

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.

Resultados de aprendizaje:

1. Comprender los conceptos básicos de Git y GitHub: Identificar y explicar los principales términos y procesos asociados con Git y GitHub, como repositorios, ramas, commits, forks, etiquetas y repositorios remotos.
2. Manejar comandos esenciales de Git: Ejecutar comandos básicos para crear, modificar, fusionar y gestionar ramas, commits y repositorios, tanto en local como en remoto.
3. Aplicar técnicas de colaboración en GitHub: Configurar y utilizar repositorios remotos, realizar forks, y gestionar pull requests para facilitar el trabajo colaborativo.
4. Resolver conflictos en un entorno de control de versiones: Identificar, analizar y solucionar conflictos de merge generados en un flujo de trabajo con múltiples ramas.

Actividades

- 1) Contestar las siguientes preguntas utilizando las guías y documentación proporcionada (Desarrollar las respuestas) :
 - ¿Qué es GitHub?
Git es un sistema de control de versiones que permite rastrear y gestionar los cambios en archivos de un proyecto
 - ¿Cómo crear un repositorio en GitHub?
Para crear un repositorio con Git, hay que navegar a la carpeta de tu proyecto y ejecutar **git init** para inicializar. Luego, agregar un origen remoto con **git remote add origin [url del repositorio]** . Subir tus archivos con **git add .** , confirma los cambios con **git commit -m "mensaje"** y finalmente enviarlos a GitHub con **git push -u origin main**
 - ¿Cómo crear una rama en Git?
Se utiliza el comando **git branch "NombreRama"**
 - ¿Cómo cambiar a una rama en Git?
Se utiliza el comando **git checkout NombreRama**
 - ¿Cómo fusionar ramas en Git?
Se utiliza el comando **git merge NombreRama**
Para hacer esto primero debemos estar parados en la rama principal o master

- ¿Cómo crear un commit en Git?
Se utiliza el comando **git commit -m "mensaje"**
- ¿Cómo enviar un commit a GitHub?
Se utiliza el comando **git push origin main** Si tu repositorio ya está vinculado con un remoto sino **git push origin nombre-de-la-rama**
- ¿Qué es un repositorio remoto?
Un **repositorio remoto** en Git es una versión del repositorio almacenada en un servidor
- ¿Cómo agregar un repositorio remoto a Git?
Se utiliza el comando **git remote add origin [url del repositorio]**
- ¿Cómo empujar cambios a un repositorio remoto?
Se utiliza el comando **git push origin nombre_de_la_rama**
- ¿Cómo tirar de cambios de un repositorio remoto?
Se utiliza el comando **git pull origin nombre_de_la_rama**
- ¿Qué es un fork de repositorio?
Un **fork** en GitHub es una copia independiente de un repositorio en tu propia cuenta.
- ¿Cómo crear un fork de un repositorio?
Se crea haciendo clic en el botón "**Fork**" en la página del repositorio original. Para trabajar localmente, se clona el fork con **git clone**

- ¿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 expplayarnos 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. Para aceptar una **solicitud de extracción (Pull Request, PR)** en GitHub, se debe ir al repositorio, entra en la pestaña "**Pull Requests**", seleccionar la PR, revisar los cambios y clickear en "**Merge Pull Request**". Luego, confirmar con "**Confirm Merge**" y opcionalmente eliminar la rama con "**Delete Branch**". 🚀
- ¿Qué es un etiqueta en Git?
Una **etiqueta en Git** es un marcador que señala un punto específico en la historia del repositorio, suele usarse para versiones importantes como lanzamientos
- ¿Cómo crear una etiqueta en Git?
Para crear una etiqueta en Git, se usa **git tag**. Una **etiqueta anotada** se crea con **git tag -a v1.0 -m "Versión 1.0"**, mientras que una **ligera** solo con **git tag v1.0**. Para subirla al remoto, se usa **git push origin v1.0** o **git push --tags** para todas.
- ¿Cómo enviar una etiqueta a GitHub?
Para subirla al remoto, se usa **git push origin v1.0** o **git push --tags** para todas.
- ¿Qué es un historial de Git?
El **historial de Git** es el registro de todos los cambios realizados en un repositorio, incluyendo commits, fusiones y ramas. Se puede visualizar con **git log**, que muestra los commits en orden cronológico con detalles como autor, fecha y mensaje.
- ¿Cómo ver el historial de Git?
Se utiliza el comando **git log**
- ¿Cómo buscar en el historial de Git?
se puede usar **git log** con opciones específicas. Para buscar por autor, usa **git log --author="nombre"**. Para encontrar un mensaje, usa **git log --grep="palabra clave"**. Para ver cambios en un archivo, usa **git log -- nombre-del-archivo**. También puedes combinar filtros, como **git log --author="nombre" --grep="bugfix"**

- ¿Cómo borrar el historial de Git?

El comando **git reset** se utiliza sobre todo para deshacer las cosas, como posiblemente puedes deducir por el verbo. Se mueve alrededor del puntero **HEAD** y opcionalmente cambia el índice 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 asegúrate de entenderlo antes de usarlo.

- **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 sólo es accesible para usuarios específicos que han sido autorizados

- ¿Cómo crear un repositorio privado en GitHub?

Para crear un **repositorio privado** en GitHub, se debe iniciar sesión y e ir a "**New Repository**". Luego, darle un nombre y seleccionar "**Private**", configurar opciones adicionales si es necesario y clicar en "**Create repository**"

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

Para invitar a alguien, hay que ir a la pestaña "**Settings**", luego a "**Manage access**", hacer clic en "**Invite a collaborator**", escribir el nombre de usuario o correo de la persona y seleccionar "**Add**". Luego de eso la persona recibirá una invitación para acceder al repositorio.

- ¿Qué es un repositorio público en GitHub?

- Un repositorio público en GitHub es un tipo de repositorio en el que el contenido es accesible para todos los usuarios que contengan la url

- ¿Cómo crear un repositorio público en GitHub?

Para crear un **repositorio público** en GitHub, se debe iniciar sesión y e ir a "**New Repository**". Luego, darle un nombre y seleccionar "**Public**", configurar opciones adicionales si es necesario y clicar en "**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"

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 *

Repository name *

 mationxz


 /


RepositorioTP2

 RepositorioTP2 is available.

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

Description (optional)

☒  **Public**

☐  **Private**

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

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**

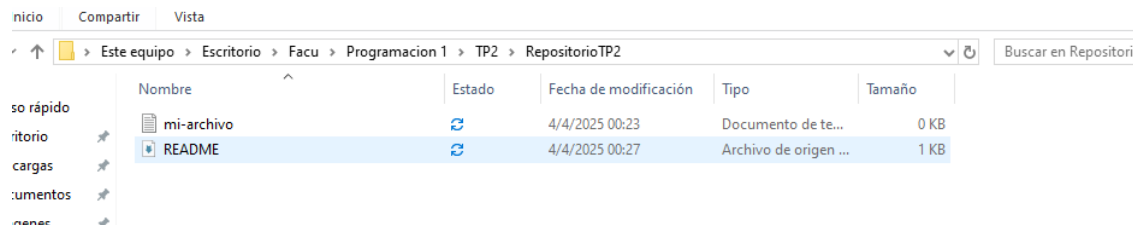
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

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

- 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.



```
matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git add .

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git commit -m "Agregando mi-archivo.txt"
[main 870850a] Agregando mi-archivo.txt
1 file changed, 1 insertion(+)
create mode 100644 mi-archivo.txt
```

- Sube los cambios al repositorio en GitHub con `git push origin main` (o el nombre de la rama correspondiente).

```
matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git push origin main
info: please complete authentication in your browser...
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 317 bytes | 317.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/matioxz/RepositorioTP2.git
   6ae68a7..870850a  main -> main

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ |
```

- Creando Branchs
 - Crear una Branch

```
matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git branch ramal

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git branch
* main
  ramal

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git checkout ramal
Switched to branch 'ramal'

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git branch
main
* ramal
```

- Realizar cambios o agregar un archivo
- Subir la Branch

```
matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git branch ramal

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git branch
* main
  ramal

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (main)
$ git checkout ramal
Switched to branch 'ramal'

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git branch
main
* ramal

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ echo "Nueva rama" > archivo2.txt

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git add .
warning: in the working copy of 'archivo2.txt', LF will be replaced by CRLF the next time Git touches it

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git status
On branch ramal
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   archivo2.txt

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git commit -m "Nuevo archivo txt"
[ramal 3b1528f] Nuevo archivo txt
1 file changed, 1 insertion(+)
create mode 100644 archivo2.txt

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git status
On branch ramal
nothing to commit, working tree clean

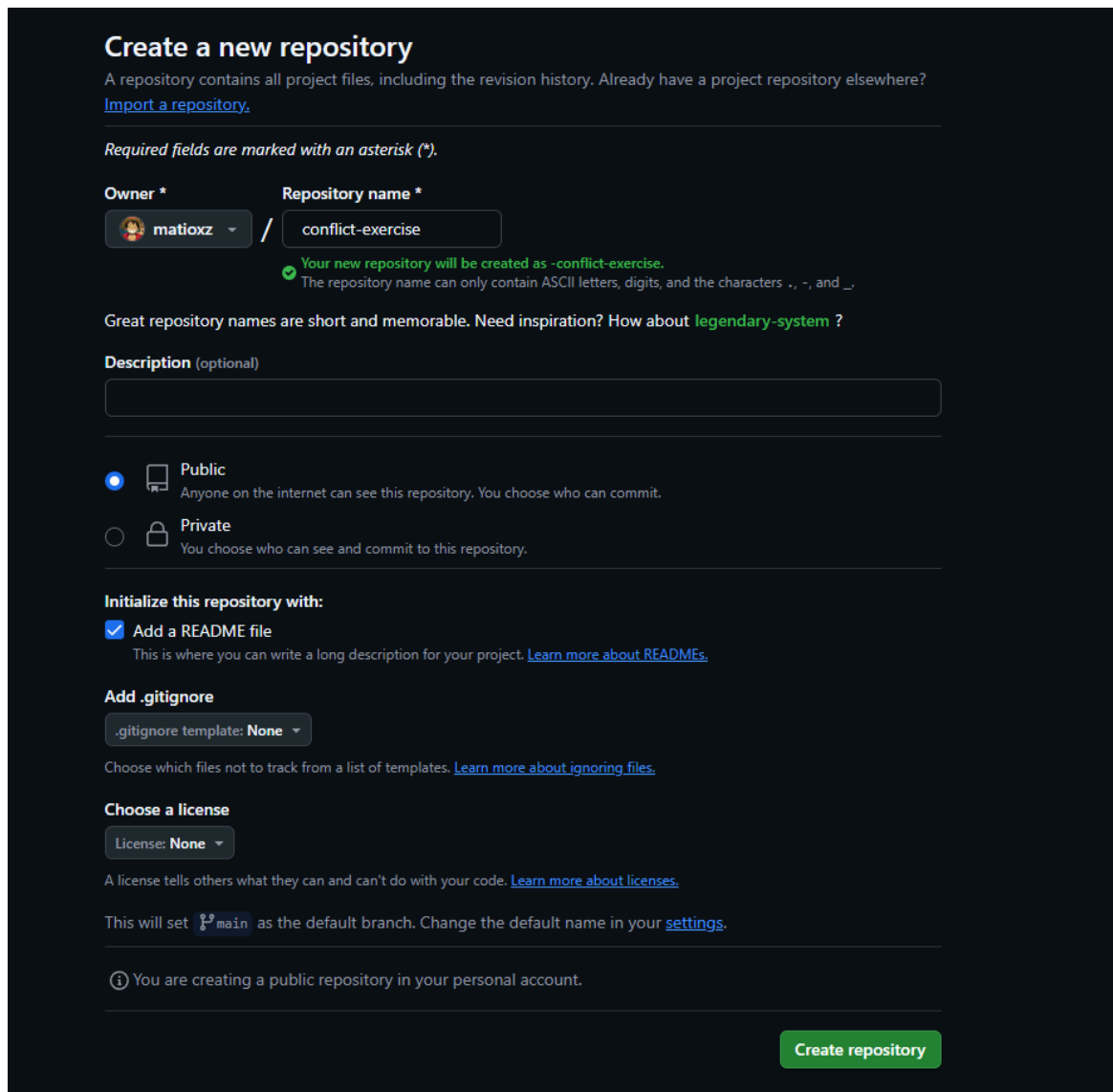
matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$ git push origin ramal
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 329 bytes | 329.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'ramal' on GitHub by visiting:
remote:   https://github.com/matioxz/RepositorioTP2/pull/new/ramal
remote:
To https://github.com/matioxz/RepositorioTP2.git
 * [new branch]      ramal -> ramal

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/RepositorioTP2 (ramal)
$
```


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".



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 *  **matioxz** / **Repository name ***

✓ Your new repository will be created as -conflict-exercise.
The repository name can only contain ASCII letters, digits, and the characters -, ., and _.

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

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: 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"
```

```
atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1 (master)
$ cd list
bash: cd: list: No such file or directory

atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1 (master)
$ cd conflict-exercise
bash: cd: conflict-exercise: No such file or directory

atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1 (master)
$ cd conflict-exercise

atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict
exercise (main)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict
exercise (feature-branch)
$ git add README.md

atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict
exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
[feature-branch 7747294] Added a line in feature-branch
1 file changed, 1 insertion(+), 1 deletion(-)

atic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict
exercise (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
```

```
matic@DESKTOP-BJ008S7 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict-exercise (main)
$ git commit -m "Added a line in main branch"
[main cdab4fc] Added a line in main branch
1 file changed, 1 insertion(+), 1 deletion(-)

matic@DESKTOP-BJ008S7 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/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.

matic@DESKTOP-BJ008S7 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict-exercise (main)
$ |
```

- 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.

```
=====
```

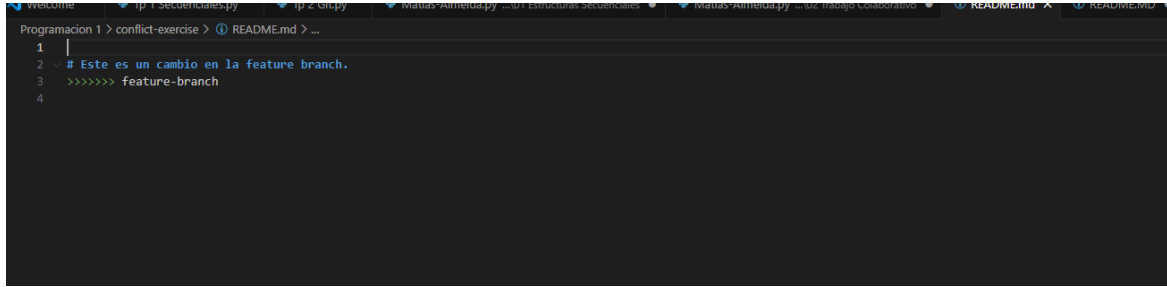
Este es un cambio en la feature branch.

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

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<<< HEAD (Current Change)
2 # el archivo esta en problemas :C
3 =====
4 # Este es un cambio en la feature branch.
5 >>>>>> feature-branch (Incoming Change)
6
```

- 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`

```
matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict-exercise (main|MERGING)
$ git add README.md

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict-exercise (main|MERGING)
$ git commit -m "Resolved merge conflict"
[main 718fd8b] Resolved merge conflict

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/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% (4/4), done.
Writing objects: 100% (9/9), 804 bytes | 804.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/matioxz/conflict-exercise.git
 2f604db..718fd8b  main -> main

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/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/matioxz/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/matioxz/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch

matic@DESKTOP-BJ00857 MINGW64 ~/OneDrive/Escritorio/Facu/Programacion 1/conflict-exercise (main)
$
```

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.

