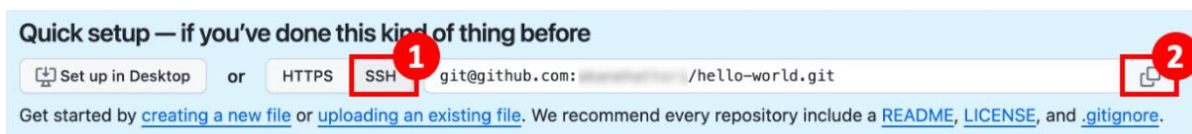


## Empezando con GitHub

GitHub es una plataforma esencial en el mundo de DevOps y CI/CD (Integración Continua y Entrega Continua). Permite a los equipos de desarrollo gestionar, compartir y colaborar en el código fuente de manera eficiente. En el contexto de CI/CD, GitHub se integra con herramientas como GitHub Actions, Jenkins, y otros sistemas de automatización para desencadenar procesos de compilación, pruebas, y despliegue cada vez que se realizan cambios en el repositorio. Esta integración garantiza que el software se desarrolle, pruebe y entregue de manera continua y confiable, permitiendo a los equipos liberar nuevas características y correcciones de forma rápida y segura.

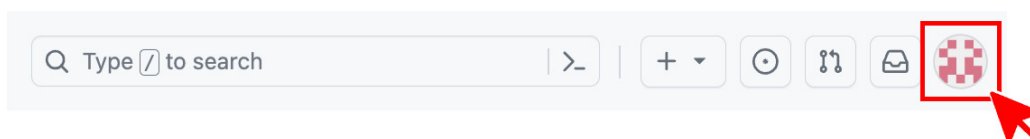
En el ámbito de DevOps, GitHub actúa como el punto central donde los desarrolladores, operadores, y otros interesados pueden colaborar en todo el ciclo de vida del software, desde el desarrollo hasta el despliegue. Con GitHub, se pueden definir flujos de trabajo automatizados, realizar revisiones de código, y mantener una versión controlada del software, todo bajo un entorno que favorece la colaboración y la eficiencia.

Después de haber creado tu repositorio para el curso, el siguiente paso es conectarlo a tu entorno de desarrollo local. A continuación, conectaremos tu repositorio local a un repositorio remoto, lo que incluye generar y registrar una clave SSH para GitHub. Los detalles de la configuración de conexiones SSH serán cruciales en las secciones siguientes, así que asegúrate de tomar nota de la cadena de URL SSH. Haz clic en el botón SSH y luego en el ícono de copia para copiar los valores, como se muestra en la siguiente captura de pantalla.

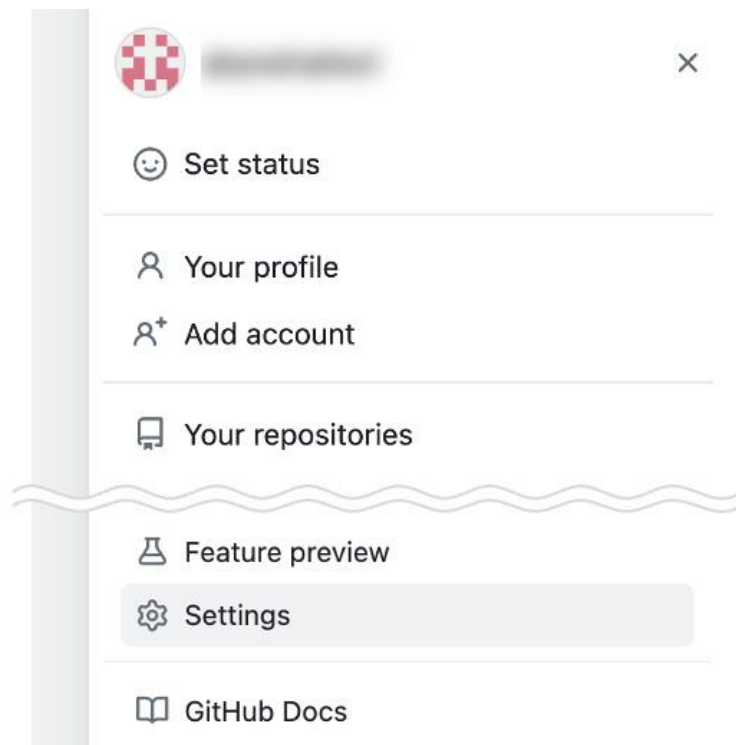


## Registrando tu clave SSH

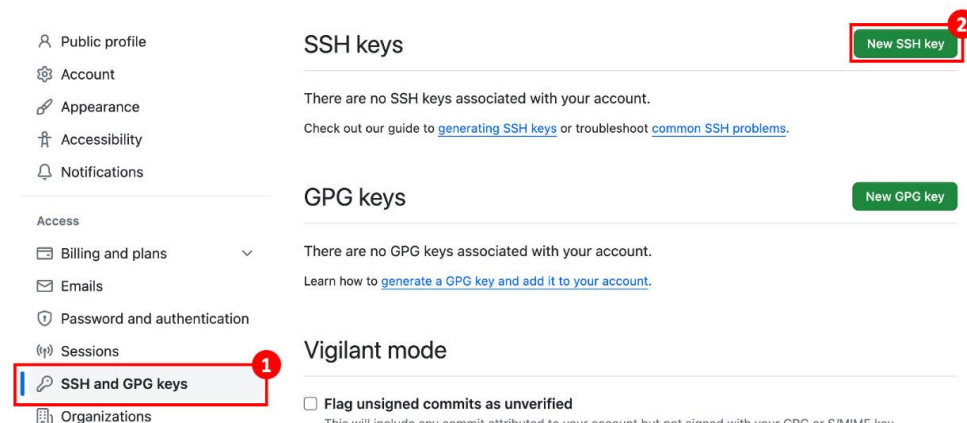
Ahora es el momento de configurar tus claves SSH. Para hacer esto, navega a la configuración haciendo clic en el botón del menú superior derecho en GitHub:



Desde esta barra de menú, tienes la capacidad de navegar a varias secciones de tu cuenta de GitHub. Puedes ver tu perfil y gestionar los repositorios y organizaciones de las que formas parte. Ahora, selecciona "Settings" en la barra de menú:



Una vez que estés en la sección de Settings, busca "SSH and GPG keys" en el menú de la izquierda. Aquí tendrás la opción de crear una nueva clave SSH:



Después de hacer clic en el botón "New SSH key", se te pedirá que ingreses un título y selecciones el tipo de clave. En esta etapa, debes elegir "Authentication Key". Luego, puedes ingresar la clave SSH en el campo proporcionado.

¡Ahora, necesitarás tu clave SSH para registrarte! Veamos cómo hacer una. Puedes omitir este proceso si ya has registrado una.

Crear tu clave SSH es un paso crucial para configurar un entorno seguro para gestionar tu código en GitHub. Este proceso es especialmente importante si eres nuevo en las claves SSH o no tienes una. Puedes verificar si tienes una clave existente navegando al directorio `~/.ssh` en tu terminal. Este directorio típicamente contiene tus archivos de configuración SSH y claves.

Si no encuentras claves existentes o deseas crear una nueva específicamente para GitHub, hagámoslo.

Primero, abre tu terminal y navega al directorio SSH:

```
$ cd ~/.ssh
```

Luego, genera una nueva clave SSH con el comando `ssh-keygen`. Utilizarás una clave RSA para este propósito:

```
$ ssh-keygen -t rsa
```

Este comando inicia el proceso de generación de claves y muestra el siguiente mensaje:

**Generating public/private rsa key pair.**

Cuando se te solicite, ingresa el nombre del archivo en el que guardar la clave. Aquí hay un ejemplo:

Enter file in which to save the key

**(/Users/username/.ssh/id\_rsa): [Your Key Name]**

En este escenario, usaré `git_key` como el nombre de la clave.

A continuación, se te pedirá que ingreses una frase de contraseña. Esto agrega una capa adicional de seguridad:

Enter passphrase (empty for no passphrase): [Your Passphrase]

Enter same passphrase again: [Repeat Your Passphrase]

Para mayor claridad, agregar una frase de contraseña a tu clave SSH es un paso clave de seguridad. Si alguien obtiene tu clave privada sin permiso, aún no podrá usarla sin la frase de contraseña. Esto protege contra el uso no autorizado. Necesitarás ingresar la frase de contraseña cada vez que uses la clave, asegurando que solo las personas con la clave y la frase de contraseña puedan acceder. Esta capa adicional de seguridad hace que tus conexiones SSH sean mucho más seguras y se recomienda para proteger información y acceso importantes.

Después de estos pasos, verás una confirmación de que tu identificación (clave privada) y tu clave pública se han guardado. También se mostrará una huella digital única de la clave y una imagen de arte aleatoria.

Ahora, necesitas verificar y copiar tu nueva clave SSH pública:

```
$ cat ~/.ssh/git_rsa.pub
```

Copia la clave SSH que se muestra (comienza con `ssh-rsa`):

```
ssh-rsa AAAAB3NzaC1yc2EAAA...x4CWuT2U= a1b2@c3d4.e5f6
```

Ahora, agreguemos esta clave SSH a tu cuenta de GitHub. Nuevamente, navega a la configuración de GitHub, encuentra la sección "SSH and GPG keys", y pega y guarda tu clave allí:

## Add new SSH Key

Title

your-ssh-key

Key type

Authentication Key

Key

ssh-rsa

Add SSH key

Para configurar o modificar tus conexiones SSH, comencemos creando un archivo de configuración utilizando el comando `touch`, que crea un nuevo archivo o actualiza la hora de modificación de uno existente:

```
$ touch config
```

Ahora, con el archivo de configuración listo, es momento de agregar configuraciones específicas para las conexiones SSH de GitHub. Asegúrate de reemplazar `git_rsa` con el nombre de tu archivo de clave SSH privada. Para aquellos que siguieron los pasos anteriores para crear una nueva clave, tu nombre de archivo debería ser `git_rsa`.

Si eres un usuario experimentado de Git, es posible que ya tengas algunas configuraciones SSH en este archivo. En tal caso, deberías actualizar o reemplazar las configuraciones existentes. Para aquellos que estén agregando nuevas configuraciones, ingresen estas líneas:

```
Host github github.com
```

```
HostName github.com
```

```
IdentityFile ~/.ssh/git_rsa
```

```
User git
```

Después de ingresar estas líneas, guarda y cierra el archivo. Esta configuración le indica a tu sistema qué clave SSH usar para GitHub y bajo qué cuenta de usuario.

También se recomienda verificar tu conexión SSH a GitHub para asegurarte de que todo esté configurado correctamente. Puedes hacerlo ejecutando el comando `ssh -T git@github.com`. Este paso ayuda a confirmar que tu sistema puede comunicarse con éxito con GitHub utilizando la clave SSH especificada en tu configuración.

**git remote: Conectando repositorios locales y remotos**

Es hora de conectar tu desarrollo local con el mundo de GitHub. Si estás continuando desde un repositorio existente en una sección anterior, simplemente muévete a ese directorio con el comando `cd`, como sigue:

```
$ cd path/to/your/repository
```

Para aquellos que quieran crear un nuevo proyecto, configurar un nuevo repositorio es un proceso sencillo. Comienza creando un nuevo directorio, inicializando un repositorio Git y preparando un archivo README, el sello distintivo de cualquier nuevo proyecto:

```
$ mkdir new-project
```

```
$ cd new-project
```

```
$ git init
```

```
$ echo "# README" >> README.md
```

```
$ git add README.md
```

```
$ git commit -m "Commit inicial"
```

Recuerda usar la URL SSH que anotaste en la sección anterior.

Conectar tu repositorio local a un repositorio remoto en GitHub implica agregar una URL remota. Este enlace te permite subir tus cambios locales a GitHub. Establece esta conexión con el comando `git remote add`, asegurándote de reemplazar `[Username]` y `[Repository]` con tu nombre de usuario de GitHub y el nombre del repositorio:

```
$ git remote add origin git@github.com:[Username]/[Repository].git
```

Para casos donde tu repositorio ya tiene una URL remota pero necesita una actualización, el comando `git remote set-url` es la opción preferida. Este comando actualiza tu configuración de Git a la nueva URL del repositorio remoto:

```
$ git remote set-url origin git@github.com:[Username]/[Repository].git
```

Con estos pasos, has vinculado con éxito tus repositorios local y remoto. Esta conexión es un punto crucial en la gestión de tu proyecto, asegurando que tus desarrollos locales se reflejen en GitHub para un mayor progreso y colaboración.

A medida que avanzamos, el siguiente paso será subir tu código local a GitHub.

### **git push: Haciendo que tu código cuente**

Finalmente, es hora de subir tus commits locales al repositorio remoto. Este paso actualiza el repositorio remoto con los cambios que has hecho localmente:

```
$ git push -u origin main
```

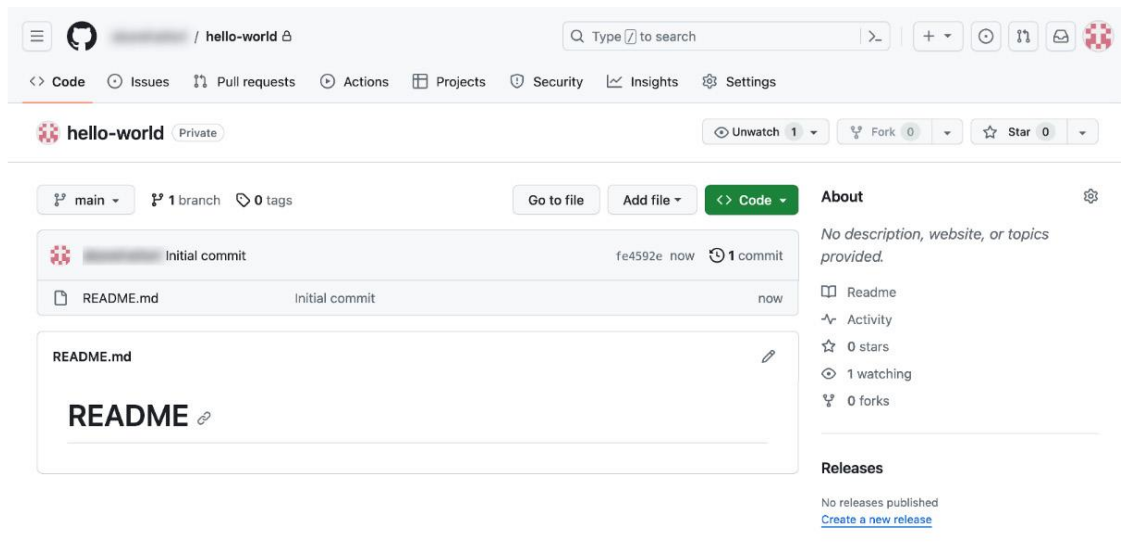
Cuando usas `-u` o `--set-upstream` con `git push`, estás efectivamente configurando el upstream para la rama actual en tu repositorio local al origen `main`.

Al especificar este upstream, simplificas tus interacciones futuras con el repositorio remoto. Una vez que el upstream está configurado, puedes usar `git push` sin parámetros adicionales para subir a la misma rama en el repositorio remoto. Esto significa que una operación de `git push` subsecuente sabrá automáticamente que debe subir tus commits al origen `main`.

## Examinando el código en GitHub

Ahora, examinemos cómo se ve el código subido en GitHub.

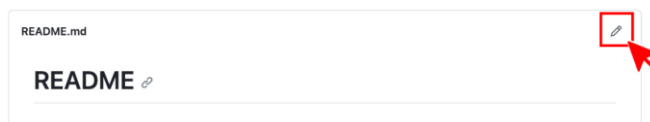
Cuando visitas un repositorio de GitHub, lo primero que te recibe es la página principal del repositorio. Aquí, puedes ver los archivos actualizados más recientemente, el archivo `README.md` (si está disponible) y varios detalles del repositorio. Esta vista proporciona una instantánea rápida del contenido y propósito del proyecto:



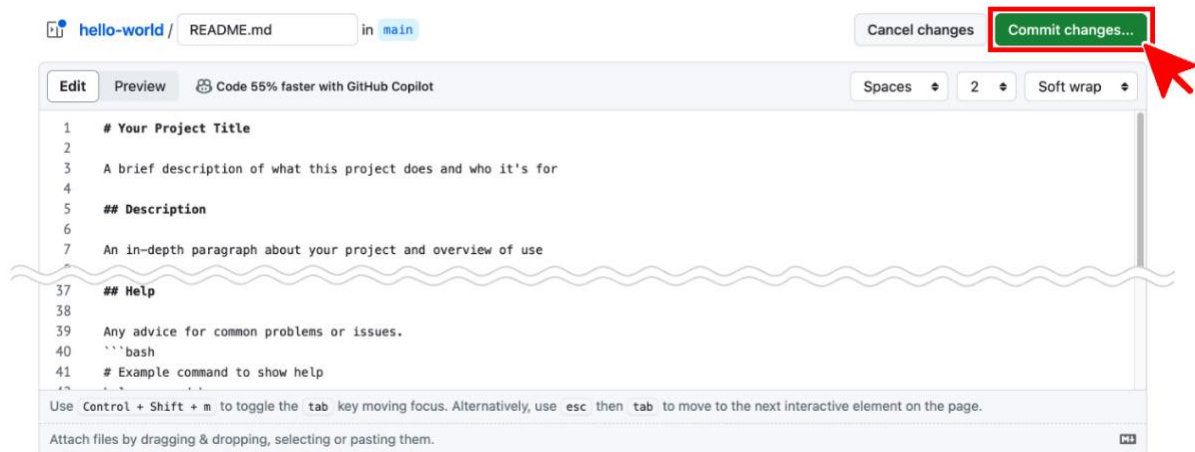
## Editando código en GitHub

Una de las características clave es la capacidad de interactuar con el código directamente en GitHub. Puedes agregar o editar archivos usando la propia interfaz, lo cual es particularmente útil para cambios pequeños o correcciones rápidas. Aquí te mostramos cómo hacerlo:

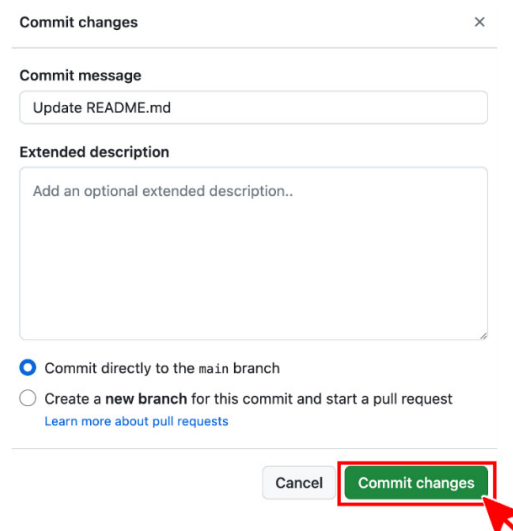
1. Haz clic en el botón de edición en la esquina superior derecha para ingresar al modo de edición:



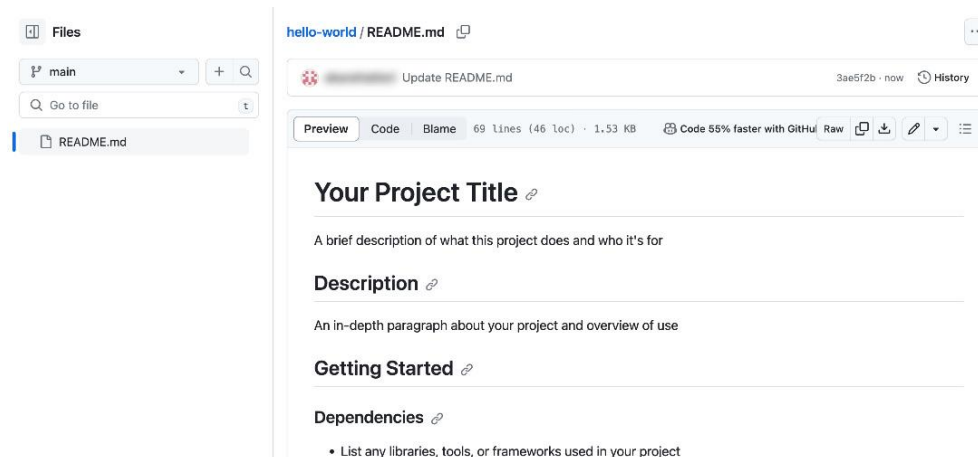
2. Luego, puedes editar y confirmar tus cambios. Recuerda que esto se confirma directamente en tu base de código:



3. Esta acción confirma tus cambios directamente en tu repositorio remoto. Sin embargo, también tienes la flexibilidad de seleccionar un destino y una rama para tu commit. Aunque el valor predeterminado es típicamente la rama en la que estás trabajando, tienes la opción de crear una nueva rama simultáneamente, lo cual puede ser particularmente útil al iniciar una nueva contribución. Si eliges crear una nueva rama, aún puedes fusionarla más tarde en la línea principal:



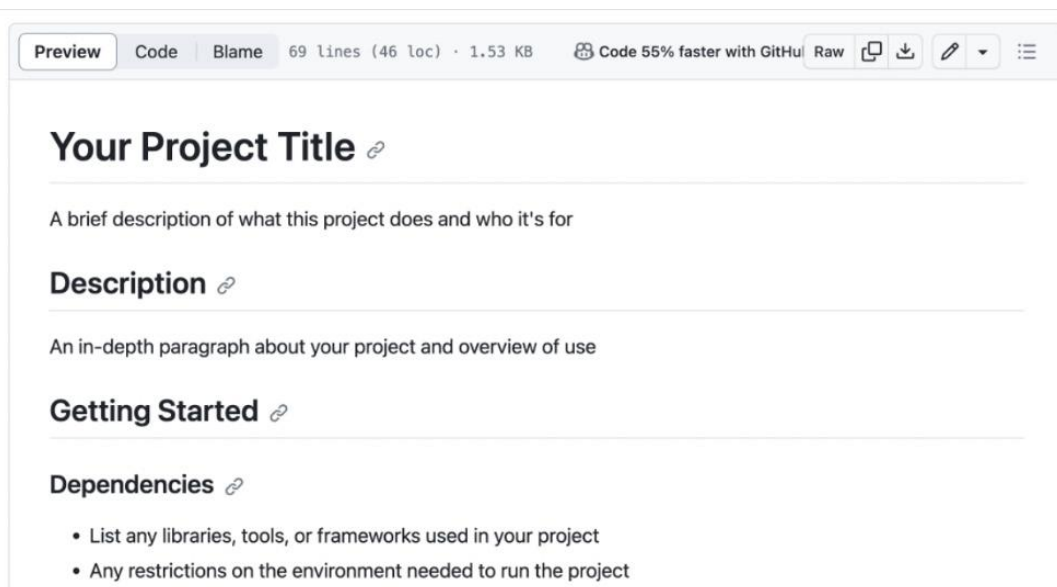
4. Ahora, deberías ver los cambios reflejados, como se muestra en la siguiente captura de pantalla:



## Revisando código y cambios en GitHub

Ahora, echemos un vistazo dentro de GitHub a través del código que acabamos de actualizar. Para una mirada más cercana al código, GitHub proporciona varias vistas en el navegador de archivos:

El modo de **vista previa** es el predeterminado, disponible para ciertos tipos de archivos como Markdown, y muestra el archivo tal como aparecería en su estado formateado:



El modo de **código** muestra el contenido del archivo tal como está en el commit más reciente, con un hermoso resaltado de sintaxis:

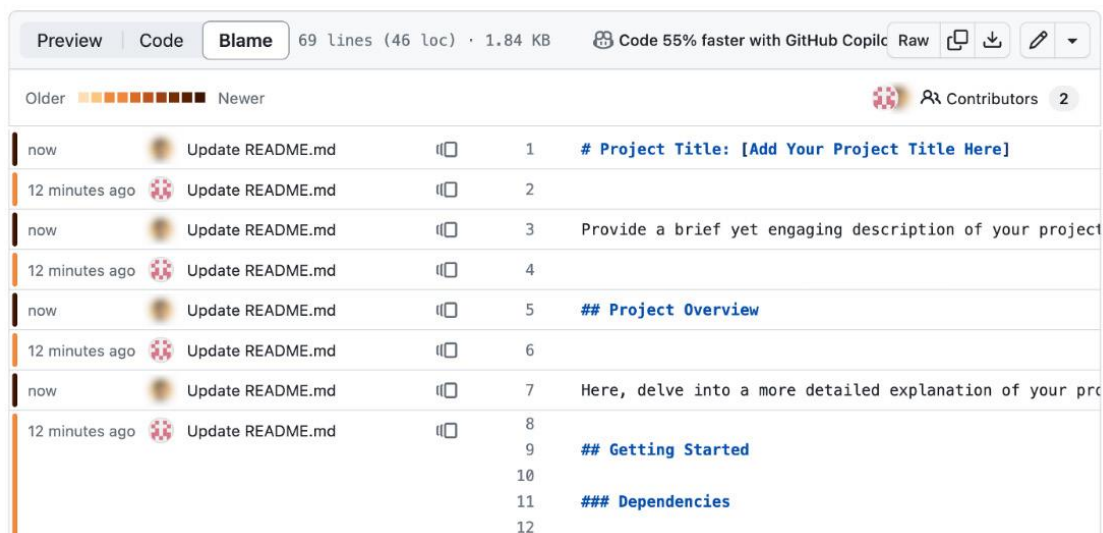




The screenshot shows the GitHub Code view for a file named README.md. The interface includes tabs for 'Preview', 'Code', and 'Blame'. The 'Code' tab is active, displaying the file's content. The file is 69 lines long (46 loc) and 1.53 KB. The content is a template for a README file, starting with a title '# Your Project Title', followed by a description, a '## Description' section, a '## Getting Started' section, and a '### Dependencies' section. The dependencies section lists two items: '- List any libraries, tools, or frameworks used in your project' and '- Any restrictions on the environment needed to run the project'. The file ends with a '### Installing' section.

```
1  # Your Project Title
2
3  A brief description of what this project does and who it's for
4
5  ## Description
6
7  An in-depth paragraph about your project and overview of use
8
9  ## Getting Started
10
11 ### Dependencies
12
13 - List any libraries, tools, or frameworks used in your project
14 - Any restrictions on the environment needed to run the project
15
16 ### Installing
17
```

El modo **blame** es particularmente perspicaz, ya que desglosa el archivo por líneas, mostrando quién modificó por última vez cada línea y en qué commit. Esto es invaluable para comprender la evolución del código y rastrear cambios:

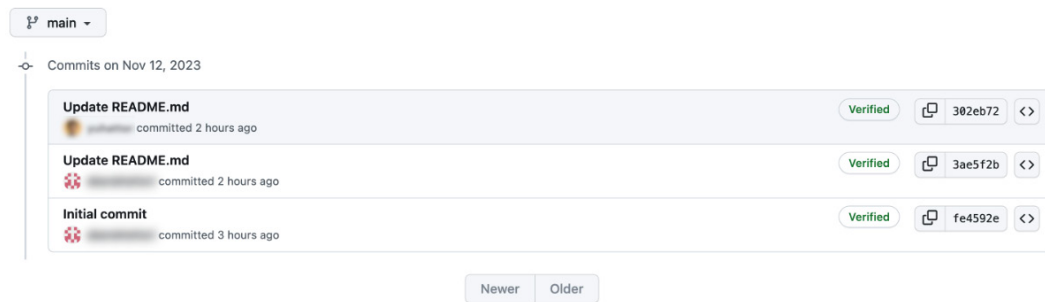


The screenshot shows the GitHub Blame view for the same README.md file. The 'Blame' tab is active, displaying a table of changes. The table has columns for 'Older', 'Newer', 'Commit', 'Author', 'File', 'Line', and 'Content'. The 'Commit' column shows the commit hash, and the 'Author' column shows the author's name. The 'File' column shows the file name, and the 'Line' column shows the line number. The 'Content' column shows the content of the line. The table shows that the file was updated 12 minutes ago by a user named 'Update README.md'. The content of the file is the same as in the Code view, but the lines are color-coded to show which commit they belong to. The lines are grouped by commit, with the most recent commit at the top. The lines are color-coded to show which commit they belong to. The lines are grouped by commit, with the most recent commit at the top.

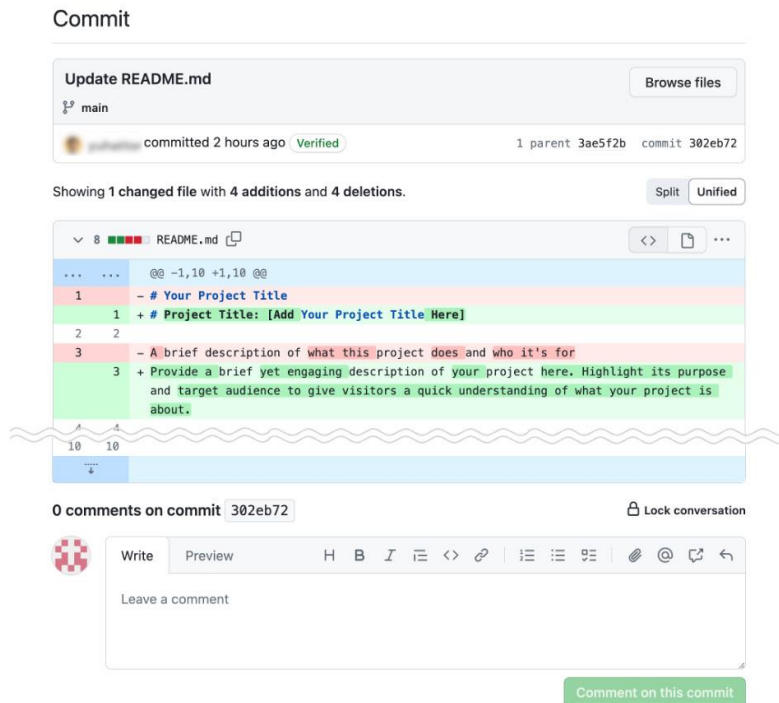
| Older          | Newer | Commit | Author           | File      | Line | Content  |
|----------------|-------|--------|------------------|-----------|------|--|
| now            |       |        | Update README.md | README.md | 1    | # Project Title: [Add Your Project Title Here]               |
| 12 minutes ago |       |        | Update README.md | README.md | 2    |  |
| now            |       |        | Update README.md | README.md | 3    | Provide a brief yet engaging description of your project     |
| 12 minutes ago |       |        | Update README.md | README.md | 4    |  |
| now            |       |        | Update README.md | README.md | 5    | ## Project Overview  |
| 12 minutes ago |       |        | Update README.md | README.md | 6    |  |
| now            |       |        | Update README.md | README.md | 7    | Here, delve into a more detailed explanation of your project |
| 12 minutes ago |       |        | Update README.md | README.md | 8    |  |
|                |       |        |                  | README.md | 9    | ## Getting Started   |
|                |       |        |                  | README.md | 10   |  |
|                |       |        |                  | README.md | 11   | ### Dependencies   |
|                |       |        |                  | README.md | 12   |  |

Esto es útil cuando no deseas obtener el código localmente y revisarlo, pero deseas echar un vistazo al código en GitHub primero. Dentro de la interfaz del repositorio de GitHub, tienes la capacidad de explorar el historial de commits del proyecto. Esta función te permite profundizar en los detalles de cada commit, examinando los cambios realizados:

## Commits



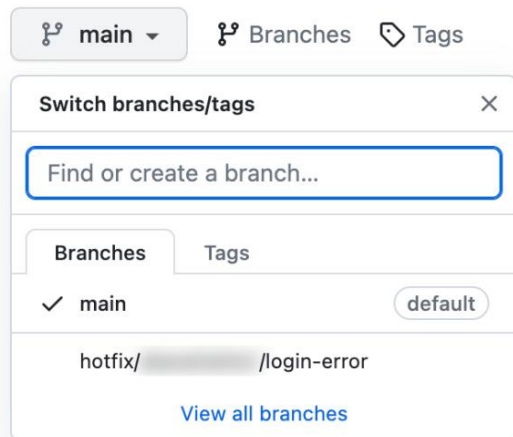
Además, GitHub ofrece la funcionalidad de comentar sobre estos cambios, permitiéndote participar en discusiones o proporcionar comentarios sobre alteraciones específicas hechas en el repositorio:



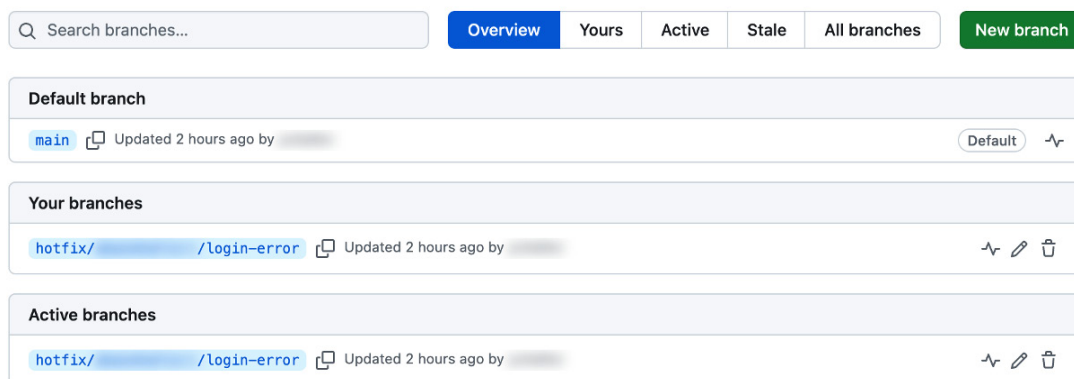
## Gestión de ramas en GitHub

Gestionar ramas en GitHub es fácil. Aquí tienes una breve descripción de la gestión de ramas en GitHub.

Primero, navegar entre diferentes ramas en un repositorio es una tarea común. En GitHub, puedes cambiar fácilmente entre ramas usando el menú desplegable de ramas, que normalmente se encuentra en la parte superior de la página del repositorio. Esta función te permite moverte rápidamente de una rama a otra, permitiéndote revisar diferentes versiones o etapas del proyecto de manera eficiente:



En repositorios que tienen múltiples ramas, encontrar una rama específica puede volverse un desafío. GitHub proporciona una funcionalidad de búsqueda dentro del menú desplegable de ramas. Esta función te permite filtrar y encontrar rápidamente la rama que estás buscando, ahorrando tiempo y mejorando tu flujo de trabajo:



Iniciar una nueva línea de desarrollo a menudo se hace creando una nueva rama. GitHub simplifica este proceso para crear una nueva rama. Puedes nombrar la nueva rama y basarla en una existente, haciendo que sea sencillo ramificar para nuevas características o experimentos:

Create a branch

×

New branch name

feature/logout-page

📄

Source

🔗 main ▾

Cancel

Create new branch

Las funciones de gestión de ramas de GitHub proporcionan una manera fluida y eficiente de manejar múltiples líneas de desarrollo dentro de un solo repositorio. Ya sea que estés cambiando para ver diferentes estados del proyecto, buscando una rama específica o creando una nueva rama para el desarrollo, la interfaz de GitHub hace que estas tareas sean intuitivas y accesibles. Este enfoque simplificado de la gestión de ramas es integral para mantener un entorno de desarrollo organizado y productivo.

### **git pull: Conectando el trabajo local y remoto**

En el ámbito del control de versiones con Git, mantenerse actualizado con los últimos cambios en un repositorio remoto es crucial para una colaboración y desarrollo fluidos. El comando `git pull` es la herramienta diseñada precisamente para este propósito. Sirve como un puente, trayendo los cambios realizados remotamente en plataformas como GitHub a tu directorio de trabajo local.

Usar `git pull` es particularmente importante cuando se trabaja en un entorno colaborativo. Supongamos que los miembros de tu equipo están haciendo commits a un repositorio compartido en GitHub. En ese caso, al extraer regularmente estos cambios, aseguras que el trabajo de todos esté alineado y reduces la probabilidad de conflictos o inconsistencias.

Para usar `git pull`, navega al directorio de tu repositorio en tu línea de comandos o terminal e ingresa el siguiente comando:

```
$ git pull origin main
```

Es un comando simple pero poderoso que mantiene la armonía de tus esfuerzos colaborativos y mantiene tu repositorio local actualizado con los últimos desarrollos.

Cuando ejecutas `git pull`, lo que esencialmente sucede es un proceso de dos pasos. Primero, Git obtiene las actualizaciones del repositorio remoto, lo que incluye todos los commits y ramas que se han subido desde tu última actualización. Luego, fusiona estas actualizaciones en tu repositorio local. Esta fusión es crítica ya que integra los cambios remotos con tu trabajo local, manteniendo tu repositorio local en sincronía con su contraparte remota.

El comando `git pull` podría parecer sencillo a primera vista, pero en realidad es una combinación de dos comandos fundamentales de Git: **`git fetch`** y **`git merge`**. Esta naturaleza dual convierte a `git pull` en una herramienta poderosa en el arsenal de Git.

Siguiendo adelante, profundicemos en el primer componente de este proceso: `git fetch`. Este comando es una pieza esencial del rompecabezas del control de versiones, permitiéndote ver lo que otros han estado trabajando sin fusionar esos cambios en tu propio trabajo todavía.

### **`git fetch`: Sincronizando sin interrupciones**

`git fetch` juega un papel crucial en cómo los desarrolladores interactúan con los repositorios remotos. En su núcleo, `git fetch` se trata de actualizar de manera segura y eficiente tu repositorio local con cambios de su contraparte remota.

Cuando ejecutas `git fetch`, Git contacta al repositorio remoto especificado y extrae todos los datos de él que aún no tienes. Esto incluye nuevos commits, ramas y etiquetas. Lo hermoso de `git fetch` es que hace esto sin hacer ningún cambio en tus archivos de trabajo. Es como echar un vistazo a lo que otros han estado haciendo sin integrar realmente sus cambios en tu trabajo. Esta característica lo convierte en una operación no destructiva, asegurando que tu trabajo de desarrollo actual permanezca intacto.

Los datos obtenidos se almacenan en tu repositorio local, pero se mantienen separados de tus archivos de proyecto reales. Para incorporar estos cambios obtenidos en tu trabajo, típicamente seguirías con un comando `git merge`, que fusiona la rama obtenida en tu rama actual. Sí, `git fetch` puede mostrar su verdadero poder cuando se usa con `git merge`.

### **Fetch vs Pull**

Volvamos un poco a `git pull` aquí. Cuando ejecutas `git pull`, primero inicia una operación `git fetch`. Esta parte del proceso alcanza al repositorio remoto y extrae todos los nuevos datos que encuentra. Estos datos incluyen commits, archivos y referencias actualizadas en el repositorio remoto desde tu última extracción. Es un paso vital para asegurarte de que tienes toda la información más reciente del repositorio remoto, pero no integra automáticamente estos cambios en tus archivos de trabajo.

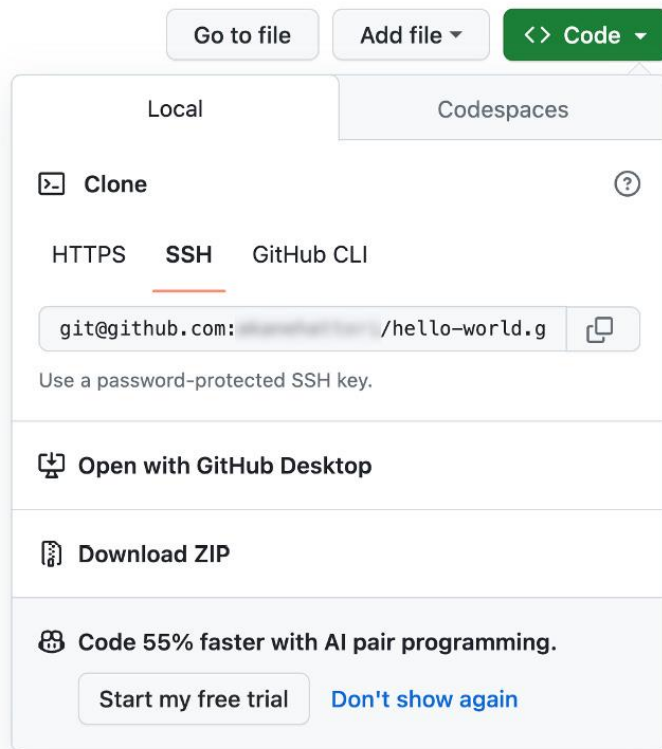
La segunda parte del comando `git pull` es donde `git merge` entra en juego. Después de obtener actualizaciones, `git merge` toma estas referencias recién descargadas y las incorpora en tu repositorio local. Este proceso de fusión es lo que realmente actualiza tus archivos de trabajo actuales con los cambios del repositorio remoto. Es una integración fluida de los cambios remotos con tu trabajo local, manteniendo tu repositorio en perfecta sincronía con su contraparte remota.

Comprender la naturaleza dual de `git pull` como una combinación de operaciones de obtención y fusión revela su verdadero poder para gestionar y sincronizar cambios de código en un entorno colaborativo.

También, conocer la diferencia entre `git fetch` y `git pull` es crucial. Te permite controlar con más precisión cuándo y cómo los cambios de tu repositorio remoto se incorporan a tu trabajo local. Esta clara comprensión es esencial para una colaboración fluida y una gestión experta del repositorio, ya que te permite decidir estratégicamente si solo deseas revisar cambios o integrarlos completamente.

## git clone: Trayendo repositorios de GitHub a tu espacio de trabajo

Hablando de clonar y descargar, estas opciones están disponibles para cada repositorio. Al ir a la página del repositorio y presionar el botón **< > Code**, verás que existen esas opciones. Clonar crea una copia local del repositorio en tu máquina, permitiéndote trabajar en el proyecto sin conexión, mientras que descargar proporciona un archivo ZIP del proyecto para respaldo o propósitos de revisión:



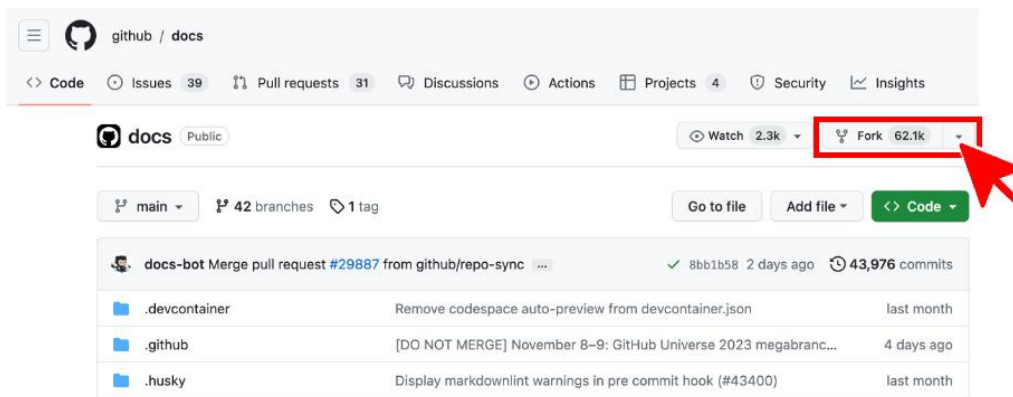
Ejecutar el comando git clone es un proceso sencillo, lo que lo hace fácilmente accesible para cualquier persona que quiera involucrarse en un proyecto alojado en una plataforma como GitHub. Para clonar un repositorio, todo lo que necesitas es la URL del repositorio que deseas clonar:

```
$ git clone [Your SSH URL]
```

git clone se destaca como un comando central, permitiéndote crear una copia local precisa de un repositorio existente. Este proceso implica más que solo duplicar archivos actuales; replica el repositorio en su totalidad. Esto incluye todas las versiones de archivos, el historial completo de commits y todas las ramas. Al usar git clone, traes una versión completa y funcional del proyecto a tu máquina local. Esto no solo te brinda la capacidad de trabajar sin conexión, sino que también proporciona una visión completa del historial de desarrollo del proyecto, lo que facilita la comprensión y la contribución efectiva.

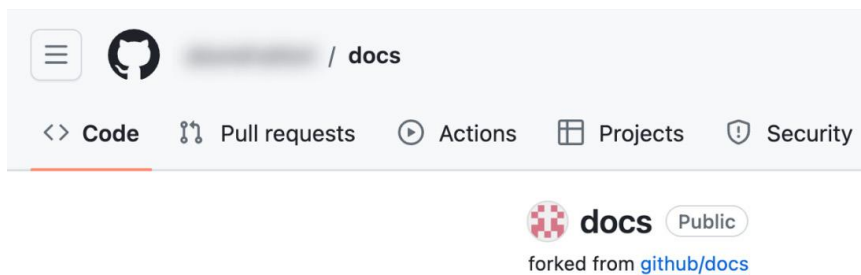
## Forking: Más que solo copiar código

Además de git clone, hay otra forma de duplicar un repositorio en GitHub. Esto es especialmente útil para el desarrollo de código abierto. El concepto de forking en Git, particularmente en plataformas como GitHub, es una piedra angular del desarrollo colaborativo y de código abierto. Hacer un fork de un repositorio significa crear tu propia copia personal del proyecto de otra persona bajo tu cuenta:



Cuando haces un fork de un repositorio, creas una copia personal del proyecto de otra persona dentro de tu cuenta de GitHub. Si bien esta copia comienza como un espejo del original, opera de manera independiente, lo que significa que puedes hacer modificaciones, adiciones o experimentos sin afectar el proyecto original. Sin embargo, es importante comprender que esta independencia tiene límites. Por ejemplo, si el repositorio original se elimina o su visibilidad cambia, puede afectar el estado del fork. A pesar de estas dependencias, hacer forks sigue siendo una práctica fundamental en el desarrollo de código abierto, permitiendo a los desarrolladores contribuir a través de pull requests sin necesidad de acceso directo de escritura al repositorio fuente.

Hacer forks es particularmente significativo en el mundo del código abierto. Permite a los desarrolladores contribuir a proyectos realizando cambios en sus forks y luego proponiendo estos cambios al proyecto original a través de un proceso llamado pull request. Así es como puedes contribuir a un proyecto sin tener acceso directo de escritura al repositorio fuente. Cuando haces un fork, se crea una copia en tu nuevo entorno, como se muestra en la siguiente captura de pantalla:



Hacer forks proporciona una plataforma única donde cualquiera, independientemente de su relación o nivel de confianza con los mantenedores del proyecto original, puede experimentar y contribuir libremente. Este enfoque reduce significativamente las barreras de entrada en la programación colaborativa.

Al hacer un fork de un repositorio, creas un entorno donde puedes agregar tus ideas, mejoras o correcciones al proyecto sin impactar el repositorio upstream. Esto es particularmente empoderador para los nuevos colaboradores que aún no han ganado la confianza de los mantenedores del proyecto para tener acceso directo al repositorio principal. Les permite demostrar sus capacidades y contribuciones en un espacio separado y personal.

Esta naturaleza independiente pero conectada de un fork es crucial. Permite un proceso de contribución en dos fases: primero, en tu propio fork, donde experimentas y haces cambios libremente, y segundo, a través de un pull request, donde propones que estos cambios se integren en el proyecto original. Este flujo de trabajo fomenta una cultura de colaboración abierta, donde las ideas y contribuciones se intercambian libremente y las mejores se integran sin problemas en los proyectos.

En esencia, hacer forks es más que solo copiar un repositorio; se trata de participar en una comunidad más grande. Ya sea que estés contribuyendo a un proyecto existente, comenzando uno nuevo basado en el trabajo de otra persona o simplemente experimentando, hacer forks es un aspecto esencial de trabajar con Git y GitHub.