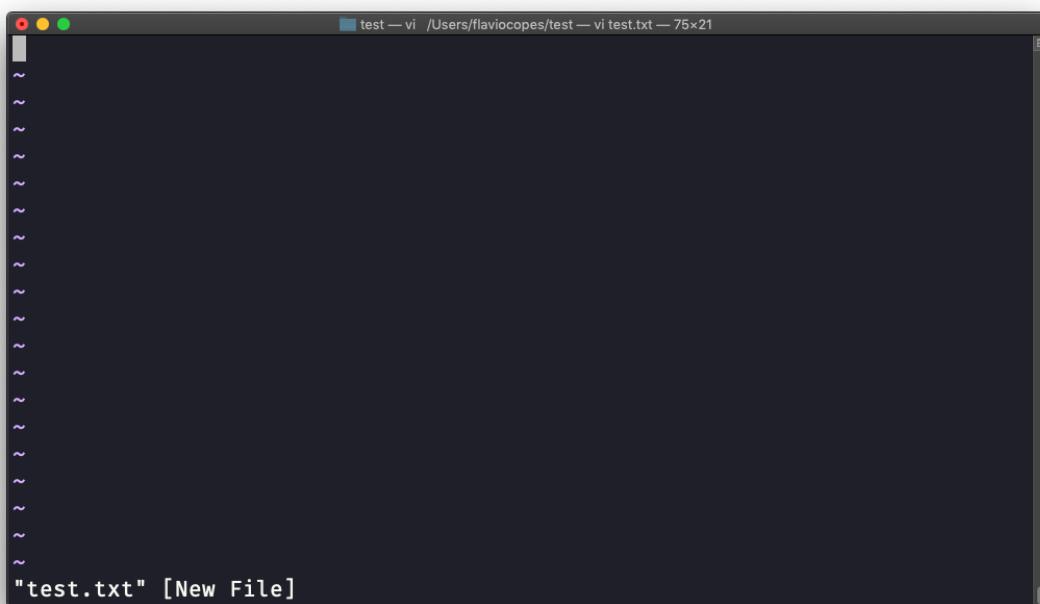
VIM - Vi IMproved
version 8.0.1283
by Bram Moolenaar et al.
Vim is open source and freely distributable

Help poor children in Uganda!
type :help iccf<Enter> for information

type :q<Enter> to exit
type :help<Enter> or <F1> for on-line help
type :help version8<Enter> for version info

Puedes especificar un nombre de archivo en el momento de la invocación para editar ese archivo específico:

```
vi test.txt
```



"test.txt" [New File]

DEBES SABER QUE VI TIENE 2 MODOS PRINCIPALES.

- modo de comando (*command* en inglés) también llamado *normal*
- modo de inserción (*insert* en inglés)

Cuando inicias el editor, estás en modo de comando, No puedes introducir texto como esperas de un editor basado en una interfaz gráfica de usuario. Tienes que entrar en el modo de inserción.

Puedes hacerlo pulsando la tecla `i`. Una vez que lo hagas, la palabra `-- INSERT --` aparece en la parte inferior del editor.



Ahora puedes empezar a escribir y llenar la pantalla con el contenido del archivo:

Puedes moverte por el archivo con las teclas de flechas, o usando las teclas `h - j - k - l`.
`h-l` para izquierda-derecha, `j-k` para abajo-arriba.

Una vez que hayas terminado de editar puedes pulsar la tecla `esc` para salir del modo de inserción y volver al **modo de comando**.

```
Hey

This is a very cool editor

It's called Vim!
~
~
```

cuidado con las teclas que pulses, ya que pueden ser comandos).

Una cosa que quizá quieras hacer ahora es guardar el archivo. Puedes hacerlo pulsando : (dos puntos), luego w .

Puedes **guardar y salir** presionando : luego w y q : :wq

Puedes **salir sin guardar** presionando : luego q y ! : :q!

Puedes **deshacer** y editar yendo al modo de comando y presionando u . Puedes **rehacer** (cancelar un deshacer) presionando ctrl-r .

Esos son los fundamentos de trabajar con Vim. A partir de aquí comienza el "cuento sin fin" en el cual no podemos entrar en esta pequeña introducción.

Sólo mencionaré los comandos que te permitirán empezar a editar con Vim:

- pulsando la tecla x borra el carácter actualmente resaltado
- pulsando A te dirige el cursor al final de la línea actualmente seleccionada y entras a modo de inserción
- pulsando 0 te dirige al comienzo de la línea
- ve al primer carácter de una palabra y pulsa d seguido de w para borrar una palabra. Si la sigues con una e en lugar de w , el espacio en blanco antes de la siguiente palabra se conserva.
- usar un número entre d y w para borrar más de una palabra, por ejemplo, usar d3w para borrar 3 palabras hacia adelante.
- pulsando d seguido de d para borrar una línea entera. Pulsa d seguido de \$ para borrar toda la línea desde donde está el cursor hasta el final

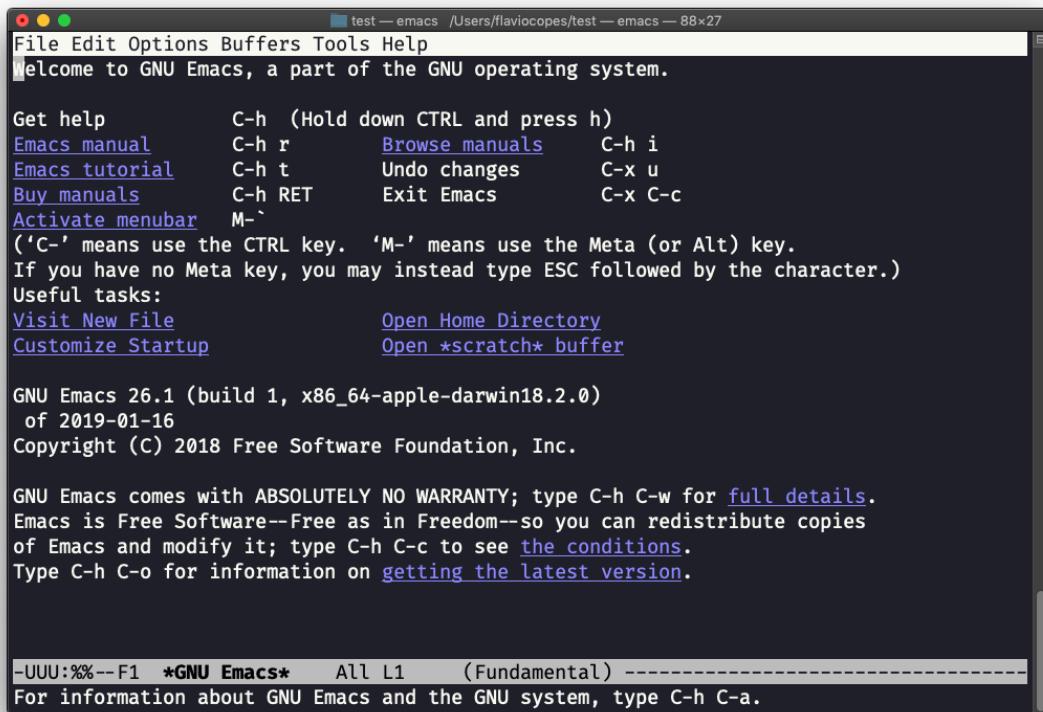
Para saber más sobre Vim, puedo recomendar el [Vim FAQ](#). También puedes ejecutar el comando vimtutor , que ya debería estar instalado en tu sistema y te ayudará enormemente a comenzar tu exploración de vim .

El comando emacs del editor de Linux

emacs es un editor impresionante y es históricamente considerado como el editor de los sistemas UNIX. Famosamente, las guerras y acaloradas discusiones de vi vs emacs han causado muchas horas improductivas para los desarrolladores de todo el mundo.

operativo (<https://news.ycombinator.com/item?id=19127258>). Hablaremos de lo básico aquí.

Puedes abrir una nueva sesión de emacs simplemente invocando a `emacs`:

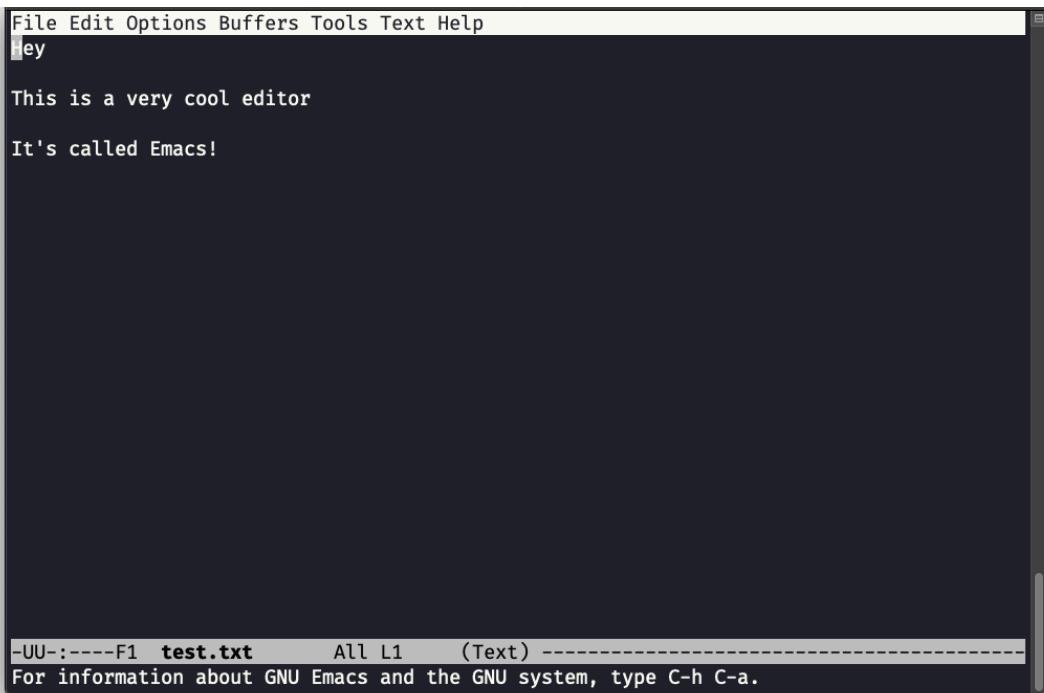


Usuarios de macOS, paren un segundo ahora. Si estás en Linux, no hay problema, pero macOS no es cargado con aplicaciones usando GPLv3, y cada comando UNIX incorporado que ha sido actualizado GPLv3 no ha sido actualizado.

Aunque hay un pequeño problema con los comandos que he enumerado hasta ahora, en este caso usar una versión de emacs de 2007 no es exactamente lo mismo que usar una versión con 12 años de mejoras y cambios.

Esto no es un problema con Vim, que está actualizado. Para arreglar esto, ejecuta `brew install emacs` y ejecutando `emacs` usarás la nueva versión de Homebrew (asegúrate de tener Homebrew instalado).

También puedes editar un archivo existente llamando a `emacs <filename>`:

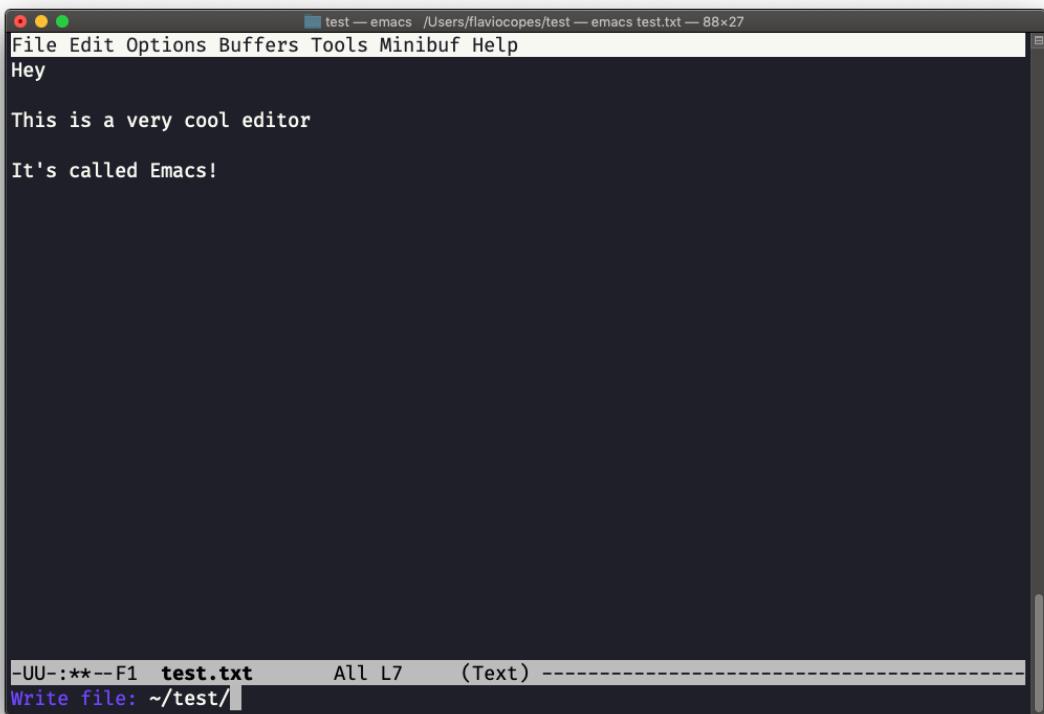


A screenshot of a dark-themed Emacs window. The menu bar at the top includes "File", "Edit", "Options", "Buffers", "Tools", "Text", and "Help". The main buffer contains the following text:

```
Hey
This is a very cool editor
It's called Emacs!
```

The status bar at the bottom displays the file name "test.txt", line count "All L1", mode "(Text)", and buffer size "-----". It also includes the message "For information about GNU Emacs and the GNU system, type C-h C-a."

Ya puedes empezar a editar. Una vez que hayas terminado, presiona `ctrl-x` seguido de `ctrl-w`. Confirma la carpeta:

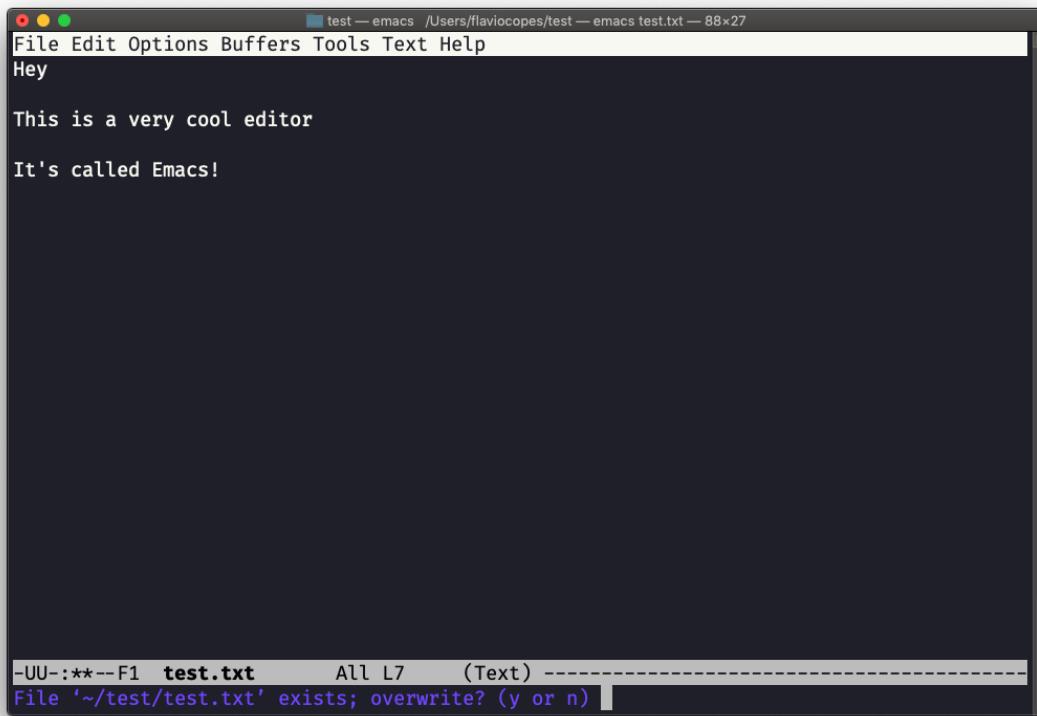


A screenshot of a dark-themed Emacs window. The menu bar at the top includes "File", "Edit", "Options", "Buffers", "Tools", "Minibuf", and "Help". The main buffer contains the same text as the previous screenshot:

```
Hey
This is a very cool editor
It's called Emacs!
```

The status bar at the bottom displays the file name "test.txt", line count "All L7", mode "(Text)", and buffer size "-----". It also includes the message "Write file: ~/test/".

Emacs te avisa que el archivo existe, preguntándote si querer la sobrescritura.



Responde `y`, y tendrás una confirmación de éxito:

```
File Edit Options Buffers Tools Text Help
Hey

This is a very cool editor

It's called Emacs!

-UU-:----F1 test.txt      All L7      (Text) -----
Wrote /Users/flaviocopes/test/test.txt
```

Puedes salir de Emacs presionando `ctrl-x` seguido de `ctrl-c`. O `ctrl-x` seguido de `c` (mantén presionado `ctrl`).

Hay mucho que saber sobre Emacs, ciertamente más de lo que soy capaz de escribir en esta pequeña introducción. Te invito a abrir Emacs y a pulsar `ctrl-h r` para abrir el manual incorporado y `ctrl-h t` para abrir el tutorial oficial.

El comando nano del editor de Linux

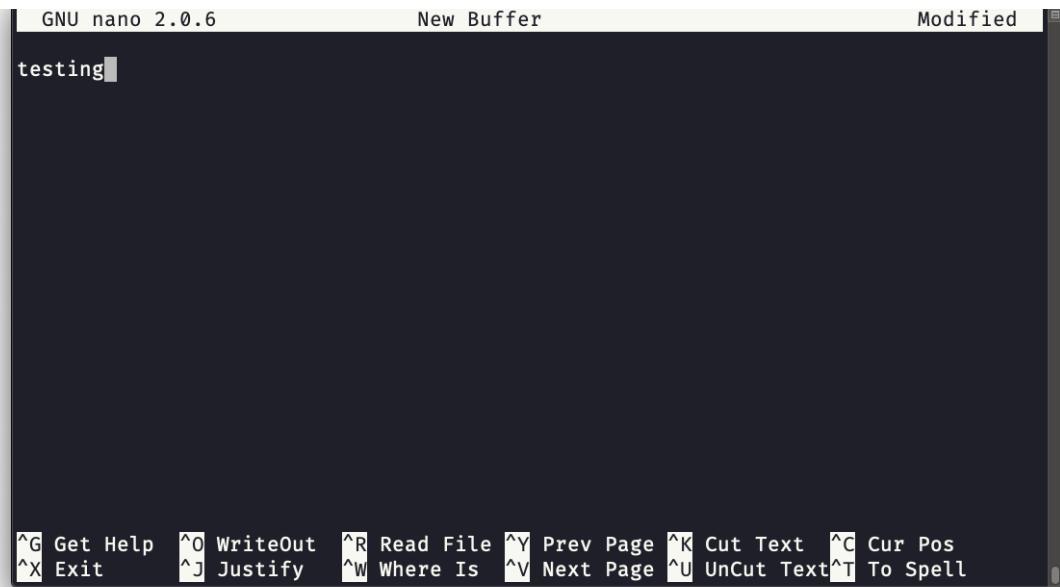
`nano` es un editor amigable para principiantes.

Ejecútalo usando `nano <filename>`.

Puedes escribir directamente los caracteres en el archivo sin preocuparte por los modos.

Puedes salir sin editar usando `ctrl-X`. Si has editado el búfer del archivo, el editor te pedirá confirmación y podrás guardar las ediciones o descartarlas.

La ayuda de la parte inferior muestra los comandos del teclado que te permiten trabajar con el archivo:



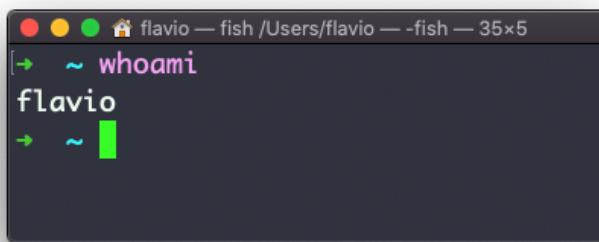
The screenshot shows a terminal window titled "GNU nano 2.0.6" with the status bar indicating "New Buffer" and "Modified". The main area contains the text "testing". At the bottom, there is a menu of keyboard shortcuts:

- ^G Get Help
- ^O WriteOut
- ^R Read File
- ^Y Prev Page
- ^K Cut Text
- ^C Cur Pos
- ^X Exit
- ^J Justify
- ^W Where Is
- ^V Next Page
- ^U UnCut Text
- ^T To Spell

pico es más o menos lo mismo, aunque nano es la versión GNU de pico que en algún momento de la historia no era de código abierto. El clon nano fue hecho para satisfacer los requisitos de la licencia del sistema operativo GNU.

El comando whoami de Linux

Escribe whoami para imprimir el nombre de usuario que está conectado a la sesión de la terminal:

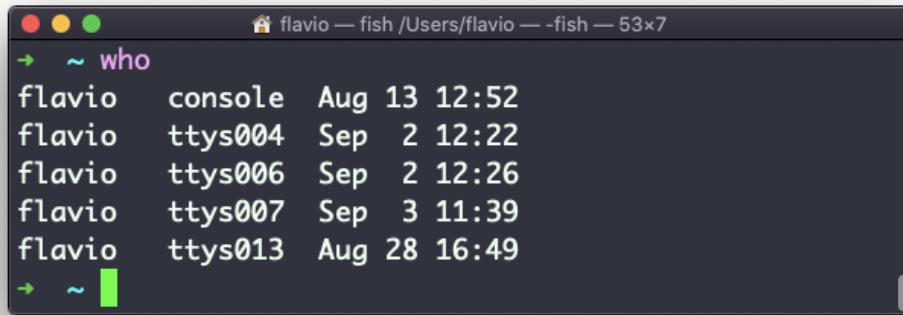


The screenshot shows a terminal window with a dark theme and a green status bar icon. The prompt is "flavio ~ whoami". The output of the command "whoami" is "flavio". The status bar at the bottom right shows the terminal is running "fish" and has a size of "35x5".

Nota: esto es diferente del comando `who am i`, el cual imprime más información

El comando who de Linux

A menos que estés usando un servidor al que varias personas tienen acceso, lo más probable es que seas el único usuario conectado, varias veces:

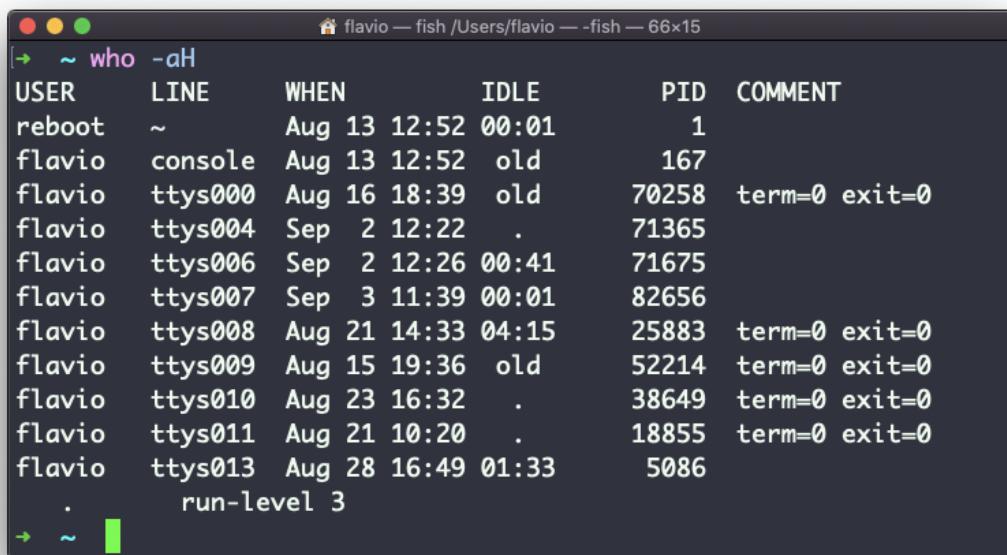


```
flavio — fish /Users/flavio — -fish — 53x7
~ who
flavio  console Aug 13 12:52
flavio  ttys004 Sep  2 12:22
flavio  ttys006 Sep  2 12:26
flavio  ttys007 Sep  3 11:39
flavio  ttys013 Aug 28 16:49
~
```

¿Por qué varias veces? Porque cada shell abierto cuenta como un acceso.

Puedes ver el nombre de la terminal utilizada, y la hora/día en que se inició la sesión.

Las banderas `-ah` le dirán a `who` a mostrar más información, incluyendo el tiempo de inactividad y el identificador del proceso de la terminal:



```
flavio — fish /Users/flavio — -fish — 66x15
~ who -ah
USER   LINE    WHEN      IDLE      PID  COMMENT
reboot ~ Aug 13 12:52 00:01      1
flavio  console Aug 13 12:52 old      167
flavio  ttys000 Aug 16 18:39 old    70258 term=0 exit=0
flavio  ttys004 Sep  2 12:22 .      71365
flavio  ttys006 Sep  2 12:26 00:41  71675
flavio  ttys007 Sep  3 11:39 00:01  82656
flavio  ttys008 Aug 21 14:33 04:15  25883 term=0 exit=0
flavio  ttys009 Aug 15 19:36 old    52214 term=0 exit=0
flavio  ttys010 Aug 23 16:32 .      38649 term=0 exit=0
flavio  ttys011 Aug 21 10:20 .      18855 term=0 exit=0
flavio  ttys013 Aug 28 16:49 01:33  5086
.      run-level 3
~
```



```
flavio — fish /Users/flavio — -fish — 45x5
|~ who am i
flavio  ttys004  Sep  2 12:22
|~
```



```
flavio — fish /Users/flavio — -fish — 65x5
|~ who -ah am i
USER   LINE   WHEN      IDLE      PID      COMMENT
flavio  ttys004  Sep  2 12:22    .        71365
|~
```

El comando su de Linux

Mientras estás conectado a la terminal con un usuario, puede que tengas que cambiar a otro usuario.

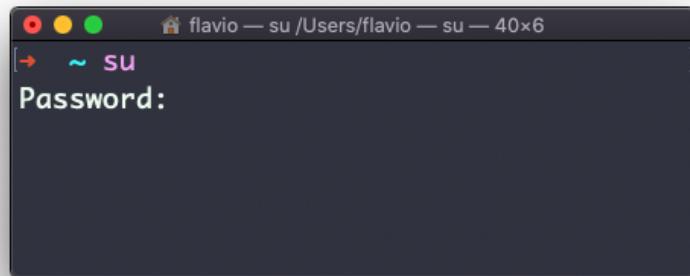
Por ejemplo, estás conectado como root para realizar algo de mantenimiento, pero luego quieres cambiar a una cuenta de usuario.

Puedes hacerlo con el comando `su` :

```
su <username>
```

Por ejemplo: `su flavio`.

contraseña del usuario `root`, ya que es el comportamiento predeterminado.



`su` iniciará un nuevo shell como otro usuario.

Cuando termines, al escribir `exit` en el shell cerrará el mismo, y te devolverá al shell del usuario actual.

El comando sudo de Linux

`sudo` se usa comúnmente para ejecutar un comando como root.

Debes estar habilitado para usar `sudo`, y una vez que lo estés, puedes ejecutar comandos como root introduciendo la contraseña de tu usuario (*no* la contraseña del usuario root).

Los permisos son altamente configurables, lo que es estupendo especialmente en un entorno de servidor multiusuario. A algunos usuarios se les puede conceder acceso a la ejecución de comandos específicos a través de `sudo`.

Por ejemplo, se puede editar un archivo de configuración del sistema:

```
sudo nano /etc/hosts
```

que de otra manera no se salvaría ya que no tienes los permisos para ello.

Puedes ejecutar `sudo -i` para iniciar un shell como root:



Puedes usar `sudo` para ejecutar comandos como cualquier usuario. `root` es el predeterminado, pero usa la opción `-u` para especificar otro usuario:

```
sudo -u flavio ls /Users/flavio
```

El comando `passwd` de Linux

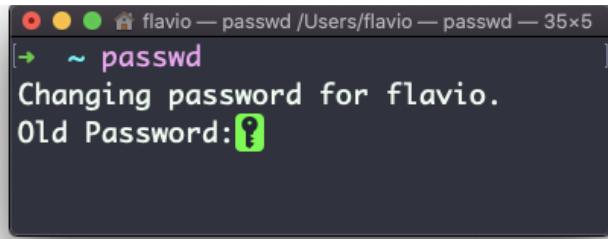
Los usuarios en Linux tienen una contraseña asignada. Puedes cambiar la contraseña usando el comando `passwd`.

Hay dos situaciones aquí.

La primera es cuando quieras cambiar tu contraseña. En este caso escribes:

```
passwd
```

y un aviso interactivo te pedirá la contraseña anterior, y luego te pedirá la nueva:



```
flavio — passwd /Users/flavio — passwd — 35x5
[~] ~ passwd
Changing password for flavio.
Old Password: [REDACTED]
```

Cuando eres `root` (o tienes privilegios de superusuario) puedes establecer el nombre de usuario para el que quieres cambiar la contraseña:

```
passwd <username> <new password>
```

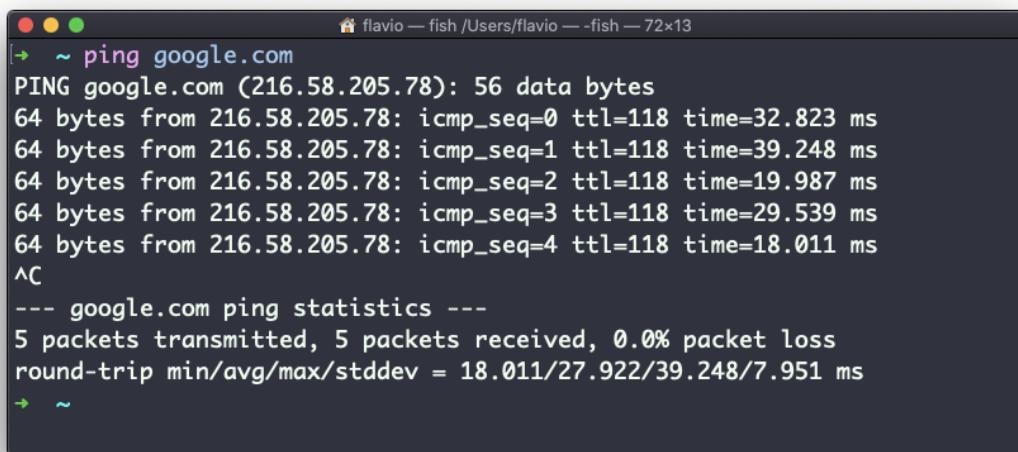
En este caso no necesitas introducir la contraseña anterior.

El comando ping de Linux

El comando `ping` hace un ping de un host de red específico, en la red local o en Internet.

Lo usas con la sintaxis `ping <host>` donde `<host>` podría ser un nombre de dominio, o una dirección IP.

Aquí hay un ejemplo de "pinging" a `google.com`:



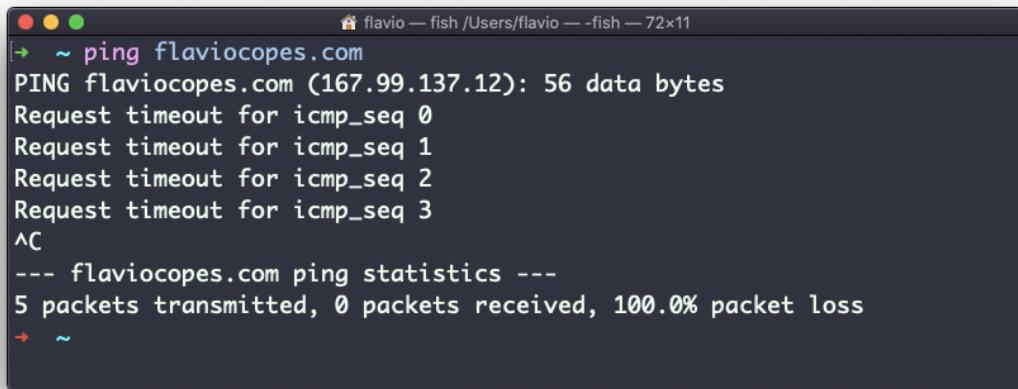
```
flavio — fish /Users/flavio — -fish — 72x13
[~] ~ ping google.com
PING google.com (216.58.205.78): 56 data bytes
64 bytes from 216.58.205.78: icmp_seq=0 ttl=118 time=32.823 ms
64 bytes from 216.58.205.78: icmp_seq=1 ttl=118 time=39.248 ms
64 bytes from 216.58.205.78: icmp_seq=2 ttl=118 time=19.987 ms
64 bytes from 216.58.205.78: icmp_seq=3 ttl=118 time=29.539 ms
64 bytes from 216.58.205.78: icmp_seq=4 ttl=118 time=18.011 ms
^C
--- google.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 18.011/27.922/39.248/7.951 ms
[~]
```

`ping` sigue enviando la solicitud cada segundo por defecto. Seguirá funcionando hasta que lo detengas con `ctrl-C`, a menos que pases el número de veces que quieras intentarlo con la opción `-c`: `ping -c 2 google.com`.

Una vez que se detenga el `ping`, imprimirá algunas estadísticas sobre los resultados: el porcentaje de paquetes perdidos, y estadísticas sobre el rendimiento de la red.

Como puedes ver, la pantalla imprime la dirección IP del host, y el tiempo que tardó en obtener la respuesta.

No todos los servidores soportan hacerles un ping, en este caso la solicitud se agota:



```
[~] flaviocopes.com
PING flaviocopes.com (167.99.137.12): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
^C
--- flaviocopes.com ping statistics ---
5 packets transmitted, 0 packets received, 100.0% packet loss
[~]
```

A veces esto se hace a propósito, para "esconder" el servidor, o sólo para reducir la carga. Los paquetes de ping también pueden ser filtrados por los cortafuegos.

`ping` funciona usando el **protocolo ICMP** (*Internet Control Message Protocol*), un protocolo de capa de red como el TCP o el UDP.

La solicitud envía un paquete al servidor con el mensaje `ECHO_REQUEST`, y el servidor devuelve un mensaje `ECHO_REPLY`. No entrará en detalles, pero este es el concepto básico.

Hacer ping a un host es útil para saber si el host es alcanzable (suponiendo que implemente ping), y cuán distante está en términos de cuánto tiempo le lleva volver a ti.

Normalmente cuanto más cerca esté el servidor geográficamente, menos tiempo tardaré en volver a ti. Leyes físicas simples hacen que una mayor distancia introduzca más retraso en los cables.

El comando traceroute de Linux

Cuando intentas contactar con un host en Internet, lo haces a través del router de tu casa. Luego llegas a la red de tu ISP (proveedor de Internet), que a su vez pasa por su propio enrutador de red ascendente, y así sucesivamente, hasta que finalmente llegas al host.

¿Alguna vez has querido saber qué pasos siguen tus paquetes para hacer eso?

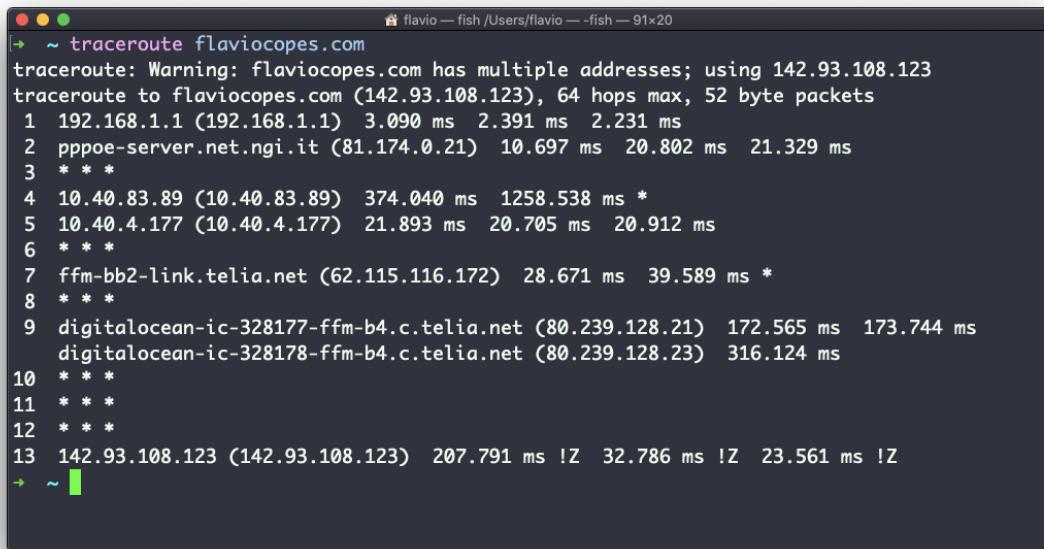
El comando `traceroute` está hecho para esto.

Lo invocas

```
traceroute <host>
```

y recogerá (lentamente) toda la información mientras el paquete viaja.

En este ejemplo, intenté buscar mi blog con `traceroute flaviocopes.com`:



```
flavio — fish /Users/flavio — -fish — 91x20
[+ ~ traceroute flaviocopes.com
traceroute: Warning: flaviocopes.com has multiple addresses; using 142.93.108.123
traceroute to flaviocopes.com (142.93.108.123), 64 hops max, 52 byte packets
 1  192.168.1.1 (192.168.1.1)  3.090 ms  2.391 ms  2.231 ms
 2  pppoe-server.net.ngi.it (81.174.0.21)  10.697 ms  20.802 ms  21.329 ms
 3  * * *
 4  10.40.83.89 (10.40.83.89)  374.040 ms  1258.538 ms *
 5  10.40.4.177 (10.40.4.177)  21.893 ms  20.705 ms  20.912 ms
 6  * * *
 7  ffm-bb2-link.telia.net (62.115.116.172)  28.671 ms  39.589 ms *
 8  * * *
 9  digitalocean-ic-328177-ffm-b4.c.telia.net (80.239.128.21)  172.565 ms  173.744 ms
    digitalocean-ic-328178-ffm-b4.c.telia.net (80.239.128.23)  316.124 ms
10  * * *
11  * * *
12  * * *
13  142.93.108.123 (142.93.108.123)  207.791 ms !Z  32.786 ms !Z  23.561 ms !Z
+ ~
```

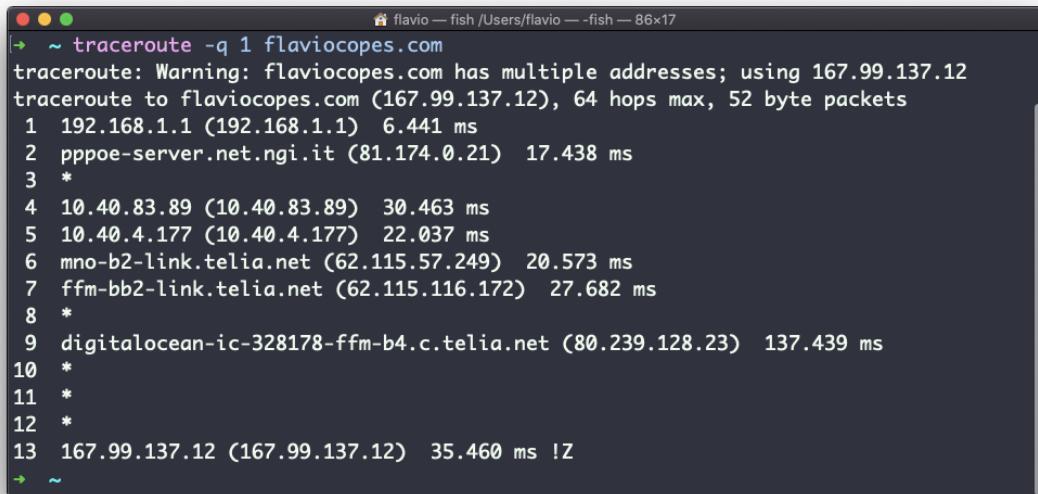
No todos los routers que viajan nos devuelven información. En este caso, `traceroute` imprime `* * *`. De lo contrario, podemos ver el nombre del host, la dirección IP y algún indicador de rendimiento.

defecto 3 veces para conseguir una buena indicación del tiempo necesario para alcanzarlo.

Por esto se tarda tanto en ejecutar `traceroute` en comparación con simplemente hacer un `ping` a ese host.

Puedes personalizar este número con la opción `-q`:

```
traceroute -q 1 flaviocopes.com
```



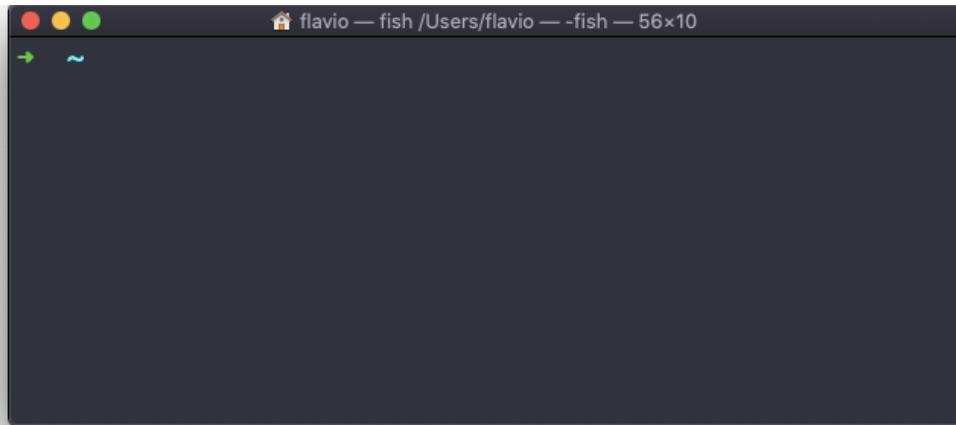
A screenshot of a macOS terminal window titled "flavio — fish /Users/flavio — -fish — 86x17". The command entered is "traceroute -q 1 flaviocopes.com". The output shows a route from the user's machine to the website flaviocopes.com, which has multiple IP addresses. The route consists of 13 hops, with the final hop being the destination at 167.99.137.12. The first few hops are local or on the same network, while subsequent ones show the path through various routers and ISPs until reaching the destination.

```
[~] ~ traceroute -q 1 flaviocopes.com
traceroute: Warning: flaviocopes.com has multiple addresses; using 167.99.137.12
traceroute to flaviocopes.com (167.99.137.12), 64 hops max, 52 byte packets
 1  192.168.1.1 (192.168.1.1)  6.441 ms
 2  pppoe-server.net.ngi.it (81.174.0.21)  17.438 ms
 3  *
 4  10.40.83.89 (10.40.83.89)  30.463 ms
 5  10.40.4.177 (10.40.4.177)  22.037 ms
 6  mno-b2-link.telia.net (62.115.57.249)  20.573 ms
 7  ffm-bb2-link.telia.net (62.115.116.172)  27.682 ms
 8  *
 9  digitalocean-ic-328178-ffm-b4.c.telia.net (80.239.128.23)  137.439 ms
10  *
11  *
12  *
13  167.99.137.12 (167.99.137.12)  35.460 ms !Z
```

El comando `clear` de Linux

Escribe `clear` para borrar todos los comandos anteriores que se ejecutaron en la terminal actual.

La pantalla se borrará y sólo verás el aviso en la parte superior:



Nota: este comando tiene un atajo práctico: `ctrl-L`

Una vez que lo hagas, perderás el acceso al desplazamiento para ver la salida de los comandos anteriormente introducidos.

Así que tal vez quieras usar `clear -x` en su lugar, que sigue despejando la pantalla, pero te permite volver a ver el trabajo anterior desplazándote hacia arriba.

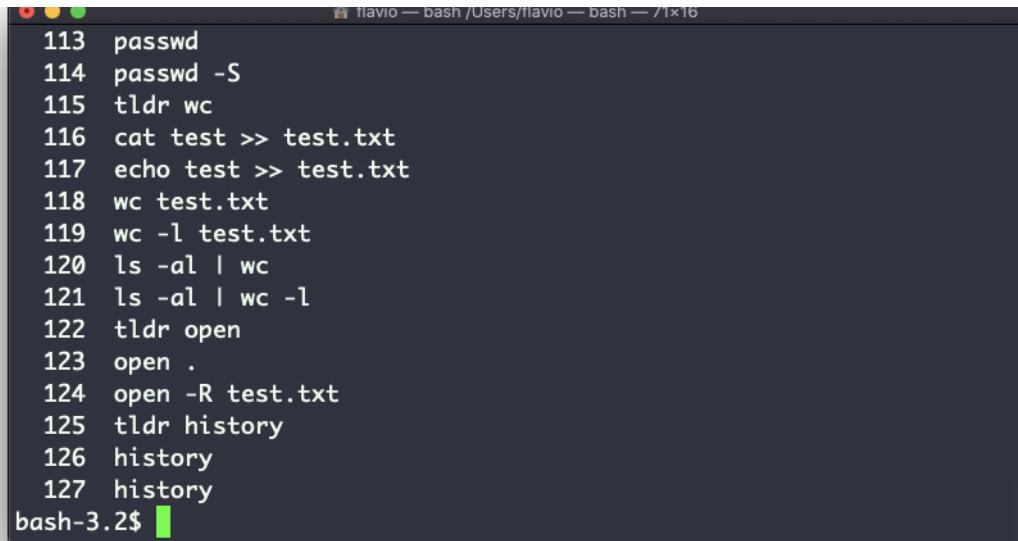
El comando `history` de Linux

Cada vez que ejecutas un comando, se memoriza en el historial.

Puedes mostrar todo el historial usando:

```
history
```

Esto muestra la historia con números:



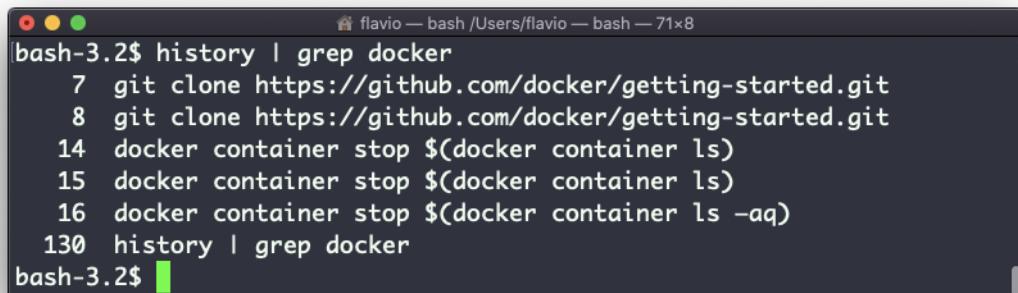
```
flavio — bash /Users/flavio — bash — 71x16
113 passwd
114 passwd -S
115 tldr wc
116 cat test >> test.txt
117 echo test >> test.txt
118 wc test.txt
119 wc -l test.txt
120 ls -al | wc
121 ls -al | wc -l
122 tldr open
123 open .
124 open -R test.txt
125 tldr history
126 history
127 history
bash-3.2$
```

Puedes usar la sintaxis `!<command number>` para repetir un comando almacenado en el historial. En el ejemplo anterior, escribiendo `!121` se repetirá el comando `ls -al | wc -l`.

Normalmente los últimos 500 comandos se almacenan en el historial.

Puedes combinar esto con `grep` para encontrar un comando que hayas ejecutado:

```
history | grep docker
```



```
flavio — bash /Users/flavio — bash — 71x8
bash-3.2$ history | grep docker
 7 git clone https://github.com/docker/getting-started.git
 8 git clone https://github.com/docker/getting-started.git
14 docker container stop $(docker container ls)
15 docker container stop $(docker container ls)
16 docker container stop $(docker container ls -aq)
130 history | grep docker
bash-3.2$
```

Para borrar el historial, ejecuta `history -c`.

El comando `export` se utiliza para exportar variables a procesos hijos.

¿Qué significa esto?

Supongamos que tienes una variable TEST definida de esta manera:

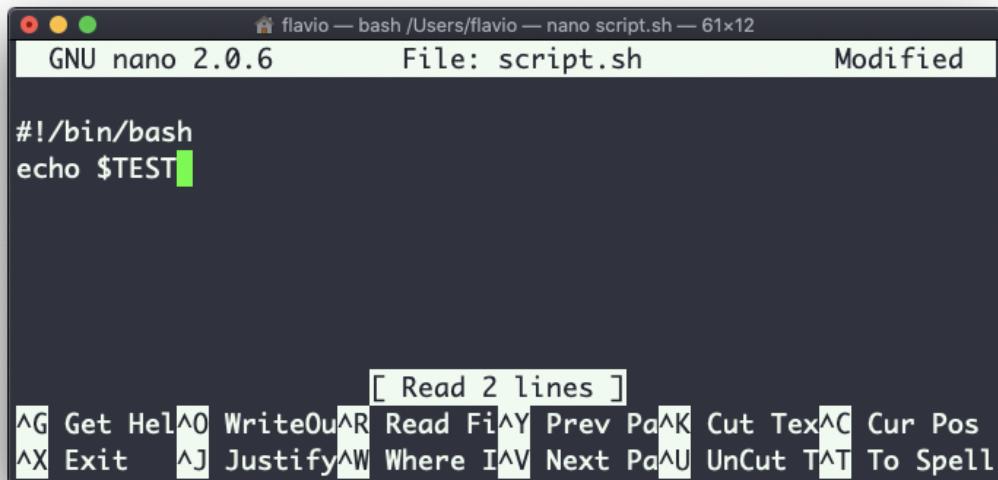
```
TEST="test"
```

Puedes imprimir su valor usando `echo $TEST`:



```
flavio — bash /Users/flavio — bash — 61x5
|bash-3.2$ echo $TEST
test
bash-3.2$
```

Pero si intentas definir un script Bash en un archivo `script.sh` con el comando anterior:



```
flavio — bash /Users/flavio — nano script.sh — 61x12
GNU nano 2.0.6          File: script.sh          Modified

#!/bin/bash
echo $TEST

[ Read 2 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit      ^J Justify ^W Where Is ^V Next Page ^U Uncut T ^T To Spell
```

ENTONCES CUANDO PONES UNA VARIABLE TEST EN TU SCRIPT.SH Y EJECUTAS ESTE SCRIPT CON ./SCRIPT.SH, PIA
línea echo \$TEST NO IMPRIMIRÁ NADA!

Esto se debe a que en Bash la variable `TEST` se definió local al shell. Cuando se ejecuta un script de shell u otro comando, se lanza un subshell para ejecutarlo, que no contiene las variables locales actuales del shell.

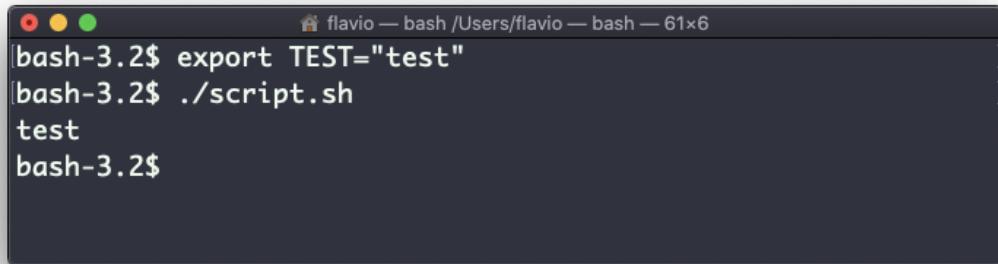
Para que la variable esté disponible allí, necesitamos no definir `TEST` de esta manera:

```
TEST="test"
```

pero de esta:

```
export TEST="test"
```

Inténtalo, y ejecutando `./script.sh` ahora debería imprimir "test":



```
flavio — bash /Users/flavio — bash — 61x6
[bash-3.2$ export TEST="test"
[bash-3.2$ ./script.sh
test
bash-3.2$
```

A veces hay que agregar algo a una variable. A menudo se hace con la variable `PATH`. Se utiliza esta sintaxis:

```
export PATH=$PATH:/new/path
```

puedes usarlo cuando creas variables en los archivos de configuración `.bash_profile` o `.bashrc` con Bash, o en `.zshenv` con Zsh.

Para remover una variable, usa la opción `-n`:

```
export -n TEST
```

Si se llama a `export` sin ninguna opción, se enlistarán todas las variables exportadas.

El comando crontab de Linux

Los trabajos "cron" son trabajos que están programados para ejecutarse a intervalos específicos. Puedes que tengas un comando que realice algo cada hora, o cada día, o cada 2 semanas. O los fines de semana.

Son muy poderosos, especialmente cuando se usan en servidores para realizar mantenimiento y automatizaciones.

El comando `crontab` es el punto de entrada para trabajar con trabajos cron.

Lo primero que puedes hacer es explorar qué trabajos cron son definidos por ti:

```
crontab -l
```

Puede que no tengas ninguno, como yo:



```
flavio — fish /Users/flavio — -fish — 62x7
[→ ~ crontab -l
crontab: no crontab for flavio
→ ~ █
```

Ejecuta

```
crontab -e
```

para editar los trabajos cron, y agregar otros nuevos.

Por defecto, esto se abre con el editor predeterminado, que suele ser `vim`. Me gusta más `nano`. Puedes usar esta línea para usar un editor diferente:

```
EDITOR=nano crontab -e
```

Ahora puedes agregar una línea por cada trabajo cron.

La sintaxis para definir los trabajos cron es un poco aterradora. Por eso suelo utilizar un sitio web que me ayuda a generarlo sin errores: <https://crontab-generator.org/>

The screenshot shows the Crontab Generator interface. At the top, there's a banner with the text "Get frustrated with Cron on your server? Try our [Webcron Service](#)". Below it, a note says: "If you want to periodically perform a task (e.g. sending Emails, backing up database, doing regular maintenance, etc.) at specified times and dates, there are two ways to set scheduled tasks: Method 1: Use our [online cron job service](#) that will save you a headache. Method 2: Use Cron available in Unix/Linux systems." A note below that says: "If you go with method 2, the following generator can help you produce a crontab syntax that you can copy & paste to your crontab file (You can open the file by using command `crontab -e`). Below the generated crontab syntax, a list of run times will be displayed too. The predictions will help you ensure that you set the time and date right." A heading "Complete the following form to generate a crontab line" is followed by a note "Ctrl-click (or command-click on the Mac) to select multiple entries". The form consists of several sections:

- Minutes:** Radio buttons for "Every Minute", "Even Minutes", "Odd Minutes", "Every 5 Minutes", "Every 15 Minutes", and "Every 30 Minutes". A dropdown menu shows values from 0 to 9.
- Hours:** Radio buttons for "Every Hour", "Even Hours", "Odd Hours", "Every 6 Hours", and "Every 12 Hours". A dropdown menu shows values like "Midnight", "1am", "2am", etc.
- Days:** Radio buttons for "Every Day", "Even Days", "Odd Days", "Every 5 Days", "Every 10 Days", and "Every Half Month". A dropdown menu shows values from 1 to 10.
- Months:** Radio buttons for "Every Month", "Even Months", "Odd Months", "Every 4 Months", and "Every Half Year". A dropdown menu shows months from Jan to Oct.
- Weekday:** Radio buttons for "Every Weekday", "Monday-Friday", and "Weekend Days". A dropdown menu shows days of the week: Sun, Mon, Tue, Wed, Thu, Fri, Sat.
- Command To Execute:** A text input field where the user can type the command to execute.
- Command Examples:** A section with the text "Execute PHP script:" and the example command: "/usr/bin/php /home/username/public_html/cron.php".

Escoge un intervalo de tiempo para el trabajo cron, y escribe el comando a ejecutar.

Elegí ejecutar un script localizado en `/Users/flavio/test.sh` cada 12 horas. Esta es la línea de `crontab` que necesito ejecutar:

```
* */12 * * * /Users/flavio/test.sh >/dev/null 2>&1
```

Ejecuto `crontab -e`:

y agrego esa línea, luego pulso `ctrl-X` y presiono `y` para salvar.

Si todo va bien, el trabajo cron está listo:



```
flavio — bash /Users/flavio — bash — 62x5
bash-3.2$ EDITOR=nano crontab -e
crontab: no crontab for flavio - using an empty one
crontab: installing new crontab
bash-3.2$
bash-3.2$
```

Una vez hecho esto, puedes ver la lista de trabajos cron activos ejecutándose:

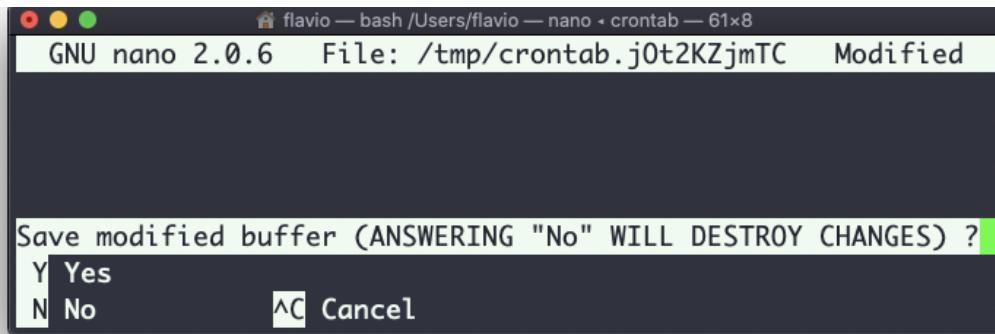
```
crontab -l
```



```
flavio — bash /Users/flavio — bash — 61x8
bash-3.2$ crontab -l
* */12 * * * /Users/flavio/test.sh >/dev/null 2>&1

bash-3.2$
```

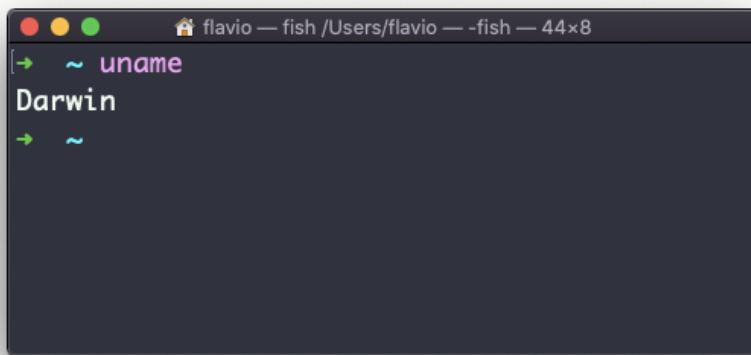
Puedes eliminar un trabajo cron ejecutando `crontab -e` de nuevo, eliminando la línea y saliendo del editor:



```
flavio — bash /Users/flavio — bash — 61x8
bash-3.2$ EDITOR=nano crontab -e
crontab: installing new crontab
bash-3.2$ crontab -l
bash-3.2$
```

El comando `uname` de Linux

Llamar a `uname` sin opciones devolverá el nombre en clave del Sistema Operativo:



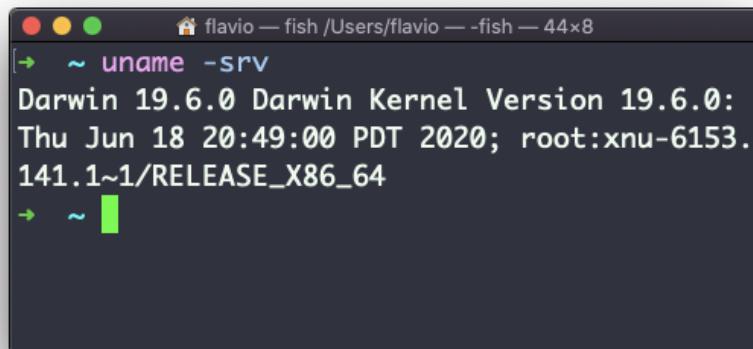
La opción `m` muestra el nombre del hardware (`x86_64` en este ejemplo) y la opción `p` imprime el nombre de la arquitectura del procesador (`i386` en este ejemplo):



```
flavio — fish /Users/flavio — -fish — 44x8
[~] ~ uname -mp
x86_64 i386
[~]
```

A screenshot of a macOS terminal window titled "flavio — fish /Users/flavio — -fish — 44x8". The command "uname -mp" is run, and the output "x86_64 i386" is displayed. The terminal has a dark theme with red, yellow, and green window controls.

La opción `s` imprime el nombre del Sistema Operativo. `r` imprime el último lanzamiento (release) y `v` imprime la versión:



```
flavio — fish /Users/flavio — -fish — 44x8
[~] ~ uname -srv
Darwin 19.6.0 Darwin Kernel Version 19.6.0:
Thu Jun 18 20:49:00 PDT 2020; root:xnu-6153.
141.1~1RELEASE_X86_64
[~]
```

A screenshot of a macOS terminal window titled "flavio — fish /Users/flavio — -fish — 44x8". The command "uname -srv" is run, and the output "Darwin 19.6.0 Darwin Kernel Version 19.6.0: Thu Jun 18 20:49:00 PDT 2020; root:xnu-6153.141.1~1RELEASE_X86_64" is displayed. The terminal has a dark theme with red, yellow, and green window controls.

La opción `n` imprime el nombre de la red de nodos:



```
flavio — fish /Users/flavio — -fish — 44x8
[~] ~ uname -n
mbp.local
[~]
```

La opción `a` imprime toda la información disponible:



```
flavio — fish /Users/flavio — -fish — 44x8
[~] ~ uname -a
Darwin mbp.local 19.6.0 Darwin Kernel Version 19.6.0: Thu Jun 18 20:49:00 PDT 2020; root
:xnu-6153.141.1~1/RELEASE_X86_64 x86_64
[~]
```

En macOS también puedes usar al comando `sw_vers` para imprimir más información sobre el Sistema Operativo de macOS. Ten en cuenta que esto difiere de la versión de Darwin (el Kernel), que arriba muestra que es 19.6.0 .

Darwin es el nombre del kernel de macOS. El kernel es el "núcleo" del Sistema Operativo, mientras que el Sistema Operativo en conjunto se llama macOS. En Linux, Linux es el kernel, y GNU/Linux sería el nombre del Sistema Operativo (aunque todos nos referimos a él como "Linux").

El comando `env` de Linux

El comando `env` puede ser usado para pasar variables de entorno sin configurarlas en el entorno exterior (el shell actual).

ella.

Puedes ejecutar

```
env USER=flavio node app.js
```

y la variable de entorno `USER` será accesible desde la aplicación Node.js a través de la interfaz de Node `process.env`.

También puedes ejecutar el comando que borra todas las variables de entorno ya establecidas, usando la opción `-i`:

```
env -i node app.js
```

En este caso obtendrás un error diciendo `env: node: No such file or directory` porque el comando `node` no es alcanzable, ya que la variable `PATH` usada por el shell para buscar comandos en las rutas comunes no está establecida.

Así que tienes que pasar la ruta completa al programa de `node`:

```
env -i /usr/local/bin/node app.js
```

Inténtalo con un simple archivo `app.js` con este contenido:

```
console.log(process.env.NAME)
console.log(process.env.PATH)
```

Verás la salida como

```
undefined
undefined
```

```
env -i NAME=flavio node app.js
```

y la salida será

```
flavio
undefined
```

Eliminado la opción `-i` hará que `PATH` esté disponible de nuevo dentro del programa:



```
flavio — fish /Users/flavio — -fish — 73x5
|~ env NAME=flavio node app.js
flavio
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/Apple/usr/bin
|~
```

El comando `env` también puede ser usado para imprimir todas las variables de entorno. Si se ejecuta sin opciones:

```
env
```

devolverá una lista de las variables de entorno establecidas, por ejemplo:

```
HOME=/Users/flavio
LOGNAME=flavio
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/Apple/usr/bin
PWD=/Users/flavio
SHELL=/usr/local/bin/fish
```

usando la opción `-u`. Por ejemplo, este código elimina la variable `HOME` del entorno de comandos:

```
env -u HOME node app.js
```

El comando `printenv` de Linux

Aquí está una guía rápida del comando `printenv`, usado para imprimir los valores de las variables de entorno.

En cualquier shell hay un buen número de variables de entorno, establecidas ya sea por el sistema, o por tus propios scripts de shell y configuración.

Puedes imprimirlas todas a la terminal usando el comando `printenv`. La salida será algo como esto:

```
HOME=/Users/flavio
LOGNAME=flavio
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/Apple/usr/bin
PWD=/Users/flavio
SHELL=/usr/local/bin/fish
```

con unas pocas líneas más, por lo general.

Puedes agregar un nombre de variable como parámetro, para mostrar solo el valor de la variable:

```
printenv PATH
```



```
flavio — fish /Users/flavio — -fish — 46x5
[→ ~ printenv PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/
Library/Apple/usr/bin
→ ~ ]
```

Conclusión

Muchas gracias por leer este manual.

Espero que te inspire para aprender más sobre Linux y sus capacidades. Es un conocimiento perdurable que no quedará desactualizado pronto.

¡Recuerda que puedes [descargar este manual en formato PDF / ePUB / Mobi](#) si así lo deseas!

Publico tutoriales de programación todos los días en mi sitio web flaviocopes.com si quieres ver más contenido genial como este.

Puedes contactarme en Twitter [@flaviocopes](https://twitter.com/flaviocopes).

Traducido del artículo de [Flavio Copes' The Linux Command Handbook](#)



Juan Carrillo

A Software Engineer who builds, writes and draws stuff on the web.

If you read this far, thank the author to show them you care. [Say Thanks](#)

Aprende a codificar de forma gratuita. El plan de estudios de código abierto de freeCodeCamp ha ayudado a más de 40,000 personas a obtener trabajos como desarrolladores. [Empezar](#)

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) charity organization (United States Federal Tax Identification Number: 82-0779546)

Nuestra misión: ayudar a las personas a aprender a programar de forma gratuita. Logramos esto mediante la creación de miles de videos, artículos y lecciones de programación interactivas, todo disponibles gratuitamente para el público. También tenemos miles de grupos de estudio de FreeCodeCamp alrededor del mundo.

Las donaciones a freeCodeCamp van dirigidas a nuestras iniciativas educativas y ayudan a pagar servidores, servicios y personal.

Puedes hacer [una donación deducible de impuestos aquí](#).

Guías de tendencias

Git Clone	UX
Métodos Agile	Proceso de Diseño
Python Main	Números Primos
Callback	Diseño de Producto
Debounce	Digital Design
URL Encode	Juegos de Código
Blink HTML	SVM
Python Tupla	JavaScript forEach
JavaScript Push	Google BERT
Java List	Create Table SQL
Diseño Web Responsive	¿Qué es TLS?
¿Qué es un archivo SVG?	¿Qué es una red LAN?
PDF Password Remover	¿Qué es npm?
¿Qué es un Archivo PDF?	Ejemplos de RSync
What Is Python?	Random Forest

Our Charity

[Acerca de](#) [Red de ex-Alumnos](#) [Código abierto](#) [Tienda](#) [Soporte](#) [Patrocinadores](#) [Honestidad Académica](#)

[Código de Conducta](#) [Política de privacidad](#) [Términos de servicio](#) [Política de derechos de autor](#)