

TRABAJO PRÁCTICO N°2: PROGRAMACIÓN (GIT/GITHUB)

Alumno: Romero Juan Agustin

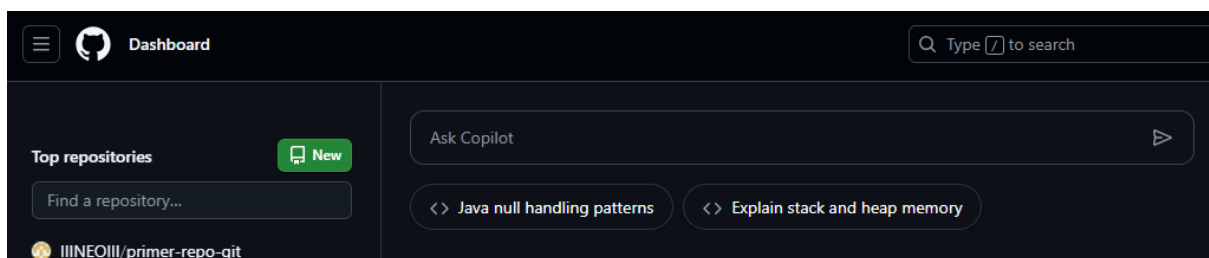
Actividades 1) Contestar las siguientes preguntas utilizando las guías y documentación proporcionada (Desarrollar las respuestas) :

1. ¿Qué es GitHub?

GitHub es una plataforma de código abierto, que permite alojar repositorios de control de versiones para que las personas almacenen su código, intercambien los mismos con otros desarrolladores o realicen trabajos en conjunto en el desarrollo de software de una forma más organizada.

2. ¿Cómo crear un repositorio en GitHub?

Para crear un repositorio en GitHub, primero debemos contar con una cuenta o crear una en caso de no contar con la misma. Esta cuenta te permitirá crear tu repositorio y configurar la misma, como por ejemplo: Si queremos que nuestro repositorio sea Privado o Público.



Habiendo ingresado a nuestra cuenta GitHub, debemos ir a “HOME” que es donde nos encontramos en el ejemplo gráfico del recorte de pantalla, en el cual

ingresamos en la caja rectangular color verde, que dice “New” haciendo click en la misma, en la cual nos dirá si queremos Crear un nuevo repositorio.

Nos saldrá un formulario para completar de configuraciones previas a crear el mismo. Luego tendremos unas líneas de comando como por ejemplo:

...or create a new repository on the comand line

Nos indica cada uno de los pasos que debemos de hacer para poder mandar desde nuestro repositorio local al repositorio que hemos creado en github y enlazarlos (recomendado cuando empiezas desde 0).

...or push an existing repository from the command line

Hace referencia a que ya tienes un repositorio local y solamente deseas mandarlo a la plataforma. Para ello deberás de tipear los siguientes comandos en la terminal bash:

```
$ git remote add origin https://github.com/tucanal/nombre-repo git
```

```
$ git push -u origin master
```

Una vez realizado esto, refrescamos la pantalla con las teclas “Ctrl + F5” o con el símbolo ubicado en la esquina superior izquierda en forma de flecha circular. Habiendo realizado este paso veremos como se ha subido nuestro repositorio exitosamente

3. ¿Cómo crear una rama en Git?

Para crear una rama en Git, utilizaremos el comando:

```
$ git branch nombreNuevaRama
```

Donde el fragmento “git branch” nos creará la rama y “nombreNuevaRama” es un ejemplo referencial en donde nosotros elegiremos como se llamará esta nueva rama que se creará a continuación.

4. ¿Cómo cambiar a una rama en Git?

Para cambiar de rama en Git, utilizaremos el comando:

```
$ git checkout nombreRama
```

En este caso al utilizar el comando “git checkout”, lo que hace este fragmento de código, es mover el apuntador HEAD a la rama “nombreRama”, que es la que utilizamos como ejemplo referencial.

En caso de que se quiera crear una nueva rama y saltar a ella en un solo movimiento, utilizaremos el comando:

```
$ git checkout -b nombre-rama
```

En el cual al agregar el fragmento “-b” hace que esta fuerce el salto directo a esa rama que llamamos “nombre-rama” en este ejemplo.

5. ¿Cómo fusionar ramas en Git?

Para fusionar dos ramas en Git:

- 1) primero debemos estar en la rama que queramos fusionar.
Ejemplo: rama “Master o Main”.

En el cual debemos movernos a ella con el comando:

```
$ git checkout master
```

- 2) Estando en la rama “Master o Main”, utilizaremos el comando:

```
$ git merge nuevaRama
```

En el cual este comando “merge” lo que hará es fusionar la rama “Master o Main” que es la rama, en la cual nos encontramos posicionado, con la rama “nuevaRama” que es la rama que le estaremos indicando que nos funcione el comando. Esta acción lo que hará es que se incorporen los cambios de “nuevaRama” en “Master o Main”.

6. ¿Cómo crear un commit en Git?

Para crear un “commit” primero debemos entender que significa este. Un “commit” es un registro en el cual guardamos los cambios realizados en nuestro repositorio en un momento dado, quedando como una marca en nuestro historial del proyecto que estamos realizando.

comando a utilizar:

```
$ git commit
```

Para realizar debemos:

- 1) Primero: Realizaremos los cambios en nuestro proyecto.
- 2) Segundo: Agregamos los cambios al área de preparación con el comando

`$git add archivo` (Agrega solamente el archivo “archivo”).

`$git add .` (Agrega todos los archivos).

- 3) Estando ya todo listo es hora de realizar un “commit” en el cual, debemos agregar un mensaje para que nosotros sepamos, que se realizó un cambio, este mensaje cumple la función de recordatorio, que en un momento dado se realizó un cambio.

`$git commit -m “Mensaje”`

Quedando guardado en el historial de nuestro repositorio, el estado actual con la leyenda que nosotros agreguemos en nuestro “mensaje”.

7. ¿Cómo enviar un commit a GitHub?

Primero debemos clonar el repositorio de GitHub a nuestra máquina local con el comando;

`$ git clone https://github.com/tu_usuario/tu_repositorio.git`

`cd tu_repositorio`

Haz cambios en los archivos del repositorio y agregalos con el comando:

`$ git add nombreArchivo` o `$ git add .`

Creemos un commit para tener un registro de nuestros cambios.

`$ git commit -m “mensaje”`

Luego debemos empujar los commits al repositorio en GitHub con el comando:

`$ git push origin nombreRama`

8. ¿Qué es un repositorio remoto?

Un repositorio remoto es una copia de tu proyecto que se encuentra alojado en una base de datos en la red. Pudiendo tener uno/varios proyectos, en los cuales podrás tener generalmente permisos de solo lectura, o de lectura y escritura. Poder colaborar con otras personas implica gestionar estos

repositorios remotos enviando y descargando datos de ellos cada vez que necesites compartir tu trabajo. Gestionar repositorios remotos incluye saber cómo añadir un repositorio remoto, eliminar los remotos que ya no son válidos, gestionar varias ramas remotas, definir si deben rastrearse o no y más.

9. ¿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 repositorio remoto, ejemplo:

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

Donde en “nombre” agregaremos el nombre que queramos y en “url”, agregaremos la dirección “URL”. La ventaja del nombre es que esto nos permitirá buscarlo o llamarlo en la terminal de una forma más rápida y fácil que colocando la dirección “URL”, que esta suele ser más larga y compleja de tipear.

Para asegurarnos de que el remoto se ha agregado correctamente, verificamos usando el comando:

```
$ git remote -v
```

Esto mostrará una lista de todos los remotos configurados con sus respectivas “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.

10. ¿Cómo empujar cambios a un repositorio remoto?

Para eso antes de empujar los cambios, es una buena práctica obtener los últimos cambios del repositorio remoto para evitar conflictos con el comando:

```
$ git pull origin nombreRama
```

Empuja tus cambios al repositorio remoto con el comando:

```
$ git push origin nombreRama
```

11. ¿Cómo tirar de cambios de un repositorio remoto?

Para poder tirar los cambios del repositorio remoto, usa el comando:

```
$ git pull
```

Para descargar y fusionar los cambios del repositorio remoto con tu rama local con el comando:

```
$ git pull origin nombreRama
```

Por ejemplo, si estás trabajando en la rama “ Master o Main” usamos el comando:

```
$ git pull origin master
```

12. ¿Qué es un fork de repositorio?

En github existen repositorios de otros usuarios en el cual nosotros podemos hacer una copia y modificarlas, para ello github nos ofrece la implementación de fork.

En el cual seleccionamos un repositorio de nuestro interes, este puede ser cualquier proyecto de código abierto que quieras modificar o utilizar como base para tu propio trabajo.

1ro) En la página del repositorio, busca el botón "Fork" ubicado en la parte superior derecha de la página.

2do) Haz clic en el botón "Fork". Esto creará una copia del repositorio en tu propia cuenta de GitHub.

Esta copia será independiente del repositorio original. Cualquier cambio que realices en tu fork no afectará al repositorio original, los cambios que hagamos no irán al repositorio original del autor de ese proyecto, si no a nuestra copia local. Entonces solo resta clonar ese repositorio que está en nuestro canal para poder trasladarlo a alguna carpeta deseada y seguir trabajando desde nuestro repositorio local.

13. ¿Cómo crear un fork de un repositorio?

Una forma por ejemplo es crear una cuenta adicional en github para que puedas desde allí hacer un Fork de alguno de los repositorios de tu cuenta original, clonarlo en alguna carpeta y en algún archivo que nosotros notamos que podría mejorarse algo, procedemos a realizar algún cambio, lo agregamos al escenario y lo cometemos (todo esto desde nuestro repositorio local remoto) y luego mandamos los cambios a el forking que hicimos con un git push origin master.

Ese cambio que hemos hecho queremos que sea visto por el creador del repositorio original (en este caso nosotros, desde nuestra cuenta original), debemos hacérselo notar puesto que en el repositorio original los cambios que hicimos previamente, no se verán reflejados.

14. ¿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 porqué ese cambio que hemos realizado nosotros, sería considerado como algo que a el repositorio original le vendría bien agregarlo.

15. ¿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 clickear en Merge pull request y de esta manera sumará a su repositorio los cambios que hizo un usuario (en modo de ayuda).

16. ¿Qué es un etiqueta en Git?

Como muchos VCS, Git tiene la posibilidad de etiquetar puntos específicos del historial como importantes. Esta funcionalidad se usa típicamente para marcar versiones de lanzamiento (v1.0, por ejemplo).

17. ¿Cómo crear una etiqueta en Git?

En Git utiliza dos tipos principales de etiquetas: ligeras y anotadas. Una etiqueta ligera es muy parecida a una rama que no cambia, simplemente es un puntero a un commit específico. Sin embargo, las etiquetas anotadas se guardan en la base de datos de Git como objetos enteros. Tienen un checksum; contienen el nombre del etiquetador, correo electrónico y fecha; tienen un mensaje asociado; y pueden ser firmadas y verificadas con GNU Privacy Guard (GPG). Normalmente se recomienda que crees etiquetas anotadas, de manera que tengas toda esta información; pero si quieres una etiqueta temporal o por alguna razón no estás interesado en esa información, entonces puedes usar las etiquetas ligeras.

18. ¿Cómo enviar una etiqueta a GitHub?

Primero, debes crear una etiqueta en tu repositorio local. Puedes crear una etiqueta anotada (que incluye información adicional como el autor y la fecha) o una etiqueta ligera (simplemente un puntero a un commit).

Ej de etiqueta ligera:

```
$ git tag v1.0
```

Podes verificar las etiquetas que has creado localmente con:

```
$git tag
```

Una vez que has creado la etiqueta en tu repositorio local, necesitas empujarla al repositorio remoto en GitHub.

Puedes hacer esto con el siguiente comando:

```
$ git push origin v1.0
```

(origin es el nombre del repositorio remoto, por defecto suele ser origin y v1.0 es el nombre de la etiqueta.)

Para empujar todas las etiquetas creadas, usar el comando:

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

19. ¿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

20. ¿Cómo ver el historial de Git?

Esto lo conseguimos con el comando:

```
$ git log
```

Con tipear este comando en el bash de Git podremos apreciar el histórico de commits, estando situados en la carpeta de nuestro proyecto.

El listado de commits estará invertido, es decir, los últimos realizados aparecen los primeros.

El comando:

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

Es una forma compacta de visualizar el historial de commits en un repositorio Git.

Muestra un resumen conciso de los commits recientes, con cada commit representado en una sola línea. Si tu proyecto ya tiene muchos commits, quizás no quieras mostrarlos todos, ya que generalmente no querrás ver cosas tan antiguas como el origen del repositorio.

Para ver un número de logs determinado introducimos ese número como opción, con el signo "-" delante (-1, -8, -12...). Por ejemplo, esto muestra los últimos tres commits con el comando:

```
$ git log -3
```

Si queremos que el log también nos muestre los cambios en el código de cada commit podemos usar la opción -p. Esta opción genera una salida mucho más larga, por lo que seguramente nos tocará movernos en la salida con los cursores y usaremos CTRL + Z para salir.

```
$ git log -2 -p
```

21. ¿Cómo buscar en el historial de Git?

Para buscar en el historial de commits de Git, podemos utilizar varios comandos y opciones que te permiten filtrar y localizar commits específicos.

Para buscar commits que contengan una palabra o frase específica en el mensaje de commit, usa git log con la opción `--grep`:

```
$ git log --grep="palabra clave"
```

Para buscar commits que han modificado un archivo específico, usa git log seguido del nombre del archivo:

```
$git log -- nombreArchivo
```

Para buscar commits en un rango de fechas específico, usa las opciones `--since` y `--until`:

```
$ git log --since="2025-01-01" --until="2025-01-31"
```

Para encontrar commits hechos por un autor específico, usa `--author`:

```
$ git log --author="NombreAutor"
```

22. ¿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 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 Aplicación:

Quita del stage todos los archivos y carpetas del proyecto:

```
$ git reset
```

Quita del stage el archivo indicado:

```
$ git reset nombreArchivo
```

Quita del stage todos los archivos de esa carpeta:

```
$ git reset nombreCarpeta/
```

Quita ese archivo del stage (que a la vez está dentro de una carpeta).

\$ git reset nombreCarpeta/nombreArchivo

Quita todos los archivos que cumplan con la condición indicada previamente dentro de esa carpeta del stage.

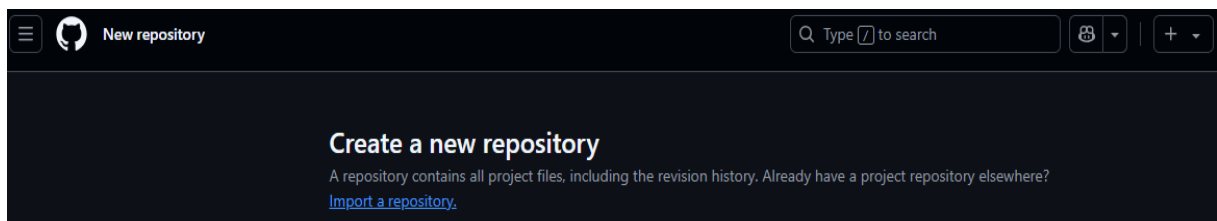
\$ git reset nombreCarpeta/*.extensión

23. ¿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.

24. ¿Cómo crear un repositorio privado en GitHub?

- 1) Ingresa a la página de GitHub y ve a la parte de creación de repositorio
- 2) En la esquina superior de la derecha de la página, has click sobre el simbolo “+” y selecciona “New Repository” o has click en “New”:



Completamos la información del repositorio:

-nombre del repositorio

-descripción


Seleccionar la configuración de privacidad:

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 ***

 IIINEOIII /

Great repository names are short and memorable. Need inspiration? How about **silver-palm-tree** ?

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

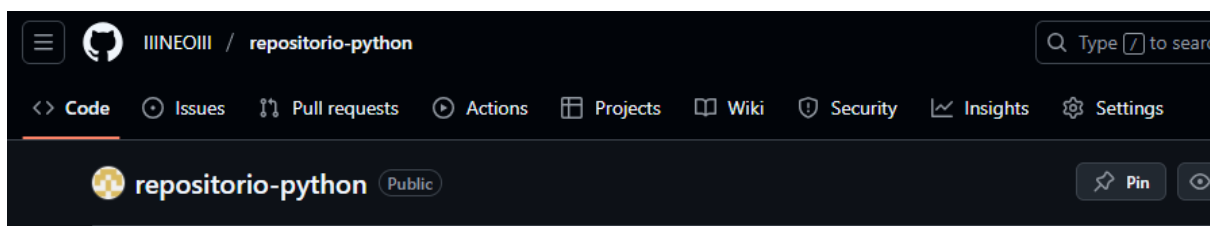
.gitignore template: **None** ▼

Como podemos apreciar en la imagen, en la parte de “Private” en la cual está tildada con color azul. Esto generará que sea accesible para los colaboradores que tu elijas.

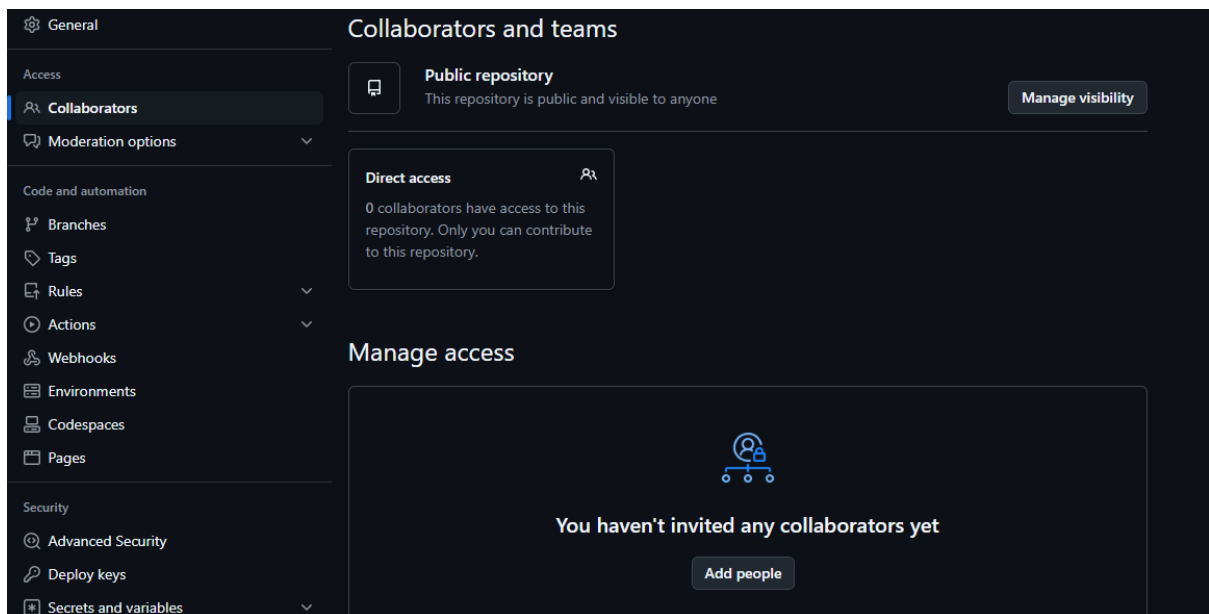
25. ¿Cómo invitar a alguien a un repositorio privado en GitHub?

Invitar a alguien a un repositorio privado en GitHub es un proceso sencillo, pero requiere permisos adecuados.

Accede al repositorio, haz clic en la pestaña "Settings" que se encuentra, en la esquina superior derecha



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



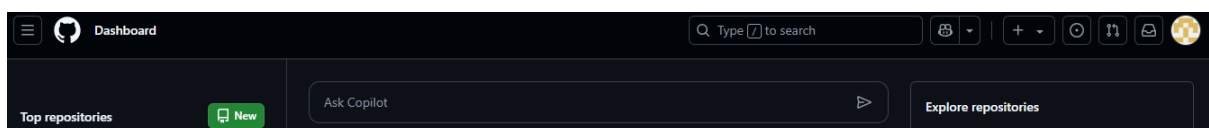
En la sección "Collaborators", haz clic en el botón "Add people" e ingresa el nombre de usuario de GitHub de la persona que deseas invitar. Selecciona el nivel de acceso que deseas otorgar: Read, Triage, Write, Maintain, o Admin. Haz clic en el botón "Add" para enviar la invitación.

26. ¿Qué es un repositorio público en GitHub?

Un repositorio público en GitHub es un repositorio cuyo contenido es accesible a cualquier persona en Internet. A diferencia de un repositorio privado, que está restringido a un grupo específico de colaboradores, un repositorio público permite que cualquier persona pueda ver, clonar y si tienen los permisos adecuados, contribuir al proyecto.

27. ¿Cómo crear un repositorio público en GitHub?

- 1) Inicia sesión en GitHub
- 2) Ingresa a la página de creación de repositorios
- 3) En la esquina superior derecha de la página principal, debes hacer clic en el botón "+" y seleccionar "New Repository" o hacer clic en "New" que es la casilla color verde:



Completar la información del repositorio

- nombre del repositorio.
- descripción.

Seleccionar la configuración de privacidad:

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 ***

IIIINEOIII /

Great repository names are short and memorable. Need inspiration? How about [reimagined-octo-guide](#) ?

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.](#)

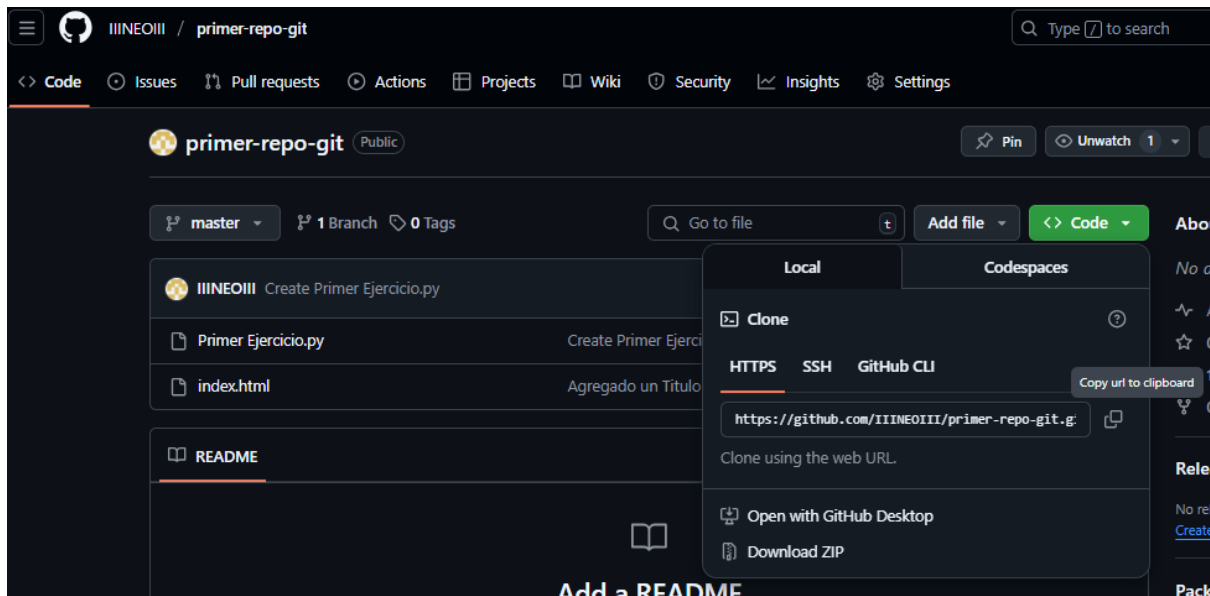
i You are creating a public repository in your personal account.

Create repository

tildando en “Public” con color azul, haremos que nuestro repositorio sea publico y visible para todos.

28. ¿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" en color verde:



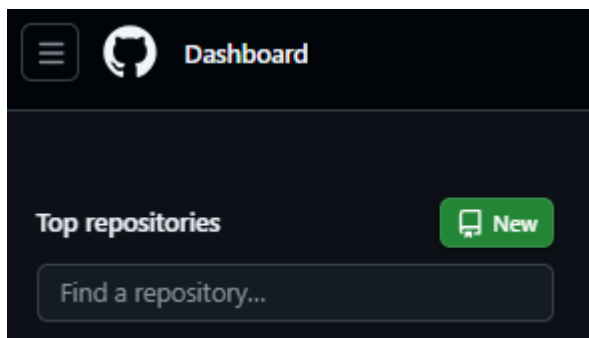
Puedes copiar la URL directamente haciendo clic en el botón de copiar a la derecha de la URL, ejemplo:

<https://github.com/IIINEOIII/primer-repo-git.git>

2) Realizar la siguiente actividad:

Crear un repositorio.

Selecciona la casilla color verde que dice “New”.



Dale un nombre al repositorio.

Elige que el repositorio sea público.

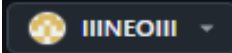
Inicializa el repositorio con un archivo.

Create a new repository

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

Required fields are marked with an asterisk (*).

Owner *



Repository name *

/ systemRepositorio

systemRepositorio is available.

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

Description (optional)

Este repositorio es de Arquitectura y OS



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](#).

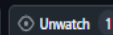


systemRepositorio

Public



Pin



Unwatch

1

main



1 Branch



0 Tags

Go to file



Add file

<> Code



IINEOIII Initial commit

9299bed · now

1 Commit



README.md

Initial commit

now



README



systemRepositorio

Este repositorio es de Arquitectura y OS

- Agregando un Archivo o 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).

Clonar el Repositorio a tu Computadora

Utilizando el comando:

\$ git clone <https://github.com/IIINEOIII/systemRepositorio.git>

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git (master)
$ git clone https://github.com/IIINEOIII/systemRepositorio.git
Cloning into 'systemRepositorio'...
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.
```

Agregar un Archivo al Repositorio

- 1) Ingresa a la carpeta del repositorio con el comando:

\$ cd systemRepositorio

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git (master)
$ cd systemRepositorio
```

- 2) Crear un archivo simple llamado "systemRepositorio"

\$ echo "Este es mi archivo de prueba" > system.txt

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (main)
$ echo "Este es mi archivo de prueba" > system.txt
```

- 3) Agregar el archivo al área de preparación:

\$ git add .

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (main)
$ git add .
```

4) Realiza el commit con un mensaje descriptivo:

\$ git commit -m "Agregando system.txt"

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (main)
$ git commit -m "Agregando system.txt"
[main be41d29] Agregando system.txt
1 file changed, 1 insertion(+)
create mode 100644 system.txt
```

5) Sube los cambios al repositorio en GitHub:

\$ git push origin main

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 319 bytes | 159.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/IIINEOIII/systemRepositorio.git
9299bed..be41d29 main -> main
```

Crear una Nueva Rama (Branch)

1) Crea una nueva rama llamada Nueva-Rama:

\$ git checkout -b nueva-rama

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (main)
$ git checkout -b nueva-rama
Switched to a new branch 'nueva-rama'

kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
```

2) Agrega un nuevo archivo o edita uno existente:

\$ echo "Este es un archivo en nueva rama" > system.txt

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ echo "Este es un archivo en nueva rama" > system.txt
```

3) Agrega y confirma los cambios:

```
$ git add .
$ git commit -m "Agregando archivo en nueva rama"
```

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git add .
git commit -m "Agregando archivo en nueva rama"
warning: in the working copy of 'system.txt', LF will be replaced by CRLF the next time Git touches it
[nueva-rama 3770ef5] Agregando archivo en nueva rama
1 file changed, 1 insertion(+), 1 deletion(-)
```

4) Sube la rama al repositorio en GitHub:

```
$ git push origin nueva-rama
```

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git push origin nueva-rama
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 330 bytes | 165.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'nueva-rama' on GitHub by visiting:
remote:   https://github.com/IIINEOIII/systemRepositorio/pull/new/nueva-rama
remote:
To https://github.com/IIINEOIII/systemRepositorio.git
 * [new branch]      nueva-rama -> nueva-rama
```

```
$ git status
```

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git status
On branch nueva-rama
nothing to commit, working tree clean
```

The screenshot shows the GitHub web interface for the repository 'systemRepositorio' by user 'IIINEOIII'. The repository is public and has 2 branches and 0 tags. A notification banner indicates that the 'nueva-rama' branch has recent pushes 41 seconds ago, with a 'Compare & pull request' button. Below the notification, the repository's commit history is displayed. The most recent commit is 'Agregando system.txt' by 'IIINEOIII', committed 32 minutes ago. The commit message is 'Agregando system.txt'. The commit hash is 'be41d29'. The commit is linked to a pull request. The repository also contains a 'README.md' file, which was committed 1 hour ago.

| Commit Hash | Commit Message | Author | Time Ago |
|-------------|----------------------|-----------|----------------|
| be41d29 | Agregando system.txt | IIINEOIII | 32 minutes ago |
| | Initial commit | | 1 hour ago |

| PROGRAMACION > Repositorio-git > systemRepositorio > | | | | |
|--|------------|-----------------------|-----------------------|--------|
| | Nombre | Fecha de modificación | Tipo | Tamaño |
| | .git | 4/4/2025 00:20 | Carpeta de archivos | |
| | README.md | 3/4/2025 23:01 | Archivo de origen ... | 1 KB |
| | system.txt | 4/4/2025 00:13 | Documento de te... | 1 KB |

Creando Branchs

Crear una Branch

\$ git branch

```
kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git branch
main
* nueva-rama
```

Realizar cambios

\$ echo "Modificacion echa desde nueva-rama" > system.txt

\$ git add .

\$ git status

```
kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ echo "Modificacion echa desde nueva-rama" > system.txt

kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git add .
warning: in the working copy of 'system.txt', LF will be replaced by CRLF the next time Git touches it

kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git status
On branch nueva-rama
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   system.txt
```

agregar un archivo

```

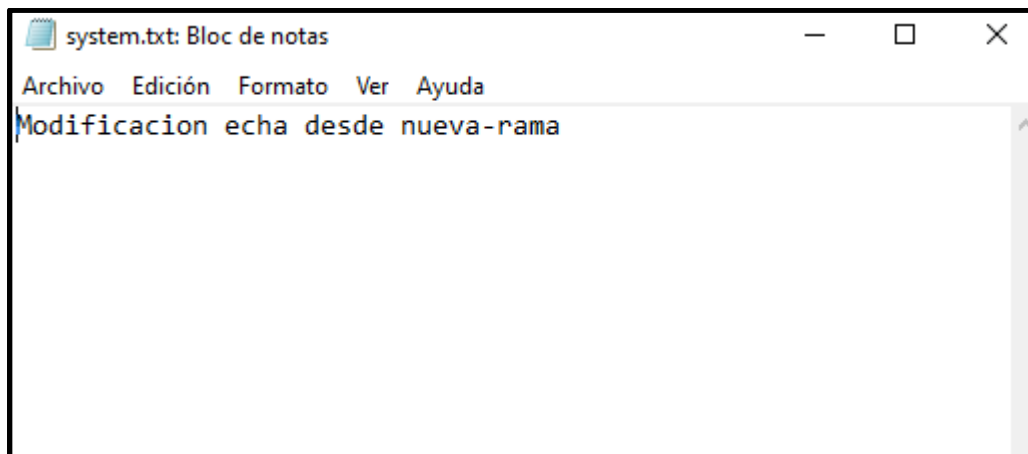
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git commit -m "Agregando modificacion desde nueva-rama"
[nueva-rama df85f0f] Agregando modificacion desde nueva-rama
1 file changed, 1 insertion(+), 1 deletion(-)

kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git status
On branch nueva-rama
nothing to commit, working tree clean

```

\$ git commit -m "Agregando modificacion desde nueva-rama"

\$ git status



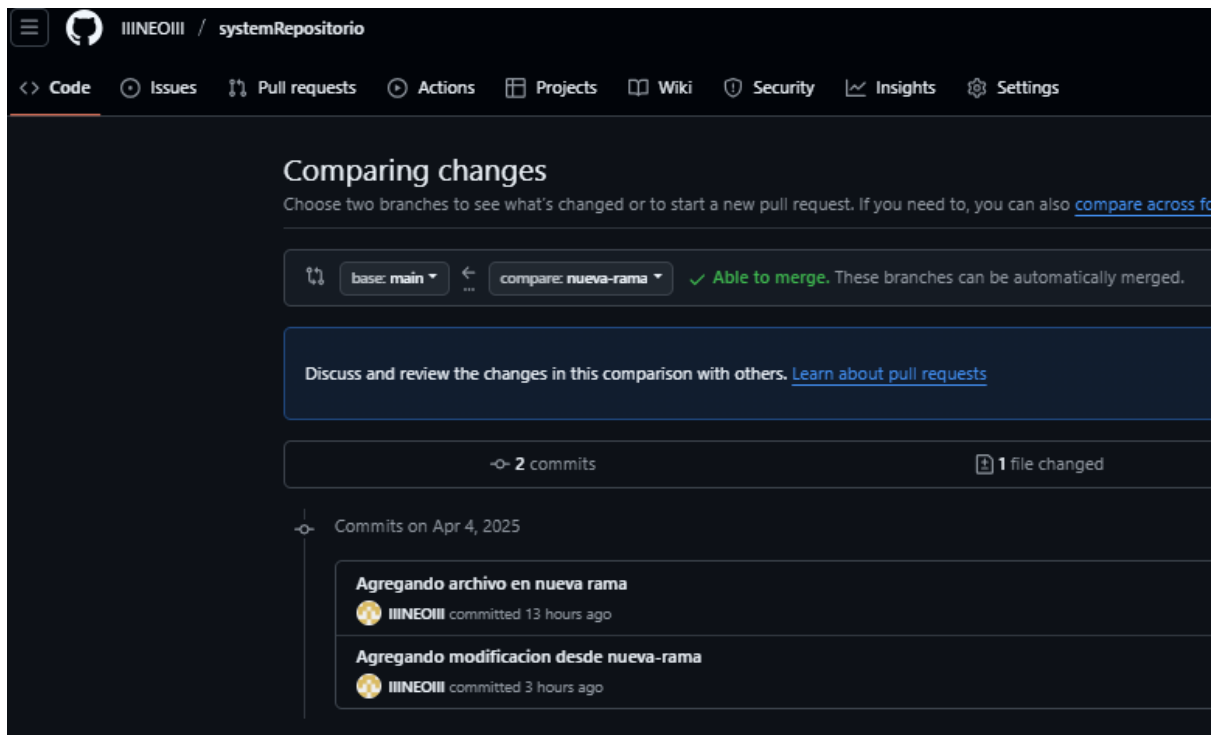
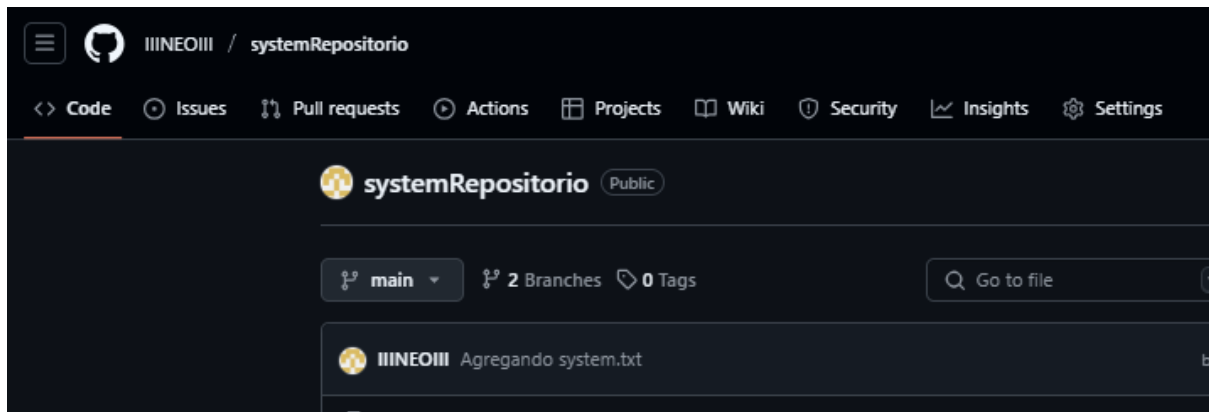
Subir la Branch

```

kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/systemRepositorio (nueva-rama)
$ git push origin nueva-rama
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 337 bytes | 168.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/IIINEOIII/systemRepositorio.git
3770ef5..df85f0f  nueva-rama -> nueva-rama

```

\$ git push origin nueva-rama

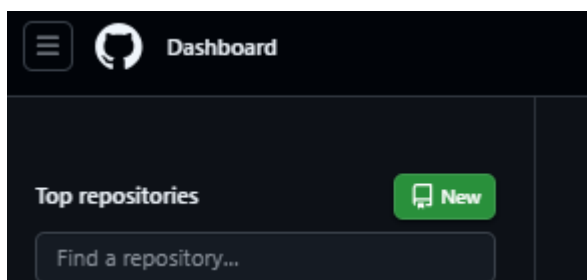


3) Realizar la siguiente actividad:

Paso 1) Crear un repositorio en GitHub

1) Ve a GitHub e inicia sesión en tu cuenta.

2) Haz clic en el botón "New" o "Create repository" para crear un nuevo repositorio.




- 3) Asigna un nombre al repositorio, por ejemplo, conflict-exercise.
- 4) Opcionalmente, añade una descripción.
- 5) Marca 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 * **Repository name ***


 IINEOIII / conflict-exercise


✔ conflict-exercise is available.

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

Description (optional)

Crear repositorio en Git

☒  **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

- 6) Haz clic en "Create repository".

Paso 2: Clonar el repositorio a tu máquina local

- 1) Copia la URL del repositorio (usualmente algo como <https://github.com/tuusuario/conflict-exercise.git>).

<https://github.com/IINEOIII/conflict-exercise.git>

- 2) Abre la terminal o línea de comandos en tu máquina.

- 3) Clona el repositorio usando el comando: `git clone` <https://github.com/tuusuario/conflict-exercise.git>

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git (master)
$ git clone https://github.com/IIINEOIII/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.
```

\$ git clone https://github.com/IIINEOIII/conflict-exercise.git

4) Entra en el directorio del repositorio: cd conflict-exercise

\$ cd conflict-exercise

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git (master)
$ cd conflict-exercise

kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main)
$
```

Paso 3: Crear una nueva rama y editar un archivo

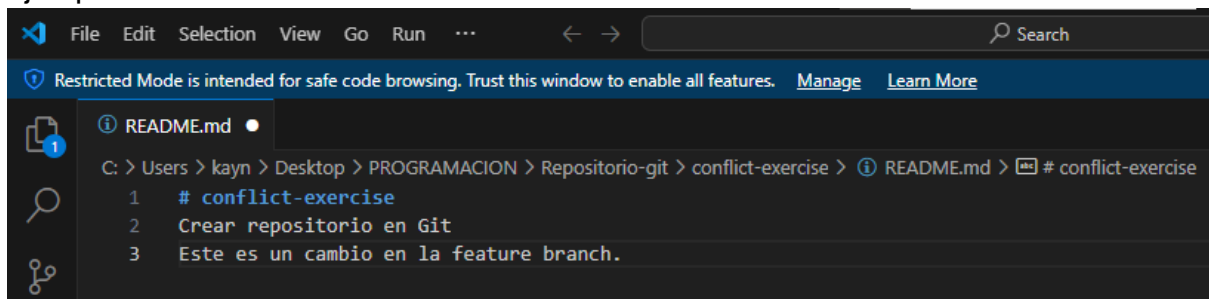
1) Crea una nueva rama llamada feature-branch:

\$ git checkout -b feature-branch

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (feature-branch)
$
```

2) 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.



3)Guarda los cambios y haz un commit:

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

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (feature-branch)
$ git add README.md
```

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

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (feature-branch)
$ git commit -m "Added a line in feature-branch"
On branch feature-branch
nothing to commit, working tree clean
```

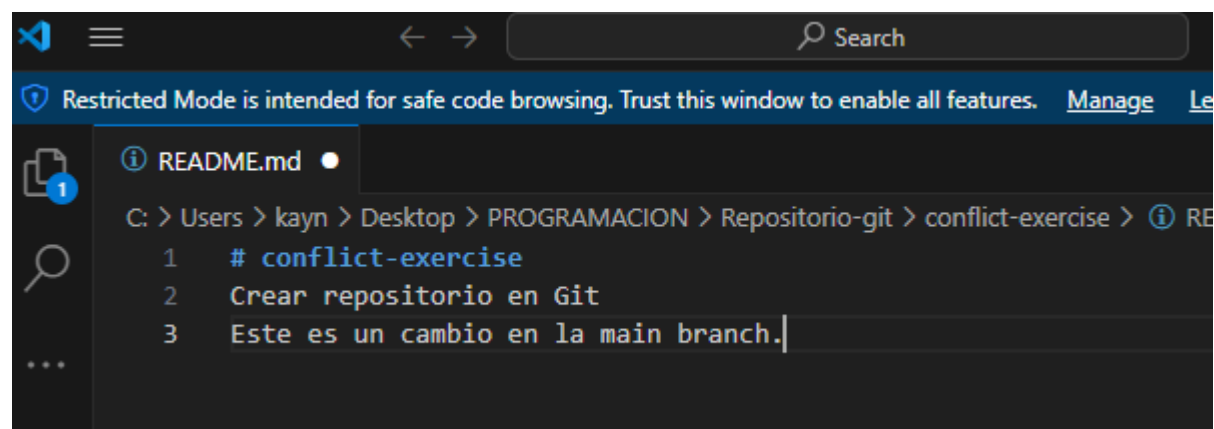
Paso 4: Volver a la rama principal y editar el mismo archivo

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

```
$ git checkout main
```

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

2)Edita el archivo README.md de nuevo, añadiendo una línea diferente: Este es un cambio en la main branch.



3)Guarda los cambios y haz un commit:

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

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main)
$ git add README.md
```

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

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main)
$ git commit -m "Added a line main branch"
[main 8d0b9a8] Added a line main branch
1 file changed, 1 insertion(+)
```

Paso 5: Hacer un merge y generar un conflicto

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

\$ git merge feature-branch

```
kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/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.
```

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

Paso 6: Resolver el conflicto

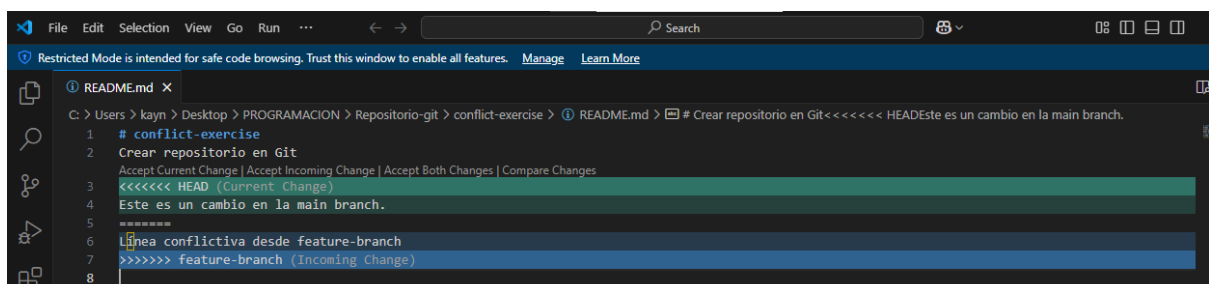
1)Abre el archivo README.md en tu editor de texto. Verás algo similar a esto:

<<<<<< HEAD

Este es un cambio en la main branch.

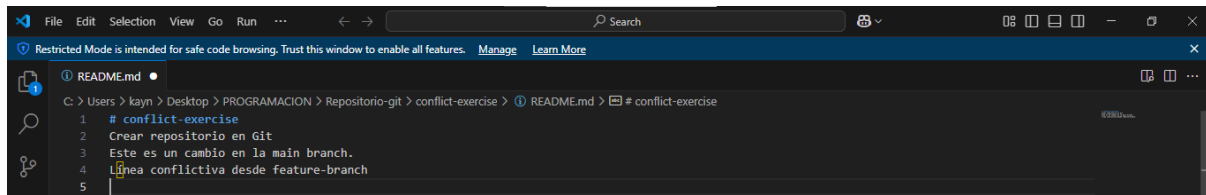
===== Este es un cambio en la feature branch.

>>>>>> feature-branch



2)Decide cómo resolver el conflicto.Puedes mantener ambos cambios, elegir uno de ellos, o fusionar los contenidos de alguna manera.

3)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).



4)Añade el archivo resuelto y completa el merge:

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

```
kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main|MERGING)
$ git add README.md
```

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

```
kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main|MERGING)
$ git commit -m "Resolved merge conflict"
[main ad40365] Resolved merge conflict
```

Paso 7: Subir los cambios a GitHub

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

```
$ git push origin main
```

```
kayn@DESKTOP-7M27OV8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/conflict-exercise (main)
$ git push origin main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 4 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (13/13), 1.24 KiB | 316.00 KiB/s, done.
Total 13 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/IIINE0III/conflict-exercise.git
af6a0a9..ad40365 main -> main
```

2)También sube la feature-branch si deseas:

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

```

kayn@DESKTOP-7M270V8 MINGW64 ~/Desktop/PROGRAMACION/Repositorio-git/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/IIINEOIII/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/IIINEOIII/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch

```

Paso 8: Verificar en GitHub

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

