**Introducción a Git y a la línea de comandos**

Hoy en día, estamos acostumbrados a interactuar con los programas a través de interfaces gráficas como botones, formularios y menús desplegables. Por ejemplo, para crear una carpeta en tu computadora, puedes hacer clic derecho y seleccionar la opción necesaria en el menú gráfico. Esta interfaz de "apuntar y hacer clic" puede ser sumamente práctica, pero existen otras interfaces.

Además de la interfaz gráfica de tu computadora, también puedes usar una interfaz basada en texto: la línea de comandos. Puedes utilizar la línea de comandos para gestionar muchas de las tareas que gestiona la interfaz gráfica, tales como crear, mover o eliminar archivos. Pero la línea de comandos también te da acceso a muchas herramientas de desarrollo que son difíciles, por no decir imposibles, de utilizar a través de una interfaz gráfica. Un ejemplo de esto es Git, un sistema de control de versiones que usarás a lo largo de este programa y en tu carrera como ingeniero o ingeniera de software.

En este capítulo, te presentaremos Git brevemente y te mostraremos cómo instalarlo en tu computadora. Después de eso, te explicaremos los conceptos básicos de la línea de comandos.

# ¿Qué es la línea de comandos?

Comencemos a explorar la línea de comandos explicando parte de la terminología. Como dijimos en la lección de introducción, la línea de comandos es una interfaz basada en texto para interactuar con una computadora. Podemos acceder a esta interfaz a través de programas llamados terminales o emuladores de terminal.

Más específicamente, el emulador de terminal interactúa con un shell, un programa que interpreta las entradas de la línea de comandos y las traduce en instrucciones para el sistema operativo.

Los programas shell más comunes en macOS y Linux son bash y zsh, los cuales comparten interfaces similares, así que sus comandos, normalmente, son interoperables.

Al emulador de terminal predeterminado en estos sistemas se le conoce, simplemente, como Terminal. Pero, los programas shell y terminal predeterminados en Windows son muy distintos de bash y zsh. Es por ello que tuviste que instalar Git en la lección anterior. Para los usuarios de Windows, Git Bash ya viene incluido en Git, lo que proporciona un shell bash y un emulador de terminal en Windows.

### Abrir la terminal

Ahora, manos a la obra. Para abrir tu emulador de terminal:

* En Windows, abre Git Bash.
* En macOS o Linux, abre Terminal.

💡 También puedes acceder a la línea de comandos directamente desde VS Code. El atajo para abrirlo es Ctrl + ` tanto en Windows como en macOS. Si estás en Windows, debes cambiar tu perfil de terminal predeterminada a Git Bash. En [este video](https://www.youtube.com/embed/3oF9diyu7hc) te explicamos cómo.

Cuando abras la terminal, lo que veas dependerá mucho del emulador de terminal que estés utilizando, pero las características generales de la línea de comandos siguen siendo las mismas. A continuación, puedes ver un ejemplo de un emulador de terminal de Linux.



1. A la izquierda, puedes ver el símbolo del sistema. La información que se incluye en el mensaje varía de sistema en sistema. En este caso, vemos el nombre de usuario student y el nombre de computadora tripleten, seguidos de los caracteres ~$. La ~ tiene un significado especial del cual hablaremos pronto. El carácter $ suele emplearse para denotar el final del símbolo del sistema.
2. A la derecha, puedes ver el campo de entrada donde escribes tus comandos.

Y eso es todo lo que necesitas saber para empezar a utilizar la línea de comandos: dónde escribir los comandos. Así que, ¡aprendamos sobre comandos!

**Familiarizándote con pwd y ls**

💡 En este capítulo, vamos a utilizar la interfaz English OS con nombres estándar de carpetas como *Desktop* ("Escritorio"), *Downloads* ("Descargas"), etc. El inglés es el idioma estándar para la tecnología, especialmente en desarrollo web. Emplear una interfaz de sistema operativo en inglés favorece tanto a la compatibilidad global como al uso de un idioma estandarizado para una comunicación efectiva en la industria de la tecnología.

Muchas de las tareas más comunes de la línea de comandos implican interactuar con el sistema de archivos de tu computadora. Piensa en cómo has interactuado con el sistema de archivos a través de la interfaz gráfica de usuario (GUI). Crear archivos, eliminar archivos, copiar o cambiar el nombre de archivos: estas tareas también se pueden gestionar en la línea de comandos. Pero antes de interactuar con un archivo o carpeta específica, debemos saber cómo encontrarla en la estructura de archivos de nuestra computadora.

Siempre que utilizas una interfaz gráfica, estás en una carpeta. Lo mismo ocurre con la línea de comandos. Al ejecutar el comando pwd, podemos mostrar nuestra ubicación actual. Las letras del comando pwd significan Print Working Directory ("mostrar directorio de trabajo") y eso es exactamente lo que pasará al introducir este comando:

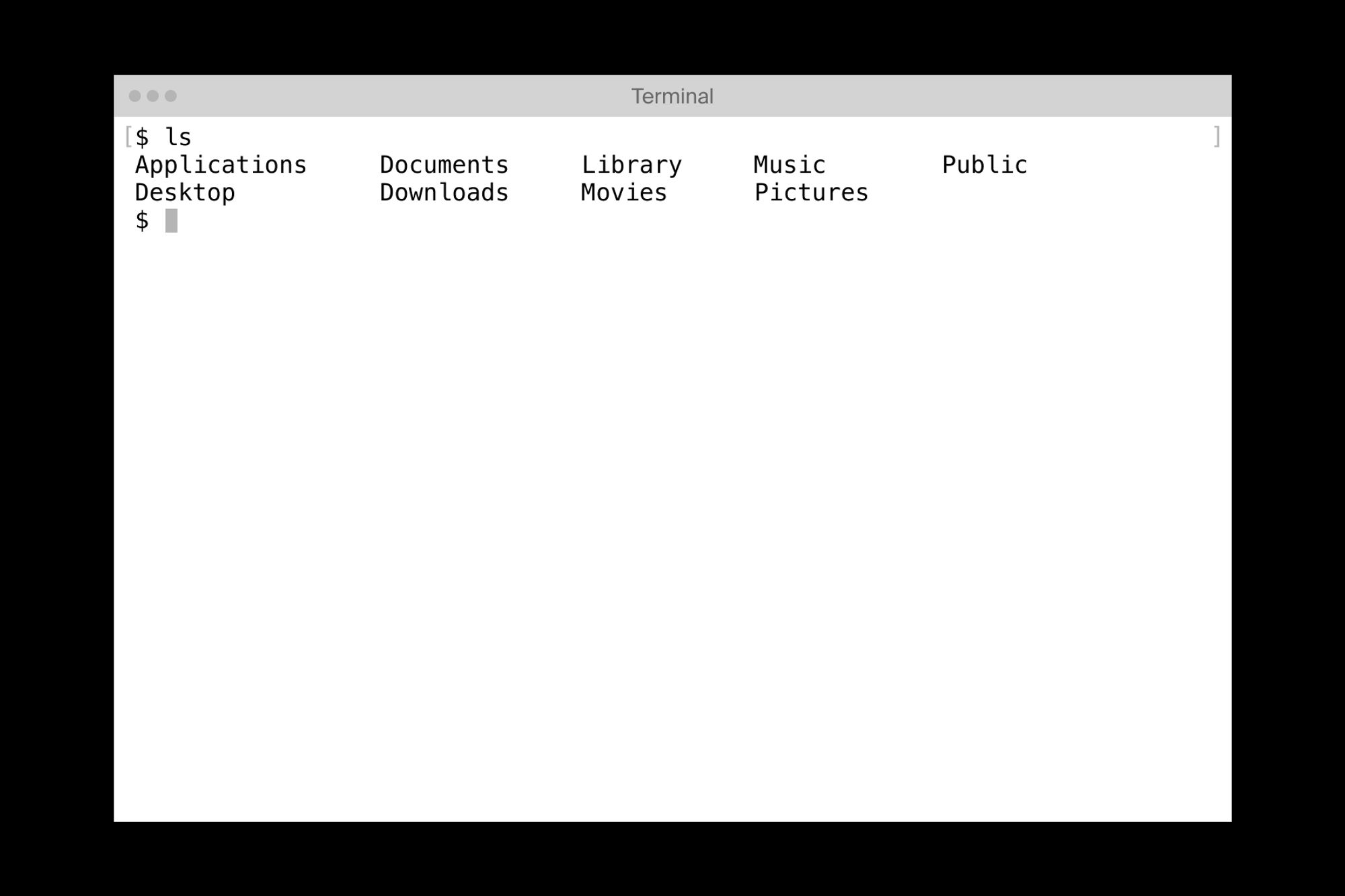
pwd

Después de introducir nuestro comando y presionar Enter, se mostrará la ruta de nuestra carpeta en la siguiente línea. Ahora verás tu nombre de usuario en lugar de TU\_NOMBREDEUSUARIO.

/Users/TU\_NOMBREDEUSUARIO

Este es tu directorio de usuario (o directorio de inicio), y siempre que abras la terminal, aquí es donde comenzarás por defecto.

En una interfaz gráfica, siempre que abras una carpeta, podrás ver el contenido de esa carpeta. Este no es el caso en la línea de comandos. En su lugar, tendremos que utilizar el comando ls para ver nuestros archivos cuando usemos la línea de comandos. Si introduces este comando, verás una lista de los archivos y carpetas dentro del directorio actual.



Pregunta

¿Qué comando mostrará la ruta hacia el directorio de trabajo actual?

pwd

¡Correcto! El comando ls se usa para enumerar el contenido del directorio actual. Y si bien a veces verás que se usa la abreviatura "cwd" para referirse al directorio de trabajo actual, el comando para mostrarlo es pwd.

cwd

ls

¡Perfecto!

# Familiarizándote con cd

A estas alturas ya sabes cómo averiguar dónde te encuentras en el sistema de archivos y ver el contenido de esa ubicación. Así que ahora vas a aprender a desplazarte mediante el comando cd. Aquí, cd significa "cambiar de directorio". La sintaxis consiste en el comando seguido por el directorio deseado, de esta manera: cd NOMBRE\_CARPETA. Aquí tienes un ejemplo:

*# comprueba dónde estamos*

pwd

*# salida:*

/Users/TU\_NOMBREDEUSUARIO

*# mover a la carpeta de Escritorio*

cd Desktop

*# comprueba dónde estamos ahora*

pwd

*# salida:*

/Users/TU\_NOMBREDEUSUARIO/Desktop

*# intentar utilizar cd para entrar a una carpeta que no existe genera un error*

cd carpeta-no-existente

*# salida*

cd: carpeta-no-existente: No such file or directory

💡 En el bloque de código anterior, hacemos uso de los comentarios al estilo bash. En el lenguaje bash, los comentarios van precedidos por un sólo carácter #. Técnicamente, estos comentarios deberían colocarse al final de las líneas de comando correspondientes. Por ejemplo:

pwd *# se ignora todo lo que está a la derecha del hash.*

Pero los hemos colocado arriba de las líneas de comando para mejorar la legibilidad. También estamos utilizando los comentarios para distinguir entre los comandos y su salida.

Observa también que estamos empleando un guion bajo para separar las letras mayúsculas para representar el contenido de la variable, como NOMBRE\_CARPETA y TU\_NOMBREDEUSUARIO. Esta es una convención que usaremos para resaltar partes de los comandos que no deberían ejecutarse tal como se muestran. Por ejemplo, en el comando cd NOMBRE\_CARPETA, debes remplazar NOMBRE\_CARPETA con el nombre de un archivo real.

## Espacios en los nombres del archivo o carpeta

Te recomendamos que evites dejar espacios en los nombres de archivos o carpetas que vayas a crear, ya que te será difícil trabajar con ese tipo de archivos en la línea de comandos y rutas de archivos de tus proyectos.

*# cd no funcionará si intentas emplear un nombre de carpeta que tenga un espacio*

cd My Videos

*# salida*

cd: too many arguments

Pero, si no pueden evitarse los espacios, tienes dos opciones para manejarlos. Primero, puedes utilizar comillas:

cd "My Videos"

Por otra parte, puedes emplear el carácter de diagonal invertida \ para escapar el espacio en el nombre de la carpeta:

cd My\ Videos

## Volver al directorio padre

Volvamos a donde dejamos el primer bloque de código en esta lección.

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO/Desktop

Hemos utilizado cd para entrar en nuestro directorio Desktop. Pero ¿qué tal si queremos volver al directorio padre, /Users/TU\_NOMBREDEUSUARIO?

Para emplear cd para entrar al padre inmediato del directorio actual de trabajo, escribe dos puntos .. en lugar del nombre de carpeta.

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO/Desktop

*# usar dos puntos nos hace subir un nivel*

cd ..

*# veamos dónde estamos ahora*

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO

También podemos agregarlos a los ... Por ejemplo, el comando cd ../UNA\_CARPETA intentaría cambiar el directorio en un directorio padre y, de ahí, en UNA\_CARPETA.

## Volviendo al inicio

Cada sesión de la línea de comandos empieza en el directorio de inicio, así que es una tarea común cambiar directorios de vuelta a este directorio. Pero, supón que estás en lo más profundo de la estructura de archivos.

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO/Desktop/pics/kittens

Podrías volver al directorio de inicio con cd de esta manera:

*# utiliza cd para subir tres niveles en la estructura de archivos*

cd ../../..

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO

Pero es tedioso y susceptible a errores. Afortunadamente, existe un atajo: cd ~. La ~ representa el directorio de inicio del usuario (o directorio de usuario), y ejecutar cd ~ te devolverá a él desde donde sea que estés en ese momento. También puedes emplear ~ como un segmento en rutas de archivo más largas. Por ejemplo, cd ~/Desktop utilizará cd para entrar en el directorio Desktop que está dentro del directorio de inicio.

Sólo ten en cuenta que existe un atajo aún más corto para volver al directorio de inicio, ya que ejecutar cd sin ningún otro argumento tiene el mismo efecto que ejecutar cd ~.

## Volver al directorio anterior

A veces resulta útil emplear cd para volver a entrar a la última ubicación en que estuviste. Por ejemplo, supongamos que estás en tu directorio web\_project\_library, pero por alguna razón, debes usar cd para entrar en tu directorio de inicio.

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO/projects/web\_project\_library

*# volver a inicio*

cd

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO

Puedes utilizar cd para volver a entrar a tu directorio del proyecto con cd /Users/TU\_NOMBREDEUSUARIO/projects/web\_project\_library, pero también existe un atajo.

*# cambia el directorio al último directorio visitado*

cd -

pwd

*# resultado*

/Users/TU\_NOMBREDEUSUARIO/projects/web\_project\_library

## Mantenerse orientados fácilmente

Te hemos mostrado cómo orientarte en la línea de comandos a través del comando pwd. Pero, en muchos sistemas, esta información está disponible de un solo vistazo. Recordemos el ejemplo de un símbolo del sistema que te mostramos en una lección anterior:

student@tripleten ~$

La ~ representa el directorio actual de trabajo, que en este caso, es el directorio de inicio.

student@tripleten ~$ pwd

*# salida*

/Users/student

Si cambiamos de directorio, podemos ver su cambio en el símbolo del sistema:

student@tripleten ~$ cd Downloads

student@tripleten ~/Downloads$

## Trabajar con rutas de archivo en la línea de comandos

Imagina que estás en tu directorio de usuario, y te gustaría usar cd para entrar en una carpeta llamada notes que está en tu escritorio. Podrías hacer esto:

cd Desktop

cd notes

pwd

*# salida*

/Users/TU\_NOMBREDEUSUARIO/Desktop/notes

Sin embargo, podemos lograr esto en un solo comando, de esta manera:

cd Desktop/notes

Observa que la ruta de archivo Desktop/notes comienza con el nombre de un directorio dentro del directorio actual de trabajo. Probablemente recuerdes del capítulo sobre sistemas de archivos, que el directorio actual de trabajo suele representarse con ./. Esta convención viene de la sintaxis de la línea de comandos, y el siguiente comando se comportaría igual que el anterior.

cd ./Desktop/notes

Pero, tal y como vimos en el capítulo sobre sistemas de archivos, una ruta de archivo que comience con una / es algo completamente diferente.

*# ups, no existe tal directorio*

cd /Desktop/notes

*# salida*

cd: /Desktop/notes: No such file or directory

Esto da como resultado un error, ya que la primera / representa el directorio raíz, no el directorio de inicio donde se ubica el directorio Desktop/notes.

Pregunta

1. Verdadero o falso. Cuando abres la terminal por primera vez, empiezas en tu directorio de usuario.

Verdadero

¡Correcto! Es importante reconocer este hecho, pues no entenderlo podría dar lugar a muchos errores.

Falso

¡Buen trabajo!

Pregunta

2. Imagina que estás en tu directorio de usuario y quieres utilizar **cd** para entrar en tu directorio de descargas, **Downloads**, que está dentro del directorio de usuario. ¿Qué comandos harán la magia?

Elige tantas como quieras

cd Downloads

Correcto.

cd /Downloads

cd ~/Downloads

Correcto.

cd /.Downloads

cd ./Downloads

Correcto.

¡Bien hecho!

Pregunta

3. Imagina que estás en el directorio **~/projects/sample-project/images**. Deseas volver al directorio **projects**. ¿Cuál de los siguientes comandos lo hará?

Elige tantas como quieras

cd ..

cd ../..

Correcto. Este comando te llevará al padre del padre del directorio images, que es projects.

cd ~/projects

Correcto. Este comando utiliza una ruta de archivo absoluta para llegar a la ubicación deseada. A veces, es más fácil hacer esto que contar mentalmente el número de segmentos ../ que debes utilizar.

cd projects

¡Buen trabajo!

# Crear archivos y carpetas

Bien, ahora que ya sabes cómo navegar por la estructura de archivos mediante la línea de comandos, llegó el momento de aprender cómo realizar algunas tareas simples del sistema de archivos con ella.

💡 En las próximas lecciones, vamos a trabajar con una carpeta de muestra llamada sample-project. Esta carpeta es sólo para fines ilustrativos. Te recomendamos que vayas siguiendo estas lecciones al tiempo que ejecutas los comandos en tu computadora. Sin embargo, recuerda que no necesitarás ninguno de los archivos ni carpetas de este capítulo en tus proyectos futuros.

En este video, te mostraremos cómo crear y eliminar archivos o carpetas con la línea de comandos.

## Crear directorios con mkdir

Con el comando mkdir (se pronuncia "méik-dir"), podemos crear una nueva carpeta justo desde dentro de la terminal. Vamos a crear una carpeta llamada projects dentro de tu directorio de usuario. Para ello, primero asegúrate de estar en tu directorio de usuario ejecutando cd ~. Después utiliza el comando mkdir para crear un nuevo directorio:

*# ve a tu directorio de usuario*

cd ~

*# mkdir seguido del nombre de directorio deseado (projects) crea un directorio*

mkdir projects

Después, vamos a ingresar cd para entrar a nuestro directorio recién creado, y luego crearemos otra carpeta dentro de esta a la que llamaremos sample-project. Y entonces, utilizaremos nuevamente cd para entrar en ella.

*# entra a la carpeta projects*

cd projects

*# comprueba dónde estamos*

pwd

*# salida*

/Users/YOUR\_USERNAME/projects

*# crea una nueva carpeta*

mkdir sample-project

*# utiliza cd para entrar en esa carpeta*

cd sample-project

*# comprueba dónde estamos*

pwd

*# salida*

/Users/YOUR\_USERNAME/projects/sample-project

## Crear archivos con touch

A continuación, vamos a crear algunos de los archivos que necesitaremos. Hay algunas maneras en que podemos crear un nuevo archivo en la línea de comandos. La primera es el comando touch.

*# este comando creará un nuevo archivo llamando a index.html en el*

*# directorio actual de trabajo, si aún no existe.*

touch index.html

Ahora, usa el comando touch para crear un archivo llamado style.css en el mismo directorio.

touch style.css

En este punto, deberías tener una estructura de archivos que puede visualizarse así:

~/

projects/

sample-project

index.html

style.css

En otras palabras, tu directorio de inicio ahora contiene un directorio projects con un directorio sample-project dentro de él. Dentro del directorio sample-project, debe haber dos archivos: index.html y style.css.

## Trabajar con rutas de archivo en la línea de comandos

Tanto touch como mkdir te permiten crear archivos en cualquier lugar, no sólo en el directorio actual de trabajo. Todo lo que debes hacer es proporcionar una ruta relativa o absoluta en la que deseas crearlo. Así que, en realidad no teníamos que utilizar el comando cd en nuestro ejemplo anterior. En cambio, pudimos haber hecho esto:

*# ir al directorio de inicio*

cd ~

*# crear un directorio en el directorio actual de trabajo*

mkdir projects

*# crear un directorio dentro del directorio projects*

mkdir projects/sample-project

*# crear un archivo dentro de projects/sample-project*

touch projects/sample-project/index.html

Ten en cuenta que todos estos comandos sólo funcionarán si existen todas las carpetas intermedias. Por ejemplo, ejecutar mkdir projects/sample-project fallaría si no hay un directorio projects en el directorio actual de trabajo. Pero, si aún no existen las carpetas, puedes indicarle a mkdir que las cree todas en la ruta de archivos a través del flag -p.

*# con el flag -p, se creará el directorio projects si este aún no existe*

mkdir -p projects/sample-project

Pregunta

1. ¿Qué condiciones deben cumplirse antes de crear un nuevo directorio mediante la línea de comandos?

Piensa en un nombre adecuado para tu directorio.

¡Correcto! Un directorio debe tener un nombre que describa su propósito.

Navega hacia una carpeta padre de tu nuevo directorio.

Poder hacer clic derecho para que aparezca el menú contextual.

¡Perfecto!

Pregunta

2. Escribe un comando para crear un directorio llamado **my-folder** en el directorio actual de trabajo.

¡Correcto!

¡Lo has entendido bien!

Pregunta

3. Ahora, escribe un comando para crear un archivo **my-file** dentro del directorio **my-folder**. No uses **cd** para entrar primero en el directorio **my-folder**.

¡Correcto!

¡Buen trabajo!

**Inicializar Git y enviar tus cambios al área de staging**

Para habilitar el control de versiones, debemos convertir nuestra carpeta del proyecto en un repositorio Git. Una vez que hagamos esto, Git comenzará a rastrear todos los cambios realizados en el directorio de nuestro proyecto.

Para empezar, ve a la terminal y navega de vuelta a la carpeta sample-project que creaste en la lección "Crear archivos y carpetas".

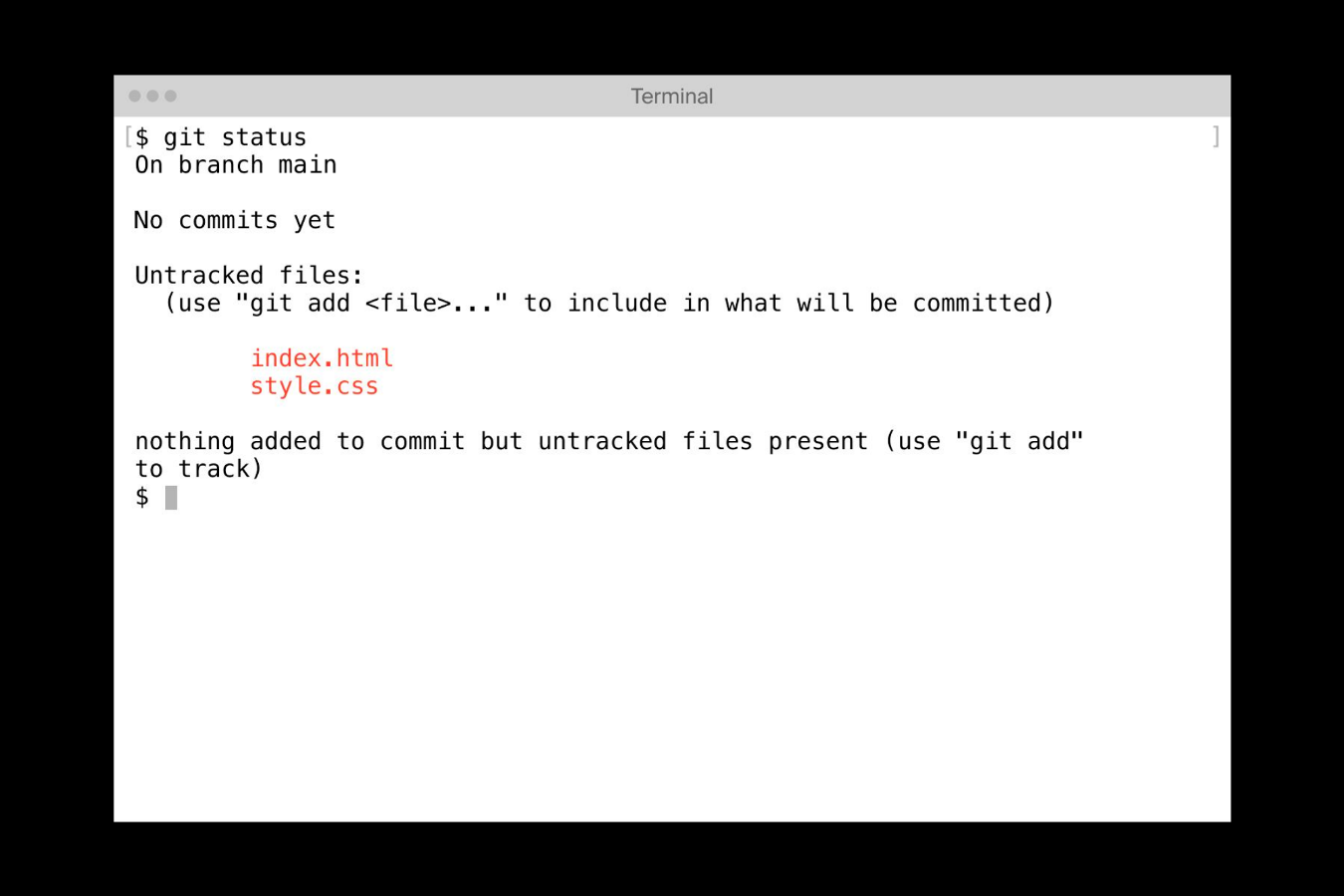
💡 Asegúrate de estar dentro del directorio de trabajo que has creado en las anteriores lecciones antes de proceder. Si seguiste las instrucciones en nuestra lección sobre la línea de comandos, puedes volver allí con cd ~/projects/sample-project.

Una vez estés en la carpeta sample-project, puedes convertirla en un repositorio Git con el comando git init:

git init

El símbolo del sistema te mostrará un mensaje confirmando que el repositorio fue iniciado.

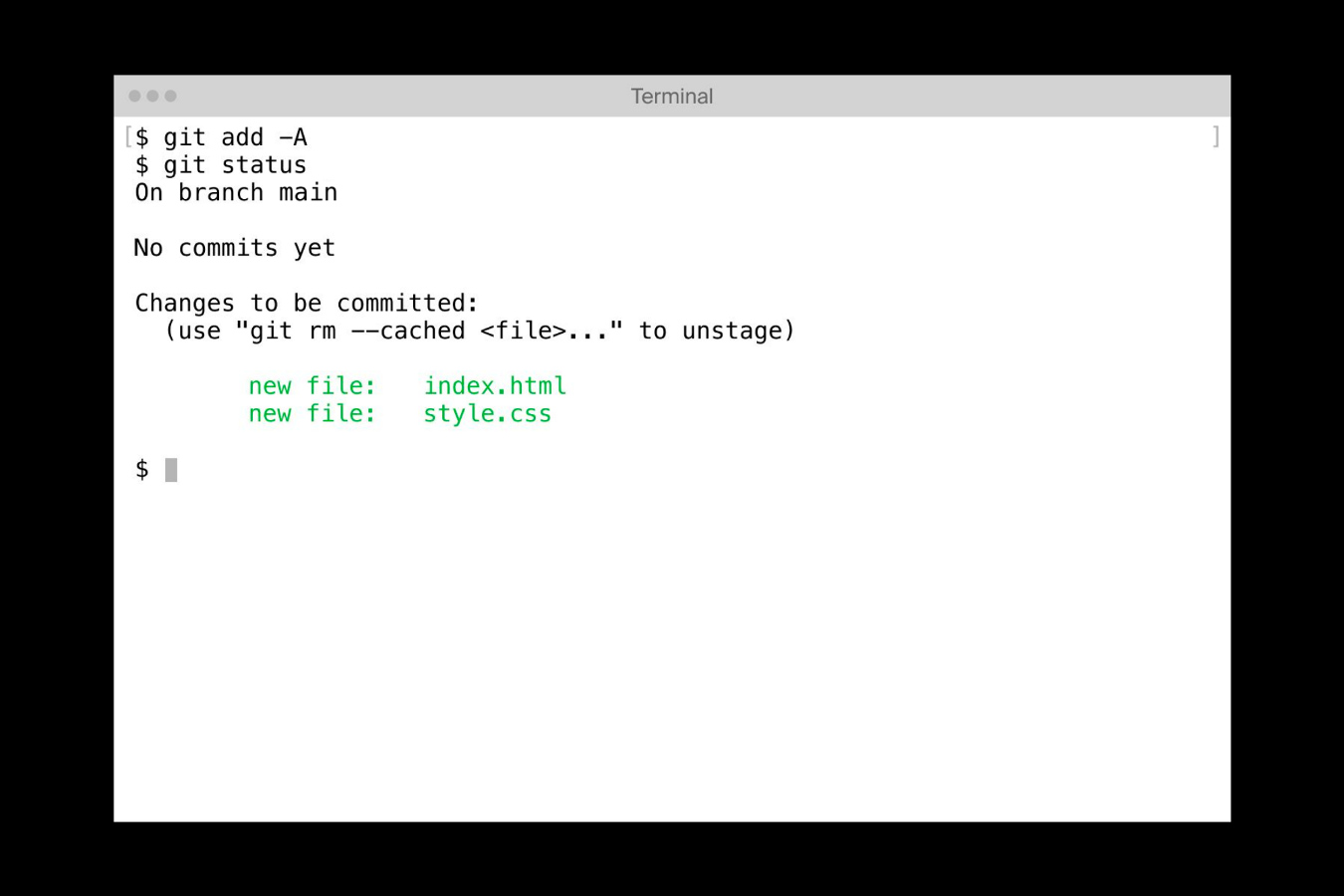
Cuando hayas terminado, en la misma carpeta ejecuta el comando git status. Recibirás un mensaje de Git mostrándote que esta carpeta contiene dos archivos: index.html y style.css.



En este momento, Git clasifica nuestros archivos como "archivos sin seguimiento". En cualquier momento, podemos decirle a Git que el código, en su versión actual, debe ser designado como una versión particular de nuestro proyecto. Puedes pensar esto como si estuviéramos creando una foto instantánea de nuestro código, a la que podemos volver cuando lo necesitemos. En Git, estas instantáneas son los commits. Sin embargo, ahora mismo, si ejecutáramos el comando git-commit, estos archivos no rastreados no formarían parte de esta versión.

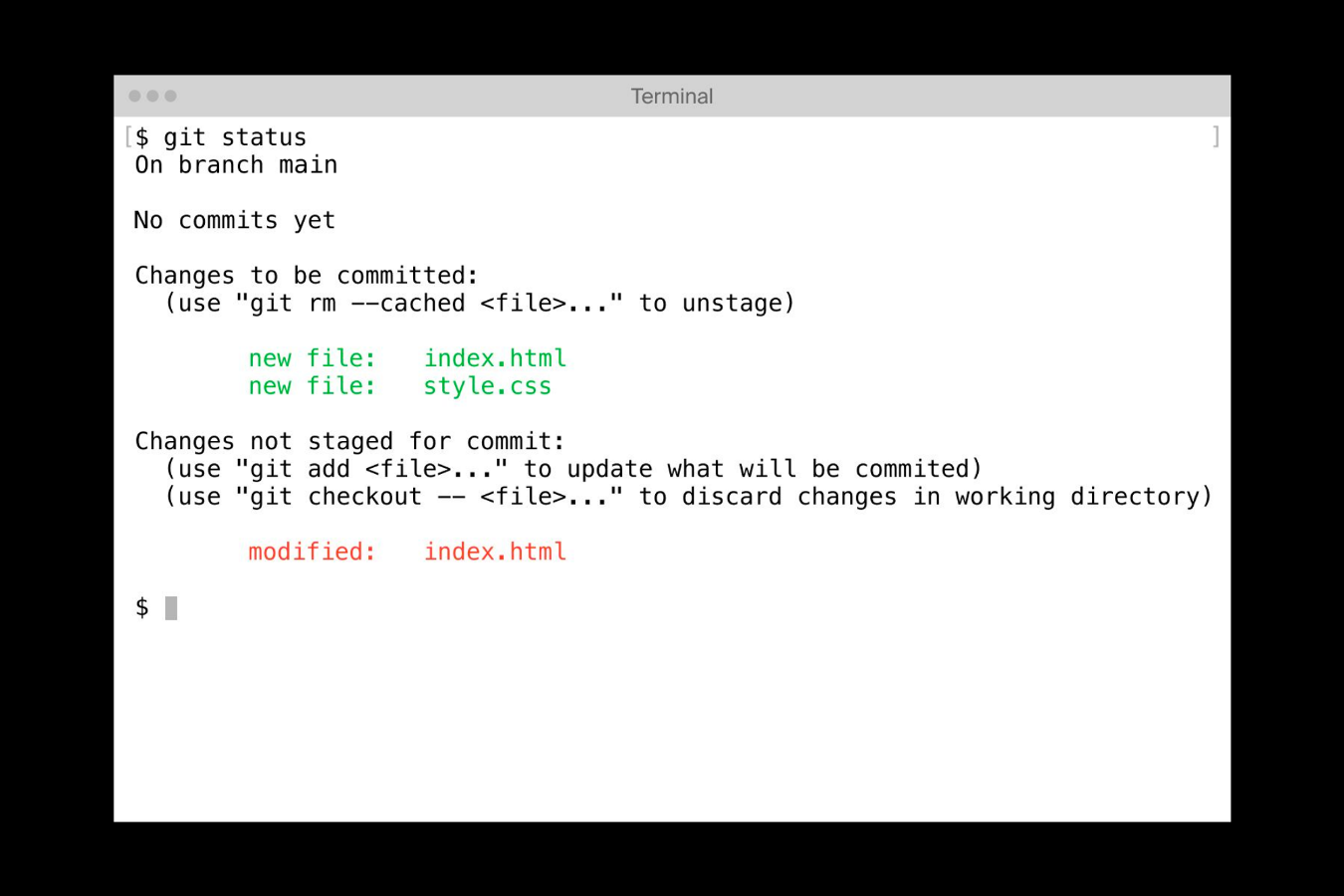
Esta vez no queremos que estos archivos queden fuera. Nuestro objetivo es aprender a guardar diferentes versiones. Para hacerlo, debemos preparar (mover al área de staging) los archivos. "Staging" significa marcar los archivos como listos para ser confirmados ("committed") para que Git sepa que debe agregarlos a nuestra nueva versión. Haremos esto utilizando el comando git add. Podemos especificar los archivos que se agregarán indicando el nombre del archivo que deseamos conservar después del comando: git add index.html.

Si queremos guardar el estado actual de todos los archivos, podemos agregar la opción --all. git add --all o git add -A para abreviar:



Si ejecutamos git status ahora, las versiones actuales de los archivos resaltados en verde están listas para ser guardadas. Eso es lo que el comando git add le dice a Git. Pero es importante que tengas en cuenta que todavía no hemos guardado nada.

Si editamos cualquier archivo en este momento, se volverá a poner en rojo:



Si modificamos un archivo, aunque ya lo hayamos añadido para ser confirmado, tendremos que usar git add para añadirlo de nuevo:

git add index.html

Si tienes que agregar varios archivos a la vez, puedes especificarlos de esta manera:

*# agrega dos archivos al mismo tiempo*

git add index.html style.css

También puedes agregar archivos y directorios dentro del directorio actual de trabajo ejecutando git add .. Sólo ten en cuenta que este comando no lleva los archivos eliminados al área de staging. Supongamos que eliminaste index.html de la carpeta. Tras ejecutar git add ., este archivo aún existe en el área de staging y está listo para ser guardado en el repositorio Git. Es por ello que te recomendamos usar git add -A, que agrega todos los archivos modificados al área de staging y hace un seguimiento de los archivos eliminados.

Pregunta

¿Qué comando usamos para iniciar Git en un proyecto?

git --version

git config

git init

¡Eso es! Si vas al directorio del proyecto y ejecutas el comando git init, Git comenzará a rastrear los archivos en el directorio que hayas seleccionado.

¡Bien hecho!

Pregunta

¿Qué comandos le indican a Git que debe añadir todos los archivos para guardarlos?

Elige tantas como quieras

cd git

git add --all

¡Correcto! add --all significa "añadir todos los archivos".

git add -A

¡Correcto! Esta es la versión abreviada del comando git add --all.

# GitHub y las colaboraciones

Hasta ahora, sólo hemos utilizado Git localmente, pero vamos a necesitar un repositorio remoto para colaborar con otras personas. Un repositorio remoto es una versión de tu código almacenada en un servidor, y que puede accederse por varios usuarios para trabajar en colaboración.

Con un repositorio remoto, tú y tus colaboradores pueden trabajar en la misma base de código desde diferentes ubicaciones y llevar un registro de los cambios de cada quien. Muchas plataformas nos permiten hacerlo, pero GitHub es la más popular, y es la que vamos a usar.

## Más sobre GitHub

GitHub es una plataforma para el desarrollo de software colaborativo sobre la base del sistema de control de versiones de Git. Un "hub" es el centro de una actividad, una región o una red. GitHub se ha convertido en el mayor centro de flujos de datos que utilizan el control de versiones Git.

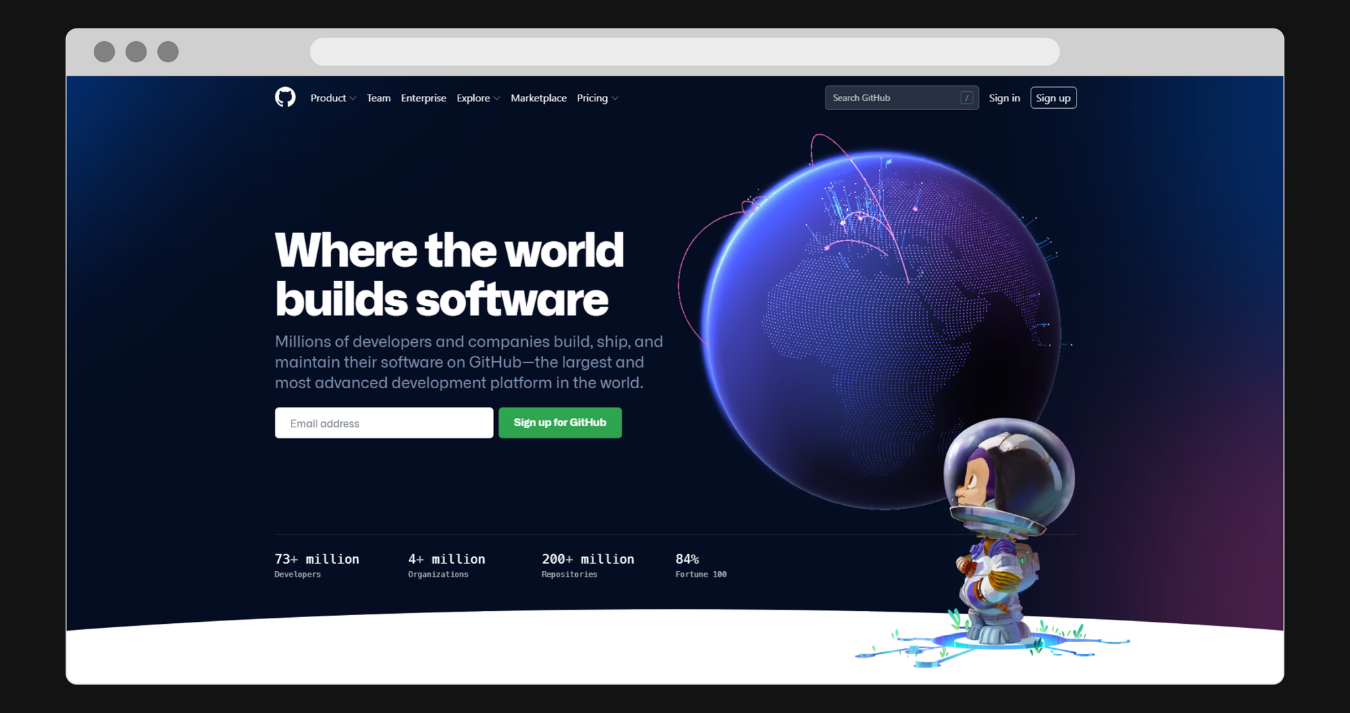
GitHub también funciona como una red social para la ingeniería de software. Desde su fundación en 2008, GitHub [según sus propias estadísticas](https://octoverse.github.com/) (materiales en inglés), ha conectado a millones de personas, permitiéndoles hacer realidad sus sueños e intercambiar información libremente.

Con GitHub, puedes crear una cuenta, publicar tu código y colaborar en cualquier proyecto de código abierto. Además, puedes publicar un sitio web de forma gratuita utilizando [GitHub Pages](https://pages.github.com/).

Muchas compañías grandes, como Facebook, Twitter, Google, Apple y Valve mantienen sus proyectos en GitHub. Es la plataforma perfecta para perfeccionar tus habilidades de Git.

GitHub también te da una ventaja en tu futura carrera en tecnología. Aquí es donde armarás tu porfolio o tus proyectos técnicos, lo que irás haciendo paso a paso durante tus estudios.

## Inscribirse

Página principal de GitHub.

Registrarse es fácil. Ve a [GitHub](https://github.com/join?ref_cta=Sign+up), introduce tu correo electrónico y crea un nombre de usuario y una contraseña. Se te formularán varias preguntas sobre tu profesión, tus conocimientos y tus habilidades. Si quieres, puedes saltarte la encuesta. Desplázate hasta el final de la página web y haz clic en el botón "Complete setup" (Completar configuración). Recibirás un correo electrónico con un enlace para activar tu cuenta. Haciendo clic en ese enlace, verificarás tu cuenta.

¡Ya te has registrado en GitHub! Afortunadamente, todo ha sido sencillo. Ahora debes estar en una página donde puedes crear tu primer repositorio remoto. Eso es lo que haremos en la próxima lección.

# Autenticar tu cuenta con SSH

Tras crear un repositorio en GitHub necesitarás clonarlo a tu equipo local y autenticar tu cuenta, ya sea a través de HTTPS o de SSH. Para utilizar HTTPS, tienes que configurar un "personal access token": un token de acceso personal que sustituye a tu contraseña cuando se interactúa con GitHub mediante la línea de comandos. Las instrucciones para hacer esto las encontrarás en los [documentos de GitHub](https://docs.github.com/es/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens).

La segunda solución, y la preferida por muchos desarrolladores, es emplear una herramienta llamada SSH. En esta lección se describirá de qué trata y qué necesitas hacer para utilizarla con GitHub.

## ¿Qué es el SSH?

En una red, los PCs intercambian datos. Con este fin, se conectan entre sí mediante protocolos de red.

Imagina que tienes que hacer llegar un mensaje desde tu balcón hasta tu amigo, que está al otro lado de la calle. Tenemos varias opciones. Puedes gritarle desde la ventana a tu amigo al otro lado. Mientras no haya nadie escuchando a escondidas, nadie sabrá de qué trató la conversación. Si hubiera alguien escuchando, oirá todo lo que dijiste.

Para evitar que la gente te pueda escuchar, puedes escribir tu mensaje en un código secreto tan solo conocido por ambos. Puedes escribir el mensaje en un papel, doblarlo en forma de avión y lanzárselo a tu amigo. Incluso si alguien lo interceptara, no sabría lo que significa y tu amigo podría decodificarlo cuando le llegue. Así es como funciona el encriptado y es uno de los principios fundamentales de SSH.

El protocolo de capa de transporte SSH (Secure Shell) es el método del "código secreto". A través de este protocolo puedes recibir datos de un equipo remoto o escribir datos en él. Encripta el tráfico y, de este modo, lo hace seguro. Por ahora, solamente necesitamos SSH para interactuar con GitHub. Más tarde, nos hará falta en el servidor remoto.

En SSH, necesitas una contraseña secreta para codificar y decodificar los mensajes. Se llaman "keys" (claves) y constan de una "public key" (clave pública) y una "private key" (clave privada). Juntas, conforman un "key pair" (pareja de claves).

Para configurar tus claves SSH, también puedes seguir la [propia guía de GitHub](https://docs.github.com/es/authentication/connecting-to-github-with-ssh). Si quieres adquirir un poco de práctica leyendo documentación de terceros, ¡échale un vistazo!

## Comprobar una clave SSH

Antes de sumergirnos en las instrucciones de texto para configurar una clave SSH en GitHub, queremos mostrarte un video que puedes usar para ayudarte a continuar:

Primero, asegúrate de que aún no tienes ninguna clave SSH. Si ya la tienes, puedes utilizar esa. Si no, vamos a crear una ahora mismo.

Por defecto, todas las claves SSH se almacenan en el directorio home (inicio) de tu PC:

*# lista las claves SSH que ya habían sido generadas*

ls -al ~/.ssh

Si la carpeta está vacía o si recibes un mensaje "No existe tal archivo o directorio", entonces todo está bien.

Si aparece una de las siguientes opciones con un nombre similar, significa que ya se había generado una clave SSH:

* id\_dsa.pub
* id\_ecdsa.pub
* id\_ed25519.pub
* id\_rsa.pub

Si tienes alguna que no hayas creado, puedes ignorarlas y seguir creando una.

Si no sabes cómo, puedes revisar la [guía de GitHub](https://docs.github.com/es/authentication/connecting-to-github-with-ssh), que examina con más detenimiento algunos aspectos.

## Generar la clave SSH

1. Vamos a generar las claves privadas y públicas en la terminal. La clave pública tiene una extensión .pub, pero no es el caso de la clave privada. Ambas claves se guardarán en tu PC. La pública es necesaria para poder trabajar con Git. Asegúrate de indicar en el campo del correo electrónico la dirección de correo que está vinculada a tu cuenta de GitHub:
2. ssh-keygen -t ed25519 -b 4096 -C "tucorreo@gmail.com"

Verás el siguiente mensaje:

> Generating public/private ed25519 key pair.

1. Especifica la ubicación de almacenamiento de las claves. Lo más sencillo es aceptar la opción predeterminada, que es un directorio oculto llamado ~/.ssh. Para hacerlo, presiona Enter:

macOS

> Enter a file in which to save the key (/Users/you/.ssh/id\_ed25519): [Press enter]

Windows

> Enter file in which to save the key (/c/Users/you/.ssh/id\_ed25519): [Press enter]

Ahora aparecerán un par de claves en el directorio especificado.

1. Crea una contraseña para acceder a la clave SSH. Esta debe ser introducida cada vez que te conectes a través del protocolo. Así que tendrás que recordar esta contraseña (o apuntarla):
2. > Enter passphrase (empty for no passphrase): [Type a passphrase]
3. > Enter same passphrase again: [Type passphrase again]

También puedes dejar el campo en blanco, para que nunca necesites introducir una contraseña. Para hacerlo, presiona Enter.

¡Se creó la clave SSH!

Después, agrega la clave al ssh-agent. Es un programa que puede almacenar y realizar un seguimiento de las claves SSH que utilizas. Ejecuta el comando ssh-agent en segundo plano:

eval $(ssh-agent -s)

Después de ejecutar este comando, deberás ver el siguiente texto:

Agent pid [number]

1. Vincula la clave privada al ssh-agent. Luego, puedes omitir introducir una contraseña cada vez que trabajes con el repositorio:
2. ssh-add ~/.ssh/id\_ed25519

La línea de comando devolverá el siguiente texto:

Identity added: /c/Users/YOUR\_USERNAME/.ssh/id\_ed25519 (YOUR\_EMAIL@DOMAIN.COM)

## Añadir la clave SSH a una cuenta de GitHub

1. Copia la clave pública en el portapapeles:

macOS

*# Copia el contenido de la clave en el portapapeles*

pbcopy < ~/.ssh/id\_ed25519.pub

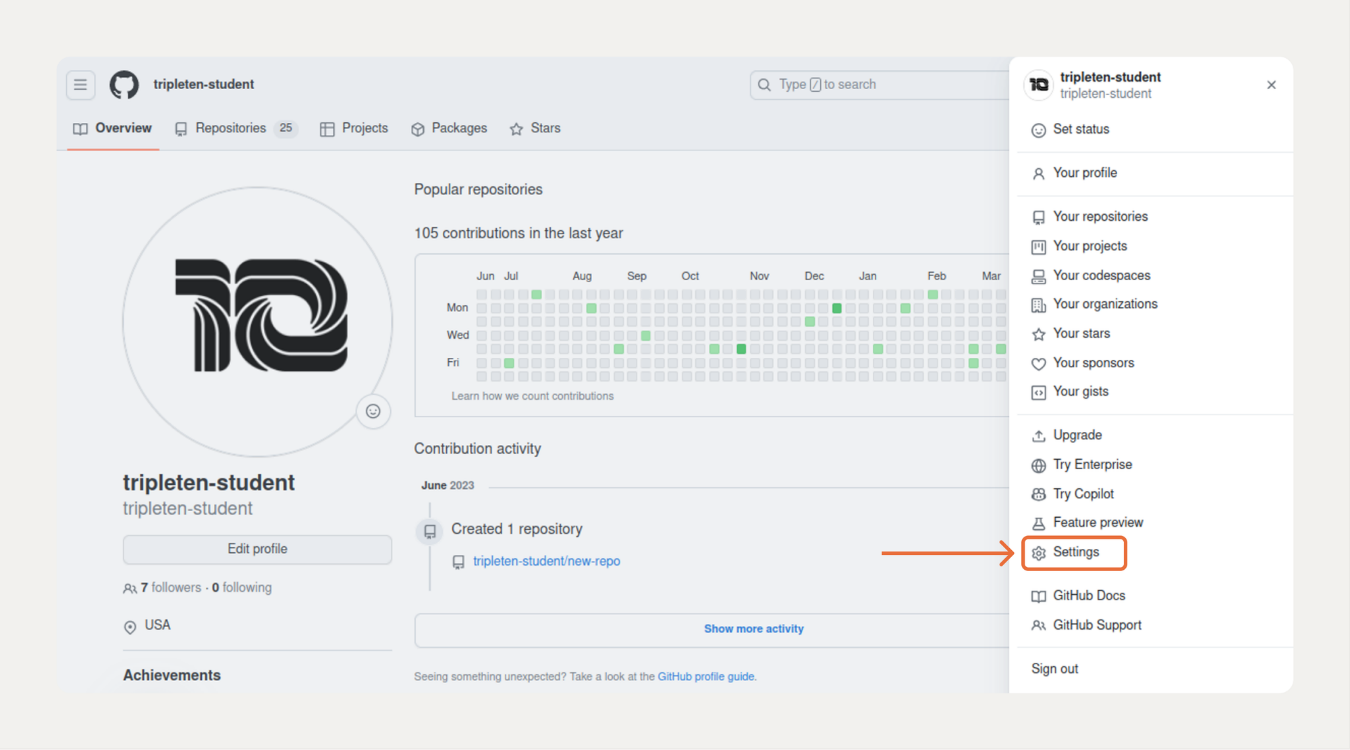
Windows

*# Copia el contenido de la clave en el portapapeles*

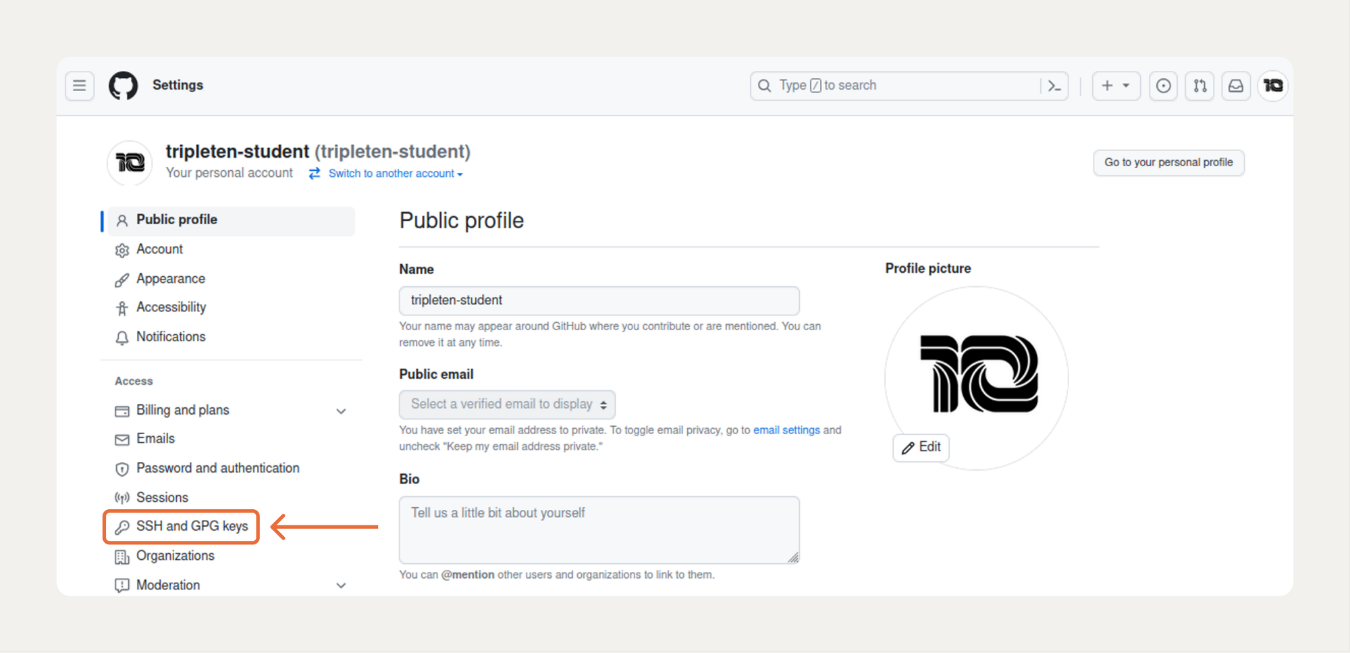
clip < ~/.ssh/id\_ed25519.pub

Si el portapapeles no funciona, busca la carpeta oculta .ssh y abre el archivo id\_ed25519.pub en cualquier editor de texto. Copia el contenido de este archivo en el portapapeles.

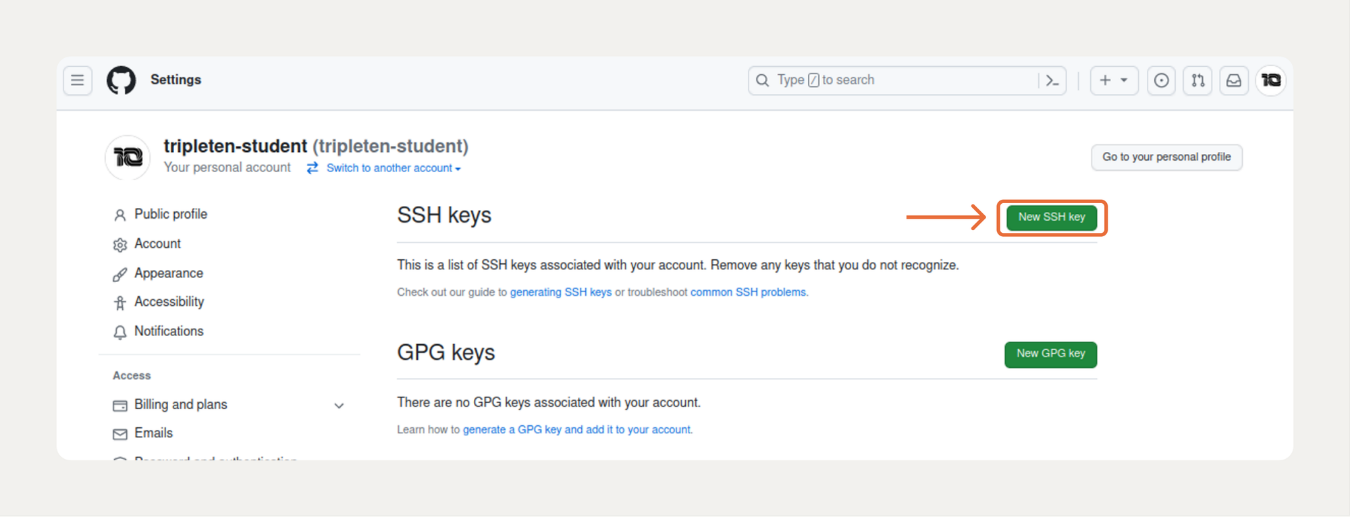
1. Accede a GitHub y abre la configuración de tu cuenta:

Configuración de la cuenta en GitHub

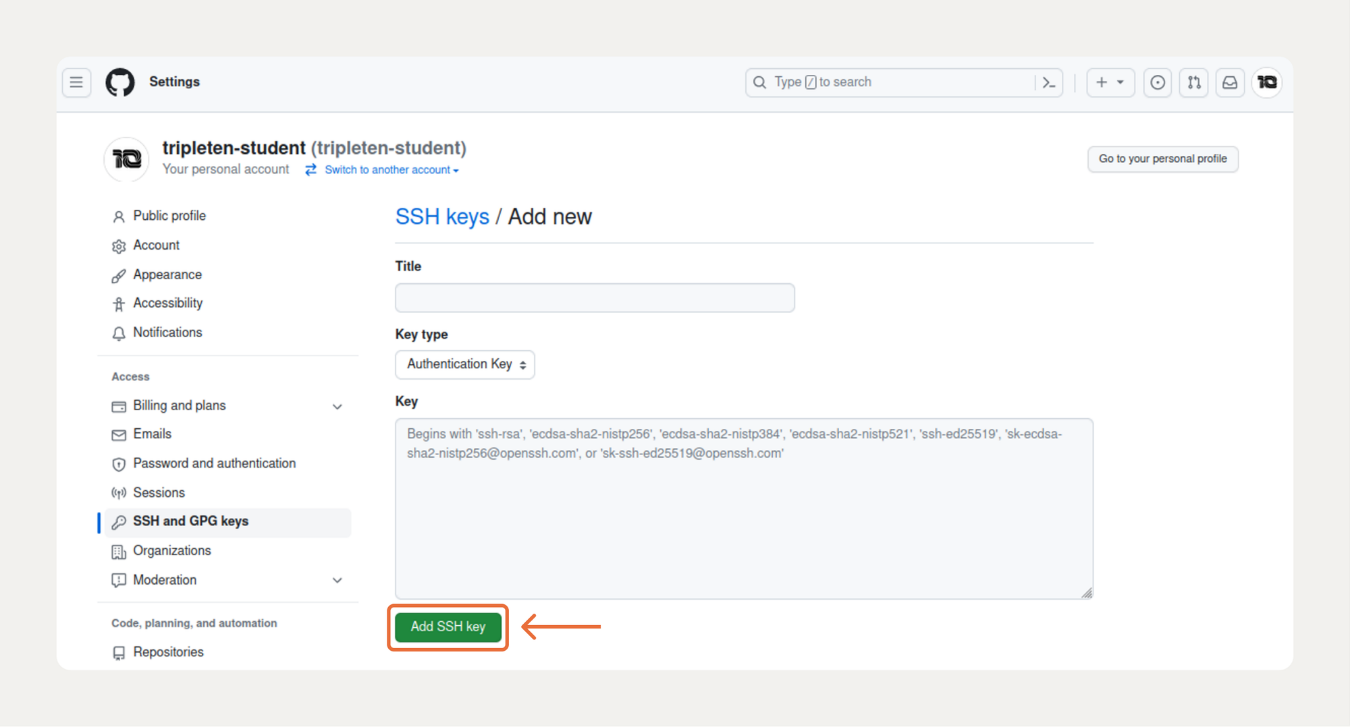
1. Selecciona "Claves SSH y GPG" en el menú de la izquierda:

Seleccionando la pestaña "SSH and GPG keys".

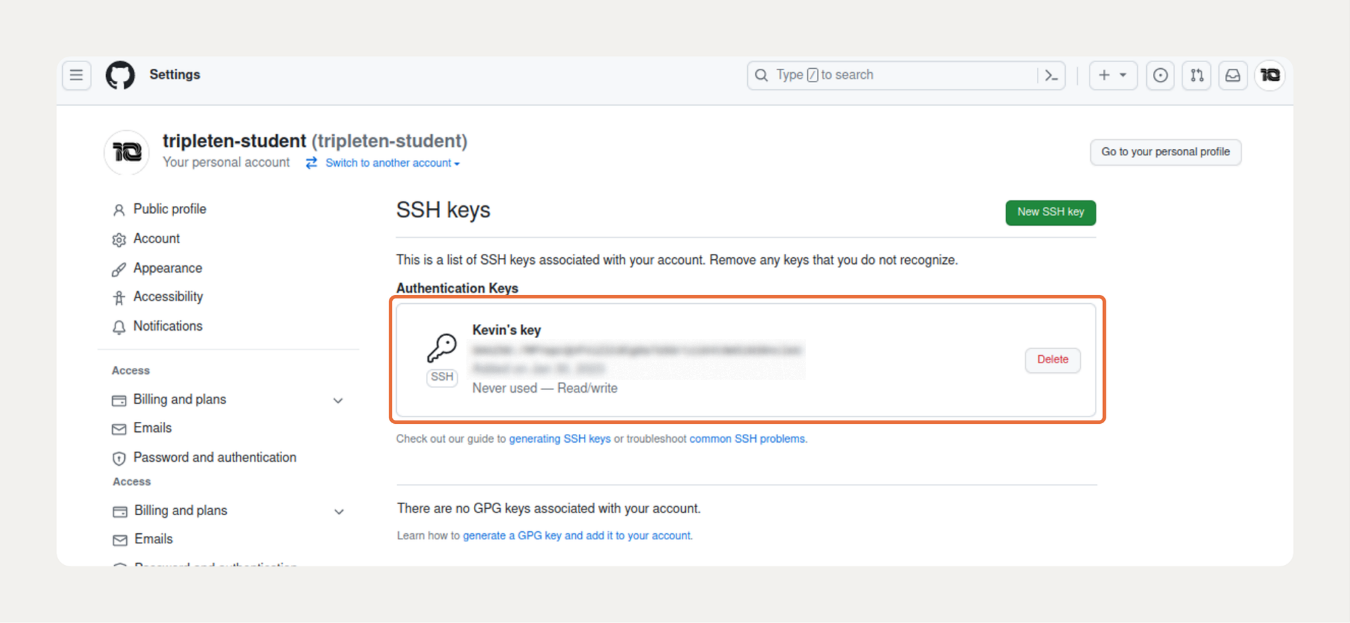
1. En la pestaña abierta, selecciona "New SSH key" (Nueva clave SSH) o "Add SSH key" (Agregar clave SSH):

Seleccionando "New SSH key" o "Add SSH key”

1. En el campo "Title", escribe un título, por ejemplo, "Clave personal".
2. En el campo "Clave", pega tu clave desde el portapapeles y haz clic en el botón "Añadir clave SSH":

Una copia de la clave SSH del portapapeles

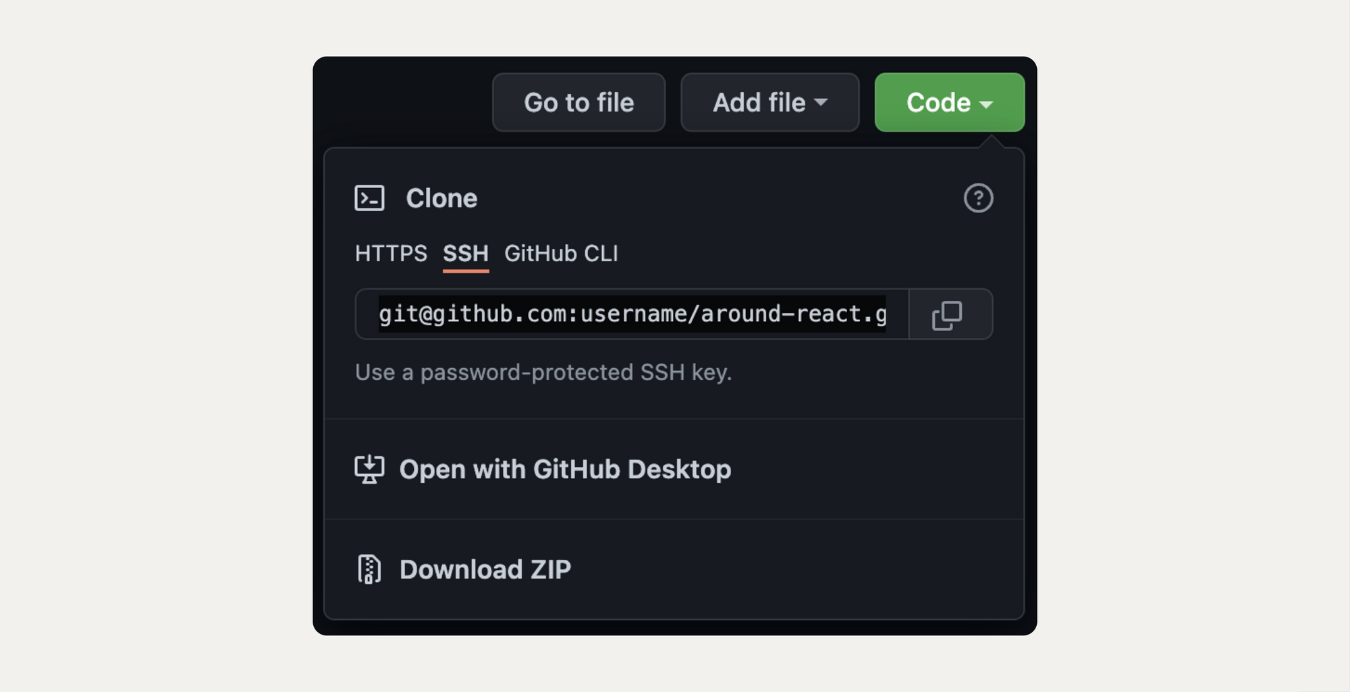
1. Ahora tu clave está vinculada a GitHub. Si has decidido establecer una contraseña para tu clave SSH, tendrás que introducirla para poder trabajar con el repositorio. *Se ha agregado tu clave SSH personal*

Tu clave SSH personal ha sido agregada

1. Comprobemos que hayas configurado todo correctamente. Pega ssh -T git@github.com en la terminal. La terminal puede pedirte que apruebes la autenticidad de la conexión. Finalmente, debe mostrar el siguiente mensaje:

Hi username! You've successfully authenticated, but GitHub does not provide shell access.

Ahora puedes conectarte a tus repositorios o clonar repositorios desde otras cuentas utilizando la URL de SSH desde GitHub y enviar nuevas confirmaciones sin introducir el token de acceso personal cada vez. En las siguientes lecciones aprenderás cómo hacerlo.



Pregunta

¿Qué comando deberíamos utilizar para comprobar qué claves SSH ya han sido generadas?

Ejecuta ssh-add ~/.ssh/id\_ed25519 en la consola.

Ejecuta ls -al ~/.ssh en la consola.

¡Correcto! Este comando mostrará el contenido de la carpeta .ssh, si es que existe.

¡Perfecto!

# Vincular los repositorios locales y remotos

El siguiente paso es vincular el repositorio remoto que creaste en GitHub al repositorio local que creaste unas lecciones atrás. Así que regresa a tu terminal y haz cd en ese repositorio porque estamos a punto de comenzar.

💡 Antes de continuar, asegúrate de estar en el repositorio local que creaste en las lecciones anteriores. Si seguiste las instrucciones en nuestra lección sobre la línea de comandos, puedes volver allí con cd ~/projects/sample-project.

## Vincular los repositorios locales y remotos

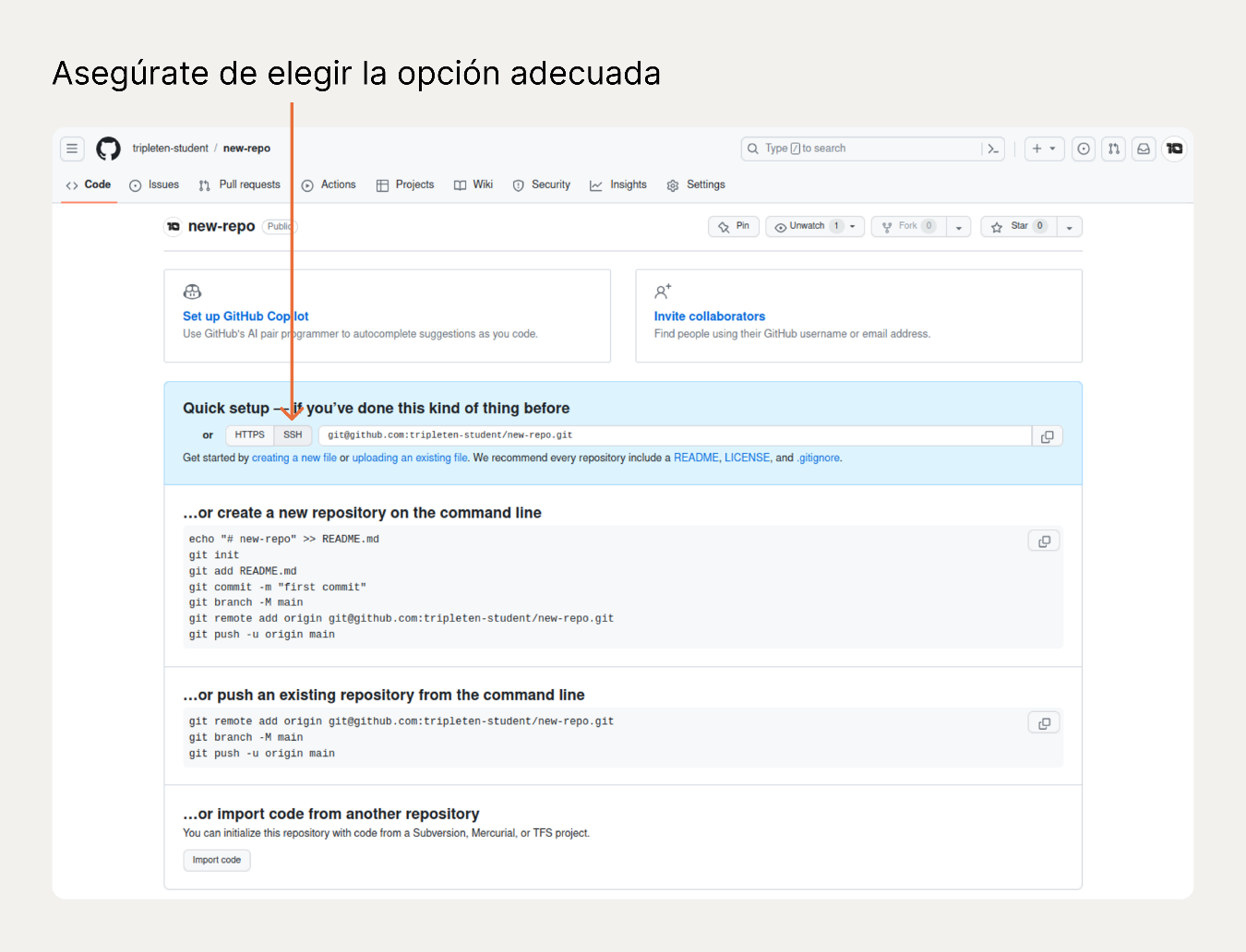
Para vincular el repositorio remoto con el local, utilizaremos el comando git remote add. Este comando toma dos argumentos: el nombre del repositorio remoto y su URL.

git remote add NOMBRE\_REMOTO URL\_REMOTA

En realidad, el argumento NOMBRE\_REMOTO no corresponde al nombre del repositorio en cuestión, es un alias que usarás para referirte al repositorio remoto desde la línea de comandos. Por convención, el alias utilizado es origin, y lo usarás la mayor parte del tiempo:

git remote add origin URL\_REMOTA

El argumento URL\_REMOTA variará dependiendo de si estás utilizando SSH o HTTPS para la autenticación, así que asegúrate de elegir la opción correcta, como se muestra a continuación.



Si estás en SSH, utiliza el siguiente comando, asegurándote de actualizarlo para emplear tu propio nombre de usuario y repositorio.

git remote add origin git@github.com:TU\_NOMBREDEUSUARIO/new-repo.git

Si estás utilizando HTTPS, el comando se verá así:

git remote add origin https://github.com/TU\_NOMBREDEUSUARIO/new-repo.git

Después de ejecutar el comando anterior, los repositorios local y remoto estarán vinculados. Ahora, debemos subir el código local a GitHub. Podemos hacerlo utilizando el comando git push:

git push -u origin main

La opción -u es necesaria la primera vez que empujas una nueva rama que no existe en un repositorio remoto, ya que esta no solo crea una relación entre las ramas local y remota, sino que también establece la rama remota como la rama ‘upstream’ predeterminada para la rama local actual. Este ajuste facilita futuros 'pushes' y 'pulls', simplificando la sincronización entre el repositorio local y el remoto.

Esto es necesario porque Git nos permite tener un nombre distinto para las versiones de la rama local y remota. También nos permite empujar el contenido de una rama a una rama remota con un nombre diferente, aunque no te recomendamos que hagas esto.

Arriba, también definimos explícitamente el nombre remoto (origin) y el nombre de la rama (main). Esto fue para dejar muy claro hacia dónde estamos empujando.

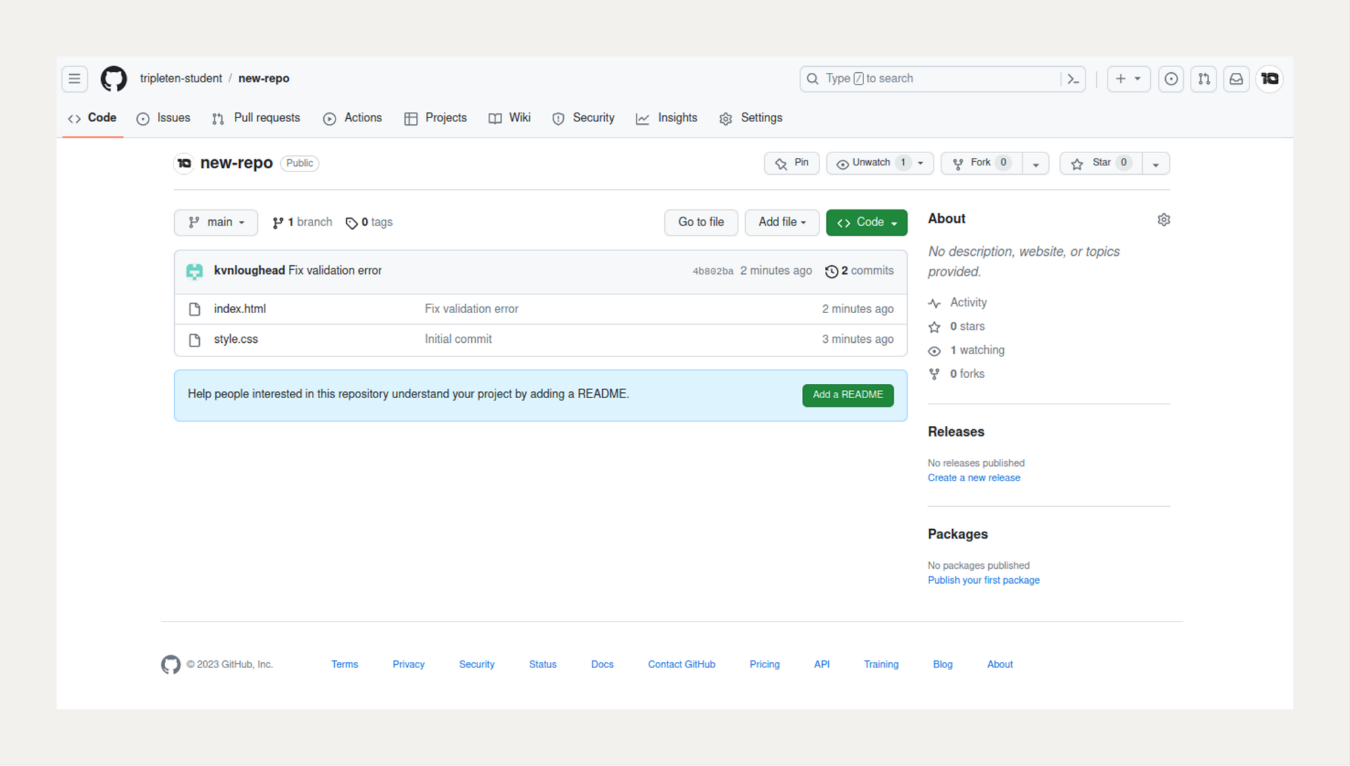
* Si utilizas el método HTTPS y configuras el token de acceso personal de la lección anterior, tras ejecutar el comando git push el símbolo del sistema te pedirá tu nombre de usuario de GitHub. Introdúcelo y presiona Enter. Después tendrás que introducir tu token de acceso personal y presionar nuevamente Enter. Si introduces toda la información correctamente, los commits se subirán a GitHub.

A veces, los usuarios de Windows deben introducir su nombre de usuario y contraseña en una ventana emergente en lugar de en la línea de comandos.

Un pequeño consejo: por motivos de seguridad, al introducir la contraseña los caracteres no aparecen en pantalla. No te preocupes, la línea de comandos seguirá recibiendo tu contraseña cuando la escribas.

💡 Recuerda que no todas las plataformas colaborativas que usan Git cambiaron el nombre de la rama master como lo hizo GitHub. En Gitlab o Bitbucket, así como en los manuales de Git, es posible que aún te encuentres con el nombre de master. En este caso, git push -u origin main dará lugar a un error. Para solucionarlo, sustituye main por master.

En la siguiente lección revisaremos el comando git push con más detalle. Por ahora, dirígete al repositorio remoto en GitHub y actualiza la página.



El código ya se ha subido (o empujado, como dicen los desarrolladores). Ahora, otros desarrolladores y desarrolladoras pueden conectarse a este repositorio, y pronto descubrirás cómo conectarte tú también a cualquier repositorio. Es muy buena idea que te familiarices con estos conceptos estudiando la interfaz de GitHub y el repositorio new-repo por tu cuenta.

### Encontrar un repositorio remoto

Supongamos que tienes un repositorio local, pero debes recordar si ya estableciste uno remoto para él o su valor. Puedes conocer esta información con el comando git remote -v. Aquí tienes un ejemplo de su uso y de su salida.

git remote -v

*# salida*

origin git@github.com:user/some-repo.git (fetch)

origin git@github.com:user/some-repo.git (push)

Si no se ha establecido un repositorio remoto, la salida estará vacía.

### Eliminar o cambiar un repositorio remoto

Si alguna vez necesitas cambiar el repositorio remoto de un repositorio local, no podrás usar git remote add. Tal como se muestra a continuación, aparecerá un error diciéndote que el repositorio remoto ya existe.

*# agrega un repositorio remoto en el repositorio local que aún no tenga establecido uno remoto*

git remote add origin git@github.com:NOMBREDEUSUARIO/REPO.git

*# intenta agregar otro repositorio remoto llamado origin*

git remote add origin git@github.com:NOMBREDEUSUARIO/OTRO\_REPO.git

*# salida*

error: remote origin already exists.

En cambio, podemos emplear el comando git remote set-url.

*# agrega un repositorio remoto en el repositorio local que aún no tenga establecido uno remoto*

git remote add origin git@github.com:NOMBREDEUSUARIO/REPO.git

*# cambia el repositorio remoto*

git remote set-url origin git@github.com:NOMBREDEUSUARIO/OTRO\_REPO.git

*# el repositorio ha sido cambiado*

git remote -v

*# salida*

origin git@github.com:NOMBREDEUSUARIO/OTRO\_REPO.git (fetch)

origin git@github.com:NOMBREDEUSUARIO/OTRO\_REPO.git (push)

Como puedes ver, el repositorio remoto se actualizó correctamente.

Si, por el otro lado, sólo quieres eliminar por completo el repositorio remoto, puedes usar el comando git remote remove.

*# comprueba qué repositorios remotos se han establecido*

git remote -v

*# salida*

origin git@github.com:NOMBREDEUSUARIO/OTRO\_REPO.git (fetch)

origin git@github.com:NOMBREDEUSUARIO/OTRO\_REPO.git (push)

*# elimina el repositorio remoto llamado "origin"*

git remote remove origin

*# comprueba si está ahí el repositorio remoto*

git remote -v

Que git remote -v no genere ninguna salida muestra que el repositorio remoto ha sido eliminado.

# Sincronizar repositorios locales y remotos

Te has registrado, has creado un repositorio local y lo has subido a GitHub. Lo más difícil ya está hecho. Buen trabajo. A continuación, vamos a averiguar cómo obtener los cambios del repositorio remoto a nuestro equipo local y cómo transferir a GitHub los cambios que hacemos. Hablemos de esto ahora mismo.

💡 Antes de continuar, asegúrate de estar en el repositorio local que creaste en las lecciones anteriores. Si seguiste las instrucciones en nuestra lección sobre la línea de comandos, puedes volver allí con cd ~/projects/sample-project.

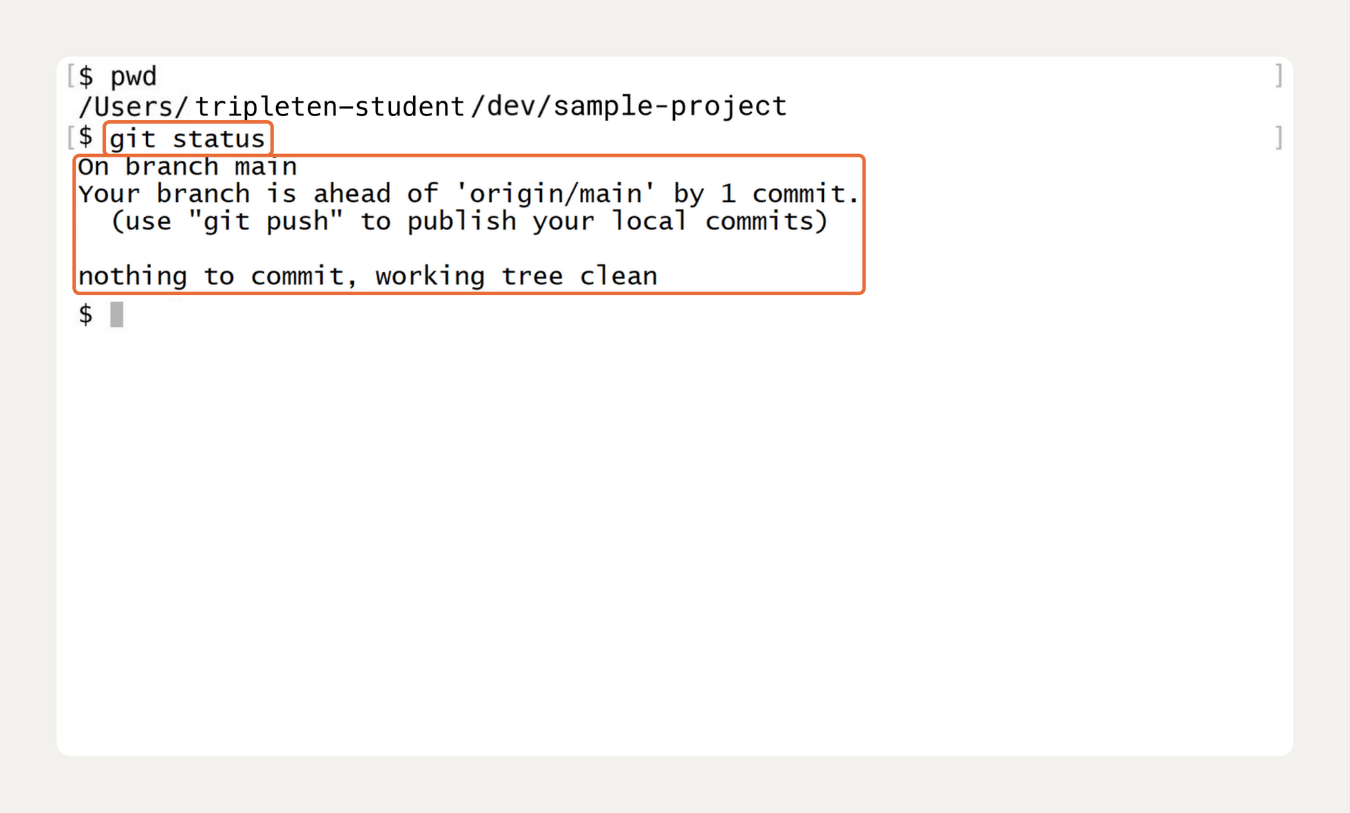
## git push

Ya has pasado por el ciclo del commit. Añadiste un archivo usando el comando git add y lo confirmaste con un comentario usando git commit. Luego, vinculaste tu repositorio local a tu repositorio de GitHub usando git remote add. Ahora podemos usar el comando git push para subir, o empujar, los cambios que hagas a tu repositorio remoto. Esto permitirá a tus compañeros de equipo ver e incorporar tu trabajo al suyo.

git push -u origin main

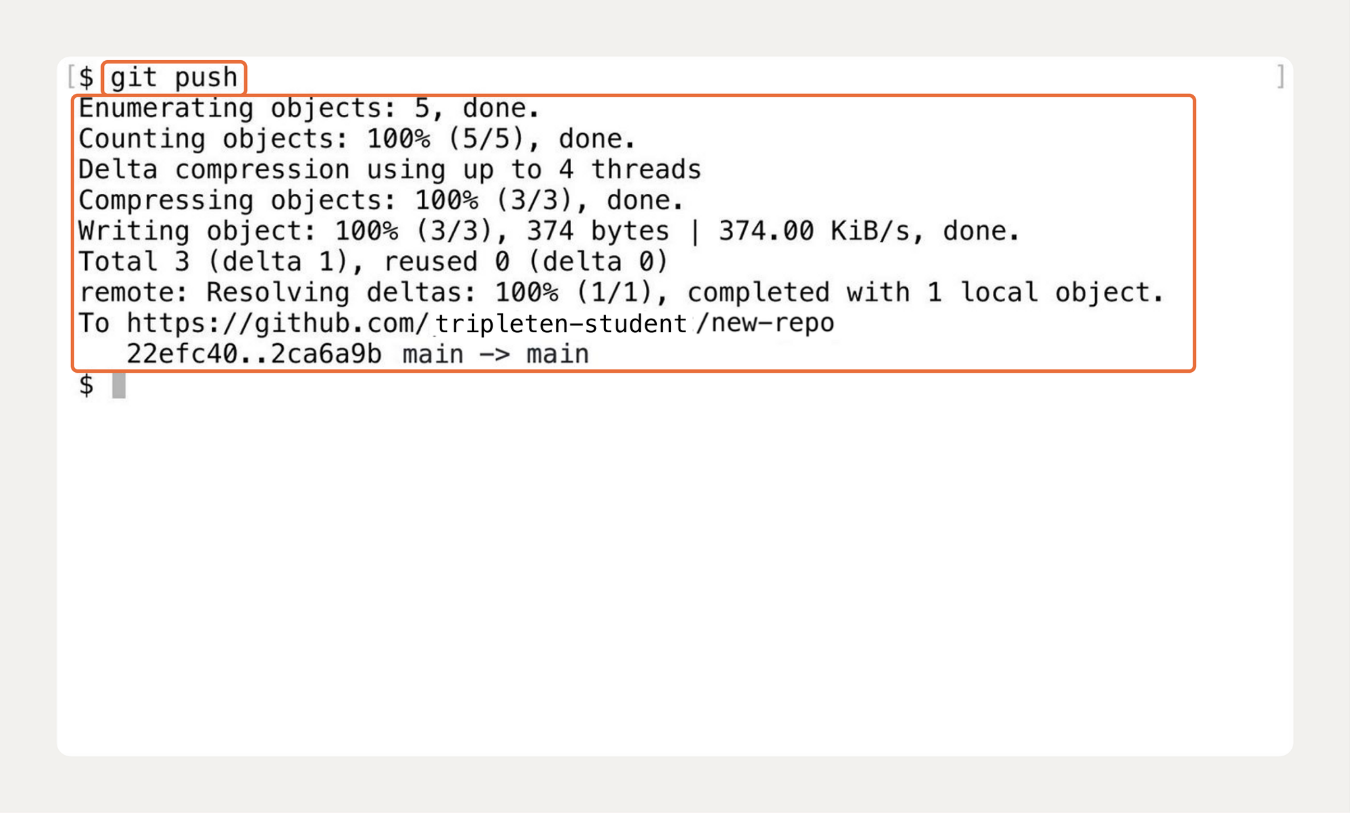
Cuando necesites subir cambios a GitHub, ejecuta el comando como se muestra arriba. Ahora vamos a practicar el uso de git push.

Abre el archivo index.html y añade algunas marcas HTML sencillas. Luego, usando git add y git commit, haz que el archivo recién actualizado esté listo para ser sincronizado con el repositorio remoto. Si todo ha ido bien, después de ejecutar el comando git status, deberías ver algo así en la línea de comandos:

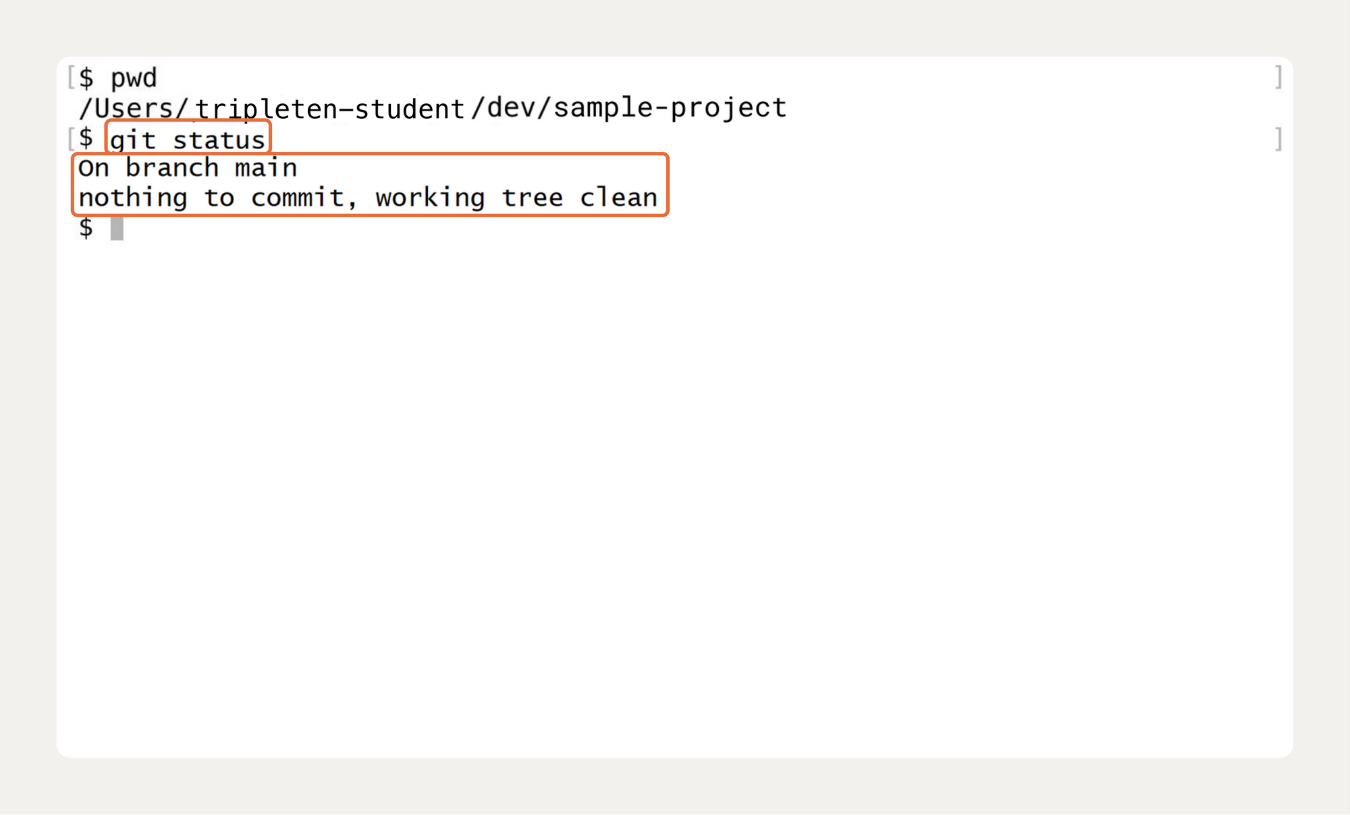


Ahora es el momento de subir los cambios a tu repositorio remoto. Introduce el comando git push.

Este commit empujará tus cambios al repositorio remoto, y si todo va bien, verás esto:



Ahora, si vuelves a ejecutar git status, deberías ver algo como esto:



Finalmente, con tu navegador web, ve al repositorio new-repo en tu cuenta de GitHub. Verás los archivos modificados en la interfaz de GitHub.

## git pull

Hasta ahora has trabajado por tu cuenta, pero en un proyecto real esto no es lo más probable. ¿Cómo podemos asegurarnos de colaborar de forma eficaz con nuestros colegas desarrolladores? Imaginemos que el viernes escribiste un montón de código, pero que no los empujaste. Durante el fin de semana, mientras descansabas, una colega siguió trabajando, pero se aseguró de utilizar git push para mantener el repositorio de GitHub sincronizado con su repositorio local. El lunes, el código del proyecto que está en GitHub tiene un aspecto bastante diferente al que tenía cuando lo dejaste. ¿Qué deberías hacer? Tienes que utilizar el comando git pull para extraer los cambios realizados por tu colega.

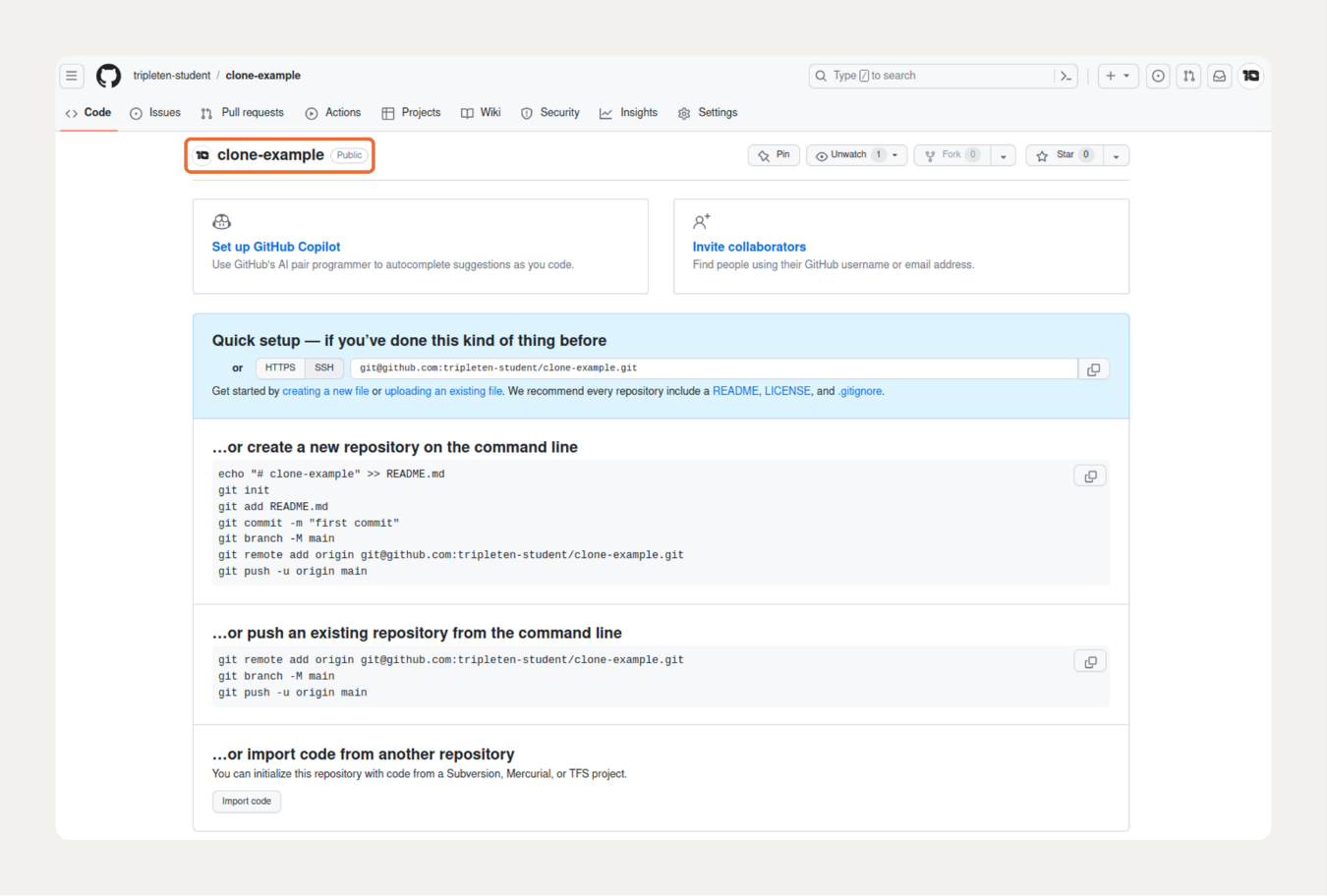
Siempre que trabajes en un repositorio con otras personas, ejecuta git pull antes de publicar cambios. Es posible que otra persona haya publicado cambios mientras escribías tu parte del código.

# Clonar un repositorio

En lecciones anteriores, creaste un repositorio local y lo vinculaste a uno remoto. Este es un flujo de trabajo común, especialmente cuando ya tienes un proyecto que quieres empujar a GitHub. Pero a veces, necesitarás copiar un repositorio ya hecho en tu computadora. Este proceso se denomina clonación.

## git clone

Clonar un repositorio suele ser lo primero que hace un desarrollador en un nuevo trabajo. Veamos una situación típica. Crea un segundo repositorio en tu cuenta y llámalo clone-example.



Verás la pantalla del nuevo repositorio. Primero, copia su dirección mediante el método de autenticación de tu preferencia (ya sea HTTPS o SSH). Después, abre la línea de comandos y ve a la carpeta donde guardaste el repositorio: el directorio del proyecto de las lecciones anteriores. Puedes llegar allí con:

cd ~/projects

Ahora, vas a introducir un comando que nunca habías visto, git clone, seguido de la dirección del repositorio (por favor, recuerda utilizar tu propia URL):

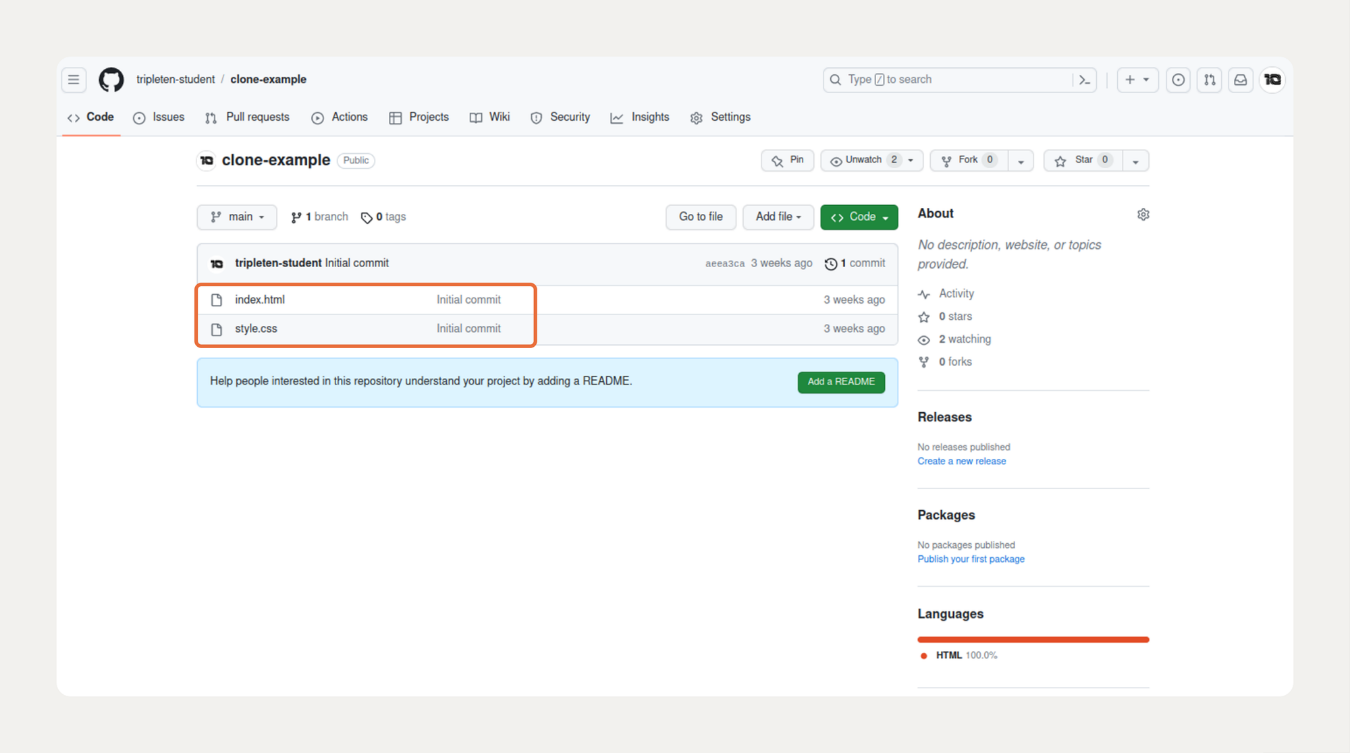
*# si estás utilizando HTTPS*

git clone https://github.com/TU\_NOMBREDEUSUARIO/clone-example.git

*# si estás utilizando SSH*

git clone git@github.com:TU\_NOMBREDEUSUARIO/clone-example.git

En cuanto hayas terminado la clonación, entra en el repositorio con el comando cd clone-example, crea los archivos index.html y style.css mediante el comando touch, y haz un commit. Después de hacer un commit, envía los cambios al repositorio remoto usando el comando git push -u origin main. Abre tu cuenta de GitHub y selecciona el repositorio clone-example; verás el commit publicado:



Ya dispones de dos formas de crear un repositorio. Pero es importante que tengas en cuenta que, aunque puedes clonar cualquier repositorio público, solo puedes enviar código a los repositorios que hayas creado tú o a los que te han dado permiso para enviar.

Pregunta

¿Solo puedo clonar mi propio repositorio?

Puedes clonar cualquier repositorio o repositorio público.

¡Así es! Si estás trabajando en un nuevo proyecto, clónalo primero. Eso es lo que hacen todos los desarrolladores.

Solo puedes clonar lo que tú creas.

¡Excelente trabajo!

Pregunta

¿Cómo se clona un repositorio de terceros?

Lo buscas en GitHub, haces clic en el icono verde del código, vas a clonar y copias el enlace. Dirígete a la línea de comandos, introduce el comando git clone y pega el enlace URL copiado.

¡Así es!

Lo mismo que en la primera opción, pero además hay que ir a la carpeta con el repositorio descargado e introducir el comando git remote add originjunto con la ruta al repositorio.

¡Lo has entendido bien!

Has aprendido los comandos básicos de terminal/bash y cómo trabajar con Git. También aprendiste a hacer commits, empujar y clonar. Necesitarás todas estas habilidades para enviar tu proyecto y terminar este sprint. Pero, más allá de la importancia de este sprint y su proyecto, estas herramientas son indispensables para los equipos de desarrollo reales.

**El flujo de trabajo de Git/GitHub**

Acabamos de ver todos los pasos indispensables del flujo de trabajo de Git y GitHub: desde inicializar un repositorio Git hasta empujar los cambios a GitHub. Así que ahora resumiremos todo el flujo: Al final de esta lección, entenderás con claridad cómo usar Git y GitHub para gestionar tus proyectos eficazmente.

Imagina que tenemos un proyecto, una recopilación de archivos en un directorio en una computadora, y queremos agregarle el control de versiones para convertirlo en un repositorio Git. Sólo tenemos que ejecutar el comando git init dentro del directorio del proyecto. Git entonces comenzará a rastrear todos los cambios hechos a los archivos del proyecto. Estos cambios se rastrean en lo que Git llama el "árbol de trabajo", que alude al directorio del proyecto habilitado por Git.

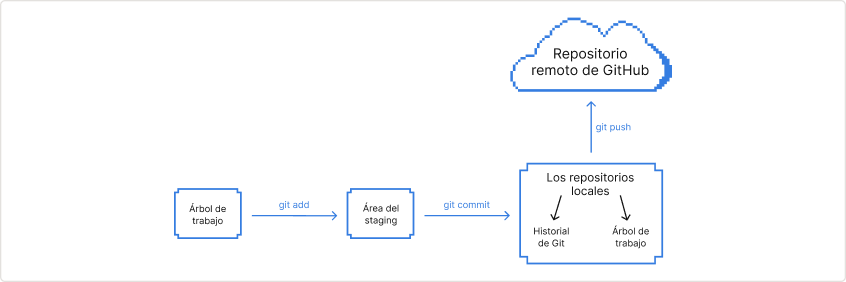
La idea central es que Git rastree los cambios a través de la historia del proyecto. Guarda el estado de un repositorio en diferentes momentos. A estos "puntos de guardado" se les llama "commits", y los creamos utilizando el comando git commit. Pero antes de poder hacer un commit, debemos decirle a Git qué cambios debe incluir en el commit. Esto se hace agregando los archivos al "área de staging" ("área de preparación") con el comando git add.

Todo lo que hemos visto hasta ahora, sucede localmente, y no tiene ninguna conexión con el repositorio remoto alojado en GitHub. Podemos resumirlo mediante los siguientes cuatro pasos:

1. Los cambios se hacen a un archivo o varios archivos.
2. Los archivos se agregan al área de staging con git add.
3. Los cambios se confirman con git commit.
4. Repite los pasos 1-3 siempre que sea necesario.

Interactuar con GitHub agrega una tarea más a nuestro flujo de trabajo básico. Cuando hayamos hecho uno o más commits, debemos "empujarlos" a nuestro repositorio remoto con el comando git push:

1. Los cambios se hacen a un archivo o varios archivos.
2. Los archivos se agregan al área de staging con git add.
3. Los cambios se confirman con git commit.
4. Repite los pasos 1-3 siempre que sea necesario.
5. Los commits se empujan a GitHub con git push.
6. Repite los pasos 1-5 siempre que sea necesario.



Ya está todo listo para que implementes este flujo en tu proyecto.