

Práctico 2: Git y GitHub

Objetivo:

El estudiante desarrollará competencias para trabajar con Git y GitHub, aplicando conceptos fundamentales de control de versiones, colaboración en proyectos y resolución de conflictos, en un entorno simulado y guiado.

Alumno: Fernando Gómez

Actividades

- 1) Contestar las siguientes preguntas utilizando las guías y documentación proporcionada (Desarrollar las respuestas) :

- ¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo basada en la web que permite a los programadores y desarrolladores almacenar y compartir su código. Está centrada en **Git**, que es un sistema de control de versiones, es decir, una herramienta que permite realizar un seguimiento de los cambios en el código a lo largo del tiempo, facilitando la colaboración entre varios desarrolladores.

Algunas de las principales características de GitHub son:

1. **Repositorios:** Son espacios donde los desarrolladores pueden almacenar su código. Un repositorio puede ser público o privado.
2. **Control de versiones:** A través de Git, GitHub permite gestionar diferentes versiones del código y realizar un seguimiento de las modificaciones, lo que ayuda a evitar conflictos entre los cambios realizados por diferentes personas.
3. **Colaboración:** GitHub facilita la colaboración entre desarrolladores. Podemos "hacer pull requests" (solicitudes de extracción), que permiten sugerir cambios a un repositorio y revisar esas modificaciones antes de integrarlas al proyecto principal.
4. **Forks:** Los desarrolladores pueden crear una copia de un repositorio para trabajar en ella sin afectar el proyecto original. Esto se conoce como "hacer un fork".
5. **Issues:** GitHub permite a los usuarios reportar problemas o errores en el código a través de los "issues" (problemas), lo que facilita el seguimiento de tareas y problemas a lo largo del desarrollo.

6. **GitHub Actions:** Es una herramienta que permite la automatización de flujos de trabajo, como la integración continua (CI) o la entrega continua (CD), lo que facilita el proceso de pruebas y despliegue de software.

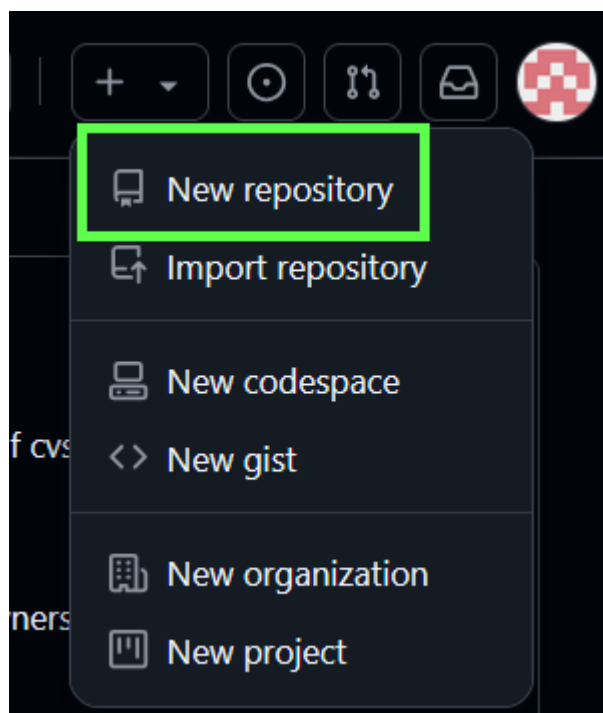
GitHub se utiliza no solo para proyectos de código abierto, sino también para proyectos privados dentro de empresas o equipos de desarrollo. Además, se ha convertido en una red social para desarrolladores, permitiendo compartir proyectos, colaborar, y aprender unos de otros.

- ¿Cómo crear un repositorio en GitHub?

Para crear un repositorio en Github debemos iniciar sesión en su plataforma y hacer click en el botón + en la esquina superior derecha.



A continuación, hacemos click en New Repository.



Luego completamos los campos que nos pide como:

Nombre del Repositorio: introducimos un nombre único para el repositorio.

Descripción (opcional): podemos agregar una breve descripción sobre el propósito o el contenido del repositorio.

Visibilidad:

- **Público:** cualquiera en Internet puede ver el repositorio.
- **Privado:** solo quien lo creó y sus colaboradores pueden verlo.

Opciones de Inicialización:

- podemos marcar la casilla para crear un archivo **README** (útil para documentar el proyecto).
- opcionalmente, podemos agregar un archivo **.gitignore** para excluir ciertos archivos del control de versiones.
- también se puede elegir una **licencia** para el proyecto.


Una vez completados estos campos, hacemos click en "**Create Repository**" para finalizar.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *

 FernandoGGomez ▾

/


Repository name *

nuevo_repositorio


✓ nuevo_repositorio is available.

Great repository names are short and memorable. Need inspiration? How about [laughing-dollop](#) ?

Description (optional)

☒  Public

Anyone on the internet can see this repository. You choose who can commit.

☐  Private

You choose who can see and commit to this repository.

Initialize this repository with:

☒ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **None** ▾


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

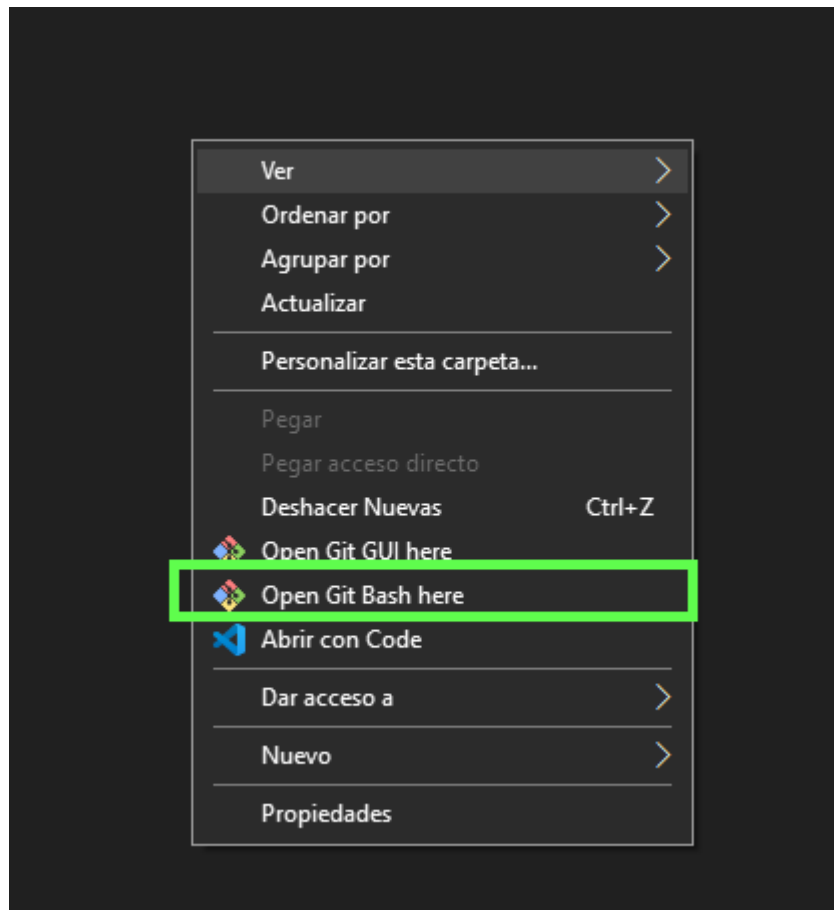
 You are creating a public repository in your personal account.

Create repository

- ¿Cómo crear una rama en Git?

Para crear una rama en Git primero debemos iniciar git.

Para hacerlo, nos posicionamos en una carpeta donde queramos crear nuestro repositorio, apretamos click derecho y abrimos la terminal de git (Git Bash) o cualquier terminal con la que trabajemos.



Después ejecutamos el comando `git init` para inicializar un nuevo repositorio.

```
$ git init
```

A continuación el comando `git add .`, para agregar los archivos que deseamos que se agreguen en el próximo commit.

```
$ git add .
```

Luego ejecutamos el comando `git commit -m ""` para tener un punto de referencia desde el cuál vamos a crear nuestras ramas.

```
$ git commit -m "rama principal"
```

Una vez que ya tenemos nuestra rama principal, podemos usar el comando git branch [nombre-de-rama] para crear nuestra rama.

```
$ git branch rama1
```

- ¿Cómo cambiar a una rama en Git?

Para cambiar a una rama en Git usamos el comando git checkout [nombre-de-rama]

```
$ git checkout rama1  
Switched to branch 'rama1'
```

- ¿Cómo fusionar ramas en Git?

Para fusionar ramas en Git debemos posicionarnos en la rama a la que queremos agregarle los cambios que se hicieron en otra rama y escribir ejecutar el comando git merge [nombre-de-rama].

Ejemplo fusionando las ramas “main” y “rama1”. Posicionado en main, ejecuto el comando git merge.

```
$ git merge rama1  
Already up to date.
```

En el ejemplo dice “Already up to date”, lo que significa que las ramas no tenían diferencias entre sí.

- ¿Cómo crear un commit en Git?

Para crear un commit, primero debemos agregar los cambios que queremos registrar con el comando git add [nombre-archivo], después de eso se puede ejecutar el comando git commit, se recomienda una descripción al commit para que el registro sea más claro con el comando -m pero no es obligatorio para que el commit funcione.

```
$ git add .
```

```
$ git commit -m "Se cambió el nombre al archivo principal"
```

- ¿Cómo enviar un commit a GitHub?

Para enviar un commit a GitHub se ejecuta el comando git remote add origin URL_REPOSITORIO para vincular nuestro repositorio local con el remoto alojado en Github. A continuación se agregan los cambios que queramos subir con git add y se confirman con git commit -m “descripción del commit”. Después se ejecuta el comando git push -u origin main, usando -u establecemos la rama main como la predeterminada para futuros git push. En los próximos envíos que hagamos no se

utiliza -u ni origin.

```
$ git remote add origin https://github.com/FernandoGGomez/nuevo_repositorio.git
```

```
$ git add .
```

```
$ git commit -m "Se cambio el nombre al archivo principal"
```

Este era el nombre del archivo antes del cambio.



Y este después del cambio.



- ¿Qué es un repositorio remoto?

Un **repositorio remoto** es una versión de nuestro repositorio Git que se encuentra **alojada en un servidor externo** (una plataforma como GitHub, Gitlab, etc.). A diferencia de un repositorio **local**, que está en nuestra máquina, un repositorio remoto nos permite colaborar con otras personas y almacenar nuestro código de manera centralizada en la nube, lo que facilita la **sincronización** y **compartición** de nuestros cambios.

- ¿Cómo agregar un repositorio remoto a Git?

Para agregar un repositorio remoto a Git se utiliza el comando `git remote add origin URL_REPOSITORIO`.

- ¿Cómo empujar cambios a un repositorio remoto?

Para empujar cambios a un repositorio remoto la primera vez se utiliza el comando `git push -u origin main`, con -u establecemos la rama main como la predeterminada

para futuros git push. En los próximos envíos que hagamos no se utiliza -u ni origin, solo se utiliza el comando git push

- ¿Cómo tirar de cambios de un repositorio remoto?

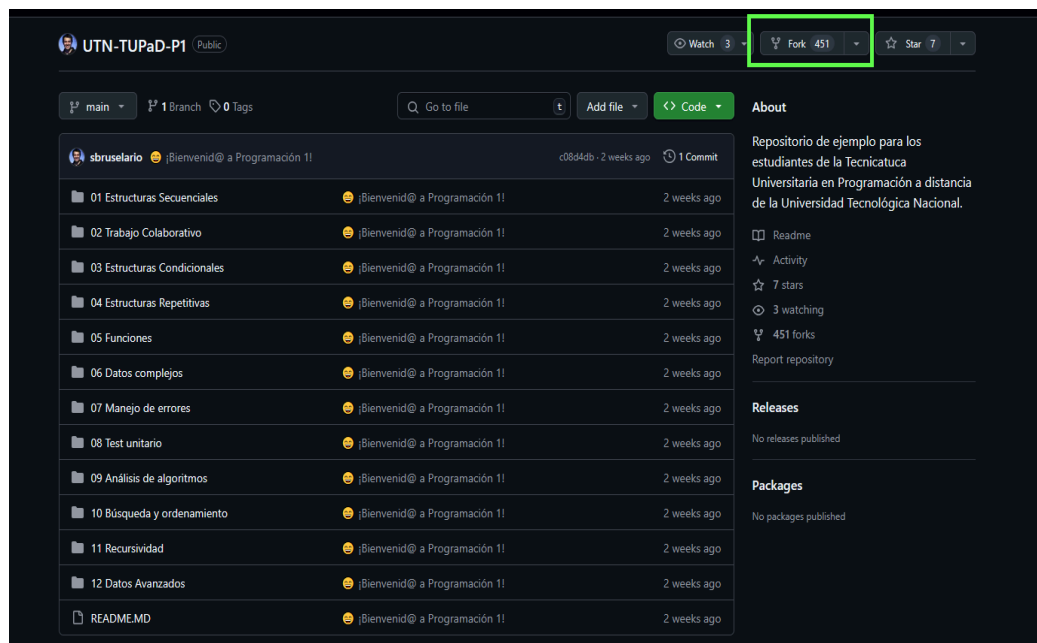
Para tirar de cambios de un repositorio remoto se utiliza el comando git pull si la rama que quieres tirar ya fue establecida como la predeterminada para hacer push y pull, sino se usa el comando git pull origin main

- ¿Qué es un fork de repositorio?

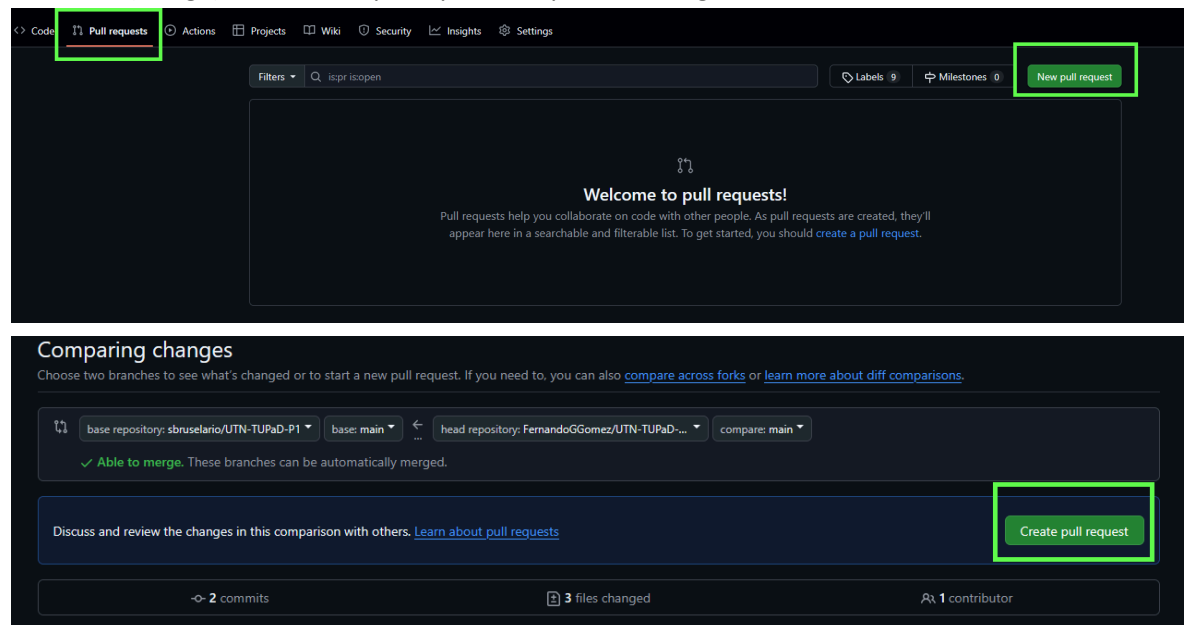
Un **fork** de un repositorio es una **copia personal** de un repositorio en plataformas como **GitHub**, **GitLab** o **Bitbucket**. Al hacer un **fork**, creamos una réplica del repositorio original en nuestra cuenta, lo que nos permite realizar cambios sin afectar al repositorio original. Este proceso es común en proyectos de código abierto y facilita la colaboración, ya que podemos trabajar en nuestras propias versiones del código y luego sugerir cambios a través de **pull requests**.

- ¿Cómo crear un fork de un repositorio?

Para hacerlo iniciamos sesión en Github, buscamos el repositorio que queremos forkear y hacemos click en el botón que dice fork, a continuación completamos los datos que nos pide, un nombre para el repositorio y una descripción opcional.



- ¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?
Para hacer un pull request nos dirigimos a nuestro repositorio forqueado en Github y nos posicionamos en la pestaña “Pull requests”, en la esquina superior derecha encontraremos un botón que dice “New pull request”. Hacemos click en él. Y después en “Create pull request”. Si se aprueban los cambios, el pull request se **fusionará** (merge) con la rama principal del repositorio original.



- ¿Cómo aceptar una solicitud de extracción?
Navegamos al repositorio en GitHub donde se ha enviado la solicitud de extracción. Hacemos click en la pestaña "Pull requests" para ver la lista de solicitudes pendientes. Seleccionamos la solicitud que deseamos revisar. Acá podemos ver los cambios propuestos, comentarios y el historial de confirmaciones.
Es aconsejable probar los cambios en nuestro entorno local antes de fusionarlos. Podemos hacerlo extrayendo la rama de la solicitud de extracción en nuestra máquina local.
Si todo está en orden y deseamos aceptar los cambios, volvemos a la página de la solicitud de extracción en GitHub y hacemos click en el botón "Merge pull request" (Fusionar solicitud de extracción). Aparecerá una ventana emergente; hacemos click en "Confirm merge" (Confirmar fusión) para completar el proceso.
Es importante asegurarse de que los cambios propuestos no introduzcan conflictos o problemas en el código base. Realizar pruebas locales y revisiones detalladas antes de fusionar es una buena práctica en el desarrollo colaborativo.
- ¿Qué es un etiqueta en Git?
En **Git**, una **etiqueta** (o **tag**) es una referencia que se usa para señalar puntos específicos en el historial de un repositorio, como, por ejemplo, una versión de lanzamiento de un software o un hito importante en el proyecto.
Las etiquetas son muy útiles para marcar versiones de un proyecto, ya que actúan como una especie de **marca** en el tiempo que nos permite volver fácilmente a ese punto o identificar un hito importante, como un **release** (lanzamiento) o **versión** del software.

- ¿Cómo crear una etiqueta en Git?

Existen dos tipos de etiquetas:

1. **Etiquetas ligeras (Lightweight tags):**

- Son básicamente un puntero a un commit específico. No contienen mucha información adicional, solo un nombre de etiqueta que apunta al commit.
- Funcionan como un alias para un commit específico en el historial.

Comando para crear una etiqueta ligera:

```
$ git tag nombre_etiqueta
```

Este comando crea una etiqueta ligera en el commit actual.

2. **Etiquetas anotadas (Annotated tags):**

- Son etiquetas más completas que almacenan información adicional, como el nombre del autor, la fecha y un mensaje. Son más como un "commit" completo, pero con el propósito de marcar un punto en el tiempo.
- Recomendadas para marcar versiones de lanzamiento o hitos importantes, ya que contienen más metadatos y son fáciles de buscar.

Comando para crear una etiqueta anotada:

```
$ git tag -a nombre_etiqueta -m "Mensaje de la etiqueta"
```

- ¿Cómo enviar una etiqueta a GitHub?

Una vez que hayamos creado la etiqueta, podemos **subirla a GitHub** usando el siguiente comando:

```
$ git push origin nombre_etiqueta
```

Si queremos enviar todas las etiquetas que hayamos creado localmente a GitHub (en lugar de enviar una por una), podemos usar el siguiente comando:

```
$ git push --tags
```

- ¿Qué es un historial de Git?

El **historial de Git** se refiere a la **lista completa de los cambios** realizados en un repositorio de Git a lo largo del tiempo. Este historial incluye todos los **commits**, es decir, los puntos en los que se ha registrado un conjunto de cambios en el proyecto. Es una herramienta fundamental para seguir el rastro de la evolución de un proyecto, ver cómo ha cambiado y quién ha realizado esos cambios.

Elementos clave del historial de Git:

1. **Commits:**

- Un commit es una instantánea del estado de los archivos en un momento dado. Cada commit en Git tiene un identificador único (un hash) y contiene información como: el autor del commit, la fecha y hora en la que se realizó el commit y un mensaje que describe los cambios realizados.
- Git mantiene el **historial de commits** en una **rama** específica (por

ejemplo, main o develop).

2. Ramas:

- Las ramas son diferentes líneas de desarrollo dentro de un repositorio. Git mantiene el historial de cambios por rama, lo que te permite trabajar en diferentes características sin interferir con el código principal.
- Podemos cambiar entre ramas para ver el historial de cada una.

3. Identificadores de commit:

Cada commit tiene un identificador único, también conocido como hash o SHA-1. Este identificador nos permite acceder a ese commit específico en el historial.

4. Rango de commits:

Podemos ver un rango de commits entre dos puntos (por ejemplo, entre dos ramas o entre dos commits específicos) para revisar los cambios que ocurrieron en ese intervalo de tiempo.

- ¿Cómo ver el historial de Git?

Existen varios comandos en Git que nos permiten ver y explorar el historial de cambios:

1. Ver el historial de commits:

Podemos ver el historial de commits usando el comando git log:

```
$ git log
```

Esto mostrará una lista de los commits más recientes, incluyendo:

- El **hash** de cada commit.
- El **autor** del commit.
- La **fecha** en que se hizo el commit.
- El **mensaje** asociado al commit.

2. Ver el historial de una rama específica:

Si deseamos ver el historial de una rama específica, podemos usar el comando git log nombre_rama:

```
$ git log rama1
```

3. Ver un resumen del historial de cambios:

Si solo queremos un resumen compacto, podemos usar el comando git log -oneline. Esto mostrará los commits en una línea por commit, lo que es útil para obtener un vistazo rápido del historial:

```
$ git log --oneline
```

4. Ver los cambios realizados en un commit específico:

Para ver qué cambios se realizaron en un commit específico, se usa el comando git show junto con el identificador del commit. Esto nos mostrará los cambios realizados en ese commit en detalle (archivos modificados, líneas añadidas o eliminadas, etc.):

```
$ git show e9c8a5e
```

5. Ver el historial de cambios de un archivo:

Si deseamos ver el historial de un archivo en particular, usamos el comando git log - - nombre_archivo:

```
$ git log -- nuevo_nombre.txt
```

- ¿Cómo buscar en el historial de Git?

En **Git**, podemos buscar en el historial de commits de varias formas, utilizando diferentes comandos y opciones para encontrar lo que necesitamos, ya sea un commit específico, un cambio en un archivo o un término en los mensajes de commit.

Buscar por mensaje de commit: git log --grep="mensaje"

Buscar por autor: git log --author="autor"

Ver el historial de un archivo específico: git log nombre_archivo

Buscar commits entre fechas: git log --since="fecha_inicio" --until="fecha_fin"

Ver un commit específico por hash: git show <commit_hash>

Buscar cambios de código específicos (por ejemplo, adición o eliminación de una palabra): git log -S"cadena_buscada"

Comparar diferencias entre dos commits: git diff <commit1> <commit2>

Estas herramientas de búsqueda nos permiten explorar y analizar el historial de un repositorio Git de manera eficiente, facilitando la navegación entre los diferentes cambios y versiones del proyecto.

- ¿Cómo borrar el historial de Git?

Si se desea eliminar todos los commits y empezar desde cero, se puede restablecer el repositorio a un estado limpio. Para hacerlo, podemos hacer un **hard reset** al primer commit o simplemente restablecer el repositorio a un estado vacío.

1. Restablecer el repositorio al primer commit:

```
git checkout --orphan nueva-rama
```

```
git add -A
```

```
git commit -m "Nuevo inicio sin historial"
```

2. Eliminar la rama principal (opcional): Si deseas eliminar la rama principal(main o master) y renombrar la nueva rama, se puede hacer:

```
git branch -D main
```

```
git branch -m main
```

3. **Forzar el push a GitHub:** Después de realizar estos cambios, se necesitará forzar el push para actualizar el repositorio remoto:

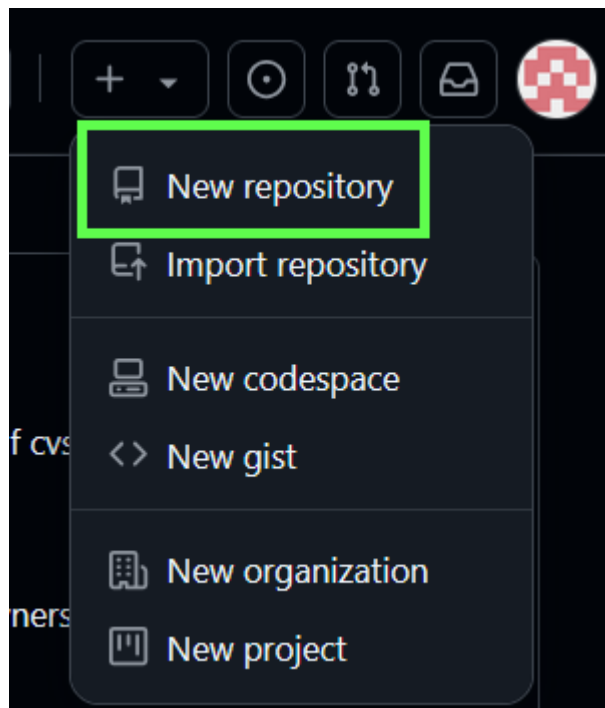
```
git push --force origin main
```

- ¿Qué es un repositorio privado en GitHub?
- Un **repositorio privado** en **GitHub** es un repositorio cuyo contenido solo es accesible y visible para los usuarios o colaboradores que tengan permisos explícitos para verlo o interactuar con él.
- ¿Cómo crear un repositorio privado en GitHub?

Para crear un repositorio en privado en Github debemos iniciar sesión en su plataforma y hacer click en el botón + en la esquina superior derecha.



A continuación, hacemos click en New Repository.



Luego completamos los campos que nos pide como:

Nombre del Repositorio: introducimos un nombre único para el repositorio.

Descripción (opcional): podemos agregar una breve descripción sobre el propósito o el contenido del repositorio.

Visibilidad : En este punto seleccionamos Private

Opciones de Inicialización:

- podemos marcar la casilla para crear un archivo **README** (útil para documentar el proyecto).
- opcionalmente, podemos agregar un archivo **.gitignore** para excluir ciertos archivos del control de versiones.
- también se puede elegir una **licencia** para el proyecto.

Una vez completados estos campos, hacemos click en "**Create Repository**" para finalizar.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner *

FernandoGGomez

Repository name *

nuevo_repositorio

✓ nuevo_repositorio is available.

Great repository names are short and memorable. Need inspiration? How about [laughing-dollop](#) ?

Description (optional)

☐

Public

Anyone on the internet can see this repository. You choose who can commit.

☒

Private

You choose who can see and commit to this repository.

Initialize this repository with:

☒ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

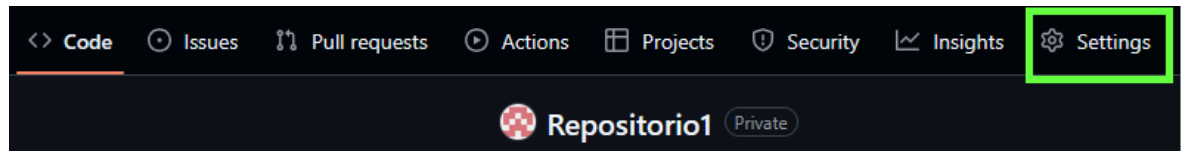
This will set `main` as the default branch. Change the default name in your [settings](#).

i You are creating a public repository in your personal account.

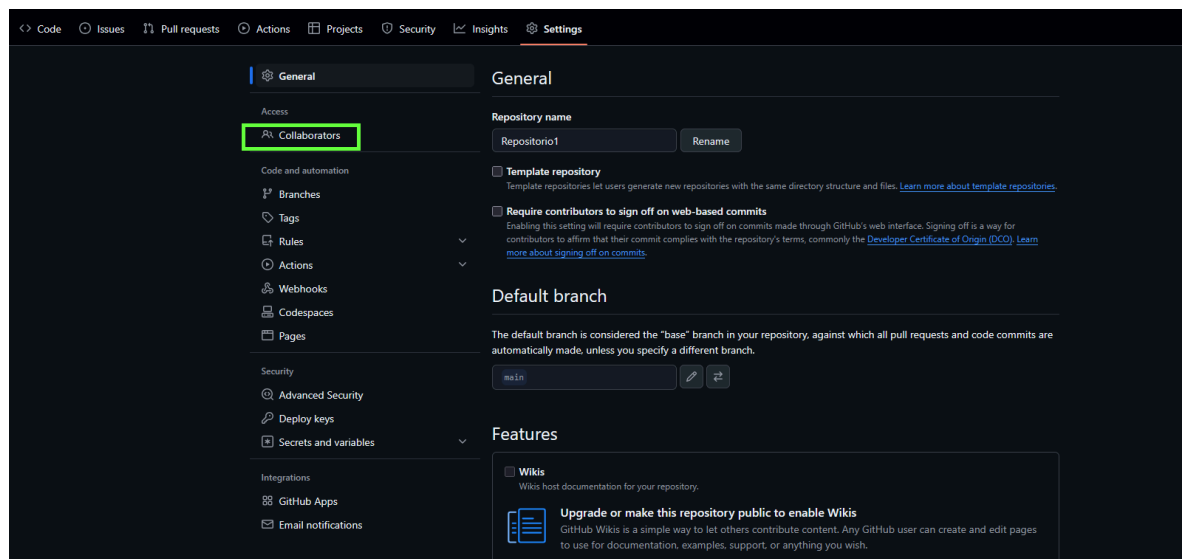
Create repository

- ¿Cómo invitar a alguien a un repositorio privado en GitHub?

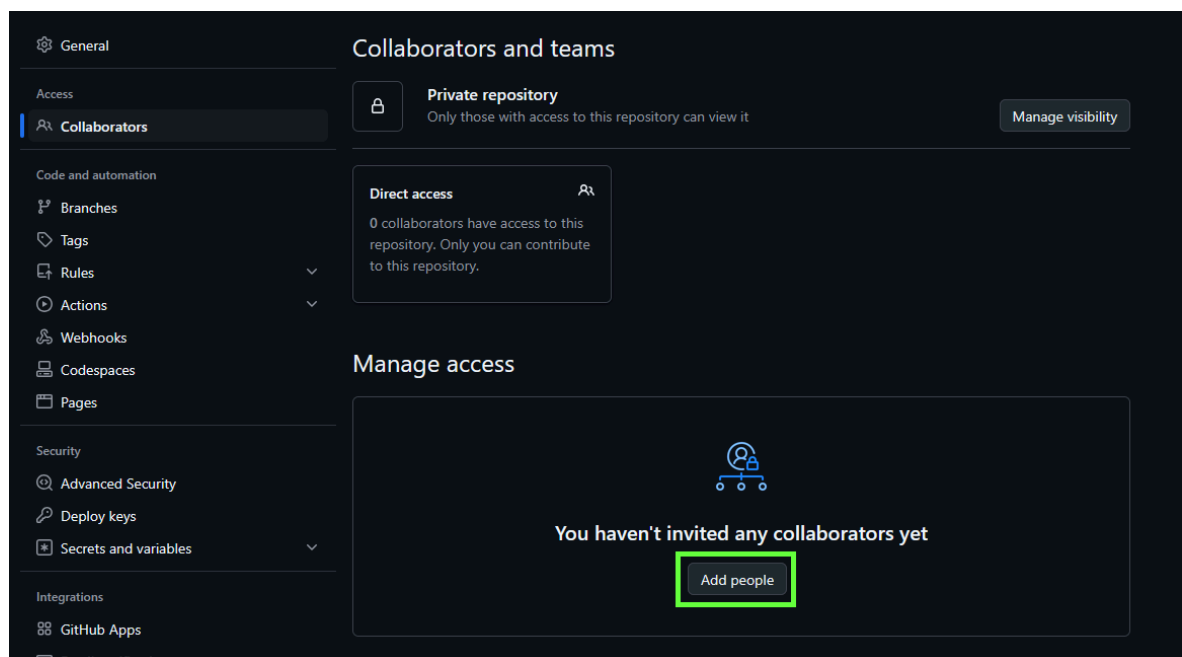
Accedemos a nuestra cuenta de Github, una vez posicionados en nuestro repositorio hacemos click en settings en la pestaña superior de nuestro repositorio.



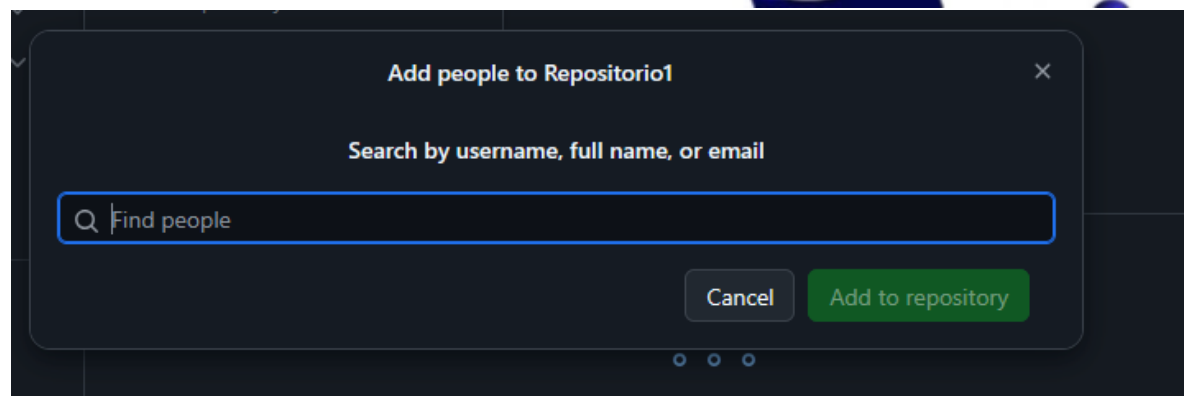
Después clickeamos a la izquierda donde dice “collaborators”.



A continuación hacemos click en el botón “Add people”.



Por último deberemos ingresar el nombre de usuario o el email de la persona que queremos invitar a nuestro repositorio y hacemos click en Add to repository.



- ¿Qué es un repositorio público en GitHub?

Un **repositorio público** en GitHub es un espacio de almacenamiento en línea donde podemos guardar, gestionar y compartir nuestro código con otros de manera abierta. Cualquier persona con acceso a Internet puede ver el contenido del repositorio, pero solo los colaboradores o dueños del repositorio pueden modificarlo.

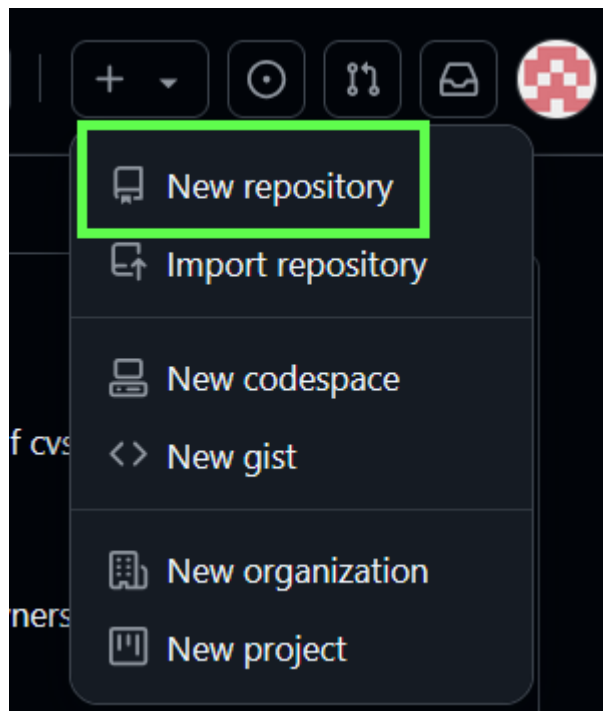
Al ser público, cualquier usuario puede:

1. **Ver el código:** Cualquier persona puede acceder y revisar el código que has subido.
 2. **Clonar el repositorio:** Los usuarios pueden hacer una copia del repositorio en su propio entorno local y trabajar en él.
 3. **Hacer "fork" del repositorio:** Los usuarios pueden crear una copia del repositorio bajo su cuenta, hacer cambios y proponer esos cambios a través de un *pull request*.
 4. **Contribuir:** *Otros usuarios pueden contribuir al proyecto enviando pull requests (solicitudes de incorporación de cambios) si les damos permisos para hacerlo.*
- ¿Cómo crear un repositorio público en GitHub?

Para crear un repositorio en privado en Github debemos iniciar sesión en su plataforma y hacer click en el botón + en la esquina superior derecha.



A continuación, hacemos click en New Repository.



Luego completamos los campos que nos pide como:

Nombre del Repositorio: introducimos un nombre único para el repositorio.

Descripción (opcional): podemos agregar una breve descripción sobre el propósito o el contenido del repositorio.

Visibilidad : En este punto seleccionamos Public

Opciones de Inicialización:

- podemos marcar la casilla para crear un archivo **README** (útil para documentar el proyecto).
- opcionalmente, podemos agregar un archivo **.gitignore** para excluir ciertos archivos del control de versiones.
- también se puede elegir una **licencia** para el proyecto.


Una vez completados estos campos, hacemos click en "**Create Repository**" para finalizar.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

 FernandoGGomez ▾

Repository name *

nuevo_repositorio

✓ nuevo_repositorio is available.

Great repository names are short and memorable. Need inspiration? How about [laughing-dollop](#) ?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

☒ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **None** ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

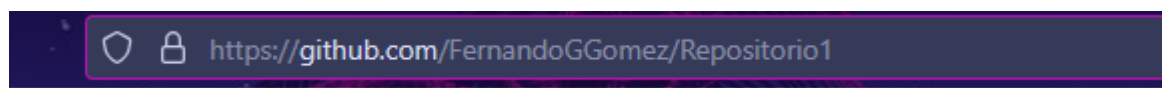
This will set  **main** as the default branch. Change the default name in your [settings](#).

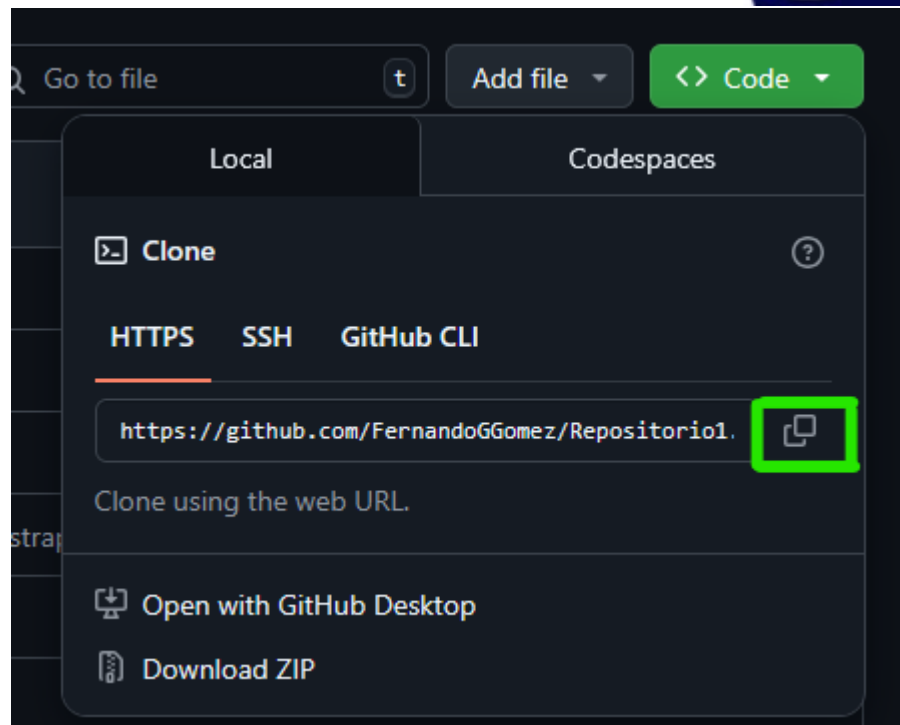
 You are creating a public repository in your personal account.

Create repository

- ¿Cómo compartir un repositorio público en GitHub?

Para compartir un repositorio público en GitHub, basta con copiar la URL que aparece en la parte superior de nuestro navegador cuando estamos en nuestro repositorio, o tocando en el botón que dice code en la esquina superior derecha del repositorio, nos aparecerá la misma URL. Una vez que la copiamos solo queda compartirla por el medio que queramos.





2) Realizar la siguiente actividad:

- Crear un repositorio.
 - Dale un nombre al repositorio.
 - Elije el repositorio sea público.
 - Inicializa el repositorio con un archivo.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk (*).


Owner * FernandoGGomez / Repository name * Nuevo_repositorio
✔ Nuevo_repositorio is available.

Great repository names are short and memorable. Need inspiration? How about [bookish-pancake](#) ?

Description (optional)

TP2 Git y Github. Actividad 2

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

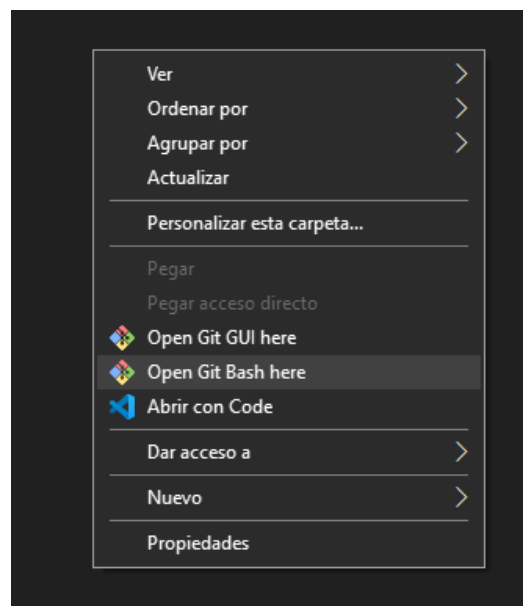
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository

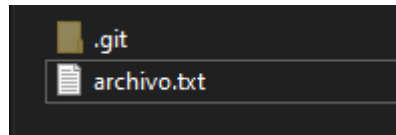
En una carpeta vacía en mi PC abro una terminal de Git:



Inicializo un repositorio con el comando git init:

```
$ git init
```

Creo un archivo en la carpeta del repositorio:



Agrego el archivo al staging area con git add .:

```
$ git add .
```

Ejecuto un commit para guardar los cambios:

```
$ git commit -m "Se creo un archivo de texto"
[main (root-commit) 13426cb] Se creo un archivo de texto
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 archivo.txt
```

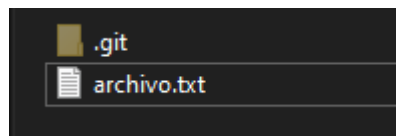
Ahora vinculo mi repositorio local con el remoto con el siguiente comando:

```
$ git remote add origin https://github.com/FernandoGGomez/Nuevo_repositorio.git
```

Lo siguiente que hago es subir los archivos al repositorio remoto alojado en Github:

```
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 239 bytes | 239.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/FernandoGGomez/Nuevo_repositorio.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

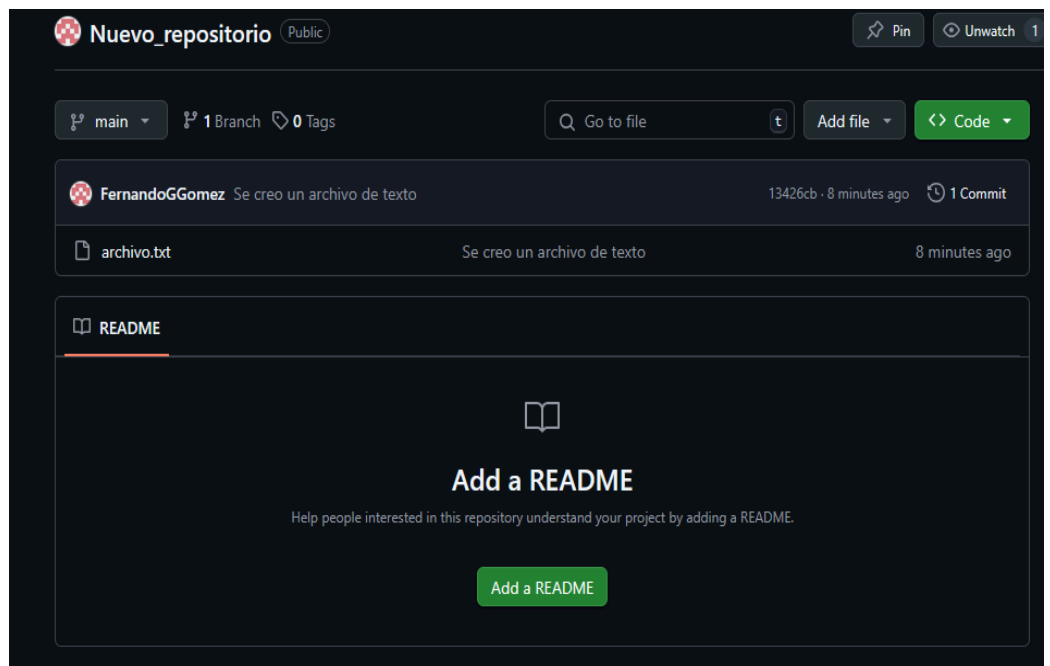
- Agregando un Archivo
 - Crea un archivo simple, por ejemplo, "mi-archivo.txt".
 - Realiza los comandos git add . y git commit -m "Agregando mi-archivo.txt" en la línea de comandos.
 - Sube los cambios al repositorio en GitHub con git push origin main (o el nombre de la rama correspondiente).



```
$ git add .
```

```
$ git commit -m "Se creo un archivo de texto"
[main (root-commit) 13426cb] Se creo un archivo de texto
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 archivo.txt
```

```
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 239 bytes | 239.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/FernandoGGomez/Nuevo_repositorio.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```



- Creando Branchs
 - Crear una Branch
 - Realizar cambios o agregar un archivo
 - Subir la Branch




Creo la rama:

```
$ git branch rama1
```

Me posiciono sobre la nueva rama:

```
$ git checkout rama1  
Switched to branch 'rama1'
```

Creo un archivo en la rama1:

	.git	31/3/2025 16:13	Carpeta de archivos	
	archivo.txt	31/3/2025 16:01	Documento de te...	0 KB
	nuevo_archivo_rama1.txt	31/3/2025 16:14	Documento de te...	0 KB

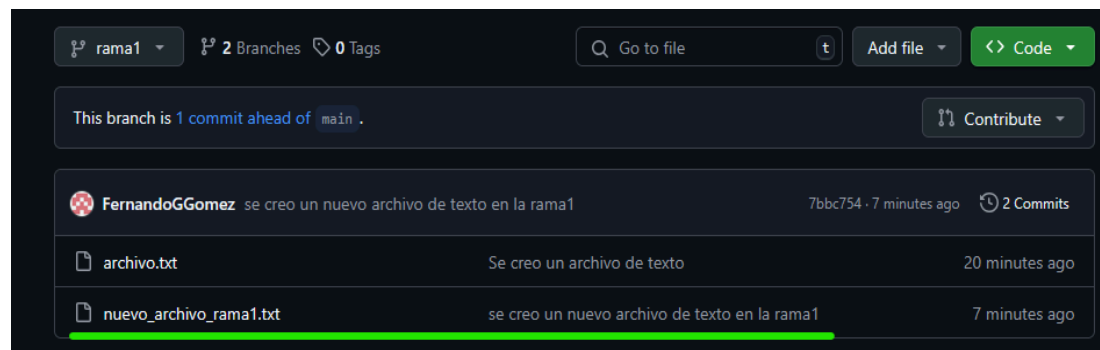
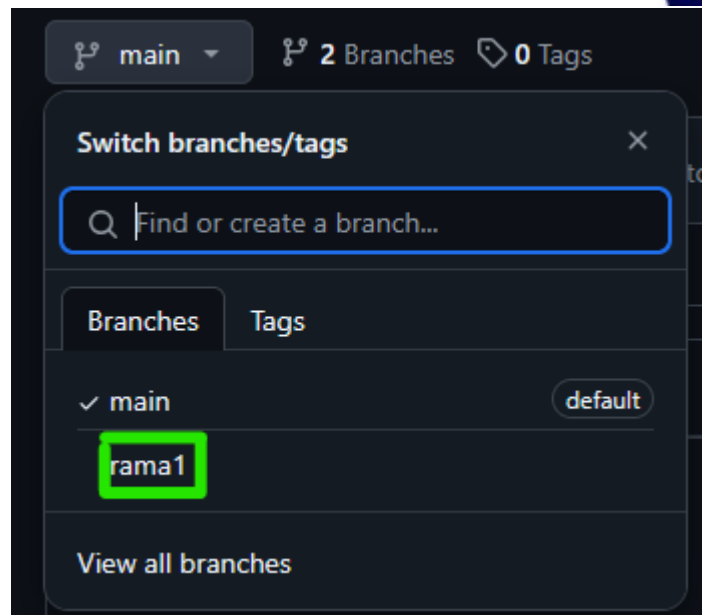
Después:

```
$ git add .
```

```
$ git commit -m "se creo un nuevo archivo de texto en la rama1"  
[rama1 7bbc754] se creo un nuevo archivo de texto en la rama1  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 nuevo_archivo_rama1.txt
```

Subo la rama al repositorio remoto:

```
$ git push -u origin rama1  
Enumerating objects: 3, done.  
Counting objects: 100% (3/3), done.  
Delta compression using up to 12 threads  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (2/2), 291 bytes | 291.00 KiB/s, done.  
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)  
remote:  
remote: Create a pull request for 'rama1' on GitHub by visiting:  
remote:   https://github.com/FernandoGGomez/Nuevo_repositorio/pull/new/rama1  
remote:  
To https://github.com/FernandoGGomez/Nuevo_repositorio.git  
* [new branch]   rama1 -> rama1  
branch 'rama1' set up to track 'origin/rama1'.
```



3) Realizar la siguiente actividad:

Paso 1: Crear un repositorio en GitHub

- Ve a GitHub e inicia sesión en tu cuenta.
- Haz clic en el botón "New" o "Create repository" para crear un nuevo repositorio.
- Asigna un nombre al repositorio, por ejemplo, conflict-exercise.
- Opcionalmente, añade una descripción.
- Marca la opción "Initialize this repository with a README".
- Haz clic en "Create repository".

Paso 2: Clonar el repositorio a tu máquina local

- Copia la URL del repositorio (usualmente algo como <https://github.com/tuusuario/conflict-exercise.git>).
- Abre la terminal o línea de comandos en tu máquina.
- Clona el repositorio usando el comando:

```
git clone https://github.com/tuusuario/conflict-exercise.git
```

- Entra en el directorio del repositorio:

`cd conflict-exercise`

Paso 3: Crear una nueva rama y editar un archivo

- Crea una nueva rama llamada feature-branch:

`git checkout -b feature-branch`

- Abre el archivo README.md en un editor de texto y añade una línea nueva, por ejemplo:

Este es un cambio en la feature branch.

- Guarda los cambios y haz un commit:

`git add README.md`

`git commit -m "Added a line in feature-branch"`

Paso 4: Volver a la rama principal y editar el mismo archivo

Cambia de vuelta a la rama principal (main):

`git checkout main`

- Edita el archivo README.md de nuevo, añadiendo una línea diferente:

Este es un cambio en la main branch.

- Guarda los cambios y haz un commit:

`git add README.md`

`git commit -m "Added a line in main branch"`

Paso 5: Hacer un merge y generar un conflicto

- Intenta hacer un merge de la feature-branch en la rama main:

`git merge feature-branch`

- Se generará un conflicto porque ambos cambios afectan la misma línea del archivo README.md.

Paso 6: Resolver el conflicto

- Abre el archivo README.md en tu editor de texto. Verás algo similar a esto:

`<<<<<<< HEAD`

Este es un cambio en la main branch.

`=====`

Este es un cambio en la feature branch.

`>>>>>>> feature-branch`

- Decide cómo resolver el conflicto. Puedes mantener ambos cambios, elegir uno de ellos, o fusionar los contenidos de alguna manera.
- Edita el archivo para resolver el conflicto y guarda los cambios (Se debe borrar lo marcado en verde en el archivo donde estes solucionando el conflicto. Y se debe borrar la parte del texto que no se quiera dejar).
- Añade el archivo resuelto y completa el merge:

```
git add README.md
```

```
git commit -m "Resolved merge conflict"
```

Paso 7: Subir los cambios a GitHub

- Sube los cambios de la rama main al repositorio remoto en GitHub:

```
git push origin main
```

- También sube la feature-branch si deseas:

```
git push origin feature-branch
```

Paso 8: Verificar en GitHub


- Ve a tu repositorio en GitHub y revisa el archivo README.md para confirmar que los cambios se han subido correctamente.
- Puedes revisar el historial de commits para ver el conflicto y su resolución.

Paso 1:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk ().*


Owner *  FernandoGGomez / **Repository name ***

✔ conflict-exercise is available.

Great repository names are short and memorable. Need inspiration? How about [curly-waffle](#) ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:


☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)


Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

[Create repository](#)

Paso 2:

```
git clone https://github.com/FernandoGGomez/conflict-exercise.git
```

```
Cloning into 'conflict-exercise'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

```
PS C:\Users\Fernando\Desktop\Actividad3> cd conflict-exercise
```

Paso 3:

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git checkout -b feature-branch  
Switched to a new branch 'feature-branch'
```

```
1 # conflict-exercise  
2 TP2 Git y Github. Actividad 3  
3 Este es un cambio en la feature branch
```

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git add README.md
```

The most similar command is

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git commit -m "Added a line in feature-branch"  
[feature-branch 5ac8cd6] Added a line in feature-branch  
1 file changed, 1 insertion(+)
```

Paso 4:

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git checkout main  
Switched to branch 'main'  
Your branch is up to date with 'origin/main'.
```

```
1 # conflict-exercise  
2 TP2 Git y Github. Actividad 3  
3 Este es un cambio en la main branch
```

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git add README.md
```

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git commit -m "Added a line in main branch"  
[main a737708] Added a line in main branch  
1 file changed, 1 insertion(+)
```

Paso 5:

```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git merge feature-branch  
Auto-merging README.md  
CONFLICT (content): Merge conflict in README.md  
Automatic merge failed; fix conflicts and then commit the result.
```

```
1 # conflict-exercise
2 TP2 Git y Github. Actividad 3
  Aceptar cambio actual | Aceptar cambio entrante | Aceptar ambos cambios | Comparar cambios
3 <<<<<< HEAD (Cambio actual)
4 Este es un cambio en la main branch
5 =====
6 Este es un cambio en la feature branch
7 >>>>>> feature-branch (Cambio entrante)
8
```

Paso 6:

```
1 # conflict-exercise
2 TP2 Git y Github. Actividad 3
3 Este es un cambio en la main branch
4 Este es un cambio en la feature branch
```


```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git commit -m "Resolved merge conflict"
[main 7efb663] Resolved merge conflict
```



Paso 7:




```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git push origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 788 bytes | 394.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/FernandoGGomez/conflict-exercise.git
 29c9f07..7efb663  main -> main
```




```
PS C:\Users\Fernando\Desktop\Actividad3\conflict-exercise> git push origin feature-branch
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:   https://github.com/FernandoGGomez/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/FernandoGGomez/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch
```


Paso 8:


 **conflict-exercise** Public



 Pin  Unwatch 1

 main  2 Branches  0 Tags

  Add file  Code

 **FernandoGGomez** Resolved merge conflict 7efb663 · 6 minutes ago 🕒 4 Commits

 README.md Resolved merge conflict 6 minutes ago

 **README** 

conflict-exercise

TP2 Git y Github. Actividad 3 Este es un cambio en la main branch Este es un cambio en la feature branch

 main

 All users  All time

 Commits on Mar 31, 2025

Resolved merge conflict
 FernandoGGomez committed 7 minutes ago 7efb663  

Added a line in main branch
 FernandoGGomez committed 39 minutes ago a737788  

Added a line in feature-branch
 FernandoGGomez committed 45 minutes ago 5ac8cd6  

Initial commit
 FernandoGGomez authored 1 hour ago Verified 29c9f87  