

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?
 - ¿Cómo crear un repositorio en GitHub?
 - ¿Cómo crear una rama en Git?
 - ¿Cómo cambiar a una rama en Git?
 - ¿Cómo fusionar ramas en Git?
 - ¿Cómo crear un commit en Git?
 - ¿Cómo enviar un commit a GitHub?
 - ¿Qué es un repositorio remoto?
 - ¿Cómo agregar un repositorio remoto a Git?
 - ¿Cómo empujar cambios a un repositorio remoto?

- ¿Cómo tirar de cambios de un repositorio remoto?
- ¿Qué es un fork de repositorio?
- ¿Cómo crear un fork de un repositorio?
- ¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?
- ¿Cómo aceptar una solicitud de extracción?
- ¿Qué es un etiqueta en Git?
- ¿Cómo crear una etiqueta en Git?
- ¿Cómo enviar una etiqueta a GitHub?
- ¿Qué es un historial de Git?
- ¿Cómo ver el historial de Git?
- ¿Cómo buscar en el historial de Git?
- ¿Cómo borrar el historial de Git?
- ¿Qué es un repositorio privado en GitHub?
- ¿Cómo crear un repositorio privado en GitHub?
- ¿Cómo invitar a alguien a un repositorio privado en GitHub?
- ¿Qué es un repositorio público en GitHub?
- ¿Cómo crear un repositorio público en GitHub?
- ¿Cómo compartir un repositorio público en GitHub?

2) Realizar la siguiente actividad:

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

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

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.

- 1) GitHub es una plataforma colaborativa que nos va a permitir llevar un control de versión sobre nuestro código.

Es un lugar en la nube donde se puede alojar los archivos de nuestros proyectos de programación de manera gratuita, solamente creando una cuenta.

- 2) Un repositorio en GitHub se crea de la siguiente forma:

1. En la esquina superior derecha de cualquier página, seleccionar + y luego haz clic en New repository(Nuevo repositorio) .
2. Escribir un nombre corto y fácil de recordar para el repositorio.
3. Opcionalmente, se puede agregar una descripción del repositorio. Por ejemplo, "Mi primer repositorio".
4. Elegir la visibilidad del repositorio.
5. Seleccionar Initialize this repository with a README(Inicializar este repositorio con un archivo Léame) en el que caso de ser necesario.
6. Hacer clic en Create repository (Crear repositorio).

- 3) Una rama en Git se puede crear con el comando git branch "nombre de la rama". También puede crearse con el comando git checkout -b "nombre de la rama". Este último comando además de crear la rama nos posiciona en la misma.
- 4) Para cambiar a una rama en Git se utiliza git checkout "nombre de la rama".
- 5) Para fusionar ramas en Git podemos usar git merge seguido del nombre de la rama que se complementará con la rama en la que estoy actualmente posicionado.
- 6) Para crear un commit en Git se utiliza git commit -m "mensaje". En el caso de haber utilizado anteriormente git add, se puede utilizar git commit -am "mensaje".
- 7) Podemos enviar un commit a GitHub con git push origin seguido del nombre de la rama en cuestión.
- 8) Un repositorio remoto se aloja en un servidor remoto (puede estar en internet o en un servidor externo; incluso podría ser la misma máquina en una ruta diferente) y se comparte entre varios miembros del equipo.
- 9) Para agregar un repositorio remoto a Git, podemos usar el comando git remote add URL en el terminal. Debemos estar dentro del directorio donde está almacenado nuestro repositorio local.

- 10) Para empujar cambios a un repositorio remoto podemos usar el comando `git push origin` seguido del nombre de la rama en cuestión. Antes de hacerlo, se debe hacer commit a todos los cambios en el repositorio local.
- 11) Para obtener los cambios de un repositorio remoto en Git, puedes usar el comando `git pull origin` seguido del nombre de la rama en cuestión. Este comando descarga y extrae el contenido de un repositorio remoto y actualiza el repositorio local.
- 12) Un Fork(bifurcación) de repositorio es una copia exacta de un repositorio, que se utiliza para colaborar en un proyecto sin tener permisos de escritura.
- 13) Para crear un fork de un repositorio en GitHub, puedes seguir estos pasos:
 1. Ir al repositorio que deseas copiar.
 2. Hacer clic en el botón "Fork".
 3. Cambiar la descripción del Fork.
 4. Hacer clic en "Create fork".
- 14) Para enviar una solicitud de extracción (pull request) a un repositorio, puedes seguir estos pasos:
 1. Ir a la página principal del repositorio y hacer un fork.
 2. Clonamos el repositorio creado por el fork a nuestro repositorio local.
 3. Creamos una rama y realizamos cambios sobre el/los archivos. Nos posicionamos en esta nueva rama.
 4. Procedemos a guardar, commitear y pushear estos cambios a la nueva rama.
 5. En GitHub, seleccionamos la rama que contiene los cambios.
 6. Hacemos clic sobre Pull request.
 7. Creamos el pull request.

También sucede generalmente que cuando subimos una rama nueva al repositorio remoto y se detectan cambios con respecto a la rama principal, nos sugiere un PR:

1. En este caso hacemos clic en "Compare & pull request".
2. Elegir las ramas de base y de comparación.

3. Escribir un título y una descripción.
 4. Hacer clic en "Create pull request".
- 15) Para aceptar una solicitud de extracción en GitHub, puedes revisar los cambios propuestos, escribir un comentario, y luego aprobar la solicitud.

Revisar los cambios propuestos:

1. Haz clic en la solicitud de extracción en la lista de solicitudes.
2. Haz clic en Archivos modificados.
3. Revisa los cambios en un archivo por vez.
4. Deja comentarios en los cambios que quieras.
5. Marca cada archivo como visto después de revisarlo.

Escribir un comentario:

1. Haz clic en Revisar cambios sobre el código modificado.
2. Escribe un comentario que resuma tu opinión sobre los cambios propuestos.

Aprobar la solicitud:

1. Selecciona Aprobar para aprobar la fusión de los cambios propuestos.
 2. Haz clic en Enviar revisión.
- 16) En Git, una etiqueta (tag) es una marca que se aplica a una confirmación para identificar un punto importante en el historial de un repositorio. Las etiquetas son útiles para:
1. Identificar confirmaciones importantes, como la versión beta de un proyecto.
 2. Crear instantáneas de un repositorio.
 3. Marcar versiones de software con números de versión semánticos.
 4. Asociar etiquetas con confirmaciones para marcar lanzamientos.

17) Para crear una etiqueta:

1. Usa el comando git tag.
2. Especifica un nombre para la etiqueta.
3. Opcionalmente, proporciona un mensaje de etiqueta.
4. Ejecuta el comando git tag.

18) Para enviar etiquetas a GitHub, puedes usar el comando git push con la opción -tags. También puedes crear etiquetas directamente desde la interfaz web de GitHub.

Usar git push para enviar etiquetas:

1. Crea las etiquetas localmente.
2. Usa el comando git push con la opción --tags para enviar las etiquetas al repositorio remoto.
3. Por ejemplo, git push origin master --tag.

Crear etiquetas desde la interfaz web de GitHub:

1. Ir a la página principal del repositorio.
2. En el enlace "releases", etiqueta nuevas versiones.
3. También puedes crear etiquetas haciendo clic en "Incidencias o Solicitudes de incorporación de cambios", luego "Labels", y luego "Nueva etiqueta".

19) El historial de Git es un registro de los cambios realizados en un repositorio, almacenado como un gráfico de instantáneas llamadas confirmaciones.

Características del historial de Git:

- Cada confirmación puede tener varios padres, lo que da forma al historial en forma de gráfico.
- El historial de Git permite:
 - Ver quiénes son los autores de los cambios.
 - Averiguar dónde se introdujeron los errores.
 - Revertir los cambios problemáticos.
 - Filtrar las confirmaciones.
 - Buscar cambios concretos.
 - Ver la gráfica de confirmación.

- Ver los archivos de los árboles de cada revisión.
- Examinar el historial de una función o línea de código.

20) Para ver el historial de Git, puedes usar el comando git log. También puedes ver el historial de cambios en GitHub.

Ver el historial de Git con git log:

- git log muestra las confirmaciones de un repositorio en orden cronológico inverso.
- git log -L muestra el historial de una función o línea de código.
- git log --since="7 days" muestra los commits realizados en los últimos 7 días.

Ver el historial de cambios en GitHub:

- En la pestaña "Historial", haz clic en la confirmación que quieres revisar.
- En la página principal del repositorio, haz clic para abrir el archivo cuyo historial de líneas quieres ver.
- Sobre el contenido del archivo, haz clic en "Atribuir".
- Usa la vista de actividad para ver un historial detallado de cambios en un repositorio.

21) Para buscar en el historial de Git, puedes usar los comandos git log y git grep. El comando git log permite ver el historial de commits, mientras que git grep busca dentro del contenido de los archivos en el historial.

- git log --grep <patrón>:
Busca en los mensajes de los commits por un patrón específico. Por ejemplo, git log --grep="feature A" mostrará los commits que contengan "feature A" en el mensaje.
- git log -p -S <string>:
Busca en el código de los commits por una cadena de texto específica. Por ejemplo, git log -p -S"funcion1" mostrará los commits que agreguen o eliminen la cadena "funcion1".
- git grep <patrón>:
Busca texto dentro de los archivos en el historial. Por ejemplo, git grep "variable1" mostrará todos los archivos y líneas que contienen "variable1" en el historial.
- gitk <path>:
Muestra el historial de commits en una ventana gráfica, lo que puede ser útil para visualizar el árbol de commits y los cambios en los archivos.

- `git log --pretty=format:<format>`:
Permite personalizar la salida del git log con diferentes formatos, por ejemplo, para mostrar solo los mensajes de los commits, `git log --pretty=format:%B`.
- `git blame <archivo>`:
Muestra el historial de cambios línea por línea en un archivo, indicando el autor, la fecha y el mensaje de confirmación de cada cambio.

22) Para borrar el historial de Git, puedes utilizar herramientas como `git filter-repo` o `BFG Repo-Cleaner` para reescribir la historia del repositorio. Estas herramientas eliminan datos específicos o archivos de commits, cambiando el historial del repositorio. También puedes utilizar `git reset --hard` para revertir a un commit anterior y eliminar cambios posteriores.

Cómo borrar el historial de Git:

1. Uso de `git filter-repo`:

Si necesitas eliminar datos sensibles de un repositorio remoto, utiliza `git filter-repo` para reescribir el historial. Este comando te permite eliminar archivos específicos o borrar datos de confirmaciones.

Ejemplo: `git filter-repo --path ./web/.env --invert-paths` (Esta línea elimina el archivo `.env` de la historia del repositorio, según el ejemplo de [DEV Community](#).)

2. Uso de `BFG Repo-Cleaner`:

[GitHub Docs](#) también recomienda `BFG Repo-Cleaner`, que también puede reescribir el historial del repositorio.

3. Uso de `git reset --hard`:

Este comando te permite revertir a un commit anterior y eliminar todos los cambios posteriores.

Ejemplo: `git reset --hard <commit>` (Reemplaza `<commit>` con el SHA del commit al que quieres revertir).

4. Borrado de archivos en repositorios remotos (GitHub):

- Borrando un directorio: Si vas a borrar un directorio, navega a la carpeta en el repositorio, selecciona el menú desplegable y haz clic en "Eliminar directorio".
- Escribiendo un mensaje de confirmación: Escribe un mensaje descriptivo que explique el cambio.

- Seleccionando la rama: Decide si deseas agregar la confirmación a la rama actual o a una nueva.
- Confirmando los cambios: Haz clic en "Confirmar cambios" o "Proponer cambios".

5. Forzando las subidas (push):

Después de eliminar datos sensibles o reescribir el historial, debes forzar la subida a GitHub para que los cambios se reflejen en el repositorio remoto.

23)En GitHub, un repositorio privado es aquel al que solo tú y las personas con las que has compartido acceso explícitamente pueden acceder. A diferencia de los repositorios públicos, que son visibles para todos en internet, los privados se diseñan para la colaboración con equipos o para proyectos sensibles que no deben ser de acceso público.

24)Para crear un repositorio privado en GitHub, sigue estos pasos: inicia sesión en GitHub, haz clic en el icono de nuevo repositorio, introduce el nombre del repositorio, selecciona la visibilidad "Private" (Privado), y luego haz clic en "Create repository" (Crear repositorio).

25)Para invitar a alguien a un repositorio privado en GitHub, debes ir a la configuración del repositorio, luego a "Colaboradores", y finalmente agregar el nombre de usuario de la persona que quieres invitar. Después, el usuario recibirá una invitación por correo electrónico para aceptar el acceso al repositorio.

26)En GitHub, un repositorio público es un lugar donde se almacena código, archivos y el historial de cambios de un proyecto. Es accesible para cualquier persona en internet, lo que facilita la colaboración y el intercambio de código con la comunidad.

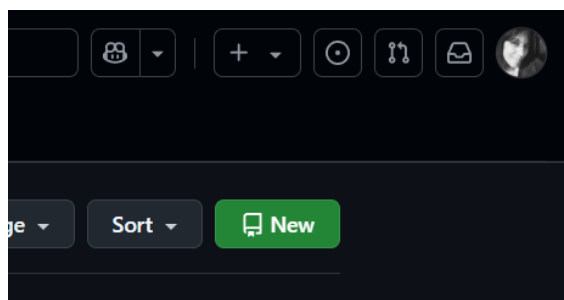
27)Para crear un repositorio público en GitHub, ve a <https://github.com/new>, elige la opción "Crear un nuevo repositorio", asigna un nombre, descripción, y selecciona "Público" como visibilidad. Puedes iniciar el repositorio con un README, y luego crear un archivo y realizar el primer commit para confirmar tus cambios.

28)Para compartir un repositorio público en GitHub, simplemente lo creas como público. Luego, puedes compartir el enlace de tu repositorio con otros usuarios o invitar a colaboradores para que tengan acceso a él. Para ello, busca en el repositorio la opción de "Colaboradores" o "Invitar colaboradores" y agrega a las personas que quieras compartir el repositorio.

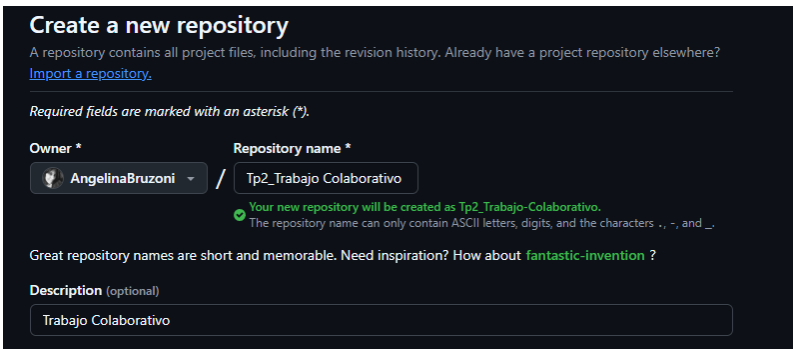
1. Crear un repositorio:
Inicia sesión en GitHub y crea un nuevo repositorio. En la configuración del repositorio, selecciona la opción "Public" para que sea visible para todos.
2. Encontrar el enlace del repositorio:
Una vez creado, el repositorio tendrá una dirección URL que puedes copiar y compartir con quien quieras. Esta dirección se muestra en la página principal del repositorio.
3. Invitar colaboradores (opcional):
Si deseas que otros usuarios tengan acceso al repositorio y puedan realizar modificaciones, puedes invitarlos como colaboradores. En la configuración del repositorio, busca la sección "Colaboradores" o "Invitar colaboradores", ingresa el nombre de usuario de la persona que quieres invitar y haz clic en el botón "Invitar".
4. Permisos de acceso:
Dependiendo de la opción que selecciones, los colaboradores podrán tener diferentes permisos, desde simplemente ver el repositorio hasta realizar modificaciones y hacer "push" de cambios, según la documentación de GitHub.
5. Compartir el repositorio:
Finalmente, puedes compartir el enlace del repositorio con quien quieras, ya sea por correo electrónico, redes sociales, o cualquier otro medio.

2)

1. Creamos un repositorio.
 - Creamos un repositorio presionando en New:



- Le damos un nombre al repositorio:



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 (*).

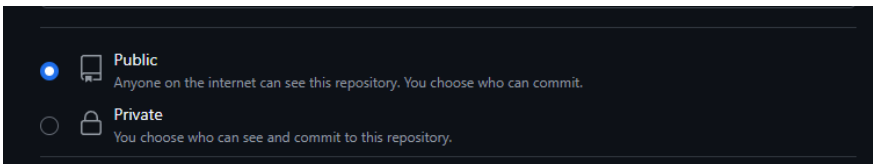
Owner * AngelinaBruzoni / Repository name * Tp2_Trabajo Colaborativo

Your new repository will be created as **Tp2_Trabajo-Colaborativo**.
The repository name can only contain ASCII letters, digits, and the characters `.`, `-`, and `_`.

Great repository names are short and memorable. Need inspiration? How about **fantastic-invention** ?

Description (optional)
Trabajo Colaborativo

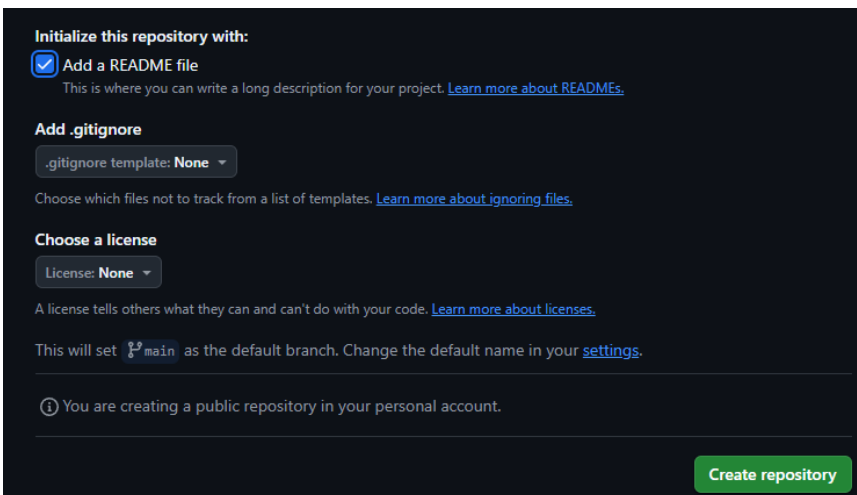
- Elegimos que el repositorio sea público:



☒ **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.

- Inicializamos el repositorio con un archivo:



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.](#)

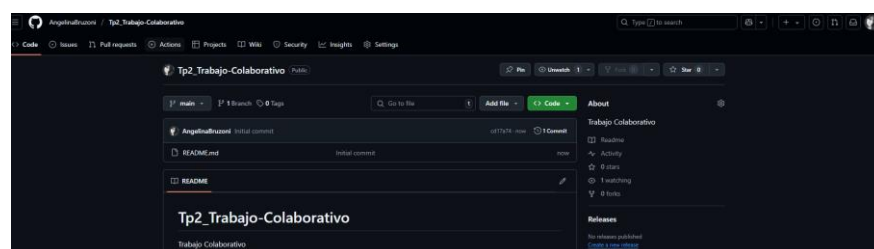
Choose a license
License: **None**

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

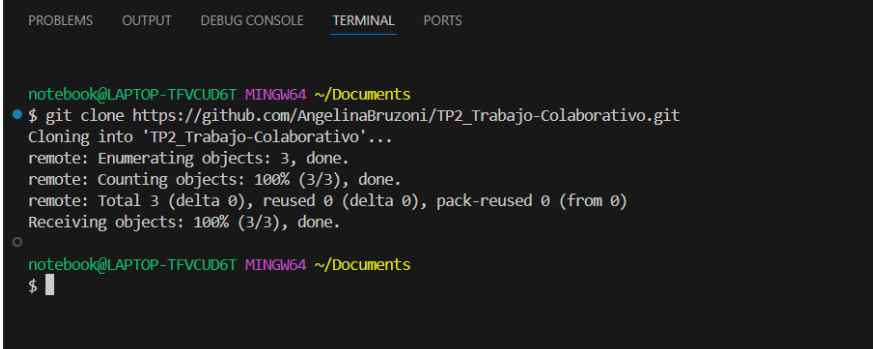
This will set `main` as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

Create repository



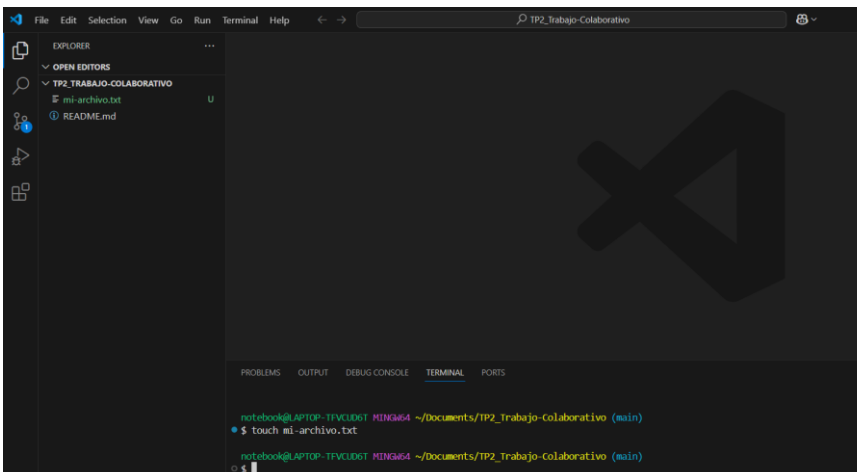
- Clonamos el repositorio remoto a nuestra Pc.



```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents
$ git clone https://github.com/AngelinaBruzoni/TP2_Trabajo-Colaborativo.git
Cloning into 'TP2_Trabajo-Colaborativo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents
$
```

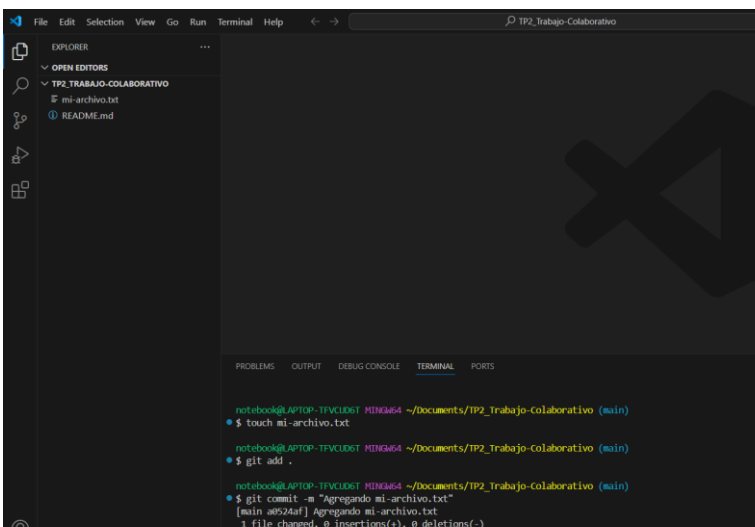
2. Agregamos un archivo.

- Creamos un archivo llamado mi-archivo.txt



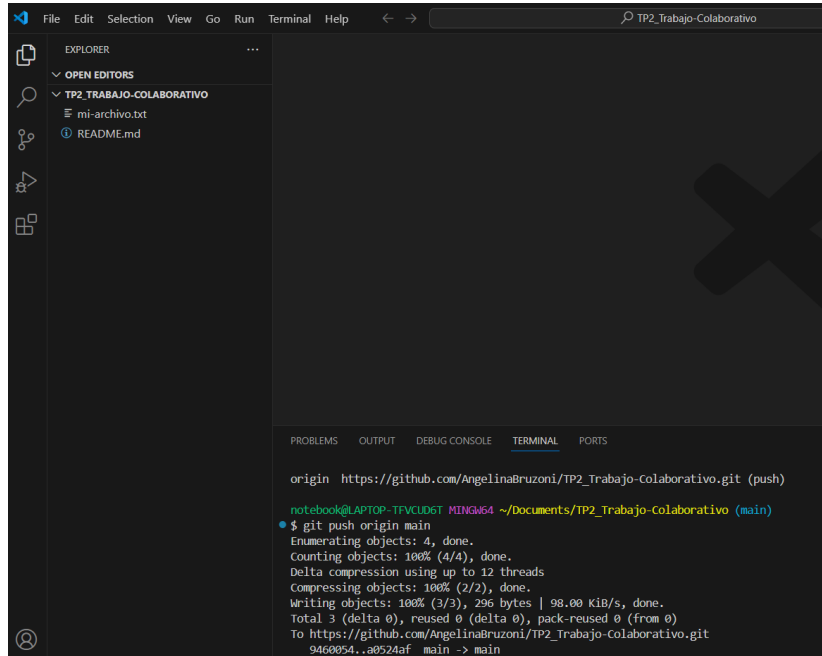
```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
$ touch mi-archivo.txt
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
$
```

- Ejecutamos los comandos git add . y git commit -m "Agregando mi-archivo.txt" en la línea de comandos.



```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
$ touch mi-archivo.txt
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
$ git add .
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
$ git commit -m "Agregando mi-archivo.txt"
[main 40524af] Agregando mi-archivo.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 mi-archivo.txt
```

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

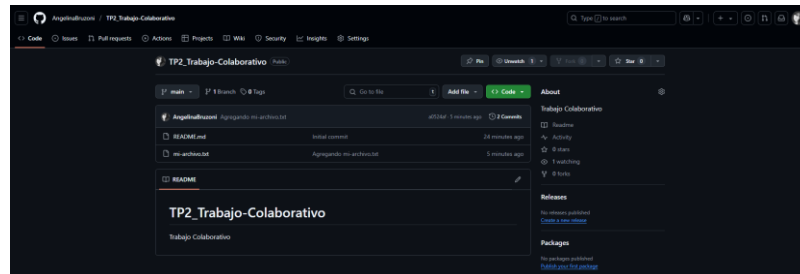


```
File Edit Selection View Go Run Terminal Help < -> TP2_Trabajo-Colaborativo

EXPLORER
  OPEN EDITORS
  TP2_TRABAJO-COLABORATIVO
    mi-archivo.txt
    README.md

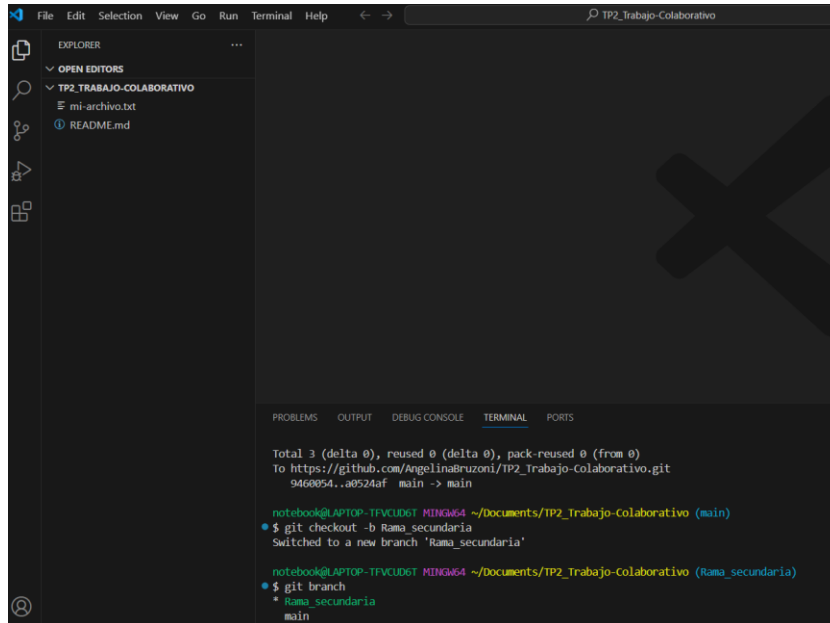
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

origin https://github.com/AngelinaBruzoni/TP2_Trabajo-Colaborativo.git (push)
notebook@LAPTOP-TPVCLDGT MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
$ git push origin main
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), 296 bytes | 98.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/AngelinaBruzoni/TP2_Trabajo-Colaborativo.git
9460054..a0524af main -> main
```



3. Creando Branchs.

- Creamos Rama_secundaria.



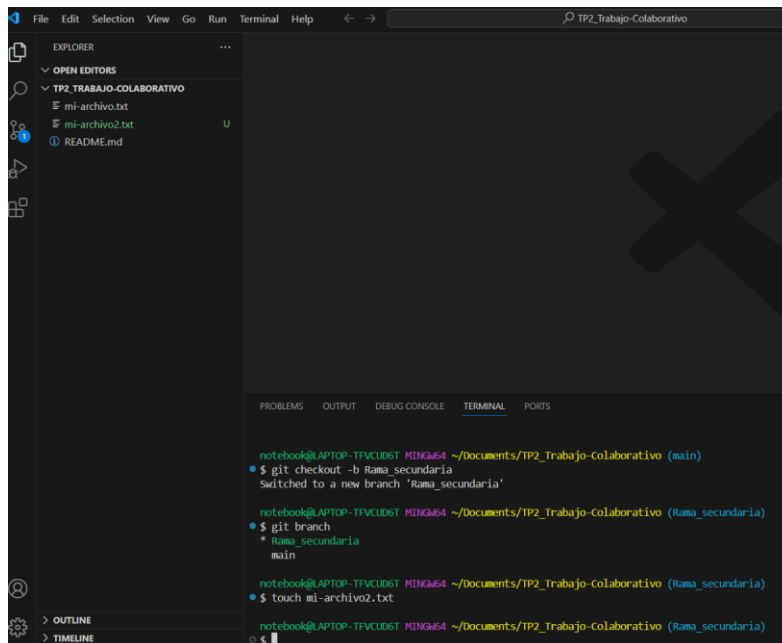
The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the file structure of 'TP2_TRABAJO-COLABORATIVO', including 'mi-archivo.txt' and 'README.md'. The main editor area is empty. The integrated terminal at the bottom shows the following commands and output:

```
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/AngelinaBruzoni/TP2_Trabajo-Colaborativo.git
9460054..a0524af main -> main

notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
• $ git checkout -b Rama_secundaria
Switched to a new branch 'Rama_secundaria'

notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
• $ git branch
* Rama_secundaria
  main
```

- Agregamos mi-archivo2.txt.



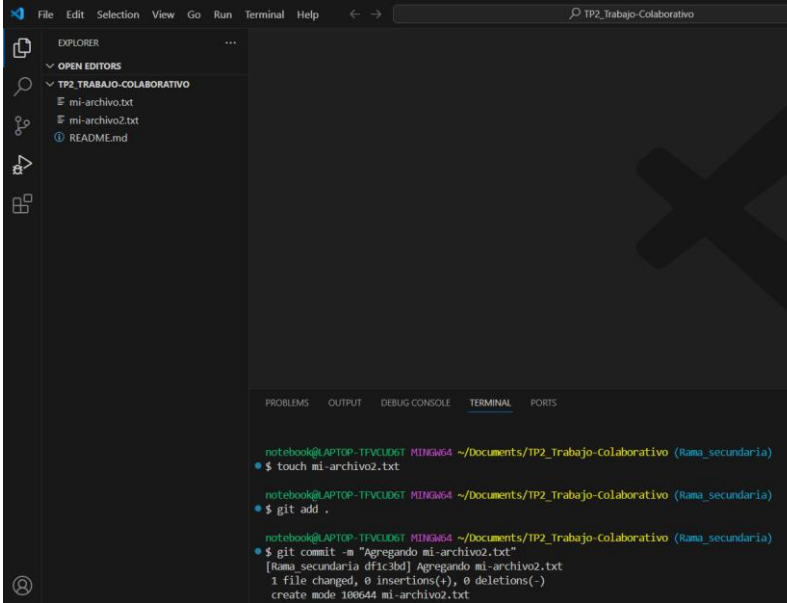
The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the file structure of 'TP2_TRABAJO-COLABORATIVO', including 'mi-archivo.txt', 'mi-archivo2.txt' (marked with a 'U' for untracked), and 'README.md'. The main editor area is empty. The integrated terminal at the bottom shows the following commands and output:

```
notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (main)
• $ git checkout -b Rama_secundaria
Switched to a new branch 'Rama_secundaria'

notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
• $ git branch
* Rama_secundaria
  main

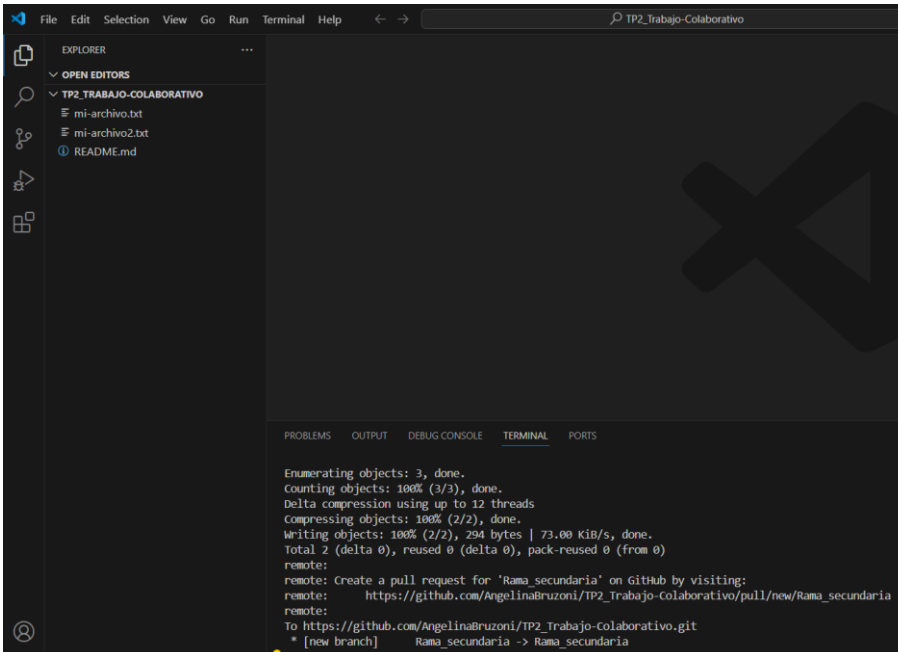
notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
• $ touch mi-archivo2.txt

notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
• $
```

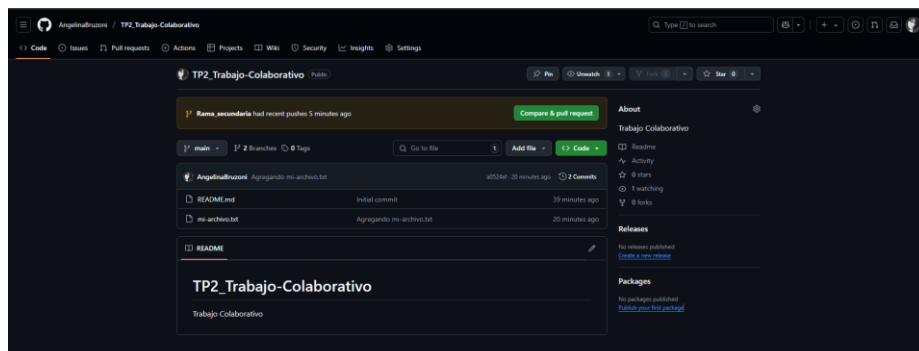


```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
$ touch mi-archivo2.txt
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
$ git add .
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/TP2_Trabajo-Colaborativo (Rama_secundaria)
$ git commit -m "Agregando mi-archivo2.txt"
[Rama_secundaria df1c3bd] Agregando mi-archivo2.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 mi-archivo2.txt
```

- Ejecutamos un git push origin Rama_secundaria.



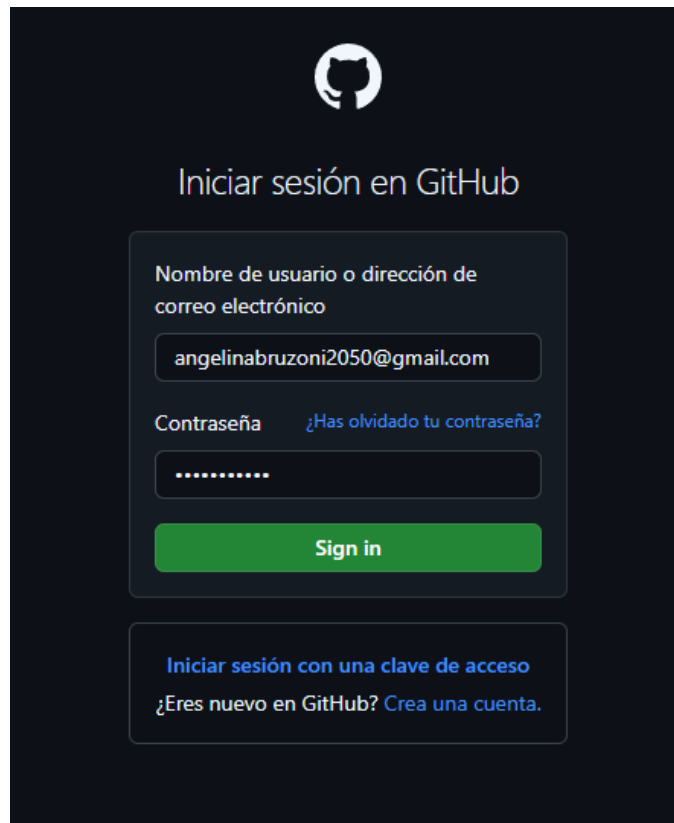
```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 294 bytes | 73.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Create a pull request for 'Rama_secundaria' on GitHub by visiting:
remote:   https://github.com/AngelinaBruzoni/TP2_Trabajo-Colaborativo/pull/new/Rama_secundaria
remote:
To https://github.com/AngelinaBruzoni/TP2_Trabajo-Colaborativo.git
 * [new branch]      Rama_secundaria -> Rama_secundaria
```



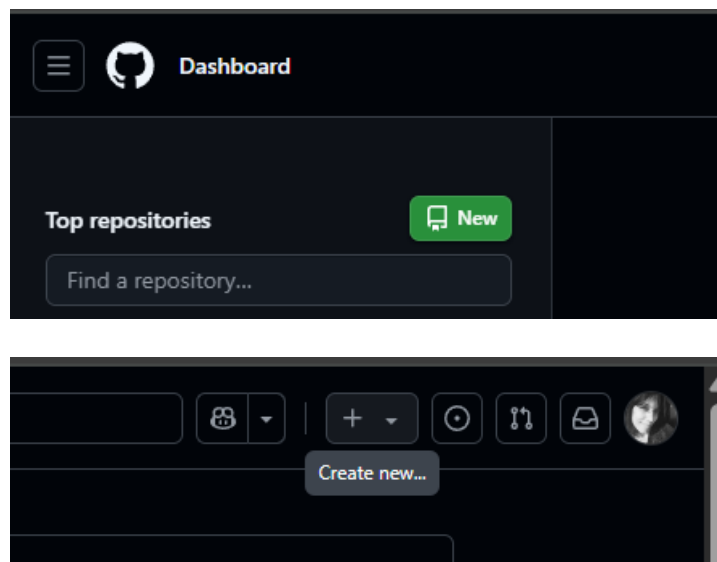
3)

1. Creamos un repositorio en GitHub:

- Nos dirigimos a <https://github.com> e iniciamos sesión en nuestra cuenta.



- Hacemos clic en el botón New o Create repository para crear un nuevo repositorio.



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 * AngelinaBruzoni / Repository name *

▲ New repository name must not be blank

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

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.](#)

① You are creating a public repository in your personal account.

[Create repository](#)

- Asignamos un nombre al repositorio, por ejemplo, conflicto-exercise.
- Añadimos una descripción.
- Marcamos la opción Initialize this repository with a README.

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 * AngelinaBruzoni / Repository name * conflict-exercise

✔ conflict-exercise is available.

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

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

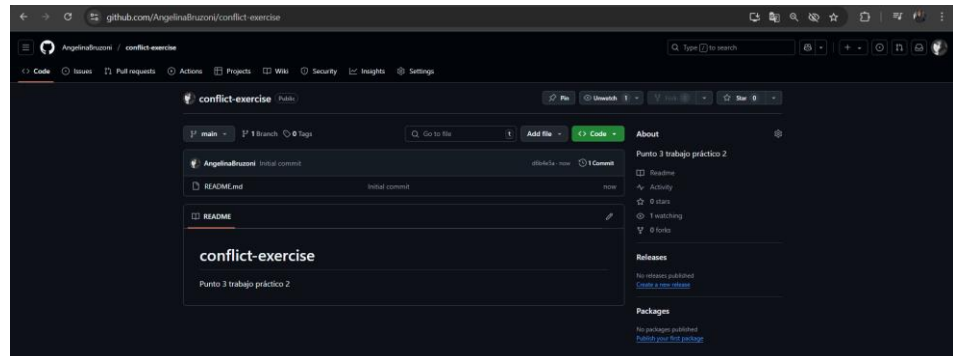
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

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

① You are creating a public repository in your personal account.

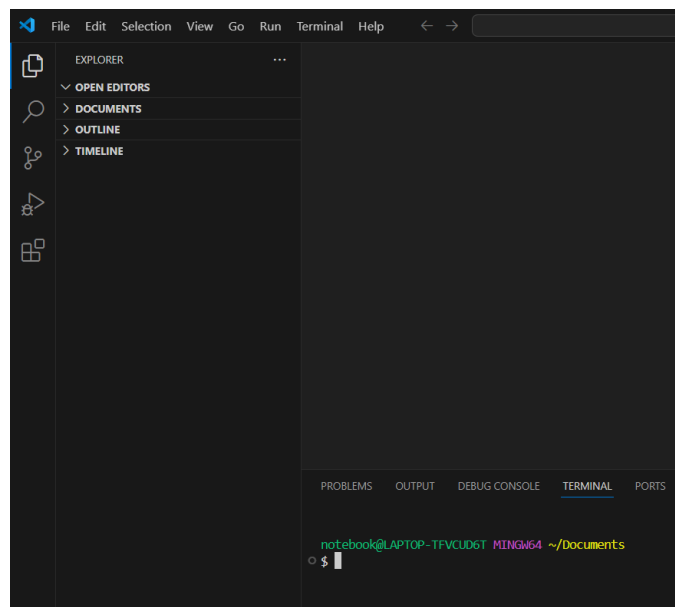
[Create repository](#)

- Hacemos clic en Create repository.

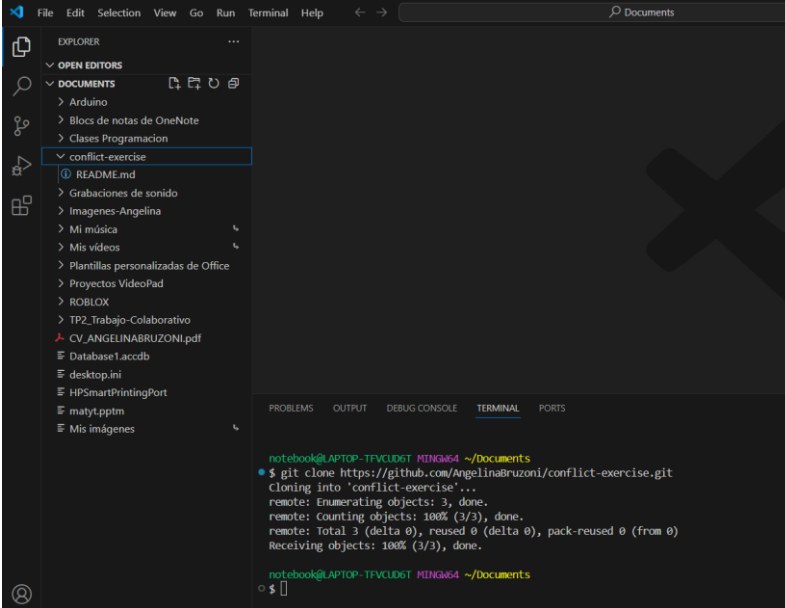


2. Clonamos el repositorio a nuestra maquina local.

- Copiamos la URL del repositorio (<https://github.com/AngelinaBruzoni/conflict-exercise.git>).
- Abrimos la terminal o línea de comandos en nuestra máquina.

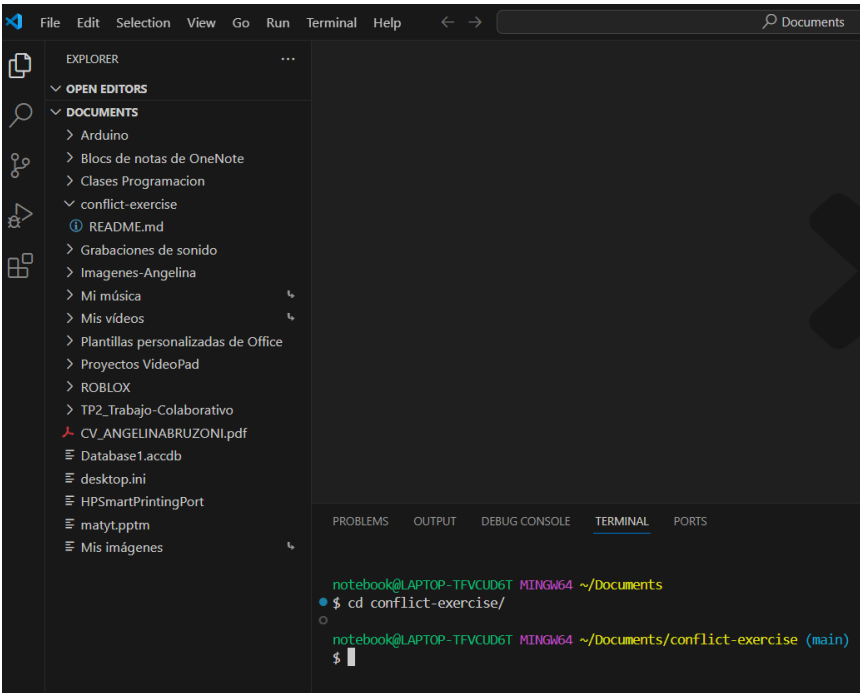


- Clonamos el repositorio usando git clone <https://github.com/AngelinaBruzoni/conflict-exercise.git>.



```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents
$ git clone https://github.com/AngelinaBruzoni/conflict-exercise.git
Cloning into 'conflict-exercise'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents
$
```

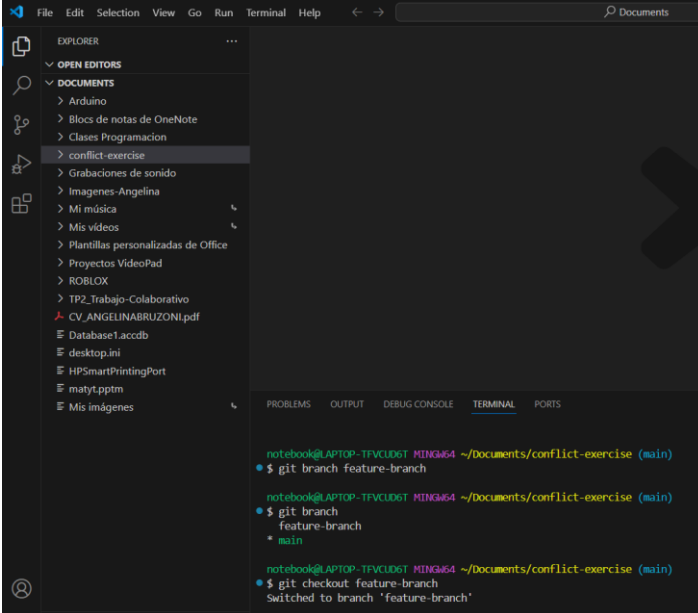
- Entramos en el directorio de nuestro repositorio: cd conflict-exercise.



```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents
$ cd conflict-exercise/
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main)
$
```

3. Creamos una nueva rama y editamos un archivo.

- Creamos una nueva rama llamada feature-branch mediante git Branch “nombre de la nueva rama”. o git checkout -b “nombre de la nueva rama”.

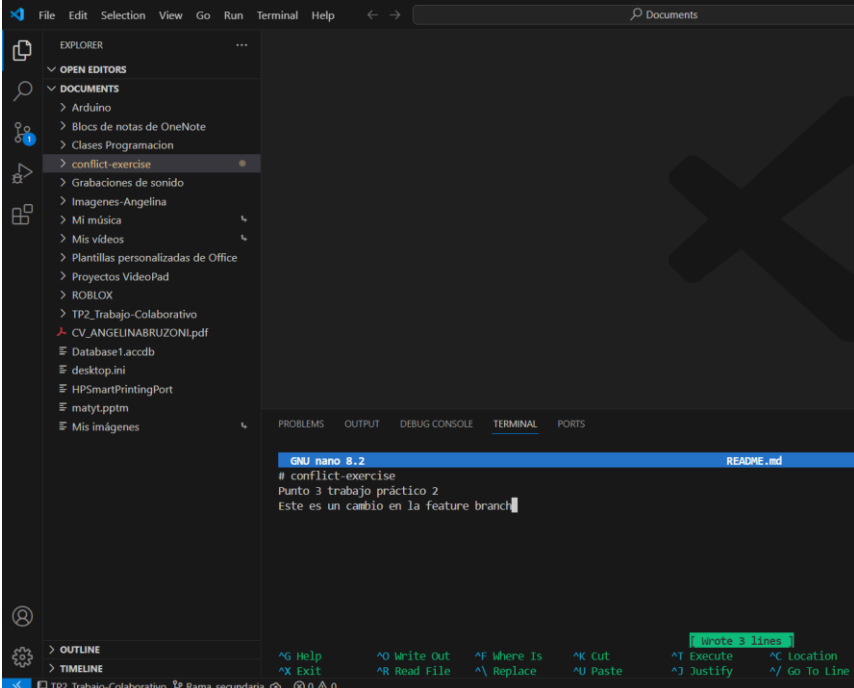


```
notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/conflict-exercise (main)
• $ git branch feature-branch

notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/conflict-exercise (main)
• $ git branch
  feature-branch
* main

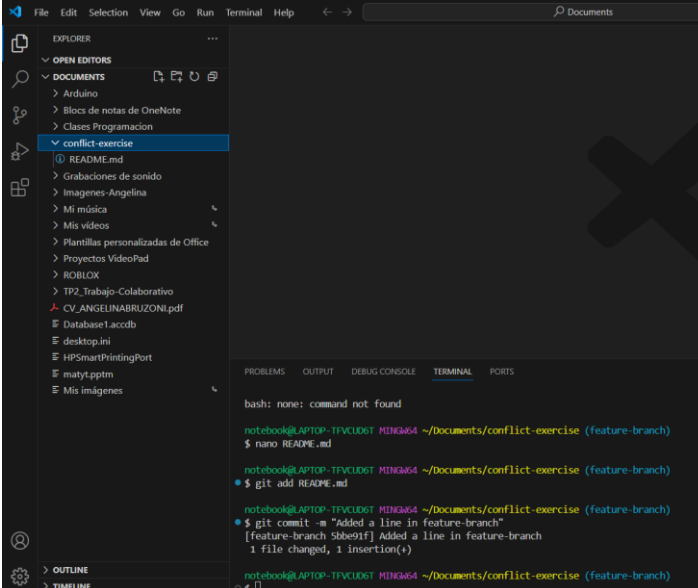
notebook@LAPTOP-TFVCLD6T MINGW64 ~/Documents/conflict-exercise (main)
• $ git checkout feature-branch
Switched to branch 'feature-branch'
```

- Abrimos el archivo README.md y añadimos una línea nueva: Este es un cambio en la feature branch.



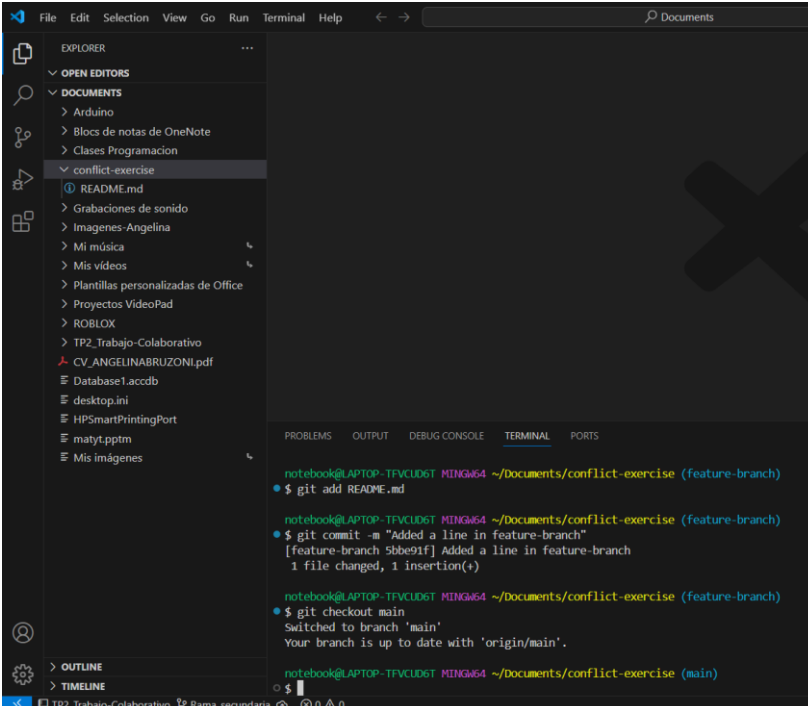
```
GNU nano 2.9.2 README.md
# conflict-exercise
Punto 3 trabajo práctico 2
Este es un cambio en la feature branch
```

- Guardamos los cambios con `git add README.md` y realizamos `git commit -m "Added a line in feature-branch"`.



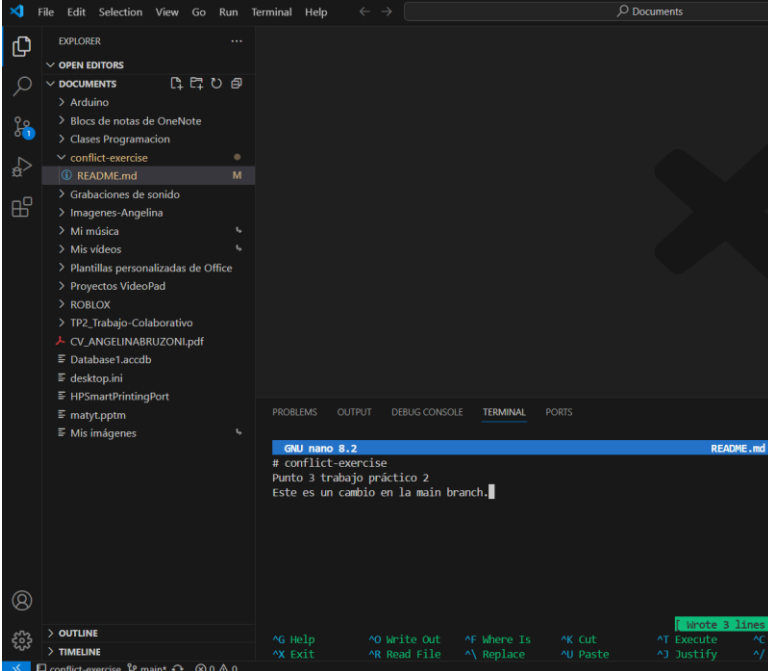
```
bash: none: command not found
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$ nano README.md
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$ git add README.md
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
[feature-branch 5bbe91f] Added a line in feature-branch
1 file changed, 1 insertion(+)
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$
```

- Volvemos a la rama principal con `git checkout main`.



```
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$ git add README.md
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
[feature-branch 5bbe91f] Added a line in feature-branch
1 file changed, 1 insertion(+)
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main)
$
```


- Editamos el archivo README.md de nuevo, añadiendo la siguiente línea: Este es un cambio en la main Branch, guardamos con git add README.md y confirmamos el cambio con git commit -m "Added a line in main Branch".



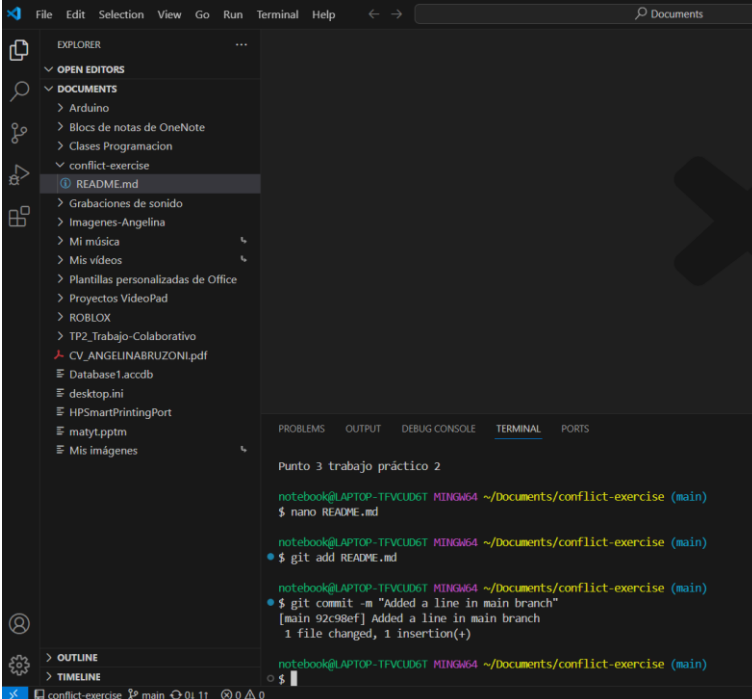
```
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DOCUMENTS
  > Arduino
  > Blocs de notas de OneNote
  > Clases Programacion
  > conflict-exercise
    README.md M
  > Grabaciones de sonido
  > Imagenes-Angelina
  > Mi música
  > Mis videos
  > Plantillas personalizadas de Office
  > Proyectos VideoPad
  > ROBLOX
  > TP2_Trabajo-Colaborativo
  > CV_ANGELINABRUZONI.pdf
  > Database1.accdb
  > desktop.ini
  > HPSSmartPrintingPort
  > matyt.pptm
  > Mis imágenes
OUTLINE
TIMELINE
conflict-exercise main 0 0 0
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

GNU nano 8.2 README.md

```
# conflict-exercise
Punto 3 trabajo práctico 2
Este es un cambio en la main branch.
```

wrote 3 lines



```
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DOCUMENTS
  > Arduino
  > Blocs de notas de OneNote
  > Clases Programacion
  > conflict-exercise
    README.md
  > Grabaciones de sonido
  > Imagenes-Angelina
  > Mi música
  > Mis videos
  > Plantillas personalizadas de Office
  > Proyectos VideoPad
  > ROBLOX
  > TP2_Trabajo-Colaborativo
  > CV_ANGELINABRUZONI.pdf
  > Database1.accdb
  > desktop.ini
  > HPSSmartPrintingPort
  > matyt.pptm
  > Mis imágenes
OUTLINE
TIMELINE
conflict-exercise main 0 0 0
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

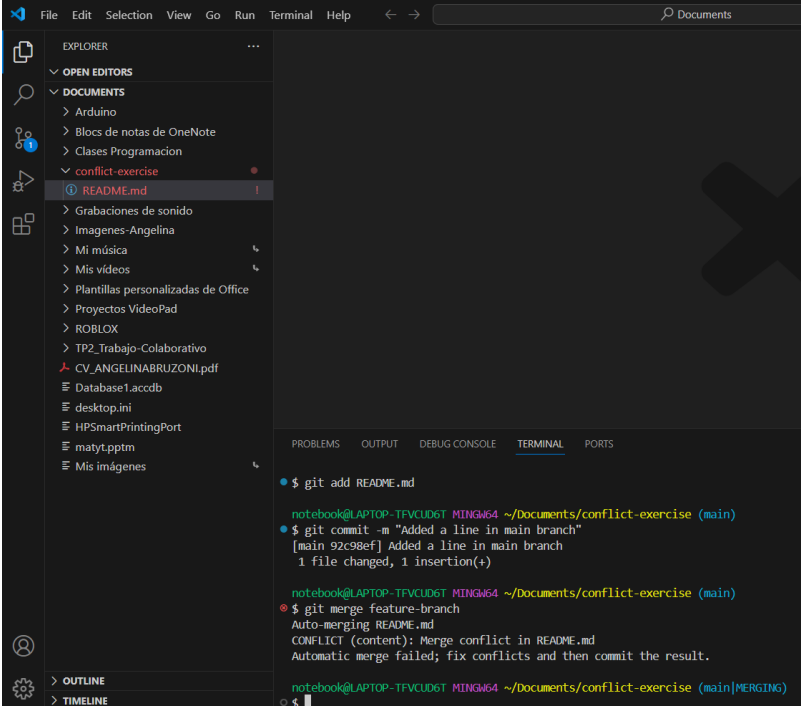
```
Punto 3 trabajo práctico 2

notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main)
$ nano README.md

notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main)
$ git add README.md

notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main)
$ git commit -m "Added a line in main branch"
[main 92c98ef] Added a line in main branch
1 file changed, 1 insertion(+)
```

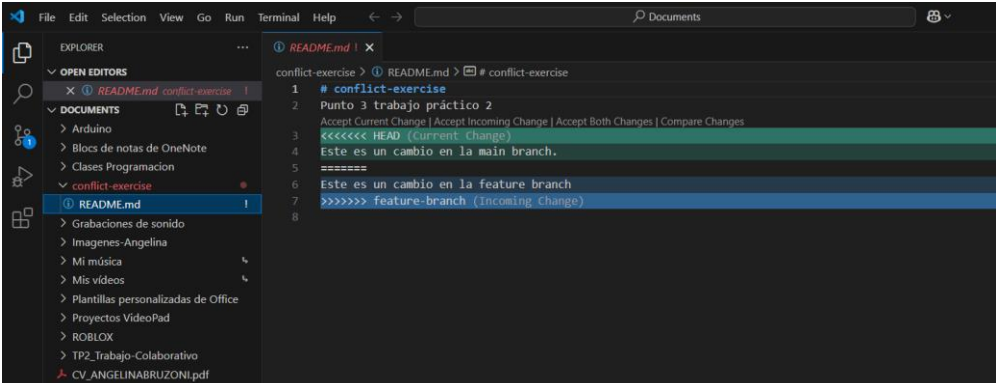
- Realizamos un Merge de la feature-branch en la rama main: git merge feature-branch.



```
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DOCUMENTS
  > Arduino
  > Blocs de notas de OneNote
  > Clases Programacion
  > conflict-exercise
    ① README.md
  > Grabaciones de sonido
  > Imagenes-Angelina
  > Mi música
  > Mis videos
  > Plantillas personalizadas de Office
  > Proyectos VideoPad
  > ROBLOX
  > TP2_Trabajo-Colaborativo
  > CV_ANGELINABRUZONI.pdf
  Database1.accdb
  desktop.ini
  HPSmartPrintingPort
  maty1.pptm
  Mis imágenes
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
$ git add README.md
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main)
$ git commit -m "Added a line in main branch"
[main 92c98ef] Added a line in main branch
1 file changed, 1 insertion(+)
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/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.
notebook@LAPTOP-TFVCUD6T MINGW64 ~/Documents/conflict-exercise (main|MERGING)
$
```

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

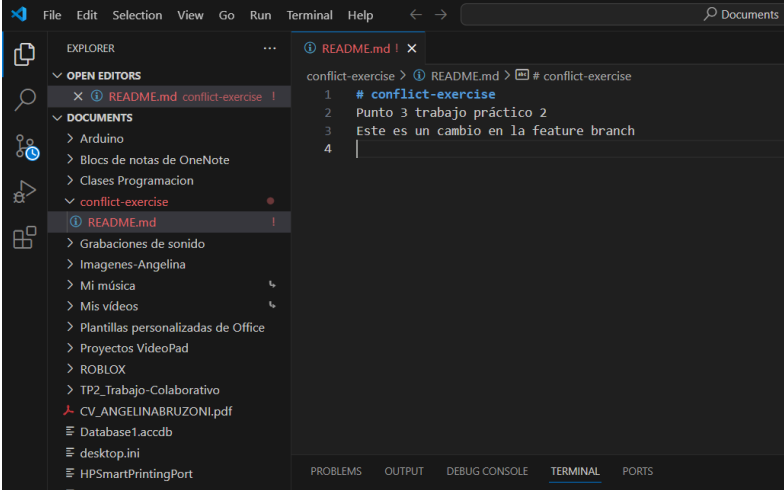
- Para resolver el conflicto, abrimos el archivo README.md y veremos:



```
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
  X ① README.md conflict-exercise
DOCUMENTS
  > Arduino
  > Blocs de notas de OneNote
  > Clases Programacion
  > conflict-exercise
    ① README.md
  > Grabaciones de sonido
  > Imagenes-Angelina
  > Mi música
  > Mis videos
  > Plantillas personalizadas de Office
  > Proyectos VideoPad
  > ROBLOX
  > TP2_Trabajo-Colaborativo
  > CV_ANGELINABRUZONI.pdf
OUTLINE
TIMELINE
conflict-exercise > ① README.md > ① # conflict-exercise
1 # conflict-exercise
2 Punto 3 trabajo práctico 2
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)
```

Decidimos cómo resolver el conflicto. Podemos mantener ambos cambios, elegir uno de ellos, o fusionar los contenidos de alguna manera.

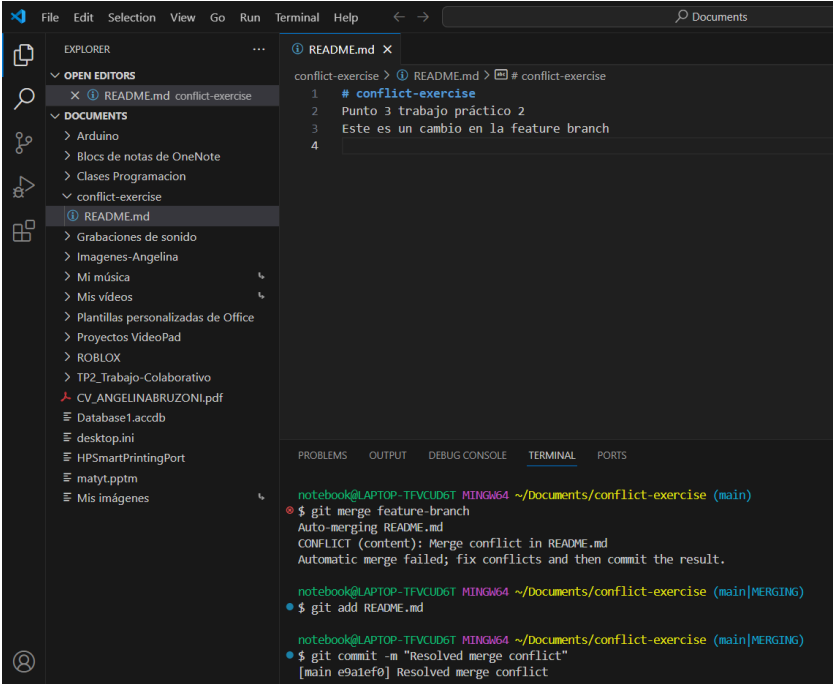
- Elegimos el cambio entrante.



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the file structure with 'conflict-exercise' selected. The main editor shows 'README.md' with a merge conflict. The conflict is between the 'conflict-exercise' branch and the 'main' branch. The conflict is resolved by choosing the 'conflict-exercise' version.

```
conflict-exercise > README.md > # conflict-exercise
1 # conflict-exercise
2 Punto 3 trabajo práctico 2
3 Este es un cambio en la feature branch
4 |
```

- Guardamos con `git add README.md` y confirmamos con `git commit -m "Resolved merge conflict"`.



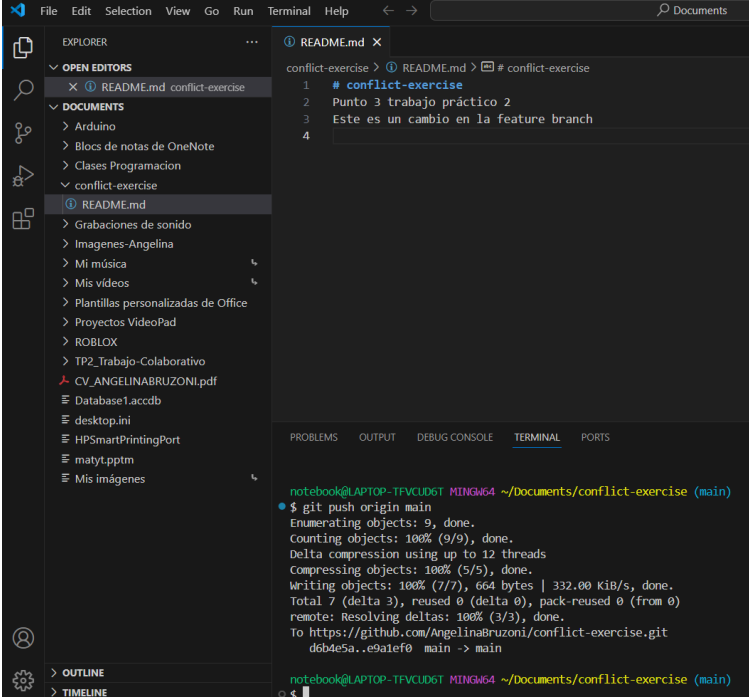
The screenshot shows the Visual Studio Code interface with the terminal panel open. The terminal shows the following commands and output:

```
notebook@LAPTOP-TEVCUD6T MINGW64 ~/Documents/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.

notebook@LAPTOP-TEVCUD6T MINGW64 ~/Documents/conflict-exercise (main|MERGING)
$ git add README.md

notebook@LAPTOP-TEVCUD6T MINGW64 ~/Documents/conflict-exercise (main|MERGING)
$ git commit -m "Resolved merge conflict"
[main e9afe0] Resolved merge conflict
```

- Subimos los cambios de la rama main y feature-branch al repositorio remoto en GitHub.

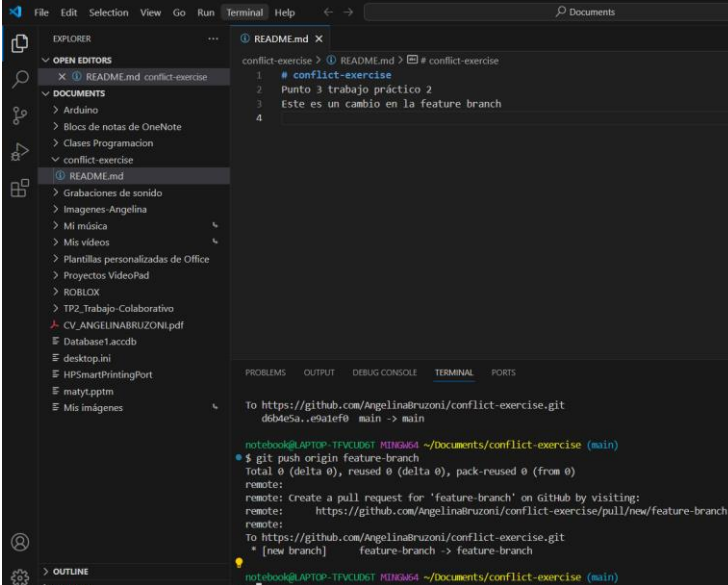


```
File Edit Selection View Go Run Terminal Help
EXPLORER
  OPEN EDITORS
    README.md conflict-exercise
  DOCUMENTS
    Arduino
    Blocs de notas de OneNote
    Clases Programacion
    conflict-exercise
    README.md
    Grabaciones de sonido
    Imagenes-Angelina
    Mi música
    Mis videos
    Plantillas personalizadas de Office
    Proyectos VideoPad
    ROBLOX
    TP2_Trabajo-Colaborativo
    CV_ANGELINABRUZONI.pdf
    Database1.acdb
    desktop.ini
    HPSSmartPrintingPort
    matyt.pptm
    Mis imágenes
  OUTLINE
  TIMELINE

conflict-exercise > README.md > # conflict-exercise
1 # conflict-exercise
2 Punto 3 trabajo práctico 2
3 Este es un cambio en la feature branch
4

notebook@LAPTOP-TFVCLUD6T MINGW64 ~/Documents/conflict-exercise (main)
$ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 664 bytes | 332.00 KiB/s, done.
Total 7 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/AngelinaBruzoni/conflict-exercise.git
d6b4e5a..e9a1ef0 main -> main

notebook@LAPTOP-TFVCLUD6T MINGW64 ~/Documents/conflict-exercise (main)
$
```



```
File Edit Selection View Go Run Terminal Help
EXPLORER
  OPEN EDITORS
    README.md conflict-exercise
  DOCUMENTS
    Arduino
    Blocs de notas de OneNote
    Clases Programacion
    conflict-exercise
    README.md
    Grabaciones de sonido
    Imagenes-Angelina
    Mi música
    Mis videos
    Plantillas personalizadas de Office
    Proyectos VideoPad
    ROBLOX
    TP2_Trabajo-Colaborativo
    CV_ANGELINABRUZONI.pdf
    Database1.acdb
    desktop.ini
    HPSSmartPrintingPort
    matyt.pptm
    Mis imágenes
  OUTLINE
  TIMELINE

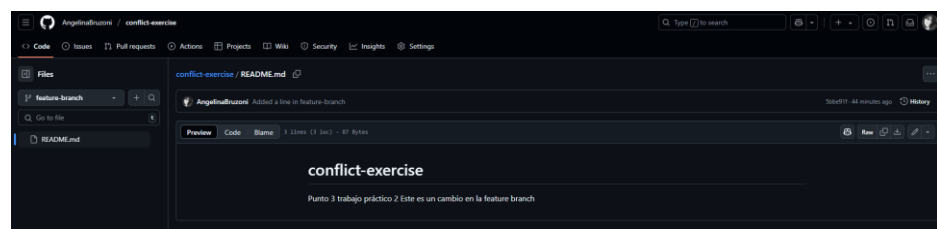
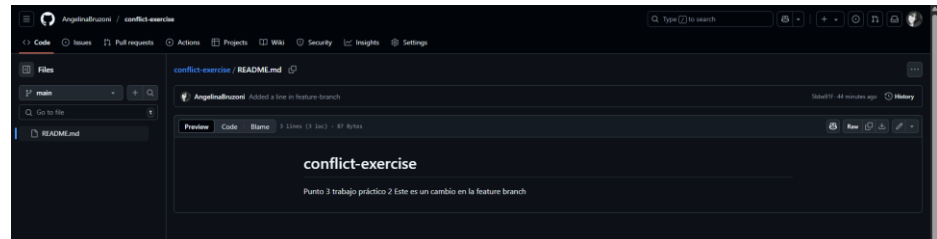
conflict-exercise > README.md > # conflict-exercise
1 # conflict-exercise
2 Punto 3 trabajo práctico 2
3 Este es un cambio en la feature branch
4

To https://github.com/AngelinaBruzoni/conflict-exercise.git
d6b4e5a..e9a1ef0 main -> main

notebook@LAPTOP-TFVCLUD6T MINGW64 ~/Documents/conflict-exercise (main)
$ git push origin feature-branch
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:   https://github.com/AngelinaBruzoni/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/AngelinaBruzoni/conflict-exercise.git
 * [new branch]   feature-branch -> feature-branch

notebook@LAPTOP-TFVCLUD6T MINGW64 ~/Documents/conflict-exercise (main)
$
```

- Nos dirigimos a nuestro repositorio en GitHub y revisamos el archivo README.md para confirmar que los cambios se hayan subido correctamente.



Revisamos el historial de commits para ver el conflicto y su resolución:

