

Práctico 2: Git y GitHub

Resolución TP2: Git & GitHub 1) Contestar las siguientes preguntas utilizando las guías y documentación proporcionada (Desarrollar las respuestas) :

1. ¿Qué es GitHub?

GitHub es una plataforma en línea que permite **alojar y gestionar proyectos de software** usando un sistema llamado **Git**, que es un sistema de control de versiones. GitHub combina este sistema con herramientas de colaboración, lo que permite a equipos de desarrollo trabajar juntos en el mismo proyecto sin pisarse el trabajo y con un historial completo de todos los cambios realizados.

1. Alojar proyectos

Puedes subir tus archivos de código y tenerlos disponibles desde cualquier lugar. Esto es útil para trabajar desde varias computadoras o compartir tu trabajo.

2. Colaborar con otros

Puedes invitar a otros desarrolladores a trabajar contigo, hacer revisiones de código, asignar tareas, y llevar un seguimiento de errores o mejoras.

3. Controlar versiones

GitHub te muestra el historial de cambios, te permite comparar versiones y revertir errores de manera sencilla.

4. Explorar código abierto

GitHub es el hogar de miles de proyectos de **código abierto**. Puedes ver cómo están hechos, usarlos, o incluso contribuir a ellos.

5. Crear portafolioS

Muchos desarrolladores usan GitHub como su portafolio profesional. Las empresas de tecnología suelen revisar los perfiles de GitHub para conocer la experiencia real de un candidato.

2. ¿Cómo crear un repositorio en GitHub?

1. Crear un nuevo repositorio

1. Una vez dentro de tu cuenta, haz clic en el botón “+” que aparece en la esquina superior derecha.
2. Selecciona la opción “New repository”.

2. Configurar el repositorio

Completa la información del formulario:

- **Repository name (Nombre del repositorio):** Escribe el nombre del proyecto (por ejemplo: “pagina-web”).
- **Description (Descripción):** Es opcional, pero puedes escribir una breve explicación del proyecto.
- **Visibility (Visibilidad):**
 - *Public (Público):* Cualquiera puede ver tu repositorio.
 - *Private (Privado):* Solo tú y quienes invites pueden verlo.
- Marca la opción “Initialize this repository with a README” si quieres crear un archivo README automáticamente.

Luego, haz clic en el botón “**Create repository**”.

3. ¿Cómo crear una rama en Git?

1. Crear una rama desde la terminal (Git instalado)

Paso 1: Abre la terminal

Asegúrate de estar dentro del directorio del proyecto. Si no estás, navega hasta él con el comando:

cd nombre-del-proyecto

Paso 2: Crear una nueva rama

Usa el siguiente comando:

git branch nombre-de-la-rama

4. ¿Cómo cambiar a una rama en Git?

Para cambiar a una rama en Git, se puede utilizar el siguiente comando:

git checkout nombre-de-la-rama

5. ¿Cómo fusionar ramas en Git?

Para fusionar ramas en Git, primero es necesario estar ubicado en la rama donde se desea aplicar la fusión. Luego se ejecuta el comando `git merge` seguido del nombre de la rama que se quiere fusionar.

Pasos para fusionar ramas:

1. Cambiar a la rama de destino (donde se quiere aplicar la fusión):
`git checkout main`
2. Ejecutar el comando de fusión, indicando la rama que se desea fusionar con la actual:
`git merge nombre-de-la-rama`

6. ¿Cómo crear un commit en Git?

Un **commit** en Git representa un conjunto de cambios guardados en el repositorio. Para crear un commit, se siguen los siguientes pasos:

1. Agregar los archivos al área de preparación (staging area)

Antes de hacer un commit, se deben seleccionar los archivos que se quieren incluir. Esto se hace con:

`git add nombre-del-archivo`

O para agregar **todos** los archivos modificados:

`git add .`

2. Crear el commit

Una vez que los archivos están en el área de preparación, se crea el commit con:

`git commit -m "Mensaje que describe los cambios"`

7. ¿Cómo enviar un commit a GitHub?

Una vez creado un commit en tu repositorio local, para enviarlo a GitHub (repositorio remoto), se utiliza el comando `git push`.

Pasos para enviar un commit a GitHub:

- 1. Verificar en qué rama estás:**

`git branch`

La rama actual estará marcada con un asterisco `*`.

Enviar los cambios al repositorio remoto:

`git push origin nombre-de-la-rama`

8. ¿Qué es un repositorio remoto?

Un **repositorio remoto** es una versión del repositorio de Git que se aloja en un servidor o plataforma en línea, como **GitHub**, **GitLab** o **Bitbucket**. Este repositorio puede ser accedido por varios usuarios para colaborar en el mismo proyecto desde diferentes computadoras.

A diferencia del repositorio local, que está en tu computadora, el remoto permite:

- **Compartir** el código con otros desarrolladores.
- **Sincronizar** cambios entre distintos equipos.

- **Guardar copias de seguridad** del proyecto en la nube.

9. ¿Cómo agregar un repositorio remoto a Git?

Para agregar un **repositorio remoto** a tu proyecto de Git (por ejemplo, un repositorio en GitHub), se utiliza el comando `git remote add`. Este comando vincula tu repositorio local con uno remoto, permitiéndote subir y sincronizar cambios.

Comando básico:

```
git remote add origin URL-del-repositorio
```

10. ¿Cómo empujar cambios a un repositorio remoto?

En Git, **empujar (push)** significa enviar los cambios realizados en el repositorio local al repositorio remoto (por ejemplo, en GitHub), para que queden guardados y compartidos en línea.

Comando para empujar cambios:

```
git push origin nombre-de-la-rama
```

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

En Git, **tirar de cambios** significa traer actualizaciones desde el repositorio remoto al repositorio local. Esto se hace con el comando `git pull`.

Comando para traer cambios:

```
git pull origin nombre-de-la-rama
```

12. ¿Qué es un fork de repositorio?

Un **fork** (bifurcación) de un repositorio es una copia completa de ese repositorio que se crea bajo tu propia cuenta de GitHub (u otra plataforma similar). Esta copia te permite experimentar, hacer cambios y desarrollar nuevas funcionalidades sin afectar el proyecto original.

¿Para qué sirve un fork?

- Contribuir a proyectos de **código abierto** sin tener acceso directo de escritura.
- Desarrollar nuevas funcionalidades o corregir errores por tu cuenta.

- Probar ideas sin modificar el repositorio original.

13. ¿Cómo crear un fork de un repositorio?

Crear un fork de un repositorio en GitHub es muy sencillo y se puede hacer directamente desde la plataforma web. Esto genera una copia del repositorio original en tu propia cuenta, lo que te permite trabajar libremente sin afectar el proyecto original.

Pasos para crear un fork:

1. Ingresar al repositorio que se desea bifurcar en **GitHub**.
2. Hacer clic en el botón **Fork**, ubicado en la parte superior derecha de la página.
3. Elegir tu cuenta (o una organización, si tenés varias).
4. GitHub creará una copia del repositorio en tu perfil.

14. ¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?

Una **solicitud de extracción** (en inglés, *pull request*) es una propuesta de cambio que se envía desde una rama (por lo general, de un *fork*) hacia el repositorio original. Es el método principal para colaborar en proyectos de código abierto.

Pasos para enviar un pull request en GitHub:

1. **Crear un fork** del repositorio original (si aún no lo hiciste).
2. **Clonar** tu fork y crear una rama nueva para hacer los cambios:

git checkout -b nombre-de-tu-rama

3. **Hacer cambios**, luego guardar y confirmar:

git add .

git commit -m "Descripción de los cambios realizados"

4. **Empujar los cambios a tu fork** en GitHub:

git push origin nombre-de-tu-rama

Ir a GitHub: abrí tu fork en el navegador. GitHub te mostrará un botón que dice "**Compare & pull request**". Hacer clic.

Completar el formulario: agregá un título y una descripción clara de los cambios realizados.

Enviar la solicitud haciendo clic en "**Create pull request**".

15. ¿Cómo aceptar una solicitud de extracción?

Aceptar una **solicitud de extracción** (*pull request*) en GitHub significa revisar y aprobar los cambios propuestos por otro usuario, para luego integrarlos al repositorio original.

Pasos para aceptar un pull request en GitHub:

1. **Ingresar al repositorio** original (donde se recibió la solicitud).
2. Hacer clic en la pestaña "**Pull requests**" (ubicada arriba).
3. Seleccionar la solicitud que se desea revisar.

4. Revisar los **cambios propuestos**, los archivos modificados y los comentarios del autor.
5. Si todo está correcto, hacer clic en el botón **"Merge pull request"**.
6. Confirmar la acción haciendo clic en **"Confirm merge"**.
7. (Opcional) Eliminar la rama si ya no es necesaria, con el botón **"Delete branch"**.

16. ¿Qué es una etiqueta en Git?

Una **etiqueta** (*tag*) en Git es un marcador que se utiliza para señalar un punto específico en la historia del repositorio, normalmente para identificar **versiones importantes** del proyecto, como lanzamientos (*releases*).

A diferencia de una rama, una etiqueta no cambia con el tiempo: apunta a un commit en particular y permanece fija.

Tipos de etiquetas:

1. **Etiquetas ligeras (lightweight)**: simplemente apuntan a un commit específico.
2. **Etiquetas anotadas (annotated)**: incluyen información adicional como el nombre del autor, la fecha y un mensaje.

17. ¿Cómo crear una etiqueta en Git?

En Git, una **etiqueta** se usa para marcar un commit específico, generalmente para indicar una versión estable del proyecto (por ejemplo, v1.0, v2.1.3, etc.).

Crear una etiqueta ligera:

Una etiqueta ligera simplemente apunta a un commit, sin información adicional:

```
git tag nombre-etiqueta
```

18. ¿Cómo enviar una etiqueta a GitHub?

Una vez que creas una etiqueta en tu repositorio local, puedes **enviarla (push)** al repositorio remoto, como GitHub, para que quede registrada y disponible para otros colaboradores.

Subir una etiqueta específica:

```
git push origin nombre-de-la-etiqueta
```

19. ¿Qué es un historial de Git?

El **historial de Git** es el registro de todos los cambios realizados en un repositorio a lo largo del tiempo. Cada vez que se realiza un **commit**, se guarda un punto en la historia del proyecto, permitiendo ver qué cambios se hicieron, quién los hizo, cuándo y por qué.

Este historial es una de las principales fortalezas de Git, ya que permite:

- Rastrear la evolución del código.
- Revertir errores.
- Colaborar con otros de forma clara.
- Analizar cuándo y por qué se introdujo un cambio.

20. ¿Cómo ver el historial de Git?

Para ver el **historial de cambios** en un repositorio Git, se utiliza el comando `git log`, que muestra todos los commits realizados, ordenados del más reciente al más antiguo.

Comando básico:

`git log`

21. ¿Cómo buscar en el historial de Git?

Git permite **buscar información específica** dentro del historial de commits, como mensajes, autores o cambios en archivos. Esto es muy útil para rastrear errores, entender modificaciones o auditar el proyecto.

Buscar por palabra clave en mensajes de commit:

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

Buscar por autor:

`git log --author="nombre o email"`

Buscar cambios en un archivo:

`git log nombre-del-archivo`

Buscar un commit por contenido (línea agregada/eliminada):

`git log -S"texto específico"`

Combinar filtros:

Podés combinar filtros para refinar la búsqueda, por ejemplo:

`git log --author="Ana" --grep="bug"`

22. ¿Cómo borrar el historial de Git?

Git no permite borrar el historial directamente, pero sí se pueden aplicar técnicas para **reemplazarlo, reescribirlo o eliminarlo completamente**. Esto puede ser útil si querés reiniciar el historial o eliminar información sensible.

Opción: Borrar todo el historial y comenzar de nuevo

Esto reinicia el historial manteniendo los archivos actuales.

```
rm -rf .git
```

```
git init
```

```
git add .
```

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

Esto elimina completamente el historial anterior.

23. ¿Qué es un repositorio privado en GitHub?

Un **repositorio privado** en GitHub es un repositorio que **no es visible públicamente**. Solo las personas autorizadas por el propietario pueden verlo o colaborar en él.

Características principales:

- **Acceso restringido:** Solo usuarios invitados pueden clonar, ver o hacer *push/pull*.
- **Confidencialidad:** Ideal para proyectos personales, académicos o de empresa.
- **Permisos personalizables:** Se puede asignar acceso de lectura, escritura o administración a colaboradores específicos.

24. ¿Cómo crear un repositorio privado en GitHub?

Crear un repositorio privado en GitHub es simple y útil para mantener tu proyecto **protegido y accesible solo para colaboradores autorizados**.

Pasos para crearlo:

1. Inicia sesión en [GitHub](#).
2. Hacé clic en el botón verde **"New"** o en el símbolo **"+" > New repository**.
3. Completá los datos:
 - a. **Repository name:** nombre del repositorio.
 - b. **Description** (opcional): una breve descripción del proyecto.
4. En la sección de visibilidad, seleccioná:
 - Private
1. Opcional: Marcá opciones como:
 - a. Inicializar con un README.

- b. Agregar `.gitignore` o licencia.
2. Hacé clic en "**Create repository**".

Confirmación:

El repositorio se creará como privado y solo vos (y quienes invites) podrán acceder a él. Podrás añadir colaboradores desde la pestaña "**Settings > Collaborators**".

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

Si tenés un repositorio privado en GitHub y querés que otras personas colaboren, podés **invitarlas manualmente como colaboradores**. Solo los usuarios invitados podrán acceder al repositorio.

Pasos para invitar a un colaborador:

1. Iniciá sesión en [GitHub](#) y abrí el repositorio privado.
2. Hacé clic en la pestaña "**Settings**" (Configuración).
3. En el menú lateral, seleccioná "**Collaborators**" o "**Manage access**".
4. Hacé clic en el botón "**Invite a collaborator**".
5. Escribí el nombre de usuario o correo electrónico de la persona que querés invitar.
6. Seleccionalo de la lista y hacé clic en "**Add**".

Confirmación:

- El usuario recibirá una notificación con la invitación.
- Una vez aceptada, podrá ver, clonar y colaborar según los permisos que le hayas asignado.

26. ¿Qué es un repositorio público en GitHub?

Un **repositorio público** en GitHub es un repositorio que **puede ser visto por cualquier persona**, incluso si no tiene cuenta en GitHub. Es la opción ideal para proyectos de código abierto (*open source*), portafolios o contenidos educativos.

Características principales:

- **Acceso abierto:** cualquier usuario puede ver el código y el historial de cambios.
- **Colaboración global:** cualquier persona puede clonar el repositorio o proponer cambios mediante *pull requests*.
- **Indexado por buscadores:** puede aparecer en resultados de Google u otros motores de búsqueda.

27. ¿Cómo crear un repositorio público en GitHub?

Para crear un repositorio público en GitHub, seguí estos pasos:

1. Iniciá sesión en GitHub y hacé clic en el botón “New” o en el símbolo “+” > New repository.
2. Escribí el nombre del repositorio y, si querés, una descripción.
3. Seleccioná la opción de visibilidad **“Public”** (pública).
4. Opcional: marcá la opción de inicializar el repositorio con un README, .gitignore o una licencia.
5. Hacé clic en **“Create repository”**.

Una vez creado, el repositorio será visible para cualquier persona en GitHub. Podrán ver el código, clonar el proyecto y, si lo permitís, colaborar.

28. ¿Cómo compartir un repositorio público en GitHub?

Compartir un repositorio público en GitHub es muy simple, ya que cualquier persona puede acceder a él mediante su URL.

Pasos para compartirlo:

1. Ingresá a GitHub y abrí el repositorio que querés compartir.
2. Copiá la URL que aparece en la barra de direcciones del navegador.
Ejemplo: <https://github.com/tu-usuario/tu-repositorio>
3. Pegá ese enlace en un mensaje, correo, red social o cualquier otro medio donde quieras compartirlo.

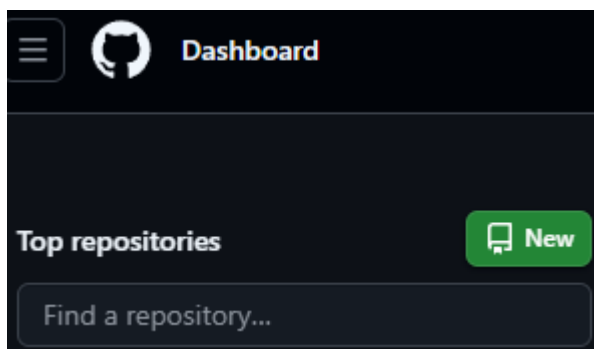
2) Realizar la siguiente actividad:

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 *

 Luciano-Cartagena ▾

Repository name *

TecRepositorio

✔ TecRepositorio is available.

Great repository names are short and memorable. Need inspiration? How about **crispy-octo-succotash** ?

Description (optional)

Este es mi repositorio de Tecnicatura en Programación

☒ 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.](#)

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

```

PS C:\Users\lucia\Desktop> cd .\Tp2\
PS C:\Users\lucia\Desktop\Tp2> LS
PS C:\Users\lucia\Desktop\Tp2> ls
PS C:\Users\lucia\Desktop\Tp2> git clone https://github.com/Luciano-Cartagena/TecRepositorio.git
Cloning into 'TecRepositorio'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
PS C:\Users\lucia\Desktop\Tp2> ls

```

Directorio: C:\Users\lucia\Desktop\Tp2

Mode		LastWriteTime	Length	Name
d-----		14/4/2025 19:48		TecRepositorio

```

PS C:\Users\lucia\Desktop\Tp2> cd .\TecRepositorio\
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> |

```

```

PS C:\Users\lucia\Desktop\Tp2> cd .\TecRepositorio\
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> echo "Este es mi primer archivo (Luciano)" > mi-archivo.txt
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> ls

```

Directorio: C:\Users\lucia\Desktop\Tp2\TecRepositorio

Mode		LastWriteTime	Length	Name
-a-----		14/4/2025 19:52	76	mi-archivo.txt
-a-----		14/4/2025 19:48	75	README.md

```

PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> |

```

```

PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git add .
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> 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

```

PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git commit -m "Agregando mi-archivo.txt"
[main 5b9e840] Agregando mi-archivo.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 mi-archivo.txt
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> |


```


 .git	14/4/2025 19:54	Carpeta de archivos	
 mi-archivo.txt	14/4/2025 19:52	Documento de te...	1 KB
 README.md	14/4/2025 19:48	Archivo de origen ...	1 KB




TecRepositorio
Public
Pin
Unwatch
1

main
1 Branch
0 Tags

Add file
Code


Luciano-Cartagena Initial commit
 cf9125b · 35 minutes ago
1 Commit

 README.md
 Initial commit
 35 minutes ago

 README
 

TecRepositorio

Este es mi repositorio de Tecnicatura en Programación

```

PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git branch
* main
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 361 bytes | 361.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Luciano-Cartagena/TecRepositorio.git
   cf9125b..5b9e840  main -> main
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> |

```



1. Creando Branchs o Crear una Branch
2. Realizar cambios o agregar un archivo
3. Subir la Branch

```
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git checkout -b nuevaRama
Switched to a new branch 'nuevaRama'
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git branch
main
* nuevaRama
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio>
```

```
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> echo "Modificación hecha desde nuevaRama" > mi-archivo.txt
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git add .
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git status
On branch nuevaRama
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   mi-archivo.txt
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio>
```

```
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git commit -m "Agregando modificación desde nuevaRama"
[nuevaRama 619e8ab] Agregando modificación desde nuevaRama
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git status
On branch nuevaRama
nothing to commit, working tree clean
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> |
```

mi-archivo modificado desde nuevaRama:




mi-archivo.txt: Bloc de notas


Archivo Edición Formato Ver Ayuda

Modificación hecha desde nuevaRama

Subimos la nueva rama a github:


```
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio> git push origin nuevaRama
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 372 bytes | 372.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'nuevaRama' on GitHub by visiting:
remote:   https://github.com/Luciano-Cartagena/TecRepositorio/pull/new/nuevaRama
remote:
To https://github.com/Luciano-Cartagena/TecRepositorio.git
 * [new branch]      nuevaRama -> nuevaRama
PS C:\Users\lucia\Desktop\Tp2\TecRepositorio>
```

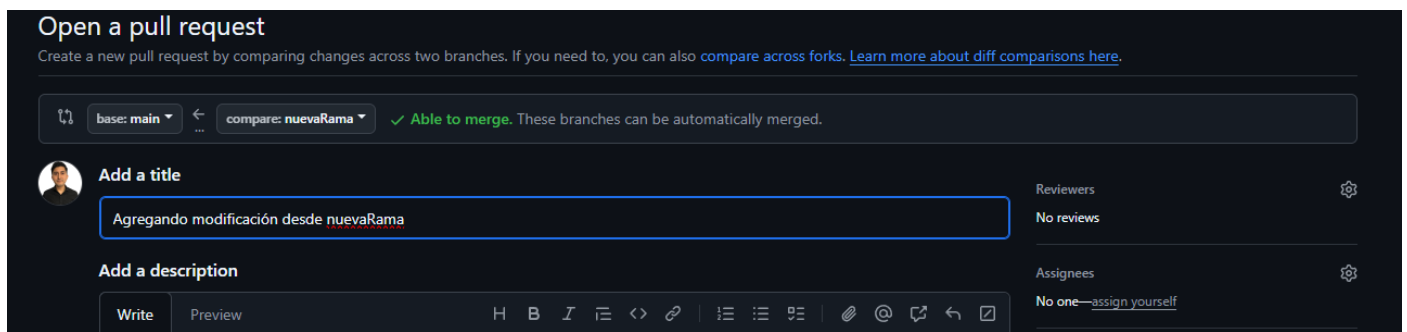

TecRepositorio
Public
Pin
Unwatch
1


nuevaRama had recent pushes 1 minute ago
 Compare & pull request

main
2 Branches
0 Tags

Add file
<> Code

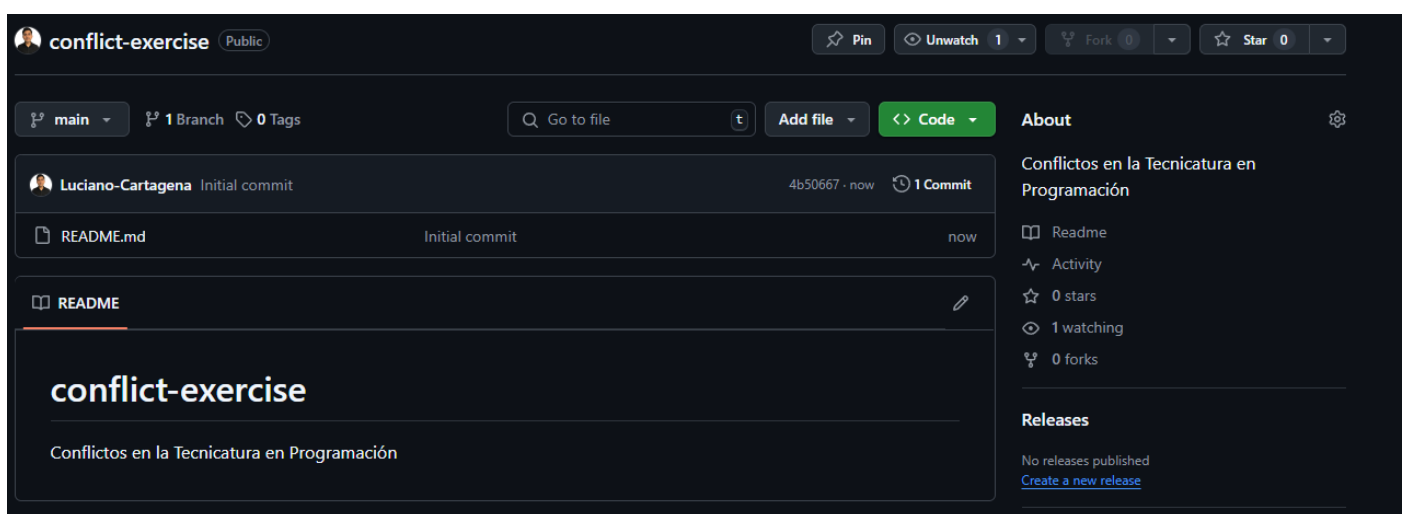

Luciano-Cartagena Agregando mi-archivo.txt
 5b9e840 · 22 minutes ago
2 Commits



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/conflictexercise.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`

```
PS C:\Users\lucia\Desktop\Tp2> git clone https://github.com/Luciano-Cartagena/conflict-exercise.git
Cloning into 'conflict-exercise'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
PS C:\Users\lucia\Desktop\Tp2>
```

- Entra en el directorio del repositorio:
`cd conflict-exercise`

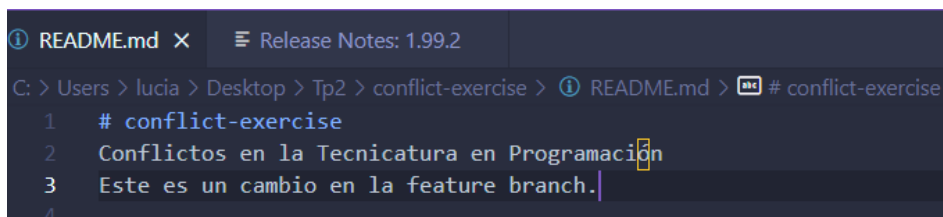
Paso 3: Crear una nueva rama y editar un archivo

- Crea una nueva rama llamada feature-branch:

`git checkout -b feature-branch`

```
PS C:\Users\lucia\Desktop\Tp2> cd .\conflict-exercise\
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git checkout -b feature-branch
Switched to a new branch 'feature-branch'
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> |
```

- 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.



The screenshot shows a code editor with a tab for 'README.md' and another for 'Release Notes: 1.99.2'. The editor's path bar shows 'C: > Users > lucia > Desktop > Tp2 > conflict-exercise > README.md'. The file content is as follows:

```
1 # conflict-exercise
2 Conflictos en la Tecnicatura en Programación
3 Este es un cambio en la feature branch.
4
```

- Guarda los cambios y haz un commit:

git add README.md

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

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git add .\README.md
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git commit -m "Added a line in feature-branch"
[feature-branch 88b3a58] Added a line in feature-branch
1 file changed, 1 insertion(+)
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> |
```

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

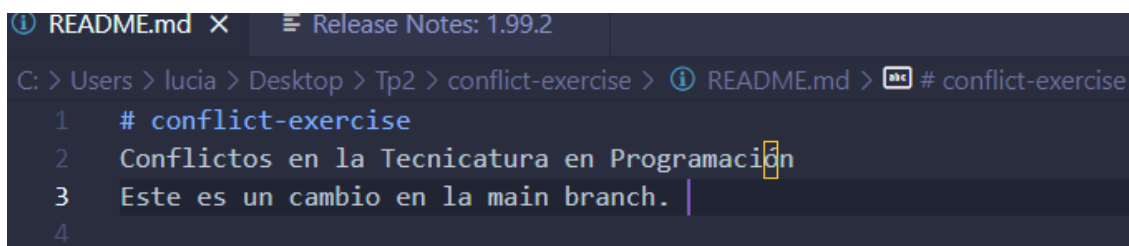
- Cambia de vuelta a la rama principal (main):

git checkout main

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise>
```

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

Este es un cambio en la main branch.



```
① README.md X  Release Notes: 1.99.2
C: > Users > lucia > Desktop > Tp2 > conflict-exercise > ⓘ README.md > # conflict-exercise
1  # conflict-exercise
2  Conflictos en la Tecnicatura en Programación
3  Este es un cambio en la main branch. |
4
```

- Guarda los cambios y haz un commit:

git add README.md

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

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git add .\README.md
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git commit -m "Added a line in main branch"
[main c3697e3] Added a line in main branch
1 file changed, 1 insertion(+)
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> |
```

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

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git merge feature-branch
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> |
```

- 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:

Copiar código <<<<<< HEAD Este es un cambio en la main branch. ===== Este es un cambio en la feature branch. >>>>>> feature-branch

```
C: > Users > lucia > Desktop > Tp2 > conflict-exercise > README.md > # Conflictos en la Tecnicatura en Programación<<<<<< HEADEste es u
1 # conflict-exercise
2 Conflictos en la Tecnicatura en Programación
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)
```

- 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.

```
C: > Users > lucia > Desktop > Tp2 > conflict-exercise > README.md > # conflict-exercise
1 # conflict-exercise
2 Conflictos en la Tecnicatura en Programación
3 Este es un cambio en la main branch.
4 Este es un cambio en la feature branch.
```

- Añade el archivo resuelto y completa el merge:
git add README.md git commit -m "Resolved merge conflict"

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git add .\README.md
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git commit -m "Resolved merge conflict"
[main 483d304] Resolved merge conflict
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> |
```

Paso 7: Subir los cambios a GitHub

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

git push origin main

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git push origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 778 bytes | 389.00 KiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Luciano-Cartagena/conflict-exercise.git
 4b50667..483d304  main -> main
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise>
```

- También sube la feature-branch si deseas:

git push origin feature-branch

```
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> git push origin feature-branch
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:   https://github.com/Luciano-Cartagena/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/Luciano-Cartagena/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch
PS C:\Users\lucia\Desktop\Tp2\conflict-exercise> |
```

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.



conflict-exercise

Public



Pin



Unwatch

1



main



2 Branches



0 Tags



Go to file



Add file



Code



Luciano-Cartagena

Resolved merge conflict

483d304 · 4 minutes ago



4 Commits



README.md

Resolved merge conflict

4 minutes ago



README



conflict-exercise

Conflictos en la Tecnicatura en Programación Este es un cambio en la main branch. Este es un cambio en la feature branch.