
Resolución TP2: Git & GitHub

Alumno: Calcatelli Renzo.

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

1. ¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Permite a los desarrolladores almacenar, gestionar y colaborar en proyectos de software. Es ampliamente utilizado para el control de versiones, el seguimiento de problemas y la colaboración en proyectos de código abierto.

2. ¿Cómo crear un repositorio en GitHub?

Creación de un Repositorio en GitHub:

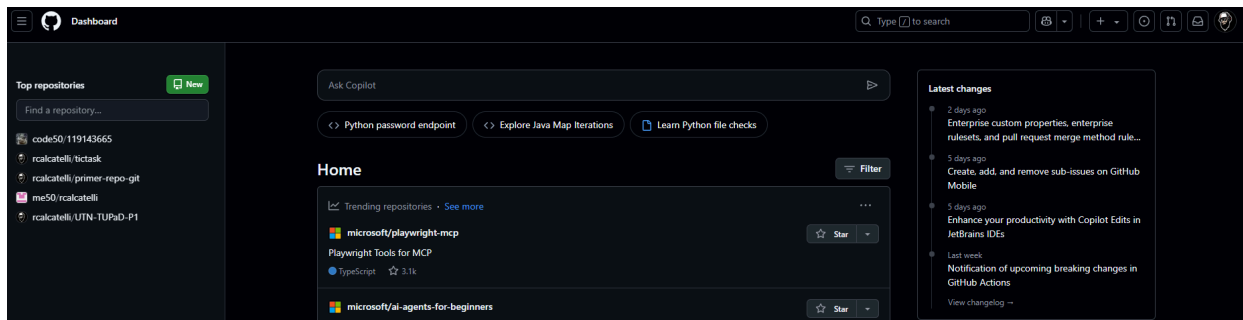
1. Crear una Cuenta:

- Si no tenés una, registrate en la página web de GitHub.

2. Crear un Nuevo Repositorio:

- Una vez iniciada la sesión, hacé clic en el botón "New" (Nuevo).
- Asignar un nombre descriptivo al repositorio.
- Opcionalmente, agregar una descripción para aclarar el propósito del repositorio.
- Seleccionar la visibilidad: "Public" (Público) para que sea accesible para todos o "Private" (Privado) para limitar el acceso.

- Se puede inicializar el repositorio con un archivo README, lo cual es recomendable.
- Hacer clic en "Create repository" (Crear repositorio).



3. Conexión con un Repositorio Local (Dos Escenarios):

- **Repositorio Local Nuevo:**
 - GitHub proporcionará comandos Git para inicializar un nuevo repositorio local y conectarlo al remoto. Seguir esos comandos.
- **Repositorio Local Existente:**
 - Si ya tienes un repositorio local, utiliza los siguientes comandos en la terminal:

```
git remote add origin <URL del repositorio> (Conecta el repositorio local al remoto).  
(https://github.com/rcalcatelli/primer-repo-git.git)
```

```
git push -u origin master (Envía el repositorio local al remoto, estableciendo el seguimiento).
```

Interacción con un Repositorio Clonado:

1. Clonar un Repositorio:

- En la página del repositorio en GitHub, hacer clic en "Clone or download" y copiar la URL.
- En tu terminal, navegar al directorio donde deseas clonar el repositorio y ejecutar:

```
git clone <URL del repositorio>  
(https://github.com/rcalcatelli/primer-repo-git.git)
```

2. Realizar y Enviar Cambios:

- Realizar las modificaciones necesarias en los archivos locales.
- Agregar los cambios al área de preparación (stage): `git add .`
- Crear un `commit` con un mensaje descriptivo: `git commit -m "Mensaje del commit"`
- Enviar los cambios al repositorio remoto: `git push origin master` (o el nombre de la rama correspondiente).

3. Consideraciones Adicionales:

- Al realizar "`push`" desde un repositorio clonado en otra computadora, es posible que se requiera autenticación (usuario y contraseña).

3. ¿Cómo crear una rama en Git?

La rama "`master`" en Git no tiene características especiales; es una rama como cualquier otra. Su presencia en la mayoría de los repositorios se debe a que el comando `git init` la crea por defecto.

Para generar una nueva rama, se utiliza el siguiente comando:

```
git branch <nombre-de-la-rama>
```

Sin embargo, crear una rama con `git branch` no implica cambiar automáticamente a ella. Git mantiene un apuntador especial llamado `HEAD`,

que sigue indicando la rama actual, que en este caso seguiría siendo "master".

4. ¿Cómo cambiar de rama en Git?

Para moverte entre ramas en Git, se utiliza el siguiente comando:

```
git checkout <nombre-de-la-rama>
```

Este comando cambia el apuntador HEAD a la rama especificada.

Si se quiere crear una nueva rama y cambiar a ella en un solo paso, se usa la opción -b:

```
git checkout -b <nombre-de-la-rama>
```

5. ¿Cómo fusionar ramas en Git?

Fusionar ramas en Git implica combinar los cambios de una rama en otra.

Primero, hay que asegurarse de estar en la rama donde deseas integrar los cambios (por lo general, master o main):

```
git checkout master
```

Luego, se usa el comando git merge para unir la rama de origen con la actual:

```
git merge <nombre-de-la-rama>
```

Esto incorporará los cambios de <nombre-de-la-rama> en master.

6. ¿Cómo crear un commit en Git?

Un commit en Git guarda los cambios en el historial del proyecto.

1. Pasos para realizar un commit:

- Modificar los archivos según lo necesario.

2. Agregar los cambios al área de preparación con:

```
git add nombre_del_archivo # Para un archivo específico
git add . # Para todos los archivos modificados
```

3. Crear el commit con un mensaje descriptivo:

```
git commit -m "Descripción de los cambios"
```

Esto registrará el estado actual del código en el historial del repositorio.

7. ¿Cómo enviar un commit a GitHub?

Para subir los cambios a GitHub, seguir estos pasos:

1. Clona el repositorio en tu máquina local (si aún no lo has hecho):

```
git clone https://github.com/rcalcatelli/primer-repo-git.git
cd primer-repo-git
```

2. Realizar cambios en los archivos y agregarlos:

```
git add nombre_del_archivo # Para un archivo específico
git add . # Para todos los archivos
```

3. Crear un commit con una descripción:

```
git commit -m "Descripción de los cambios"
```

4. Envíar los commits a GitHub con:

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

5. Esto subirá los cambios al repositorio remoto en la rama correspondiente.

8. ¿Qué es un repositorio remoto?

Un repositorio remoto en Git es una versión de tu proyecto almacenada en Internet o en una red externa, lo que permite la colaboración con otros desarrolladores. Se puede tener múltiples repositorios remotos, con permisos de solo lectura o de lectura y escritura.

Trabajar con repositorios remotos implica sincronizar datos, enviando y recibiendo cambios para compartir tu trabajo con otros. Además, incluye tareas como agregar nuevos remotos, eliminar aquellos que ya no son necesarios, administrar ramas remotas y decidir cuáles deben ser rastreadas.

9. ¿Cómo agregar un repositorio remoto en Git?

Agregar un Repositorio Remoto a Git

- **Vincular el Repositorio Local con el Remoto:**
 - Utilizar el comando `git remote add` para establecer una conexión entre tu repositorio local y un repositorio remoto.
 - La sintaxis es: `git remote add [nombre] [url]`.
 - El "[nombre]" es un alias que asignas al repositorio remoto, lo cual facilita su referencia en futuros comandos.
 - "[url]" es la dirección URL del repositorio remoto.
- **Verificar la Configuración del Remoto:**
 - Para confirmar que el repositorio remoto se agregó correctamente y para ver los nombres asignados, ejecutar el comando: `git remote -v`.
 - Este comando muestra una lista de todos los repositorios remotos configurados, junto con sus URLs correspondientes.

- **Obtener Información del Repositorio Remoto:**

- Para descargar la información del repositorio remoto que aún no está presente en tu repositorio local, utilizar el comando: `git fetch [nombre]`.
 - Este comando descarga los cambios del repositorio remoto, pero no los fusiona automáticamente con tu rama local actual.
 - Es útil para inspeccionar los cambios disponibles en el repositorio remoto antes de integrarlos en tu rama local.
-

10. ¿Cómo empujar cambios a un repositorio remoto?

- **Actualizar el Repositorio Local (Recomendado):**

- Antes de enviar tus cambios, es aconsejable actualizar tu repositorio local con los últimos cambios del repositorio remoto para minimizar la posibilidad de conflictos. Esto se logra con el comando:

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

- Este comando descarga los cambios del repositorio remoto y los fusiona con tu rama local.

- **Enviar los Cambios al Repositorio Remoto:**

- Una vez que tu repositorio local está actualizado, se pueden enviar tus cambios al repositorio remoto con el comando:

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

- "`origin`" se refiere al nombre del repositorio remoto (generalmente "origin" por defecto), y "`<nombre-de-la-rama>`" especifica la rama que deseas enviar.

- **Consideraciones Importantes:**

- El comando "`git pull`" es muy importante, ya que sin él, se podrían generar conflictos en el repositorio remoto.
- "`git push`" sube los cambios que se encuentran en el repositorio local, al repositorio remoto.

Se recomienda actualizar el repositorio local antes de realizar el "push" para evitar conflictos.

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

Obtener Cambios de un Repositorio Remoto en Git

- **Descargar y Fusionar Cambios:**
 - Utilizar el comando `git pull` para descargar los cambios del repositorio remoto y fusionarlos automáticamente con tu rama local.
 - La sintaxis es: `git pull origin <nombre-de-la-rama>`.
 - "`origin`" es una convención, y representa el nombre que se le asigna por defecto al repositorio remoto.
 - "`<nombre-de-la-rama>`" es el nombre de la rama remota que se quiere integrar a tu rama local.
- **Ejemplo:**
 - Si estás trabajando en la rama "`main`", el comando sería: `git pull origin main`.
- **Puntos claves:**
 - El comando `git pull`, realiza dos acciones, primero descarga los cambios que existen en el repositorio remoto, y luego los integra a tu rama local.
 - Es sumamente importante utilizar este comando, con el fin de mantener sincronizado el repositorio local con el remoto, y así, evitar conflictos.

12. ¿Qué es un fork de un repositorio?

Un `fork` en GitHub es una copia de un repositorio que se crea en tu propia cuenta para modificarlo libremente sin afectar el original. Esto es útil cuando se encuentra un proyecto de código abierto que te interesa y se quiere adaptar o contribuir a su desarrollo.

Para hacer un fork, simplemente hay que acceder al repositorio en GitHub y hacer clic en el botón "Fork", ubicado en la parte superior derecha. Esto generará una copia independiente en tu cuenta.

Una vez creado el fork, se puede clonar el repositorio a tu computadora para trabajar en él localmente. Cualquier cambio que realices afectará solo tu fork, y si deseas que el autor original lo incorpore, se puede enviar un **pull request**.

13. ¿Cómo crear un fork de un repositorio?

Un **fork** es una copia de un repositorio en tu cuenta de GitHub, permitiéndote hacer cambios sin afectar el original. Para probarlo, podrías crear una cuenta adicional en GitHub y, desde allí, hacer un fork de uno de los repositorios de tu cuenta principal.

Luego, clonarlo en tu máquina local:

```
git clone https://github.com/rcalcatelli/nombre_del_fork.git
cd nombre_del_fork
```

Realiza los cambios necesarios en los archivos, agregarlos al área de preparación y crear un commit:

```
git add .
git commit -m "Descripción de los cambios"
```

Después, envía los cambios a tu fork con:

```
git push origin master
```

Sin embargo, estos cambios no aparecerán automáticamente en el repositorio original. Para que el dueño del proyecto los vea y los considere, debes abrir un **pull request** desde GitHub, solicitando la fusión de tus modificaciones en el repositorio original.

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

Para crear un **pull request (PR)** en GitHub, sigue estos pasos:

1. Ve a la pestaña **Pull requests** en el repositorio donde hiciste el fork.
2. Haz clic en **New pull request**.
3. Se abrirá una ventana que mostrará un resumen de los cambios realizados en comparación con el código original.
4. Presiona **Create pull request**.
5. Agrega un título descriptivo y, en el espacio de descripción, explica por qué consideras que tus cambios deberían ser incorporados al repositorio original.

Este paso permite que los mantenedores del proyecto revisen y, si lo consideran útil, fusionen tus modificaciones en el código principal.

15. ¿Cómo aceptar una solicitud de extracción (pull request)?

Cuando alguien envía un **pull request**, el propietario del repositorio lo verá en la sección de **Pull requests**. Desde allí, podrá revisar los cambios, dejar comentarios y decidir si los acepta o no.

Si la modificación es útil y no genera conflictos con la rama principal, simplemente debe hacer clic en **Merge pull request** para incorporar los cambios al repositorio. De este modo, las contribuciones del usuario quedarán integradas en el proyecto.

16. ¿Qué es una etiqueta en Git?

En Git, una **etiqueta (tag)** permite marcar puntos específicos en el historial como referencias importantes. Se utilizan comúnmente para identificar versiones de lanzamiento, como **v1.0**, facilitando el seguimiento de hitos en el desarrollo del proyecto.

17. ¿Cómo crear una etiqueta en Git?

Git permite crear dos tipos de etiquetas: **ligeras** y **anotadas**.

- **Etiquetas ligeras:** Son simples referencias a un `commit`, similares a una rama que no cambia.
- **Etiquetas anotadas:** Se almacenan como objetos completos en la base de datos de Git e incluyen información como el nombre del creador, correo, fecha y un mensaje descriptivo. Además, pueden ser firmadas y verificadas con GPG.

Se recomienda usar **etiquetas anotadas**, ya que proporcionan más contexto. Sin embargo, si solo necesitas una referencia rápida o temporal, las etiquetas ligeras pueden ser una opción adecuada.

18. ¿Cómo enviar una etiqueta a GitHub?

Primero, crear una etiqueta en tu repositorio local. Puede ser:

- **Ligera** (simple referencia a un commit):

```
git tag v1.0
```

- **Anotada** (incluye detalles como autor y fecha):

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

Para verificar las etiquetas creadas localmente, usa:

```
git tag
```

Luego, para enviar una etiqueta específica a GitHub:

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

Si se quiere subir **todas** las etiquetas creadas:

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

(En este caso, *origin* es el nombre del repositorio remoto por defecto y *v1.0* el nombre de la etiqueta).

19. ¿Qué es el historial de Git?

El **historial de Git** es una secuencia que registra todos los cambios realizados en un repositorio. Cada cambio se guarda como un **commit**, y cada commit incluye información detallada sobre el estado del proyecto en un momento determinado, como:

- **Identificador del commit**
- **Autor**
- **Fecha**
- **Mensaje descriptivo**

Este historial permite hacer un seguimiento completo de la evolución del proyecto.

20. ¿Cómo ver el historial de Git?

Para ver el historial de commits en Git, utilizar el comando:

```
git log
```

Este comando mostrará los commits en orden inverso, es decir, los más recientes aparecerán primero.

Si se prefiere una vista más compacta, se usa:

```
git log --oneline
```

Esto te dará un resumen de los commits recientes, cada uno en una línea.

Si tu repositorio tiene muchos commits y solo se quiere ver los más recientes, se pueden limitar la cantidad con un número después del signo `-`, por ejemplo:

```
git log -3
```

Esto mostrará solo los últimos tres commits.

Si además se quieren ver los cambios realizados en cada commit, agregar la opción `-p`, lo que mostrará un detalle más extenso. Tener en cuenta que la salida será más larga, por lo que necesitarías desplazarte con los cursores. Para salir, usar **CTRL + Z**.

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

21. ¿Cómo buscar en el historial de Git?

Para buscar en el historial de commits de Git, se pueden utilizar diversos comandos y opciones que te permiten filtrar y encontrar commits específicos:

Buscar por palabra clave en el mensaje de commit:

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

Buscar commits que han modificado un archivo específico:

```
git log -- nombre_del_archivo
```

Buscar commits dentro de un rango de fechas:

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

Buscar commits hechos por un autor específico:

```
git log --author="Nombre del Autor"
```

Estos comandos te permiten filtrar los commits según lo que estés buscando en el historial.

22. ¿Cómo borrar el historial de Git?

El comando `git reset` se utiliza principalmente para deshacer cambios. Este comando mueve el puntero **HEAD** y, opcionalmente, también puede afectar el **índice (staging area)** o el **directorio de trabajo** si se usa la opción `--hard`. Sin embargo, tener en cuenta que `git reset --hard` puede borrar tu trabajo de manera irreversible, por lo que es importante entender bien su funcionamiento antes de usarlo.

Existen varias formas de utilizar `git reset`:

- `git reset`: Elimina todos los archivos del **staging area** (área de preparación).
- `git reset <nombre-archivo>`: Elimina un archivo específico del **staging area**.
- `git reset <nombre-carpeta>/`: Elimina todos los archivos dentro de una carpeta del **staging area**.
- `git reset <nombre-carpeta>/<nombre-archivo>`: Elimina un archivo específico dentro de una carpeta del **staging area**.
- `git reset <nombre-carpeta>/*.extensión`: Elimina todos los archivos con una extensión específica dentro de una carpeta del **staging area**.

Hay que recordar que `git reset` solo afecta el **staging area**, no los cambios en el **directorio de trabajo**, a menos que se utilicen opciones adicionales como `--hard`.

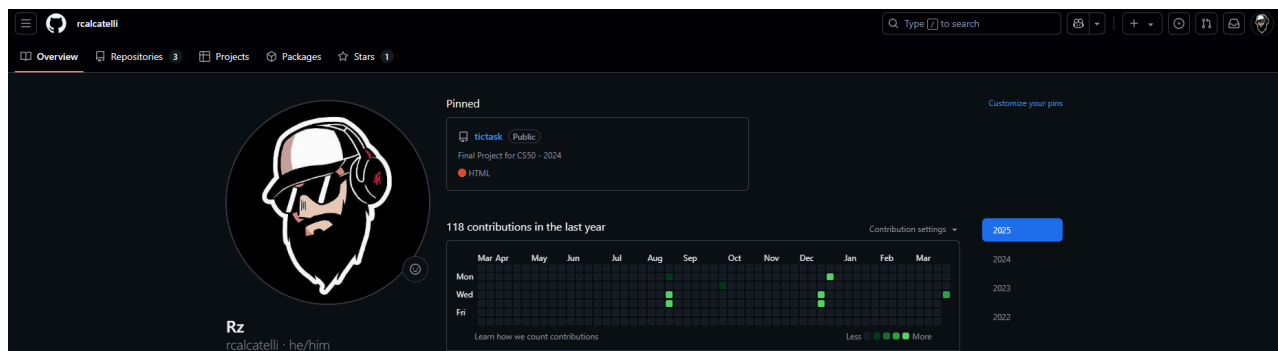
23. ¿Qué es un repositorio privado en GitHub?

Un **repositorio privado** en GitHub es un repositorio cuyo contenido solo puede ser accedido por usuarios específicos que hayan sido autorizados. A diferencia de los repositorios públicos, donde cualquier persona puede ver y clonar el contenido, los repositorios privados limitan el acceso únicamente a los colaboradores que selecciones. Esto resulta útil para proyectos que contienen información sensible o que aún están en desarrollo y no deseas que sean accesibles al público.

24. ¿Cómo crear un repositorio privado en GitHub?

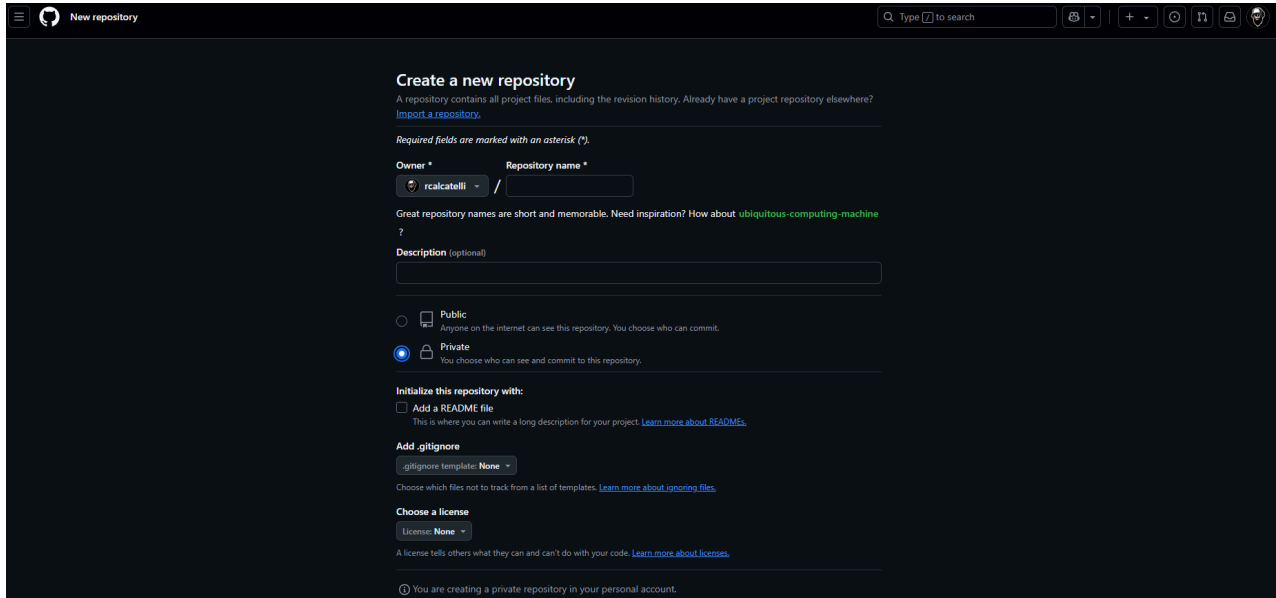
Para crear un repositorio privado en GitHub:

1. Inicia sesión en tu cuenta de GitHub.
2. Dirigirse a la página de creación de repositorios:
 - Hacer clic en el botón "+" en la esquina superior derecha de la página principal y seleccionar **"New Repository"**, o simplemente haz clic en **"New"**.



3. Completa la información del repositorio, como el nombre y la descripción.
4. Configurar la privacidad del repositorio:

- Selecciona la opción **"Private"** para asegurarte de que el repositorio sea privado y solo accesible para los colaboradores que elijas.

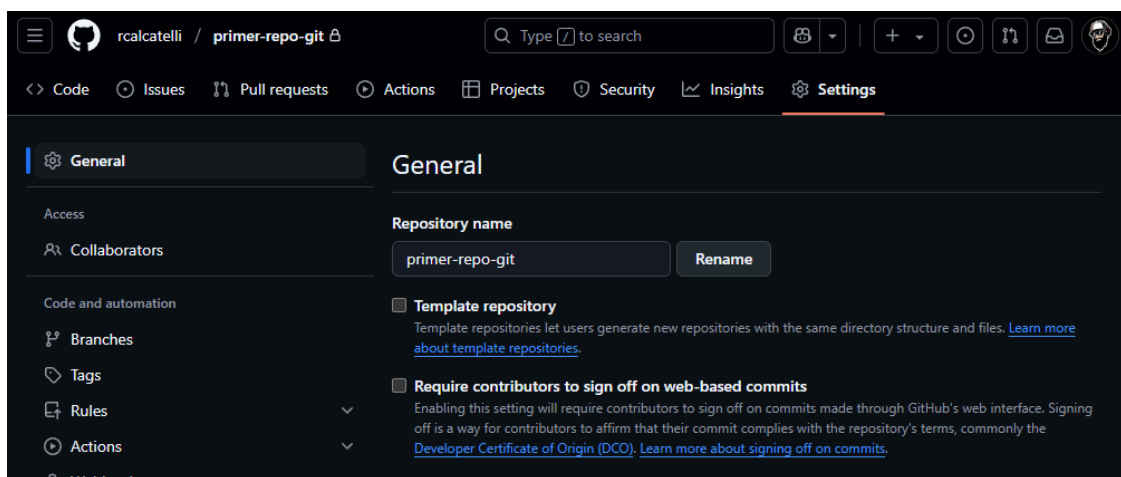


The screenshot shows the 'Create a new repository' form on GitHub. It includes fields for 'Owner' (set to 'rcalcatelli'), 'Repository name', and 'Description (optional)'. The 'Visibility' section has two radio buttons: 'Public' and 'Private', with 'Private' selected. Below this, there are checkboxes for 'Initialize this repository with:' including 'Add a README file' and 'Add .gitignore'. The 'Choose a license' section is also visible. A note at the bottom states: 'You are creating a private repository in your personal account.'

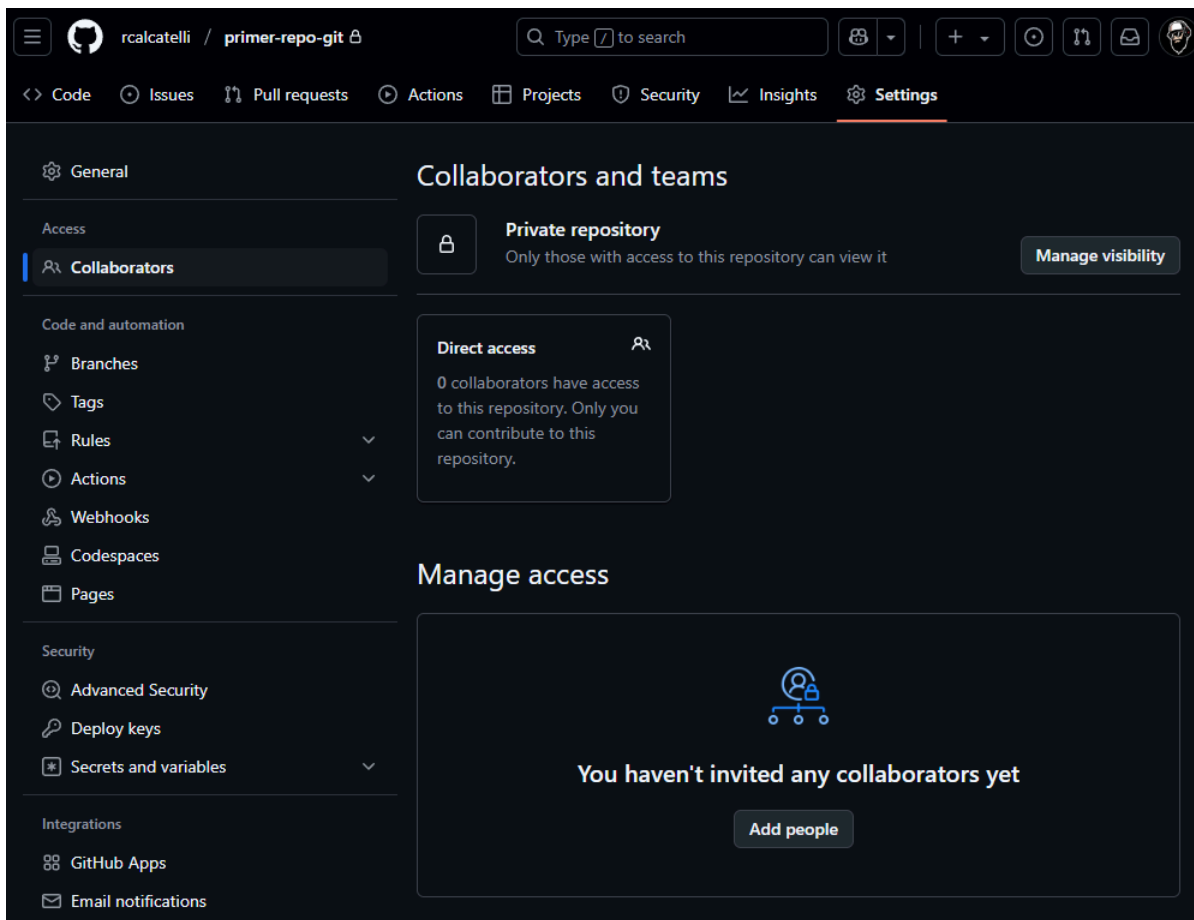
25. ¿Cómo invitar a alguien a un repositorio privado en GitHub?

Invitar a alguien a un repositorio privado en GitHub es un proceso sencillo, pero necesitas tener los permisos adecuados:

1. Acceder al repositorio y hacer clic en la pestaña **"Settings"** en la parte superior, junto a opciones como **"Code"** e **"Issues"**.



2. En el menú de la izquierda, selecciona **"Collaborators"**. Esto te llevará a la página donde gestionar a los colaboradores.



3. En la sección **"Collaborators"**, hacer clic en el botón **"Add people"**.
4. Ingresar el nombre de usuario de GitHub de la persona a invitar.
5. Elegir el nivel de acceso a otorgar (Read, Triage, Write, Maintain, o Admin).
6. Hacer clic en **"Add"** para enviar la invitación.

26. ¿Qué es un repositorio público en GitHub?

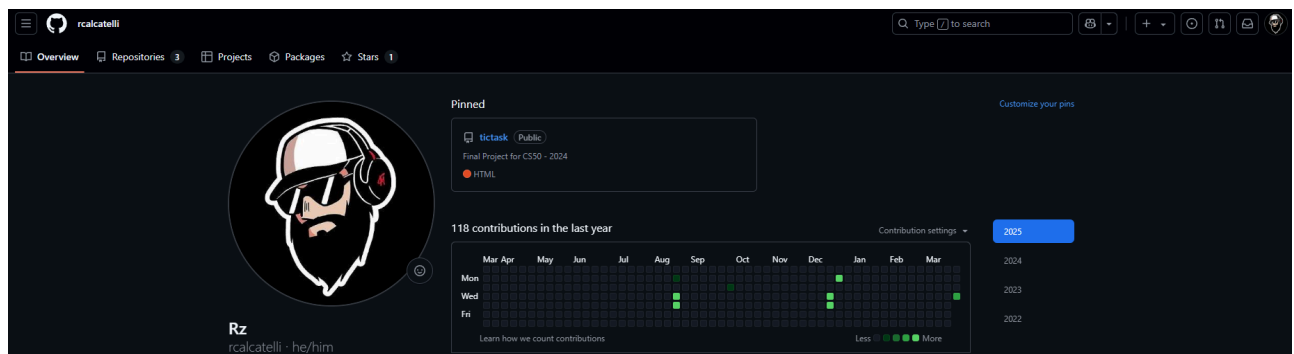
Un **repositorio público** en GitHub es aquel cuyo contenido está disponible para cualquier persona en Internet. A diferencia de un repositorio privado, que

sólo es accesible por un grupo específico de colaboradores, un repositorio público permite que cualquiera pueda ver, clonar y, si tiene los permisos adecuados, contribuir al proyecto.

27. ¿Cómo crear un repositorio público en GitHub?

Para crear un repositorio público en GitHub:

1. Inicia sesión en GitHub.
2. Ir a la página de creación de repositorios:
 - Hacer clic en el botón "+" en la esquina superior derecha y seleccionar **"New Repository"**, o hacer clic directamente en **"New"**.

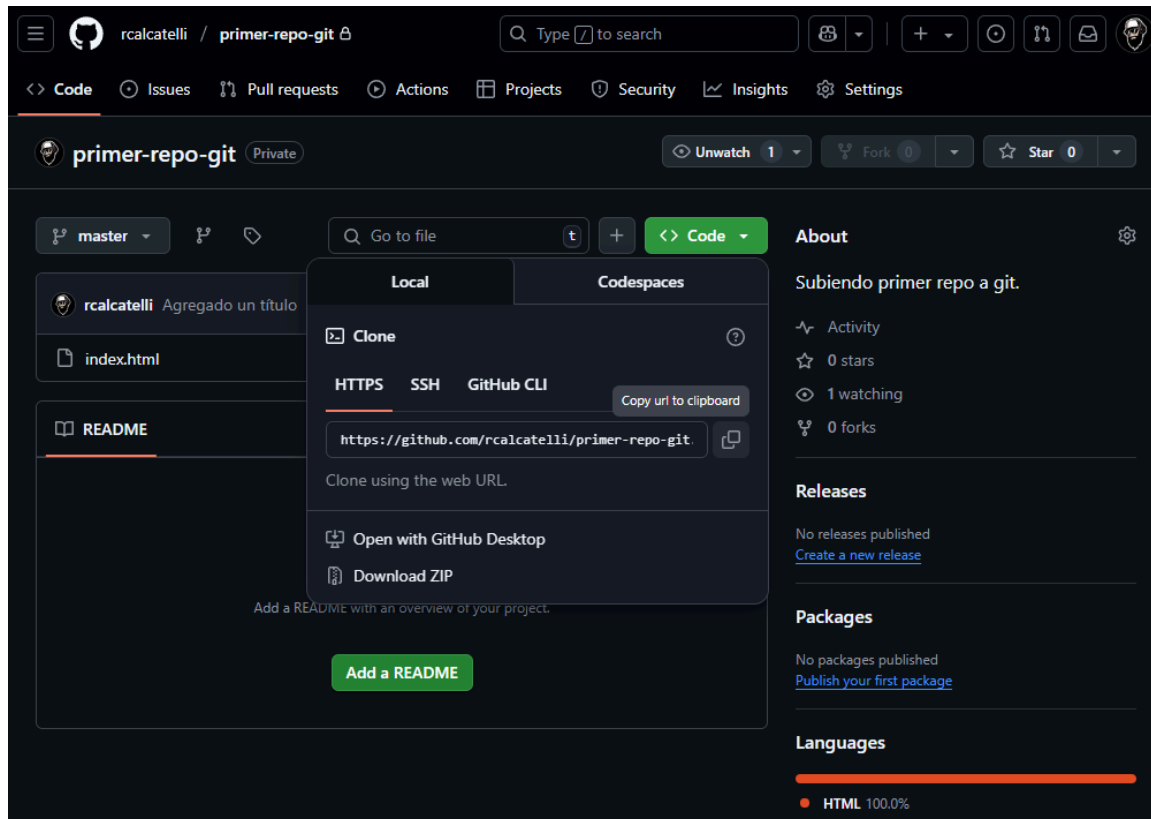


3. Completar la información del repositorio (nombre, descripción).
4. En la configuración de privacidad, seleccionar **"Public"**. Esto asegurará que el repositorio sea accesible públicamente.

28. ¿Cómo compartir un repositorio público en GitHub?

La forma más sencilla de compartir el repositorio es proporcionando su enlace directo. Para hacerlo:

1. Acceder a tu repositorio en GitHub.
2. Copiar la URL del repositorio que aparece en el cuadro de texto bajo la sección **"Code"**.
3. Hacer clic en el botón de copiar, ubicado a la derecha de la URL, para obtener el enlace y compartirlo.



2. Realizar la siguiente actividad:

Crear un repositorio.

- Dale un nombre al repositorio.
- Elige que el repositorio sea público.
- Inicializa el repositorio con un archivo.

New repository

Q Type [Z] to search

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 *

rcalcatelli / MatRepositorio

✔ MatRepositorio is available.

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

Description (optional)

Mi repositorio de MATEMÁTICA

☒ 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

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).

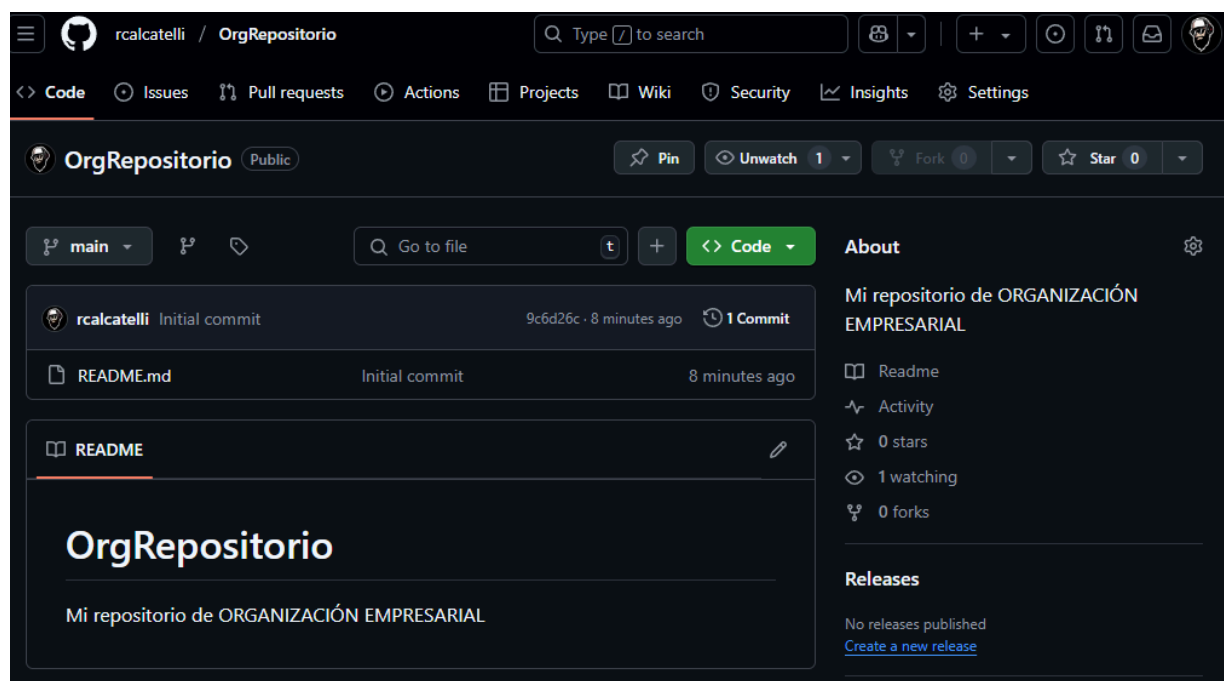
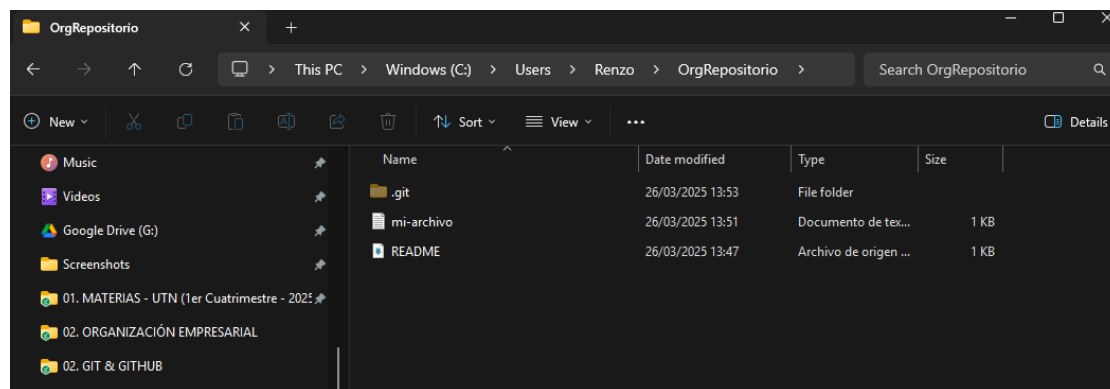
```
PS C:\Users\Renzo> git clone https://github.com/rcalcatelli/OrgRepositorio.git
Cloning into 'OrgRepositorio'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
PS C:\Users\Renzo> cd OrgRepositorio
```

```
PS C:\Users\Renzo> cd OrgRepositorio
PS C:\Users\Renzo\OrgRepositorio> echo "Este es mi archivo (Renzo)" > mi-archivo.txt
```

```
PS C:\Users\Renzo\OrgRepositorio> git add .
PS C:\Users\Renzo\OrgRepositorio> 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\Renzo\OrgRepositorio> git commit -m "Agregando mi archivo.txt"
[main ac53a98] Agregando mi archivo.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 mi-archivo.txt
```



```

PS C:\Users\Renzo\OrgRepositorio> git branch
* main
PS C:\Users\Renzo\OrgRepositorio> git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 346 bytes | 346.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/rcalcatelli/OrgRepositorio.git
9c6d26c..ac53a98  main -> main

```

The screenshot shows the GitHub interface for a repository named 'OrgRepositorio' owned by 'rcalcatelli'. The repository is public and has 1 commit, 0 stars, and 0 forks. The main branch is selected. The commit history shows two commits: 'Initial commit' (10 minutes ago) and 'Agregando mi archivo.txt' (4 minutes ago). The README file is visible, titled 'OrgRepositorio', with the description 'Mi repositorio de ORGANIZACIÓN EMPRESARIAL'. The right sidebar shows the repository's activity, including 1 watcher and 0 forks.

Creando Branchs

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

```

PS C:\Users\Renzo\OrgRepositorio> git checkout -b nuevaRama
Switched to a new branch 'nuevaRama'
PS C:\Users\Renzo\OrgRepositorio> git branch
main
* nuevaRama
PS C:\Users\Renzo\OrgRepositorio>

```

```

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

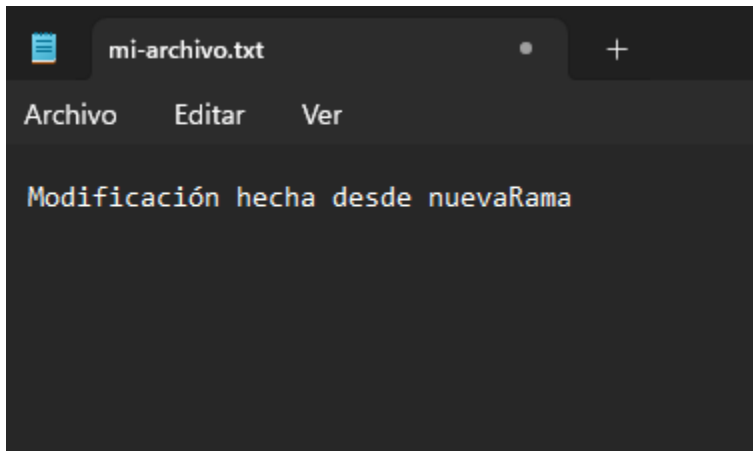
```

```

PS C:\Users\Renzo\OrgRepositorio> git commit -m "Agregando modificación desde nuevaRama"
[nuevaRama 89e2a32] Agregando modificación desde nuevaRama
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\Renzo\OrgRepositorio>

```

mi-archivo modificado desde **nuevaRama**:

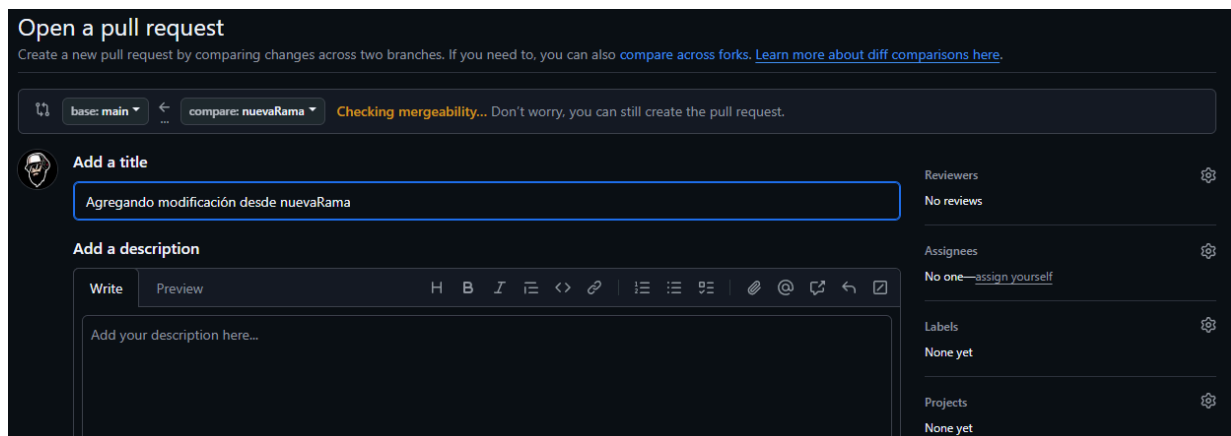
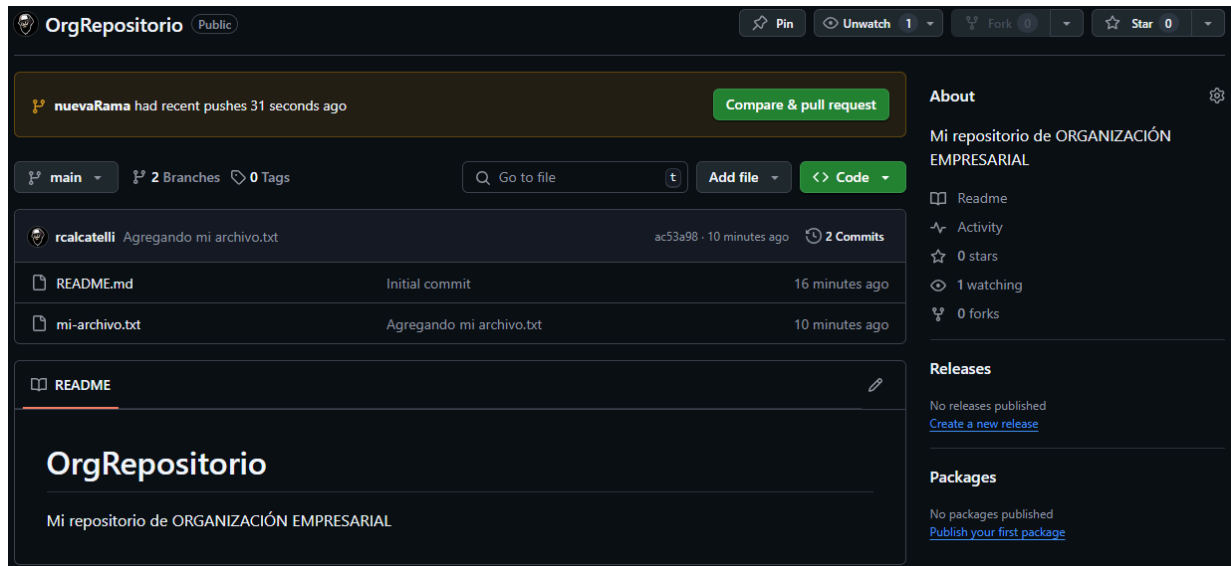


Subimos la nueva rama a GitHub:

```

PS C:\Users\Renzo\OrgRepositorio> git push origin nuevaRama
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 368 bytes | 368.00 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/rcalcatelli/OrgRepositorio/pull/new/nuevaRama
remote:
To https://github.com/rcalcatelli/OrgRepositorio.git
 * [new branch]      nuevaRama -> nuevaRama

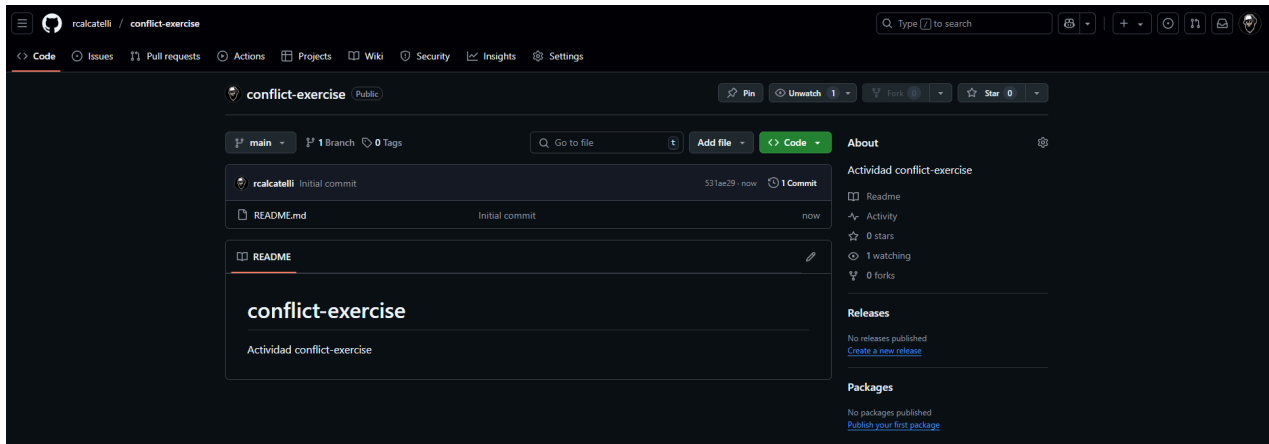
```

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/rcalcatelli/conflict-exercise.git
```

```
PS C:\Users\Renzo> git clone https://github.com/rcalcatelli/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.
PS C:\Users\Renzo>
```

- Entra en el directorio del repositorio:

```
cd conflict-exercise
```

Paso 3: Crear una nueva rama y editar un archivo

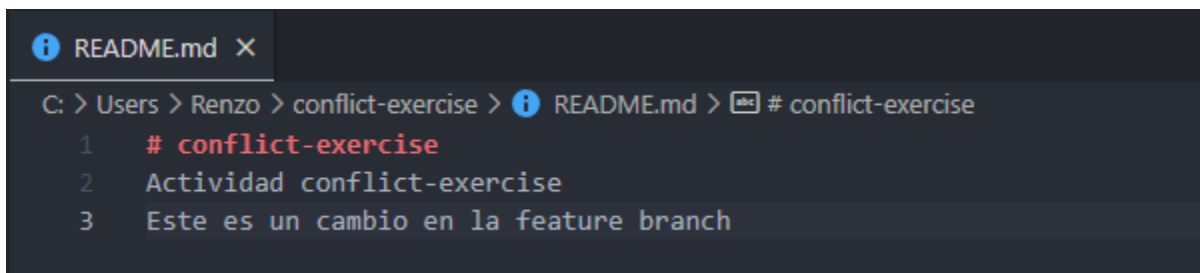
- Crea una nueva rama llamada feature-branch:

```
git checkout -b feature-branch
```

```
PS C:\Users\Renzo> cd conflict-exercise
PS C:\Users\Renzo\conflict-exercise> git checkout -b feature-branch
Switched to a new branch 'feature-branch'
PS C:\Users\Renzo\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



```
README.md X
C: > Users > Renzo > conflict-exercise > README.md > # conflict-exercise
1 # conflict-exercise
2 Actividad conflict-exercise
3 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"
```

```
PS C:\Users\Renzo\conflict-exercise> git add README.md
PS C:\Users\Renzo\conflict-exercise> git commit -m "Added a line in feature-branch"
[feature-branch 3f3e3d4] Added a line in feature-branch
1 file changed, 1 insertion(+)
PS C:\Users\Renzo\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\Renzo\conflict-exercise> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\Renzo\conflict-exercise> 
```

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

Este es un cambio en la main branch.

```
README.md X
C: > Users > Renzo > conflict-exercise > README.md > # conflict-exercise
1  # conflict-exercise
2  Actividad conflict-exercise
3  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"
```

```
PS C:\Users\Renzo\conflict-exercise> git add README.md
PS C:\Users\Renzo\conflict-exercise> git commit -m "Added a line in main branch"
>>
>>
[main 14926bb] Added a line in main branch
1 file changed, 1 insertion(+)
PS C:\Users\Renzo\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\Renzo\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\Renzo\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:

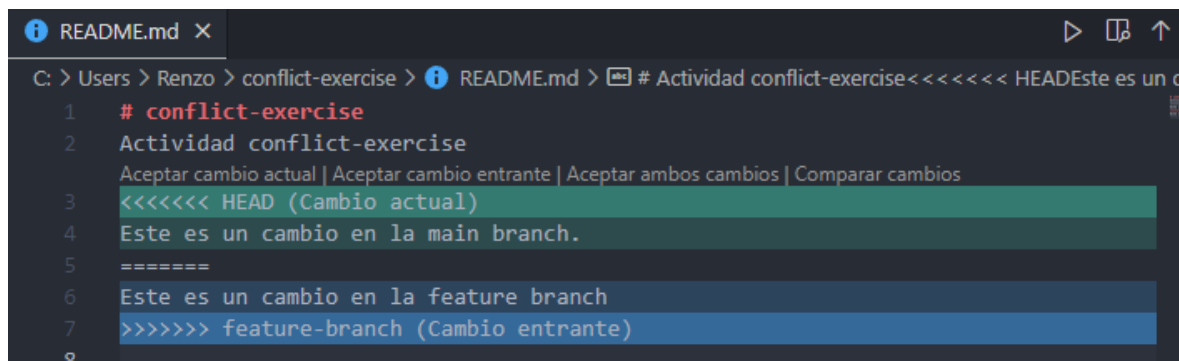
```
<<<<<<< HEAD
```

Este es un cambio en la main branch.

```
=====
```

Este es un cambio en la feature branch.

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

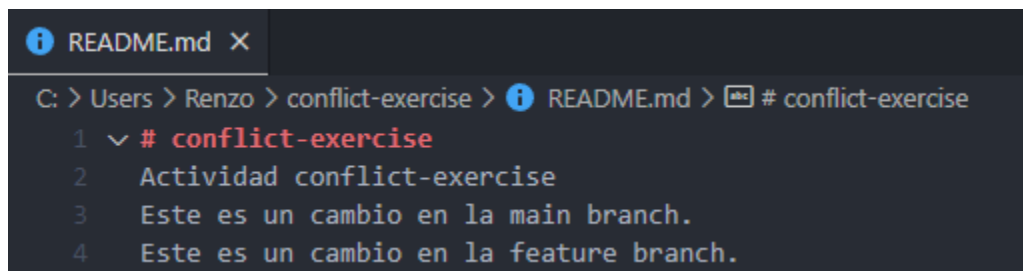


```

C: > Users > Renzo > conflict-exercise > README.md > # Actividad conflict-exercise<<<<<< HEADEste es un c
1  # conflict-exercise
2  Actividad conflict-exercise
   Aceptar cambio actual | Aceptar cambio entrante | Aceptar ambos cambios | Comparar cambios
3  <<<<<<< HEAD (Cambio actual)
4  Este es un cambio en la main branch.
5  =====
6  Este es un cambio en la feature branch
7  >>>>>>> feature-branch (Cambio entrante)
8

```

- 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 > Renzo > conflict-exercise > README.md > # conflict-exercise
1  ✓ # conflict-exercise
2  Actividad conflict-exercise
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\Renzo\conflict-exercise> git add README.md
● PS C:\Users\Renzo\conflict-exercise> git commit -m "Resolved merge conflict"
  [main 4fe8c12] Resolved merge conflict
○ PS C:\Users\Renzo\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\Renzo\conflict-exercise> git push origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 748 bytes | 748.00 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/rcalcatelli/conflict-exercise.git
  531ae29..4fe8c12  main -> main
○ PS C:\Users\Renzo\conflict-exercise> |
```

- También sube la feature-branch si deseas:

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

```
PS C:\Users\Renzo\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/rcalcatelli/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/rcalcatelli/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch
```

Paso 8: Verificar en GitHub

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

The screenshot shows the GitHub interface for a repository named 'conflict-exercise' by user 'rcalcatelli'. The repository is public and has 2 branches and 0 tags. The main branch is selected. A commit titled 'Resolved merge conflict' by 'rcalcatelli' is shown, with a commit hash of '4fe8c12' and a timestamp of '3 minutes ago'. The commit message is 'Resolved merge conflict'. The file 'README.md' is highlighted, showing a 'Resolved merge conflict' status. The README content is displayed below, showing the repository name 'conflict-exercise' and a message: 'Actividad conflict-exercise Este es un cambio en la main branch. Este es un cambio en la feature branch.'

Commits

main

All users

All time

Commits on Mar 26, 2025

Resolved merge conflict

rcalcatelli committed 3 minutes ago

4fe8c12

<>

Added a line in main branch

rcalcatelli committed 9 minutes ago

14926bb

<>

Added a line in feature-branch

rcalcatelli committed 13 minutes ago

3f3e3d4

<>

Initial commit

rcalcatelli authored 25 minutes ago

Verified

531ae29

<>