



## Práctico 2: Git y GitHub

### Programacion I

MAURO ZAVATTI

#### *Actividades*

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

#### **¿Qué es GitHub?**

Es una plataforma que ofrece alojamiento de repositorios de control de versiones, que permite a los desarrolladores almacenar y gestionar sus proyectos de software. Es una herramienta gratuita y de código abierto que permite el trabajo en equipo en proyectos, el intercambio de código y el trabajo conjunto de forma eficiente.

#### **¿Cómo crear un repositorio en GitHub?**

- ☐ Primero ir a GitHub e iniciar sesión.
- ☐ Una vez dentro, hacer clic en el ícono "+" en la esquina superior derecha.
- ☐ Seleccionar **"New repository"** (Nuevo repositorio).
- ☐ **Nombre del repositorio:** Escribir un nombre único y descriptivo.
- ☐ **Descripción:** Explica brevemente el propósito del repositorio.
- ☐ **Visibilidad:**
  - **Público:** Cualquiera puede verlo.

- **Privado:** Solo tú y las personas que invites pueden verlo.

☐ **Opcionales:**

- Se puede inicializar el repositorio con un **README.md** (recomendado).
- Se puede agregar un archivo **.gitignore** para ignorar archivos específicos.
- Se puede seleccionar una licencia para definir el uso del código.

☐ Hacer clic en "**Create repository**".

☐ Abrir la terminal (Git Bash, CMD o PowerShell).

☐ Navega a la carpeta de tu proyecto con:

```
$ cd /ruta/del/proyecto
```

Inicializa Git:

```
$ git init
```

Conéctalo a GitHub:

```
$ git remote add origin URL_DEL_REPO
```

Agrega y sube los archivos:

```
$ git add .
```

```
$ git commit -m "Primer commit"
```

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

### **¿Cómo crear una rama en Git?**

Para crear una nueva rama, se usará el comando git branch:

```
$ git branch nuevaRama
```

Git sabe en qué rama estás mediante un apuntador especial denominado HEAD. Que estemos creando una nueva rama con git branch no significa que estemos en dicha rama, estaríamos en la rama master.

¿Cómo cambiar a una rama en Git?

Para saltar de una rama a otra, tienes que utilizar el comando git checkout:

```
$ git checkout nuevaRama
```

Esto si mueve el apuntador HEAD a la rama nuevaRama.

### **¿Cómo fusionar ramas en Git?**

Paso 1: Cambiar a la rama principal (ej. main o master)

```
$ git checkout main
```

Paso 2: Traer los últimos cambios

```
$ git pull origin main
```

Paso 3: Fusionar la otra rama

```
$ git merge nombre_de_la_rama
```

Si no hay conflictos, Git fusionará automáticamente y mostrará un mensaje de confirmación.

Paso 4: Resolver conflictos (si los hay)

Si Git detecta cambios en los mismos archivos en ambas ramas, aparecerán conflictos.

1. Git marcará los archivos en conflicto.
2. Abre esos archivos y edítalos manualmente para elegir qué cambios conservar.
3. Una vez resueltos, agrega los cambios:

```
$ git add .
```

```
$ git commit -m "Resolviendo conflictos de fusión"
```

## Paso 5: Subir los cambios a GitHub

`git push origin main`

¿Cómo crear un commit en Git?

Crear un commit en Git es el proceso mediante el cual se guardan los cambios realizados en el repositorio. Un commit captura el estado actual del código en un momento dado y lo almacena en el historial del proyecto.

Pasos para hacer un commit:

Primero realiza los cambios en los archivos de tu proyecto que desees guardar en el commit.

Luego debes agregar los cambios al área de preparación. Esto se hace con el comando `git add`. Puedes agregar archivos específicos o todos los archivos modificados:

`$ git add archivo1` (para agregar el archivo "archivo1") o `$git add .` (para agregar todos los archivos)

Una vez que los cambios están en el área de preparación, puede hacer el commit con el comando `git commit`. Debes proporcionar un mensaje de commit que describa los cambios realizados:

`$ git commit -m "Mensaje de los cambios"`

Esto guarda el estado actual del código en el historial del repositorio con el mensaje correspondiente.

## **¿Cómo enviar un commit a GitHub?**

Paso 1: Asegurar que Git está configurado

`git config --global user.name "Tu Nombre"`

`git config --global user.email tuemail@example.com`

Paso 2: Ir al directorio del proyecto

`cd /ruta/del/proyecto`

Paso 3: Agregar los archivos al área de preparación

`git add .`

Si solo quieres agregar un archivo específico:

```
git add nombre_del_archivo
```

Paso 4: Crear un commit con un mensaje descriptivo

```
git commit -m "Descripción breve del cambio"
```

Paso 5: Enviar los cambios a GitHub (los empujamos con push)

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

Para commits posteriores, simplemente usa:

```
git push
```

### **¿Qué es un repositorio remoto?**

Un repositorio remoto es un espacio de almacenamiento en la nube donde se guarda el código de un proyecto. A diferencia de un repositorio local, que está en tu computadora, el remoto está en plataformas como GitHub, y permite que varias personas colaboren en el mismo proyecto desde distintos lugares.

Cuando trabajas con un repositorio remoto, puedes enviar tus cambios para que queden guardados en la nube, descargar actualizaciones realizadas por otros y sincronizar tu código con el equipo. Esto facilita la colaboración, el control de versiones y la seguridad del código, ya que siempre hay una copia respaldada en línea.

Por ejemplo, si estás desarrollando un sitio web en tu computadora, puedes subirlo a un repositorio remoto para que otros miembros del equipo puedan verlo, hacer modificaciones y trabajar en conjunto sin necesidad de compartir archivos manualmente.

### **¿Cómo agregar un repositorio remoto a Git?**

Utiliza el comando `git remote add` para vincular tu repositorio local con un repositorio remoto y asignar un nombre a ese remoto:

```
$ git remote add [nombre] [url]
```

Ponerle un nombre te ayuda a referenciarlo de manera más fácil en lugar de usar la URL completa. Puedes usar el nombre en la línea de comandos.

Para asegurarte de que el remoto se ha agregado correctamente y verificar el nombre que le has dado, usa el comando:

```
$ git remote -v
```

Esto mostrará una lista de todos los remotos configurados con sus URLs:

```
[nombre] [url]
```

Para traer toda la información del repositorio remoto que aún no tienes en tu repositorio local, usa el comando `git fetch` seguido del nombre del remoto:

```
$ git fetch [nombre]
```

Este comando descarga todos los cambios del repositorio remoto asociado con el nombre, pero no fusiona esos cambios con tu rama actual. Es útil para ver qué cambios están disponibles en el remoto.

### **¿Cómo empujar cambios a un repositorio remoto?**

Antes de empujar tus cambios, es una buena práctica obtener los últimos cambios del repositorio remoto para evitar conflictos:

```
$ git pull origin nombre_de_la_rama
```

Empuja tus cambios al repositorio remoto:

```
$ git push origin nombre_de_la_rama
```

### **¿Cómo tirar de cambios de un repositorio remoto?**

Para tirar los cambios del repositorio remoto Usa el comando `git pull` para descargar y fusionar los cambios del repositorio remoto con tu rama local:

```
$ git pull origin nombre_de_la_rama
```

Por ejemplo, si estás trabajando en la rama `main`:

```
$ git pull origin main
```

## **¿Qué es un fork de repositorio?**

Un **fork** de un repositorio es una copia independiente de un proyecto que se crea en tu propia cuenta de GitHub. Sirve para hacer modificaciones sin afectar el repositorio original.

¿Para qué se usa un fork?

1. Colaborar en proyectos abiertos: Puedes hacer cambios en un repositorio sin permiso directo del propietario y luego proponerlos mediante un Pull Request.
2. Experimentar sin riesgo: Modificar código sin afectar el proyecto original.
3. Crear una versión personalizada: Si quieres usar un proyecto como base para tu propia versión.

Ejemplo práctico

Imagina que encuentras un proyecto interesante en GitHub, pero no tienes permisos para modificarlo directamente. Puedes hacer un **fork**, trabajar en tu propia copia y luego sugerir tus cambios al autor original.

## **¿Cómo crear un fork de un repositorio?**

Para crear un fork de un repositorio en GitHub, primero debes ir al repositorio que deseas copiar. En la página del repositorio, encontrarás un botón en la parte superior derecha que dice "**Fork**". Al hacer clic en él, GitHub creará automáticamente una copia del repositorio en tu propia cuenta. Esta copia es completamente independiente del repositorio original, por lo que podrás modificarla a tu gusto sin afectar el proyecto original.

Una vez que has hecho el fork, puedes clonar el repositorio a tu computadora para trabajar de forma local. Para hacerlo, vas a tu perfil, encuentras el repositorio fork y copias la URL que aparece en el botón "**Code**". Luego, en tu terminal, usas esa URL para clonar el repositorio en tu máquina.

Cuando hayas realizado cambios en tu fork y quieras contribuir al repositorio original, puedes hacer un **Pull Request**. Esto permite que el dueño del repositorio original vea tus cambios y decida si los fusiona con su proyecto.

## **¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?**

Para hacer el pull request nos dirigiremos a la solapa de Pull requests allí daremos click en new pull request, veremos una ventana a modo de resumen en donde se reflejarán los cambios que hemos hecho nosotros en comparación al repositorio original (el código original, mejor dicho). Daremos click en Create pull request donde veremos el asunto (colocamos algún mensaje global) y más abajo tenemos suficiente lugar para poder explayarnos en mencionar el porque ese cambio que hemos realizado nosotros, sería considerado como algo que a el repositorio original le vendrían bien agregarlo.

## **¿Cómo aceptar una solicitud de extracción?**

El autor del repositorio verá en sus pull requests el mensaje que le hemos enviado, para que lo pueda observar y si lo considera realizar el cambio pertinente (además de poder responderle al usuario que le ha propuesto ese cambio). Lo bueno de todo esto es que si el usuario original considera que esta modificación es buena y no genera conflictos con la rama maestra de su repositorio local remoto, puede clicar en Merge pull request y de esta manera sumará a su repositorio los cambios que hizo un usuario (en modo de ayuda).

## **¿Qué es un etiqueta en Git?**

En **Git**, una **etiqueta** (o **tag**) es una referencia o marcador que apunta a un commit específico en el historial del repositorio. Las etiquetas se usan comúnmente para marcar puntos importantes en el desarrollo, como versiones de lanzamiento o hitos clave, para facilitar su identificación.

## **¿Cómo crear una etiqueta en Git?**

Crear una **etiqueta** en Git es un proceso sencillo. Puedes hacerlo de dos maneras: **de forma ligera** (sin detalles adicionales) o **anotada** (con información extra, como nombre de autor, fecha, etc.).



Una etiqueta ligera es simplemente un puntero a un commit específico, sin información adicional. Es la forma más básica de etiqueta.

```
$ git tag nombre_de_etiqueta
```

Si deseas crear una etiqueta en un commit específico, puedes agregar el hash del commit:

```
$ git tag nombre_de_etiqueta <commit_hash>
```

Una etiqueta anotada es más completa, ya que incluye el nombre del autor, la fecha y un mensaje. Además, se almacena como un objeto completo dentro de Git, lo que la hace más recomendable para marcar versiones importantes.

Para crear una etiqueta anotada:

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

Por ejemplo:

```
$ git tag -a v1.0 -m "Versión 1.0 - Lanzamiento inicial"
```

También puedes etiquetar un commit específico añadiendo el hash del commit al final:

```
$ git tag -a v1.0 <commit_hash> -m "Versión 1.0 - Lanzamiento inicial"
```

### **¿Cómo enviar una etiqueta a GitHub?**

Primero se puede listar las etiquetas. Para listar todas las etiquetas que has creado en tu repositorio, simplemente ejecuta:

```
$ git tag
```

Después de crear una etiqueta localmente, si deseas que esté disponible en el repositorio remoto (por ejemplo, en GitHub), debes enviarla usando:

```
$ git push origin nombre_de_etiqueta
```

Si deseas enviar todas las etiquetas a la vez, puedes usar:

```
git push origin -tags
```

## **¿Qué es un historial de Git?**

El historial de Git es una secuencia de todos los cambios realizados en un repositorio de Git. Cada cambio en el repositorio se guarda como un commit, y cada commit contiene información sobre el estado del proyecto en un momento específico, incluyendo:

Identificador del commit

Autor

Fecha de realización

Mensaje enviado

## **¿Cómo ver el historial de Git?**

Para ver el historial de cambios en un repositorio Git, puedes usar el comando **git log**, que muestra una lista de los commits realizados en el proyecto, junto con información sobre cada uno, como el autor, la fecha y el mensaje del commit.

Esto te mostrará una lista de commits en orden cronológico inverso (el más reciente primero). Cada commit incluirá:

- El hash del commit (un identificador único).
- El autor y su correo electrónico.
- La fecha del commit.
- El mensaje del commit.

## **¿Cómo buscar en el historial de Git?**

Puedes usar opciones adicionales para buscar con git log:

```
$ git log [nombre_commit]
```

```
$ git log [nombre_rama]
```

```
$ git log [nombre_archivo]
```

Si quieres ver los commits de un período de tiempo específico, puedes usar las opciones de fecha:

Desde una fecha específica

```
$ git log --since="2025-01-01"
```

Hasta una fecha específica

```
$ git log --until="2025-03-01"
```

### **¿Cómo borrar el historial de Git?**

El comando git reset se utiliza sobre todo para deshacer las cosas. Se mueve alrededor del puntero HEAD y opcionalmente cambia el index o área de ensayo y también puede cambiar opcionalmente el directorio de trabajo si se utiliza - hard. Esta última opción hace posible que este comando pueda perder tu trabajo si se usa incorrectamente, por lo que asegúrese de entenderlo antes de usarlo.

Existen distintas formas de utilizarlo:

- git reset -> Quita del stage todos los archivos y carpetas del proyecto.
- git reset nombreArchivo -> Quita del stage el archivo indicado.
- git reset nombreCarpeta/ -> Quita del stage todos los archivos de esa carpeta.
- git reset nombreCarpeta/nombreArchivo -> Quita ese archivo del stage (que a la vez está dentro de una carpeta).
- git reset nombreCarpeta/\*.extensión -> Quita todos los archivos que cumplan con la condición indicada previamente dentro de esa carpeta del stage.

### **¿Qué es un repositorio privado en GitHub?**

Un repositorio privado en GitHub es un tipo de repositorio en el que el contenido solo es accesible para usuarios específicos que han sido autorizados. A diferencia de los repositorios públicos, donde cualquier persona puede ver y clonar el contenido, un repositorio privado limita el acceso a los colaboradores que tú elijas. Esto es útil para proyectos que contienen información sensible o que aún están en desarrollo y no deseas que estén disponibles públicamente.

## **¿Cómo crear un repositorio privado en GitHub?**

☐ Inicia sesión en tu cuenta de GitHub:  
Ve a GitHub y accede con tu cuenta.

☐ Crear un nuevo repositorio:  
En la parte superior derecha de la página de inicio, haz clic en el ícono "+" y selecciona "New repository" (Nuevo repositorio).

☐ Configurar el repositorio:

- Nombre del repositorio: Elige un nombre único para tu repositorio.
- Descripción (opcional): Puedes añadir una breve descripción sobre el propósito de tu repositorio.

☐ Elegir la visibilidad del repositorio:  
Debajo de la descripción, encontrarás la opción de "Visibility" (Visibilidad):

- Selecciona "Private" para que solo las personas que invites puedan ver el repositorio y acceder a él.

☐ Opciones adicionales:

- Puedes inicializar el repositorio con un README.md, un .gitignore (para ignorar ciertos archivos) y elegir una licencia (opcional).

☐ Crear el repositorio:  
Haz clic en el botón "Create repository" (Crear repositorio) para finalizar la creación del repositorio privado.

## **¿Cómo invitar a alguien a un repositorio privado en GitHub?**

Accede al repositorio, haz clic en la pestaña "Settings" del repositorio. Está en la parte superior del repositorio, junto a las pestañas como "Code" y "Issues".

Selecciona "Collaborators" en el menú de la izquierda. Esto te llevará a la página donde puedes administrar colaboradores.

En la sección "Collaborators", haz clic en el botón "Add people" e ingresa el nombre de usuario de GitHub de la persona que deseas invitar. Selecciona el nivel de acceso que deseas otorgar: Read, Triage, Write, Maintain, o Admin. Haz clic en el botón "Add" para enviar la invitación.

## **¿Qué es un repositorio público en GitHub?**

Un repositorio público en GitHub es un espacio donde el código fuente y otros archivos de un proyecto son accesibles para cualquier persona en internet. Esto significa que cualquier usuario puede ver, clonar, bifurcar (hacer un *fork*) y contribuir al proyecto si el propietario del repositorio lo permite.

## **¿Cómo crear un repositorio público en GitHub?**

Pasos:

Inicia sesión en GitHub

Ingresa a la página de creación de repositorios:

En la esquina superior derecha de la página principal, debes hacer clic en el botón “+” y seleccionar “New Repository” o hacer clic en “New”.

Completar la información del repositorio (nombre del repositorio, descripción)

Seleccionar la configuración de privacidad, esto asegura que el repositorio será público o privado, según elección.

Por último, seleccionar “Create repository”.

## **¿Cómo compartir un repositorio público en GitHub?**

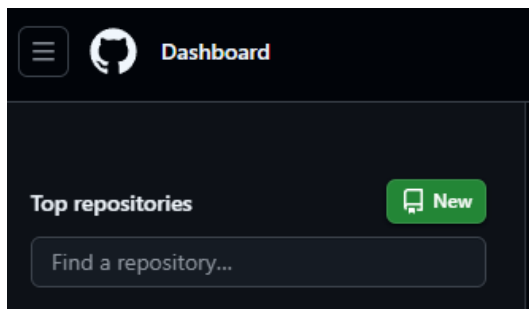
La forma más sencilla de compartir tu repositorio es proporcionar el enlace directo al mismo.

Accede a tu repositorio, copia la URL de tu repositorio que se encuentra en un cuadro de texto que dice “<> Code”:

Puedes copiar la URL directamente haciendo clic en el botón de copiar a la derecha de la URL.

2) Realizar la siguiente actividad:

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



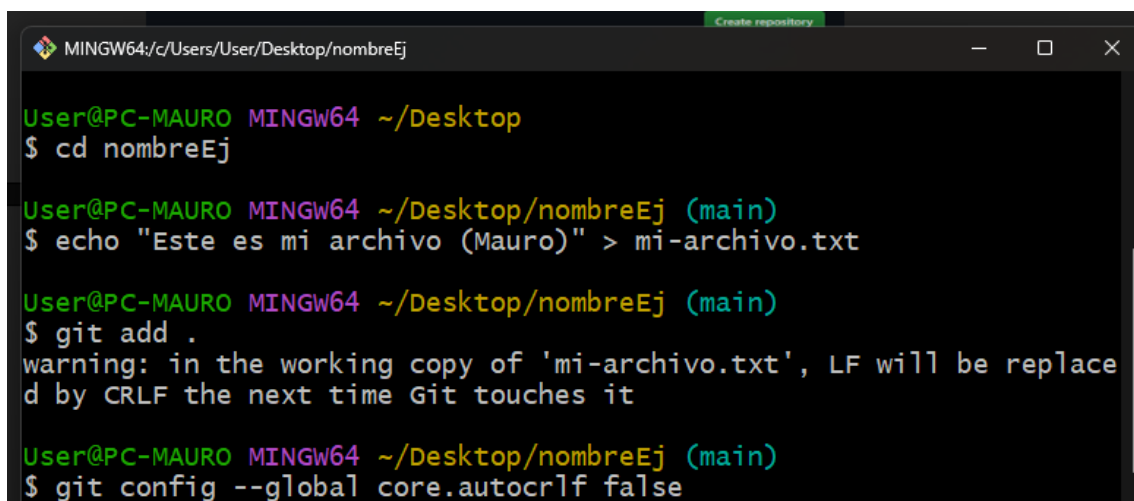
Dale un nombre al repositorio.

Elije el repositorio sea público.

Inicializa el repositorio con un archivo.

A screenshot of the 'Create a new repository' form on GitHub. The form is titled 'Create a new repository' and includes a subtitle: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, a note states: 'Required fields are marked with an asterisk (\*)'. The form has two main sections: 'Owner \*' with a dropdown menu showing 'maurozavatti' and 'Repository name \*' with a text input field containing 'nombreEj'. A green checkmark indicates 'nombreEj is available.'. Below these, a message says: 'Great repository names are short and memorable. Need inspiration? How about **cautious-palm-tree** ?'. The 'Description (optional)' field is a large text area. The 'Visibility' section has two radio buttons: 'Public' (selected) and 'Private'. The 'Initialize this repository with:' section has a checked checkbox for 'Add a README file'. Below this, there's a section for 'Add .gitignore' with a dropdown menu showing '.gitignore template: None'. The 'Choose a license' section has a dropdown menu showing 'License: None'. At the bottom, there's a note: 'This will set `main` as the default branch. Change the default name in your [settings](#).' and a green 'Create repository' button.

- **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).



```

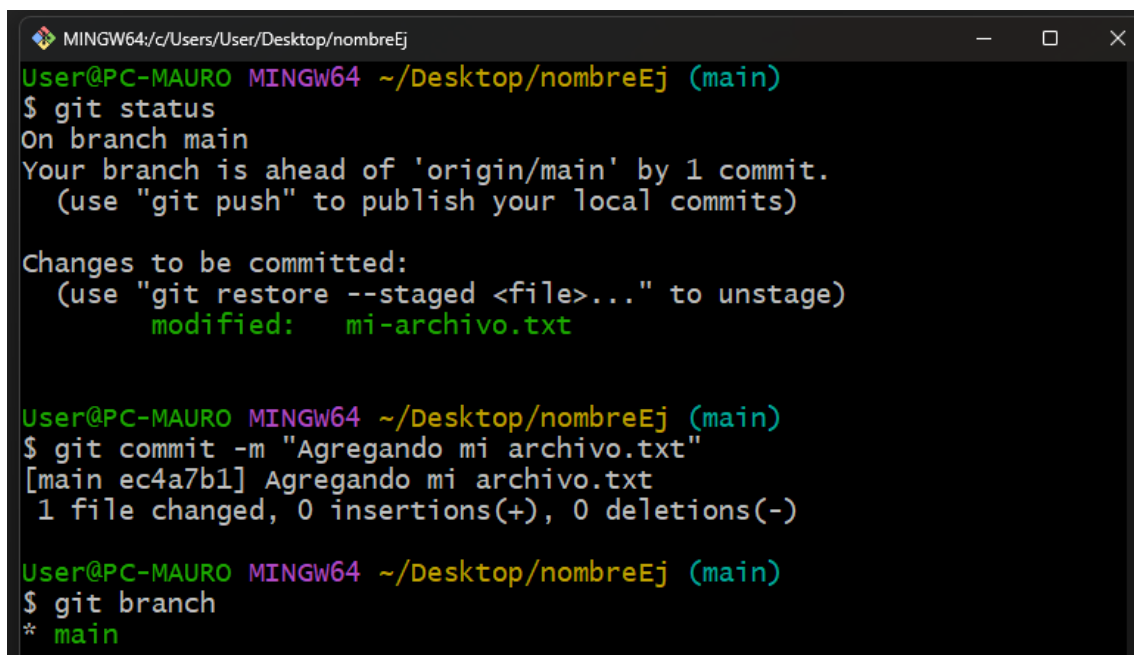
MINGW64/c/Users/User/Desktop/nombreEj
User@PC-MAURO MINGW64 ~/Desktop
$ cd nombreEj

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ echo "Este es mi archivo (Mauro)" > mi-archivo.txt

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git add .
warning: in the working copy of 'mi-archivo.txt', LF will be replaced by CRLF the next time Git touches it

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git config --global core.autocrlf false

```



```

MINGW64/c/Users/User/Desktop/nombreEj
User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   mi-archivo.txt

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git commit -m "Agregando mi archivo.txt"
[main ec4a7b1] Agregando mi archivo.txt
1 file changed, 0 insertions(+), 0 deletions(-)

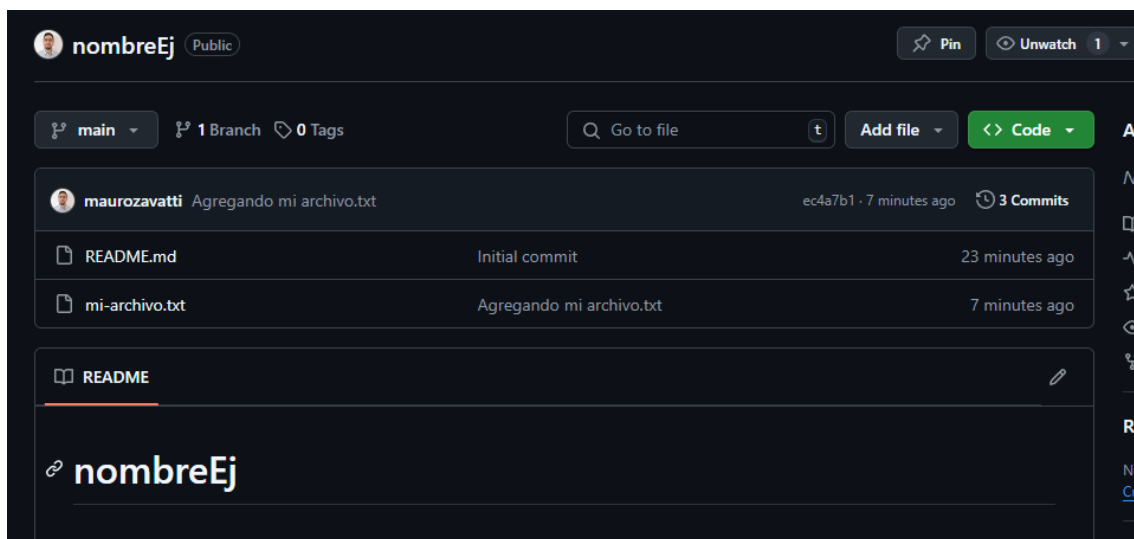
User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git branch
* main

```

```
MINGW64:/c/Users/User/Desktop/nombreEj
$ git commit -m "Agregando mi archivo.txt"
[main ec4a7b1] Agregando mi archivo.txt
1 file changed, 0 insertions(+), 0 deletions(-)

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git branch
* main

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git push origin main
info: please complete authentication in your browser...
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 622 bytes | 622.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/maurozavatti/nombreEj
582b1a8..ec4a7b1 main -> main
```



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



```
MINGW64/c/Users/User/Desktop/nombreEj
User@PC-MAURO MINGW64 ~/Desktop/nombreEj (main)
$ git checkout -b nuevaRama
Switched to a new branch 'nuevaRama'

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ git branch
  main
* nuevaRama

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ echo "Modificacion desde nuevaRama" > mi-archivo.txt

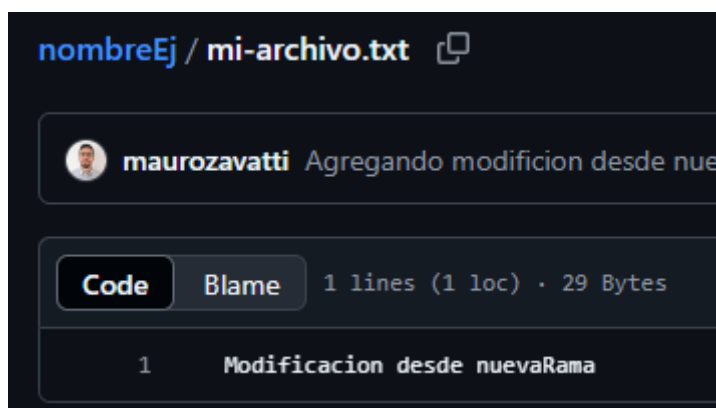
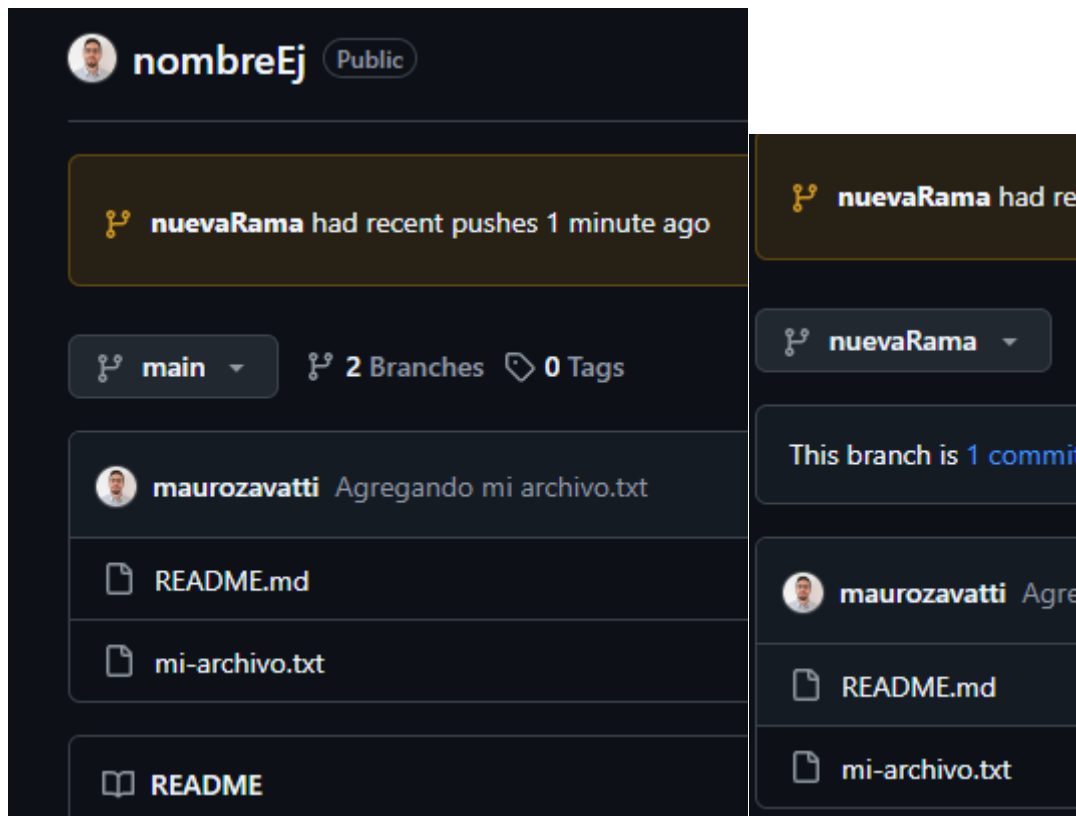
User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ git add .

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ git status
On branch nuevaRama
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   mi-archivo.txt
```

```
MINGW64/c/Users/User/Desktop/nombreEj
User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ git commit -m "Agregando modificacion desde nuevaRama"
[nuevaRama 0cdc217] Agregando modificacion desde nuevaRama
1 file changed, 1 insertion(+), 1 deletion(-)

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ git status
On branch nuevaRama
nothing to commit, working tree clean

User@PC-MAURO MINGW64 ~/Desktop/nombreEj (nuevaRama)
$ git push origin nuevaRama
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 336 bytes | 336.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'nuevaRama' on GitHub by visiting
remote:   https://github.com/maurozavatti/nombreEj/pull/new/nueva
rama
```

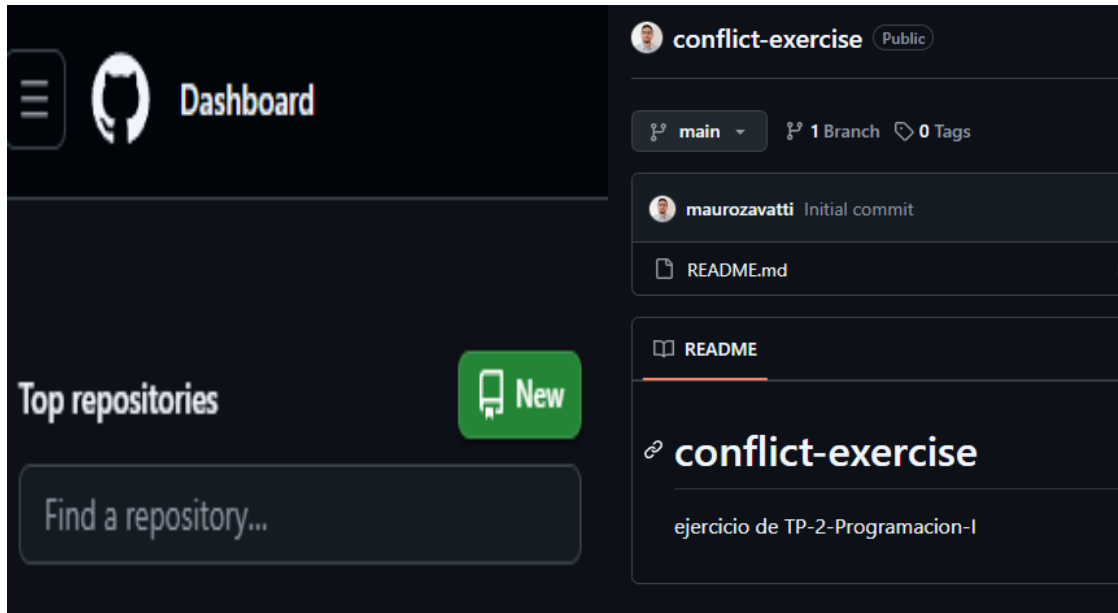


### 3) Realizar la siguiente actividad:

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
```

```

User@PC-MAURO MINGW64 ~/Desktop
$ git clone https://github.com/maurozavatti/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.

User@PC-MAURO MINGW64 ~/Desktop
$ cd conflict-exercise

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

```

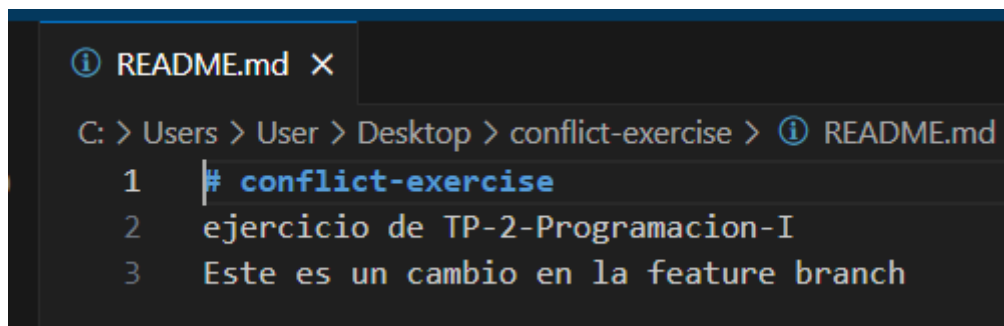
- Abre el archivo README.md en un editor de texto y añade una línea nueva, por ejemplo:

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"



```

① README.md X
C: > Users > User > Desktop > conflict-exercise > ① README.md
1  # conflict-exercise
2  ejercicio de TP-2-Programacion-I
3  Este es un cambio en la feature branch

```

```

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (feature-branch)
$ git add README.md

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
[feature-branch 91ce09c] Added a line in feature-branch
1 file changed, 1 insertion(+)

```

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

- Cambia de vuelta a la rama principal (main):

git checkout main

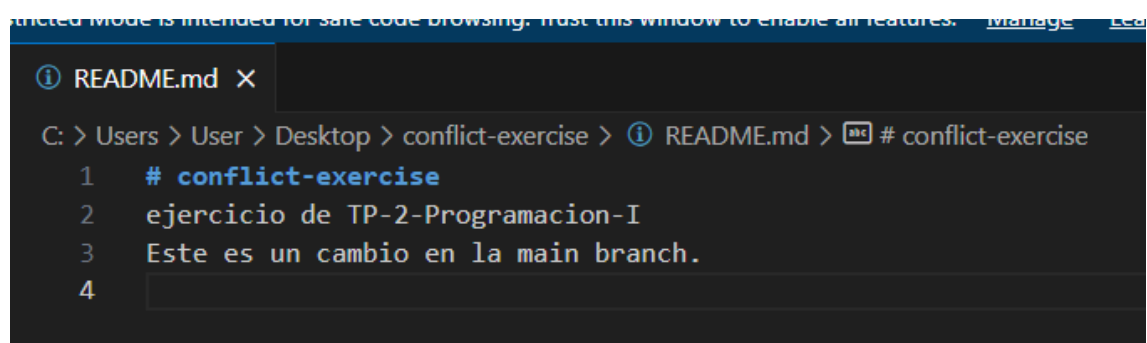
- Edita el archivo README.md de nuevo, añadiendo una línea diferente:

Este es un cambio en la main branch.

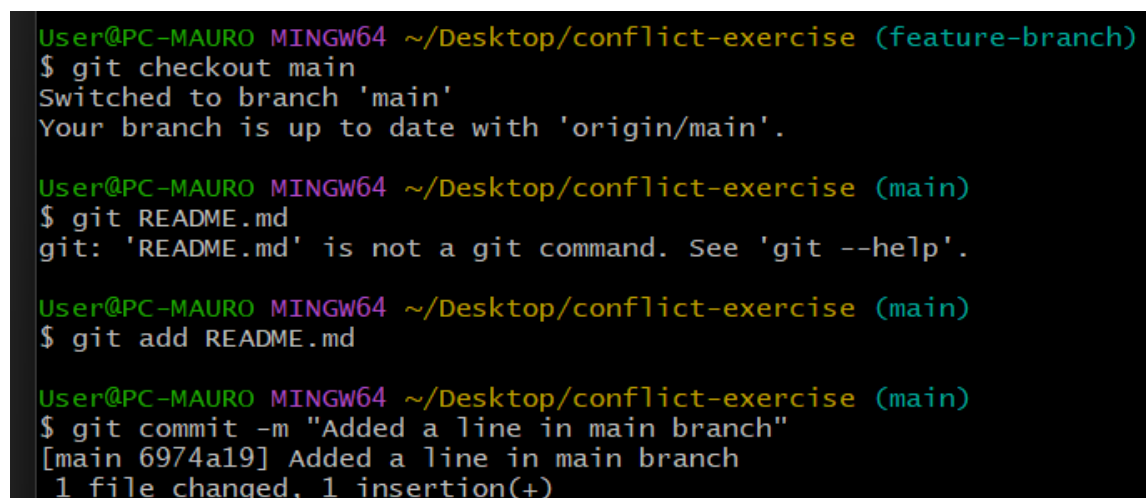
- Guarda los cambios y haz un commit:

git add README.md

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



```
① README.md X
C: > Users > User > Desktop > conflict-exercise > ① README.md > # conflict-exercise
1 # conflict-exercise
2 ejercicio de TP-2-Programacion-I
3 Este es un cambio en la main branch.
4
```



```
User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ git README.md
git: 'README.md' is not a git command. See 'git --help'.

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ git add README.md

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ git commit -m "Added a line in main branch"
[main 6974a19] Added a line in main branch
1 file changed, 1 insertion(+)
```

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.

```
User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ 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.
```

#### 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.

```
=====
```

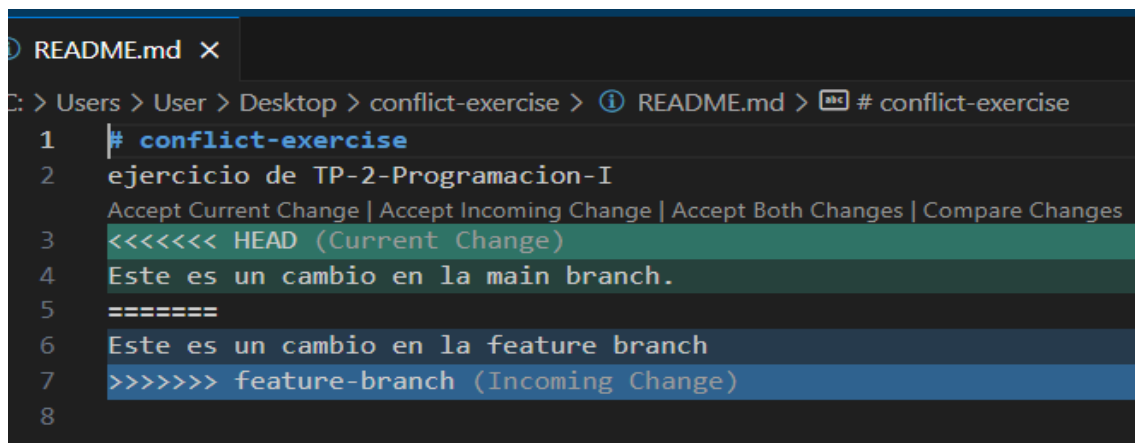
Este es un cambio en la feature branch.

```
>>>>>> 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"
```



```
README.md x
C: > Users > User > Desktop > conflict-exercise > README.md > # conflict-exercise
1  # conflict-exercise
2  ejercicio de TP-2-Programacion-I
3  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
4  <<<<<< HEAD (Current Change)
5  Este es un cambio en la main branch.
6  =====
7  Este es un cambio en la feature branch
8  >>>>>> feature-branch (Incoming Change)
```

```
① README.md ✕
C: > Users > User > Desktop > conflict-exercise > ① README.md > # conflict-exercise
1 # conflict-exercise
2 ejercicio de TP-2-Programacion-I
3 Este es un cambio en la main branch.
4 Este es un cambio en la feature branch
5
6 |
```

```
User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main|MERGING)
$ git add README.md

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main|MERGING)
$ git commit -m "Resolved merge conflict"
[main 5412ab0] 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

```
MINGW64/c/Users/User/Desktop/conflict-exercise
User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ 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), 798 bytes | 798.00 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/maurozavatti/conflict-exercise.git
8a622f8..5412ab0 main -> main

User@PC-MAURO MINGW64 ~/Desktop/conflict-exercise (main)
$ 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/maurozavatti/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/maurozavatti/conflict-exercise.git
* [new branch] feature-branch -> 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.

