

Nombre y Apellido: Daniel Alderete

Comisión: 6

## Práctico 2: Git y GitHub

1) Contestar las siguientes preguntas utilizando las guías y documentación proporcionada

(Desarrollar las respuestas) :

- ¿Qué es GitHub?

GitHub es una plataforma basada en la nube que permite gestionar y colaborar en proyectos de programación utilizando Git, un sistema de control de versiones. Con GitHub, los desarrolladores pueden almacenar, compartir y trabajar juntos en el código de manera eficiente.

- ¿Cómo crear un repositorio en GitHub?

Crear un repositorio en GitHub:

1. **Inicia sesión en GitHub:** Ve a [GitHub](#) e ingresa con tu cuenta. Si no tienes una, puedes registrarte gratuitamente.
2. **Crea un nuevo repositorio:**
  - Haz clic en el botón **+** en la esquina superior derecha de la página y selecciona **New repository** (Nuevo repositorio).
  - Ingresa un nombre para tu repositorio en el campo "Repository name" (ejemplo: mi-proyecto).
  - Opcional: Agrega una breve descripción para explicar el propósito de tu proyecto.
  - Decide si el repositorio será **público** o **privado**.
3. **Configuraciones adicionales:**
  - Si lo deseas, selecciona la opción para inicializar el repositorio con un archivo README. Este archivo es útil para agregar información sobre tu proyecto.
  - Puedes añadir un archivo .gitignore si necesitas excluir ciertos archivos del seguimiento de Git.
  - Selecciona una licencia para tu proyecto si es necesario (por ejemplo, MIT License).
4. **Crear el repositorio:**
  - Haz clic en **Create repository** (Crear repositorio).

- ¿Cómo crear una rama en Git?

Crear una rama en Git:

1. **Verifica las ramas existentes:** Usa el siguiente comando para ver las ramas del repositorio y verificar en cuál estás actualmente:
2. `git branch`

3. **Crear una nueva rama:** Para crear una nueva rama, se utiliza:
4. `git branch nombre-de-la-rama`

Cambia nombre-de-la-rama por el nombre que quieras darle. Por ejemplo: nueva-funcionalidad.

5. **Moverte a la nueva rama:** Después de crearla, cámbiate a la nueva rama con:
6. `git checkout nombre-de-la-rama`
7. **Combinar pasos 2 y 3:** Si deseas crear una rama y cambiarte a ella al mismo tiempo, puedes utilizar:
8. `git checkout -b nombre-de-la-rama`
9. **Confirmar que estás en la rama correcta:** Vuelve a usar el comando:
10. `git branch`

La rama activa aparecerá marcada con un asterisco (\*).

- ¿Cómo cambiar a una rama en Git?

Para cambiar a una rama en Git, se utiliza el comando `git checkout` seguido del nombre de la rama a la que se desea cambiar.

Paso a paso

1. Crear una rama con el comando `git branch`
2. Usar `git checkout` seguido del nombre de la rama a la que se desea cambiar

Por ejemplo, para crear una rama nueva a partir de la rama principal y cambiar a ella, se puede hacer lo siguiente:

1. `git branch new_branch`
2. `git checkout new_branch`

También se puede usar el argumento `-b` con `git checkout` para crear y cambiar a una rama al mismo tiempo. Por ejemplo, `git checkout -b new_branch`.

Para forzar a Git a cambiar de rama, se puede usar la opción `-f` o `--force` con el comando `git checkout`.

Las ramas permiten trabajar en varias funciones al mismo tiempo y facilitan la colaboración.

- ¿Cómo fusionar ramas en Git?

Fusionar ramas en Git:

1. **Cambiarte a la rama en la que quieres fusionar:** Primero, asegúrate de estar en la rama de destino (la rama donde quieres integrar los cambios). Por ejemplo, si quieres fusionar cambios en main:
2. `git checkout main`
3. **Fusionar la rama:** Utiliza el siguiente comando para fusionar la rama que contiene los cambios:
4. `git merge nombre-de-la-rama`

Sustituye nombre-de-la-rama con el nombre de la rama que deseas fusionar.

5. **Resolver conflictos (si los hay):** Si Git encuentra conflictos entre las ramas, te notificará y tendrás que resolverlos manualmente. Abre los archivos con conflictos, ajusta el código según lo que necesites, y luego marca los conflictos como resueltos usando:
6. `git add nombre-del-archivo`
7. **Confirmar la fusión:** Después de resolver los conflictos (si existen), finaliza el proceso con un commit:
8. `git commit`

- ¿Cómo crear un commit en Git?

Crear un commit en Git:

1. **Asegúrate de estar en el repositorio correcto:** Navega al directorio del proyecto en tu terminal:
2. `cd ruta-del-repositorio`
3. **Verifica el estado del repositorio:** Antes de hacer un commit, puedes ver los archivos modificados o nuevos utilizando:
4. `git status`
5. **Añade los cambios al área de preparación (staging area):** Si deseas agregar un archivo específico:
6. `git add nombre-del-archivo`

O si quieres añadir todos los cambios en el directorio actual:

`git add .`

7. **Crea el commit:** Usa el siguiente comando para guardar los cambios con un mensaje descriptivo:
8. `git commit -m "Mensaje descriptivo sobre los cambios realizados"`

El mensaje debería explicar claramente qué cambios estás incluyendo en este commit.

9. **Confirma el commit:** Para verificar que el commit se ha registrado, puedes ver el historial con:

10. `git log`

- ¿Cómo enviar un commit a GitHub?

Enviar un commit a GitHub:

1. **Asegúrate de estar en el repositorio correcto:** En tu terminal, navega al directorio del proyecto:
2. `cd ruta-del-repositorio`
3. **Verifica que tu repositorio remoto está configurado:** Para confirmar que el repositorio remoto está vinculado, usa:
4. `git remote -v`

Si no tienes un repositorio remoto configurado, agrega uno con:

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

5. **Verifica el estado del repositorio:** Asegúrate de que el commit ya se ha creado y que no quedan cambios sin preparar:
6. `git status`
7. **Empuja el commit al repositorio remoto:** Usa el siguiente comando para enviar tus cambios al repositorio remoto:
8. `git push origin nombre-de-la-rama`

Sustituye nombre-de-la-rama con la rama en la que estás trabajando (por ejemplo, main).

9. **Autenticación en GitHub:** Si GitHub requiere autenticación, introduce tus credenciales o utiliza un token de acceso personal.

- ¿Qué es un repositorio remoto?

Un repositorio remoto es una versión de tu repositorio local que está alojada en un servidor o plataforma en línea, como GitHub, GitLab o Bitbucket. Estos repositorios permiten colaborar con otras personas y mantener una copia centralizada y sincronizada de tu proyecto.

Utilidad:

- **Colaboración en equipo:** Varios desarrolladores pueden clonar o descargar el repositorio remoto para trabajar en él simultáneamente.
- **Copia de seguridad:** Mantiene una copia de tu trabajo fuera de tu máquina local, útil en caso de fallos en el sistema.

- **Sincronización:** Puedes enviar (push) o recibir (pull) cambios entre tu repositorio local y remoto para mantenerlos actualizados.
- **Accesibilidad:** Permite acceder a tu código desde cualquier lugar con acceso a Internet.

- ¿Cómo agregar un repositorio remoto a Git?

Agregar un repositorio remoto a Git:

1. **Crea un repositorio remoto:** Si estás usando una plataforma como GitHub, primero crea un repositorio allí siguiendo los pasos que ya discutimos.
2. **Obtén la URL del repositorio remoto:** Ve a tu repositorio en GitHub (o la plataforma que uses) y copia la URL. Por ejemplo, puede tener un formato como este:
3. `https://github.com/usuario/nombre-del-repositorio.git`
4. **Abre tu terminal:** Navega al directorio de tu proyecto local:
5. `cd ruta-del-repositorio`
6. **Vincula el repositorio remoto:** Usa el comando `git remote add` para vincular tu repositorio local al remoto. El formato es:
7. `git remote add origin URL-del-repositorio`

Sustituye URL-del-repositorio con la dirección que copiaste en el paso 2. Por ejemplo:

`git remote add origin https://github.com/usuario/nombre-del-repositorio.git`

8. **Verifica que el remoto se agregó correctamente:** Usa este comando para comprobar que el repositorio remoto está configurado:
9. `git remote -v`

- ¿Cómo empujar cambios a un repositorio remoto?

Empujar (o "push") cambios a un repositorio remoto es el proceso de enviar los commits realizados en tu repositorio local para que se guarden en el repositorio remoto.

El paso a paso:

1. **Asegúrate de estar en la rama correcta:** Antes de empujar los cambios, verifica que estás en la rama que deseas enviar:
2. `git branch`

Si necesitas cambiar de rama, utiliza:

`git checkout nombre-de-la-rama`

3. **Verifica el estado del repositorio:** Asegúrate de que todos los cambios estén en un commit utilizando:
4. `git status`
5. **Empuja los cambios al repositorio remoto:** Usa el siguiente comando para enviar los cambios:
6. `git push origin nombre-de-la-rama`

Cambia nombre-de-la-rama por la rama en la que estás trabajando (por ejemplo, main).

7. **Autenticación en GitHub:** Si se te solicita, ingresa tus credenciales o utiliza un token de acceso personal para completar el proceso.
8. **Confirma que los cambios se han enviado:** Ve al repositorio en GitHub y verifica que los commits aparecen en la rama correspondiente.

- ¿Cómo tirar de cambios de un repositorio remoto?

Tirar (o "pull") de cambios de un repositorio remoto en Git implica traer las actualizaciones desde el repositorio remoto a tu repositorio local. Así es como puedes hacerlo:

1. **Asegúrate de estar en el repositorio local correcto:** Navega al directorio del proyecto en tu terminal:
2. `cd ruta-del-repositorio`
3. **Actualiza la rama local desde el remoto:** Usa el siguiente comando para traer los cambios:
4. `git pull origin nombre-de-la-rama`

Sustituye nombre-de-la-rama con la rama desde la que deseas traer los cambios (por ejemplo, main).

5. **Resolver conflictos (si es necesario):** Si hay conflictos entre los cambios locales y remotos, Git te informará. Necesitarás resolverlos manualmente editando los archivos en conflicto, guardando los cambios y marcándolos como resueltos:
6. `git add nombre-del-archivo`

Luego, finaliza el proceso con:

`git commit`

7. **Confirma que los cambios fueron aplicados:** Usa este comando para ver los últimos commits y confirmar que los cambios se integraron:
8. `git log`

- ¿Qué es un fork de repositorio?

Un fork de un repositorio es una copia de un repositorio existente que se crea bajo tu cuenta en una plataforma como GitHub. Es una herramienta poderosa para colaborar en proyectos, especialmente en código abierto, ya que permite hacer modificaciones sin afectar el repositorio original.

- ¿Cómo crear un fork de un repositorio?

Crear un fork de un repositorio en GitHub:

1. **Busca el repositorio original:**

- Ve al repositorio público en GitHub que deseas forkar.

2. **Haz clic en el botón "Fork":**

- En la parte superior derecha de la página del repositorio, encontrarás un botón que dice **Fork**. Haz clic en él.

3. **Selecciona tu cuenta o una organización:**

- GitHub te pedirá que selecciones dónde quieres que se cree el fork: puede ser en tu cuenta personal o en una organización en la que participes.

4. **Espera mientras GitHub crea el fork:**

- Una vez que selecciones la opción, GitHub copiará el repositorio original y lo creará en tu cuenta. Esto puede tardar unos segundos.

5. **Confirma tu fork:**

- Cuando se complete, serás redirigido al nuevo repositorio en tu cuenta. Verás que en la parte superior aparece un mensaje indicando que es un fork del repositorio original.

6. **Clona tu fork (opcional):**

- Si deseas trabajar localmente en tu fork, clónalo en tu máquina:
- `git clone URL-de-tu-fork`
- `cd nombre-del-repositorio`

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

Enviar una solicitud de extracción (pull request) en GitHub cómo hacerlo:

1. **Asegúrate de que tus cambios están en una rama:**

- Trabaja en una rama específica y realiza todos tus commits.
- Envía los cambios al repositorio remoto usando:

- git push origin nombre-de-la-rama
- 2. **Dirígete al repositorio en GitHub:**
  - Ve a la página del fork o repositorio donde has empujado los cambios.
- 3. **Inicia el Pull Request:**
  - Aparecerá una opción para "Compare & pull request" justo después de empujar los cambios. Haz clic en ella.
  - Si no aparece automáticamente, haz clic en la pestaña **Pull requests** del repositorio y luego en el botón **New pull request**.
- 4. **Selecciona la rama de origen y la rama de destino:**
  - Elige la rama de tu repositorio (fork) que contiene los cambios.
  - Selecciona la rama del repositorio original a la que deseas fusionar tus cambios (por ejemplo, main o develop).
- 5. **Agrega detalles al Pull Request:**
  - Escribe un título claro y un mensaje que explique los cambios realizados.
  - Si es necesario, incluye capturas de pantalla, referencias o cualquier información relevante.
- 6. **Envía el Pull Request:**
  - Haz clic en el botón **Create pull request**.
- 7. **Espera revisión:**
  - Los mantenedores del repositorio revisarán tu propuesta. Es posible que te hagan preguntas o sugerencias antes de aceptar (o cerrar) tu solicitud.

• ¿Cómo aceptar una solicitud de extracción?

Aceptar una solicitud de extracción (pull request) en GitHub es el paso final para integrar cambios propuestos en tu repositorio. Aquí tienes cómo hacerlo:

1. **Revisar los cambios propuestos:**
  - Ve a la pestaña **Pull requests** en tu repositorio.
  - Haz clic en la solicitud de extracción que deseas aceptar.
  - Lee la descripción y, si es necesario, examina los archivos modificados en la sección **Files changed**.
2. **Probar los cambios (opcional):**
  - Si los cambios son importantes o complejos, puedes clonar la rama del pull request para probarla en tu entorno local:



- `git fetch origin nombre-de-la-rama-del-pull-request`
- `git checkout nombre-de-la-rama-del-pull-request`

### 3. Resolver conflictos (si los hay):

- Si Git detecta conflictos entre la solicitud de extracción y la rama de destino, deberás resolverlos antes de aceptar.
- Los conflictos se pueden resolver directamente en GitHub (si son simples) o en tu entorno local.

### 4. Aceptar el pull request:

- Si todo está en orden, haz clic en el botón **Merge pull request**.
- Escribe un mensaje descriptivo para la fusión (opcional).
- Confirma la acción haciendo clic en **Confirm merge**.

### 5. Eliminar la rama (opcional):

- Una vez que los cambios se hayan fusionado, GitHub te dará la opción de eliminar la rama asociada al pull request. Esto ayuda a mantener el repositorio limpio si ya no necesitas esa rama.

## • ¿Qué es una etiqueta en Git?

En Git, una **etiqueta (tag)** es un marcador que se utiliza para identificar un punto específico en la historia de un repositorio. Generalmente, las etiquetas se usan para marcar versiones importantes de un proyecto, como el lanzamiento de una nueva versión de software.

### Tipos de etiquetas en Git:

1. **Lightweight (ligeras):** Son básicamente punteros a un commit específico, sin ningún metadato adicional.
2. **Annotated (anotadas):** Contienen información adicional como un mensaje, la fecha, el autor y, opcionalmente, una firma criptográfica. Estas son ideales para versiones oficiales.

### Cómo funcionan las etiquetas:

- Las etiquetas no son ramas. Son inmutables, lo que significa que una vez creadas no cambian.
- Permiten referenciar versiones específicas de tu código fácilmente.

### Cómo crear etiquetas:

- **Etiqueta ligera:**
- `git tag nombre-de-la-etiqueta`
- **Etiqueta anotada:**
- `git tag -a nombre-de-la-etiqueta -m "Mensaje descriptivo de la etiqueta"`

### **Cómo enviar etiquetas a un repositorio remoto:**

Si deseas compartir las etiquetas con un repositorio remoto, usa:

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

O todas las etiquetas a la vez:

`git push --tags`

- ¿Cómo crear una etiqueta en Git?

Crear una etiqueta en Git es un excelente paso para marcar puntos importantes en tu proyecto, como el lanzamiento de una versión. Aquí tienes el proceso detallado:

### **1. Decide el tipo de etiqueta que necesitas**

- **Etiqueta ligera:** Es básica, solo marca un commit sin metadatos adicionales.
- **Etiqueta anotada:** Incluye información como el autor, fecha y un mensaje descriptivo. Ideal para versiones oficiales.

### **2. Crea la etiqueta**

- Para una etiqueta ligera:
- `git tag nombre-de-la-etiqueta`
- Para una etiqueta anotada:
- `git tag -a nombre-de-la-etiqueta -m "Mensaje descriptivo"`

Por ejemplo:

`git tag -a v1.0.0 -m "Versión 1.0.0 - Primer lanzamiento oficial"`

### **3. Verifica las etiquetas creadas**

Usa el siguiente comando para listar todas las etiquetas en tu repositorio:

`git tag`

### **4. Opcional: Comparar etiqueta con otro commit**

Si deseas ver detalles del commit asociado a una etiqueta, utiliza:

`git show nombre-de-la-etiqueta`

## 5. Sube las etiquetas al repositorio remoto

- Para subir una etiqueta específica:
- `git push origin nombre-de-la-etiqueta`
- Para subir todas las etiquetas de una vez:
- `git push --tags`

## 6. Confirma que la etiqueta se subió

Ve a tu repositorio remoto, como GitHub, y verifica que la etiqueta aparece en la sección de "Releases" o "Tags".

- ¿Cómo enviar una etiqueta a GitHub?

Enviar una etiqueta (tag) a GitHub es muy sencillo. Aquí están los pasos para compartir tus etiquetas con el repositorio remoto:

### 1. Crea una etiqueta en tu repositorio local:

- Si aún no tienes la etiqueta creada, puedes hacerlo con:
- `git tag -a nombre-de-la-etiqueta -m "Mensaje descriptivo"`

Por ejemplo:

```
git tag -a v1.0.0 -m "Versión 1.0.0"
```

### 2. Sube una etiqueta específica:

- Si solo deseas enviar una etiqueta al repositorio remoto, usa:
- `git push origin nombre-de-la-etiqueta`

Por ejemplo:

```
git push origin v1.0.0
```

### 3. Sube todas las etiquetas de una vez:

- Si quieres subir todas las etiquetas que has creado localmente, utiliza:
- `git push --tags`

### 4. Verifica en GitHub:

- Ve a tu repositorio en GitHub y revisa la pestaña **Tags** o **Releases** para asegurarte de que la etiqueta se haya subido correctamente.

- ¿Qué es un historial de Git?

El historial de Git se refiere al registro de los cambios realizados en un repositorio a lo largo del tiempo. Este historial incluye todos los commits, los cuales representan cambios individuales aplicados al proyecto. Cada commit en el historial contiene detalles como el autor, la fecha, el mensaje del commit y un identificador único (hash).

- ¿Cómo ver el historial de Git?

Ver el historial de Git te permite examinar los cambios realizados en el repositorio. Aquí tienes los comandos más útiles:

1. **Ver el historial completo:** Usa el comando:
2. `git log`

Esto muestra detalles completos de cada commit, como el autor, la fecha, el mensaje y el hash único.

3. **Historial compacto (resumido):** Si prefieres un resumen más sencillo, utiliza:
4. `git log --oneline`

Esto mostrará solo el hash y el mensaje de cada commit en una sola línea.

5. **Historial con gráfico de ramas:** Para visualizar mejor las ramas y cómo se relacionan los commits:
6. `git log --oneline --graph --all`
7. **Filtrar por autor:** Si quieres ver los commits realizados por un autor específico:
8. `git log --author="nombre-del-autor"`
9. **Buscar palabras clave en los mensajes de commit:** Para localizar commits que mencionen un tema específico:
10. `git log --grep="palabra-clave"`

- ¿Cómo buscar en el historial de Git?

Buscar en el historial de Git te permite localizar commits específicos basándote en diversos criterios, como palabras clave, autor o fecha. Aquí tienes los comandos más útiles:

1. **Buscar por mensaje de commit:**

Usa el comando `--grep` para buscar palabras clave en los mensajes de commit:

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

Ejemplo:

```
git log --grep="corrección de errores"
```

2. **Filtrar por autor:**

Si quieres ver commits realizados por un autor en particular:

```
git log --author="nombre-del-autor"
```

Ejemplo:

```
git log --author="Jennifer"
```

### 3. Buscar por fecha:

Puedes buscar commits realizados en un rango de fechas con el flag `--since` y `--until`:

```
git log --since="2025-01-01" --until="2025-03-31"
```

Esto mostrará los commits entre enero y marzo de 2025.

### 4. Combinar filtros:

Puedes usar varios filtros al mismo tiempo. Por ejemplo, buscar commits por un autor y palabras clave:

```
git log --author="Jennifer" --grep="nueva funcionalidad"
```

### 5. Historial compacto y gráfico con filtros:

Para visualizar los resultados con un resumen más simple y un gráfico de ramas:

```
git log --oneline --graph --all --grep="palabra-clave"
```

### 6. Buscar en los cambios de los archivos:

Si necesitas buscar en el contenido modificado (no solo en los mensajes), usa:

```
git log -S "cadena-de-texto"
```

Ejemplo:

```
git log -S "nombre_de_la_variable"
```

Estas herramientas hacen que sea más fácil ubicar información específica en tu historial.

- ¿Cómo borrar el historial de Git?

En Git, **no es posible borrar completamente el historial de commits de forma nativa** sin afectar el estado y estructura del repositorio. Sin embargo, puedes reescribir el historial en ciertas circunstancias. Aquí hay algunos métodos que podrías considerar según tus necesidades:

#### 1. Reescribir el historial localmente

Si solo deseas modificar el historial en tu repositorio local:

- Usa el comando `git rebase` para reorganizar y reescribir commits específicos.

- Ejemplo:
- `git rebase -i HEAD~n`

Sustituye n con el número de commits que deseas modificar. Esto abrirá un editor para que puedas modificar o eliminar los commits.

## 2. Borrar todo el historial (y comenzar desde cero)

Este enfoque reinicia el historial del repositorio. Hazlo con cuidado, ya que afecta **todas las ramas y commits**:

# Paso 1: Borra la historia actual

```
git checkout --orphan nueva-rama
```

```
git add .
```

```
git commit -m "Inicio desde cero"
```

# Paso 2: Reemplaza la rama original

```
git branch -D main
```

```
git branch -m main
```

# Paso 3: Actualiza el repositorio remoto

```
git push --force origin main
```

Esto crea un nuevo historial desde cero, pero elimina el historial existente tanto local como remotamente.

## 3. Manipular el historial remoto

Si necesitas que el historial modificado también se refleje en GitHub, realiza un **push forzado** después de reescribir los commits:

```
git push --force
```

Ten en cuenta que **esto podría afectar a otros colaboradores**, ya que sus repositorios locales podrían desincronizarse.

### Precaución:

Antes de proceder, analiza si es necesario borrar el historial o si sería mejor preservarlo. El historial de Git contiene información importante para rastrear cambios en tu proyecto.

Además, usa estos métodos solo si tienes control total del repositorio y no estás trabajando en equipo.

- ¿Qué es un repositorio privado en GitHub?

Un **repositorio privado** en GitHub es un tipo de repositorio al que solo tú y las personas que invites tienen acceso. A diferencia de los repositorios públicos, los repositorios privados no son visibles para el público en general ni para usuarios no autorizados.

#### **Características principales de un repositorio privado:**

##### **1. Control de acceso:**

- Puedes elegir quién puede ver y colaborar en tu repositorio.
- Se requiere invitación para que otros usuarios puedan acceder.

##### **2. Perfecto para trabajo confidencial:**

- Útil para proyectos personales, colaboraciones privadas o trabajo que aún no está listo para ser publicado.

##### **3. Mismas funcionalidades que un repositorio público:**

- Puedes usar todas las herramientas de GitHub, como ramas, pull requests, GitHub Actions, etc.

##### **4. Gestión de colaboradores:**

- Desde la configuración del repositorio, puedes agregar colaboradores específicos indicando su nombre de usuario o correo en GitHub.

Con un repositorio privado, tienes mayor seguridad y privacidad mientras desarrollas tus proyectos.

- ¿Cómo crear un repositorio privado en GitHub?

#### **Cómo crear un repositorio privado:**

1. Ve a GitHub y haz clic en el botón **+** (parte superior derecha), luego selecciona **New repository**.
2. Completa la información del repositorio.
3. En "Repository type", selecciona **Private**.
4. Haz clic en **Create repository**.

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

Invitar a alguien a colaborar en un repositorio privado en GitHub:

### 1. Ve a la configuración del repositorio

- Abre el repositorio privado al que deseas invitar a alguien.
- Haz clic en la pestaña **Settings** en la parte superior derecha de la página.

### 2. Accede a "Manage access"

- En el menú lateral izquierdo, busca la sección **Manage access** y haz clic en ella.
- Allí verás la lista de colaboradores actuales y opciones para agregar nuevos.

### 3. Invitar colaboradores

- Haz clic en el botón **Invite a collaborator**.
- Ingresa el nombre de usuario o correo electrónico de GitHub de la persona que quieres invitar.
- Haz clic en **Send invitation**.

### 4. Confirmación de la invitación

- La persona recibirá una invitación por correo electrónico o directamente en su cuenta de GitHub. Debe aceptar la invitación para poder acceder al repositorio.

### 5. Gestionar permisos

- Puedes decidir el nivel de acceso de los colaboradores:
  - **Read:** Solo lectura.
  - **Write:** Lectura y escritura (pueden realizar cambios en el repositorio).
  - **Admin:** Control total, incluida la capacidad de modificar la configuración del repositorio.

- ¿Qué es un repositorio público en GitHub?

Un **repositorio público** en GitHub es un repositorio que está disponible para que cualquier persona lo pueda ver y, dependiendo de su configuración, incluso interactuar con él. Es ideal para proyectos de código abierto o cualquier trabajo que desees compartir con el mundo.

### Características principales de un repositorio público:

#### 1. Visibilidad total:

- Cualquier usuario de GitHub (o incluso sin cuenta) puede ver el contenido del repositorio.
- Esto incluye archivos, commits, ramas y etiquetas.



## 2. Colaboración abierta:

- Otros usuarios pueden hacer forks de tu repositorio y enviar pull requests para contribuir.
- Puedes decidir quién tiene permisos de escritura o administración.

## 3. Perfecto para código abierto:

- Utilizado principalmente para proyectos colaborativos donde todos pueden aprender, usar o mejorar tu trabajo.

## 4. Opciones de licencia:

- Puedes definir una licencia para establecer cómo otros pueden usar, modificar o distribuir tu proyecto.

- ¿Cómo crear un repositorio público en GitHub?

### Cómo crear un repositorio público en GitHub:

1. Ve a GitHub y haz clic en el botón **+**, luego selecciona **New repository**.
2. Completa el nombre y la descripción.
3. En "Repository type", selecciona **Public**.
4. Haz clic en **Create repository**.

- ¿Cómo compartir un repositorio público en GitHub?

Compartir un repositorio público en GitHub es muy fácil, ya que estos repositorios son visibles para todos. Aquí están los pasos para hacerlo:

### 1. Asegúrate de que tu repositorio es público

- Si ya tienes el repositorio creado, verifica que sea público. En GitHub, ve a la sección **Settings** de tu repositorio y confirma que el tipo de repositorio esté marcado como **Public**. Si no lo está, cámbialo desde la configuración.

### 2. Copia la URL del repositorio

- En la página principal de tu repositorio, encontrarás una barra de URL en la parte superior (con un botón de "Code"). Copia esa URL; será algo como:
- `https://github.com/usuario/nombre-del-repositorio`

### 3. Comparte la URL

- Ahora puedes enviar la URL por correo electrónico, compartirla en redes sociales o incluirla en tu sitio web.

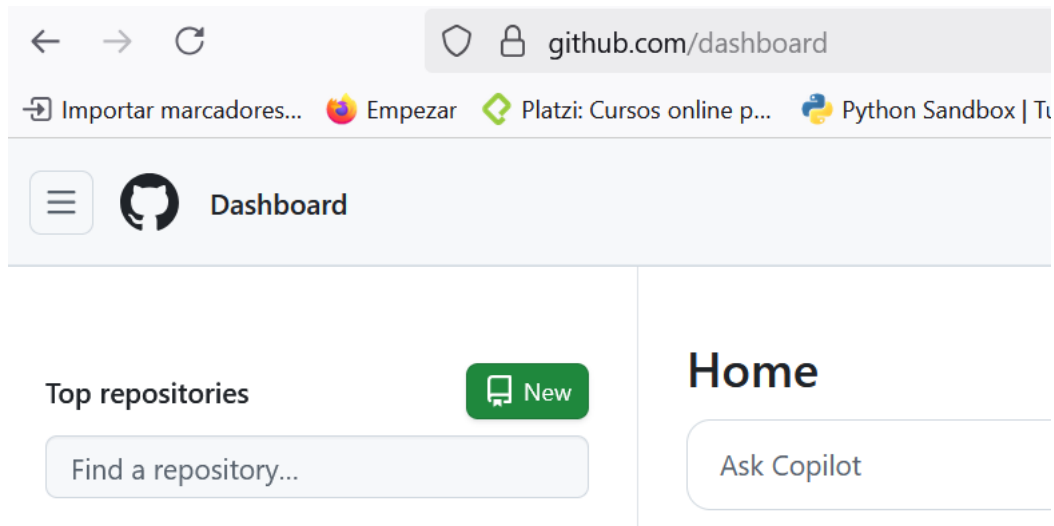
- Por ejemplo, si estás trabajando en un proyecto de código abierto, puedes publicarlo en foros, blogs o comunidades de desarrollo.

#### 4. Opcional: Destacar tu repositorio

- Agrega una descripción clara y una documentación completa (como un archivo README.md) para que sea más atractivo y fácil de entender para quienes lo visiten.
- Si deseas permitir contribuciones, incluye instrucciones sobre cómo colaborar (por ejemplo, crear pull requests).

2) Realizar la siguiente actividad:

- Crear un repositorio.




- o Dale un nombre al repositorio.
- o Elije el repositorio sea público.
- o 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 \*

 Dann07-ops ▾

Repository name \*

/

mi-proyecto

✓ mi-proyecto is available.

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

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

Activar W  
Ve a Configu

- Agregando un Archivo

o Crea un archivo simple, por ejemplo, "mi-archivo.txt".

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

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

```

PS C:\Users\User> git clone https://github.com/Dann07-ops/mi-proyecto.git
Cloning into 'mi-proyecto'...
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\User> cd mi-proyecto
PS C:\Users\User\mi-proyecto> echo "Este es mi archivo" > mi-archivo.txt
PS C:\Users\User\mi-proyecto> git add .
PS C:\Users\User\mi-proyecto> git commit -m "Agregando mi-archivo.txt"
[main 8a1867a] Agregando mi-archivo.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 mi-archivo.txt
PS C:\Users\User\mi-proyecto> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
PS C:\Users\User\mi-proyecto> git push origin main

```

```

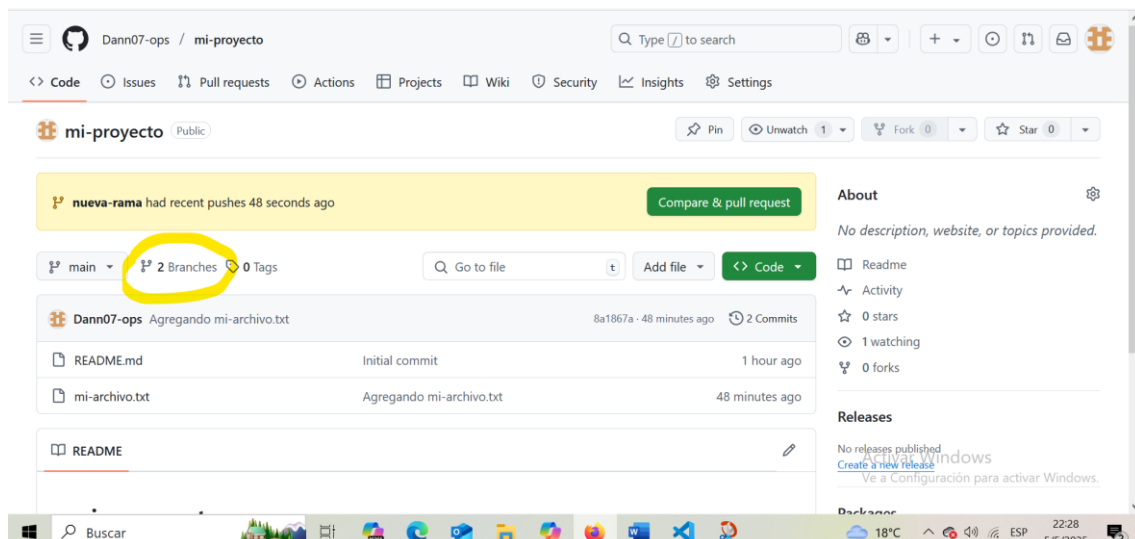
PS C:\Users\User\mi-proyecto> git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 329 bytes | 329.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Dann07-ops/mi-proyecto.git
 407848f..8a1867a  main -> main
PS C:\Users\User\mi-proyecto>

```

The screenshot shows the GitHub interface for a public repository named 'mi-proyecto' by user 'Dann07-ops'. The repository has 1 branch (main) and 0 tags. The commit history shows two commits: an initial commit for 'README.md' 11 minutes ago, and a subsequent commit 'Agregando mi-archivo.txt' 4 minutes ago. The README file is visible and contains the text 'mi-proyecto'. On the right sidebar, the 'About' section is empty, and the 'Releases' and 'Packages' sections also show no published content.

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

```
PS C:\Users\User\mi-proyecto> git branch nueva-rama
PS C:\Users\User\mi-proyecto> git checkout nueva-rama
Switched to branch 'nueva-rama'
PS C:\Users\User\mi-proyecto> echo "Nuevo contenido" > archivo-nuevo.txt
PS C:\Users\User\mi-proyecto> git add archivo-nuevo.txt
PS C:\Users\User\mi-proyecto> git commit -m "Agregando archivo-nuevo.txt"
[nueva-rama 3ee596e] Agregando archivo-nuevo.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 archivo-nuevo.txt
PS C:\Users\User\mi-proyecto> git push origin nueva-rama
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 359 bytes | 359.00 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/Dann07-ops/mi-proyecto/pull/new/nueva-rama
remote:
To https://github.com/Dann07-ops/mi-proyecto.git
 * [new branch]      nueva-rama -> nueva-rama
PS C:\Users\User\mi-proyecto>
```

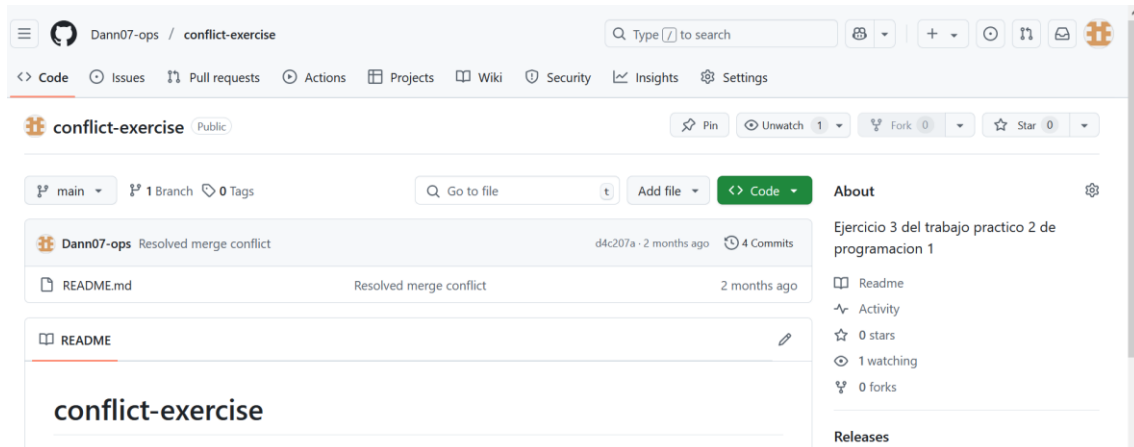


3) Realizar la siguiente actividad:

Paso 1: Crear un repositorio en GitHub

- Ve a GitHub e inicia sesión en tu cuenta.

- Haz clic en el botón "New" o "Create repository" para crear un nuevo repositorio.
- Asigna un nombre al repositorio, por ejemplo, conflict-exercise.
- Opcionalmente, añade una descripción.
- Marca la opción "Initialize this repository with a README".
- Haz clic en "Create repository".



## Paso 2: Clonar el repositorio a tu máquina local

- Copia la URL del repositorio (usualmente algo como <https://github.com/tuusuario/conflict-exercise.git>).
- Abre la terminal o línea de comandos en tu máquina.

- Clona el repositorio usando el comando:

```
git clone https://github.com/tuusuario/conflict-exercise.git
```

- Entra en el directorio del repositorio:

```
cd conflict-exercise
```

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\User> git clone https://github.com/Dann07-ops/conflict-exercise.git
Cloning into 'conflict-exercise'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 12 (delta 4), reused 9 (delta 3), pack-reused 0 (from 0)
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (4/4), done.
PS C:\Users\User> cd conflict-exercise
PS C:\Users\User\conflict-exercise>

```

## Paso 3: Crear una nueva rama y editar un archivo

- Crea una nueva rama llamada feature-branch:

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

- Abre el archivo README.md en un editor de texto y añade una línea nueva, por ejemplo:

Este es un cambio en la feature branch.

- Guarda los cambios y haz un commit:

```
git add README.md
```

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

```
PS C:\Users\User\conflict-exercise> code README.md
PS C:\Users\User\conflict-exercise> git checkout -b feature-branch
Switched to a new branch 'feature-branch'
PS C:\Users\User\conflict-exercise> git add README.md
PS C:\Users\User\conflict-exercise> git commit -m "Added a line in feature-branch"
[feature-branch 5b801e2] Added a line in feature-branch
1 file changed, 4 deletions(-)
PS C:\Users\User\conflict-exercise> git push origin feature-branch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 313.00 KiB/s, done.
Total 3 (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/Dann07-ops/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/Dann07-ops/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch
PS C:\Users\User\conflict-exercise>
```

```
C: > Users > User > conflict-exercise > ⓘ README.md > [tab] # conflict-exercise
1  # conflict-exercise
2  Este es un cambio en la feature branch.
3
4
```

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

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

git checkout main

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

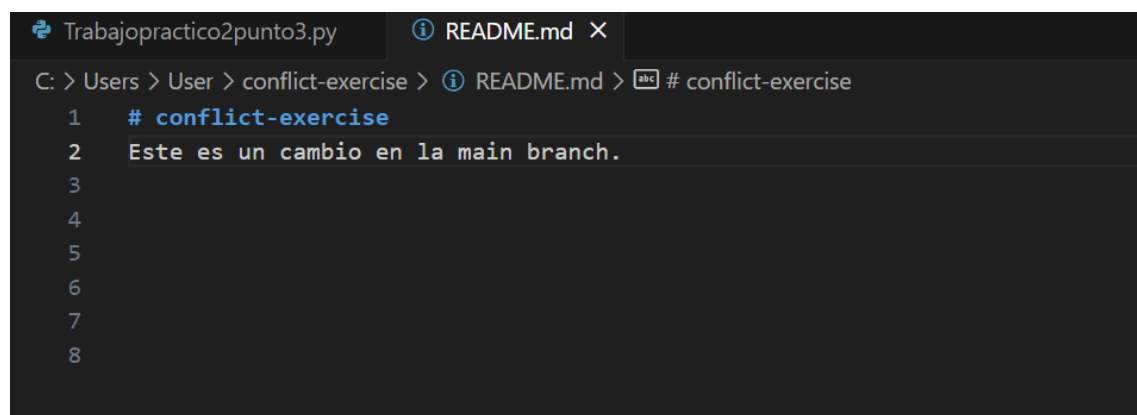
Este es un cambio en la main branch.

- Guarda los cambios y haz un commit:

git add README.md

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

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

A screenshot of a code editor window. The title bar shows two tabs: 'Trabajopractico2punto3.py' and 'README.md'. The editor content shows the following text:

```
C: > Users > User > conflict-exercise > README.md > # conflict-exercise
1  # conflict-exercise
2  Este es un cambio en la main branch.
3
4
5
6
7
8
```

```
PS C:\Users\User\conflict-exercise> git add README.md
PS C:\Users\User\conflict-exercise> git commit -m "Added a line in main branch"
[main 0b1c2cf] Added a line in main branch
1 file changed, 3 insertions(+), 3 deletions(-)
PS C:\Users\User\conflict-exercise>
```

```
PS C:\Users\User\conflict-exercise> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Dann07-ops/conflict-exercise.git
d4c207a..0b1c2cf main -> main
```

Paso 5: Hacer un merge y generar un conflicto

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

git merge feature-branch

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



```
Trabajopractico2punto3.py  README.md X
C: > Users > User > conflict-exercise > README.md > # conflict-exercise
1 # conflict-exercise
2 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
3 <<<<<< HEAD (Current Change)
4 Este es un cambio en la main branch.
5
6
7
8 =====
9 Este es un cambio en la feature branch.
10 >>>>>> feature-branch (Incoming Change)
11
12

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\User\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\User\conflict-exercise>
```

## Paso 6: Resolver el conflicto

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

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

```
Este es un cambio en la main branch.
```

```
=====
```

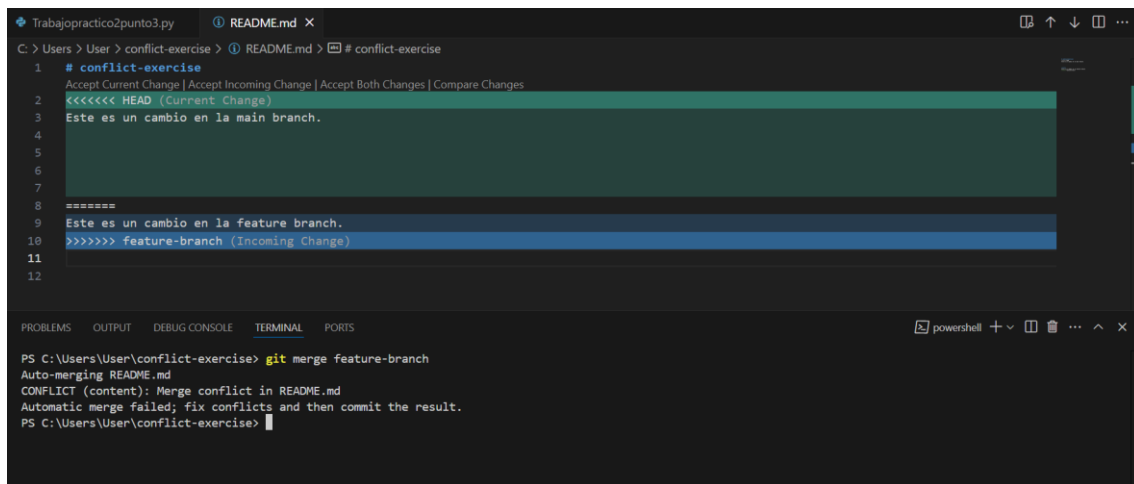
```
Este es un cambio en la feature branch.
```

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

- Decide cómo resolver el conflicto. Puedes mantener ambos cambios, elegir uno de ellos, o fusionar los contenidos de alguna manera.
- Edita el archivo para resolver el conflicto y guarda los cambios (Se debe borrar lo marcado en verde en el archivo donde estes solucionando el conflicto. Y se debe borrar la parte del texto que no se quiera dejar).
- Añade el archivo resuelto y completa el merge:

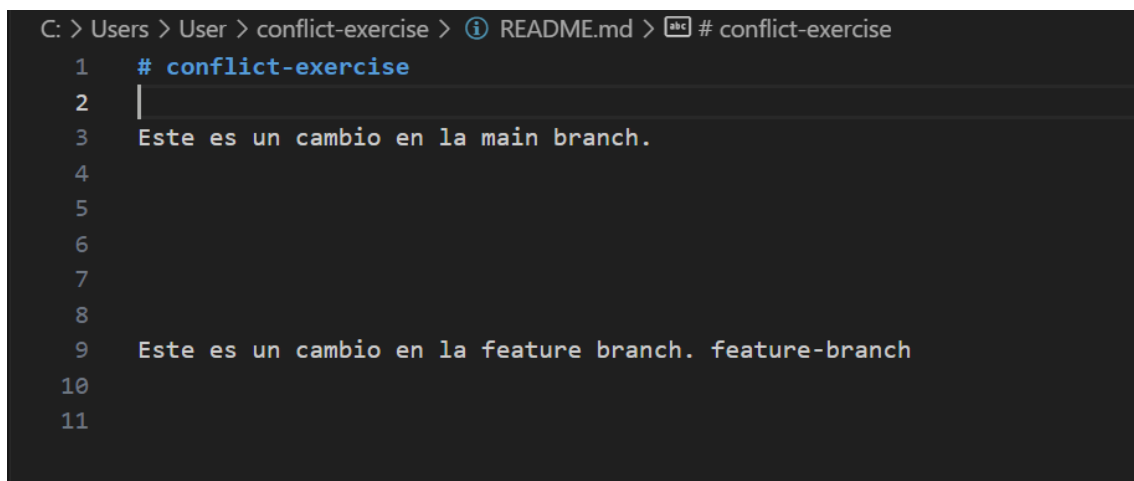
```
git add README.md
```

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

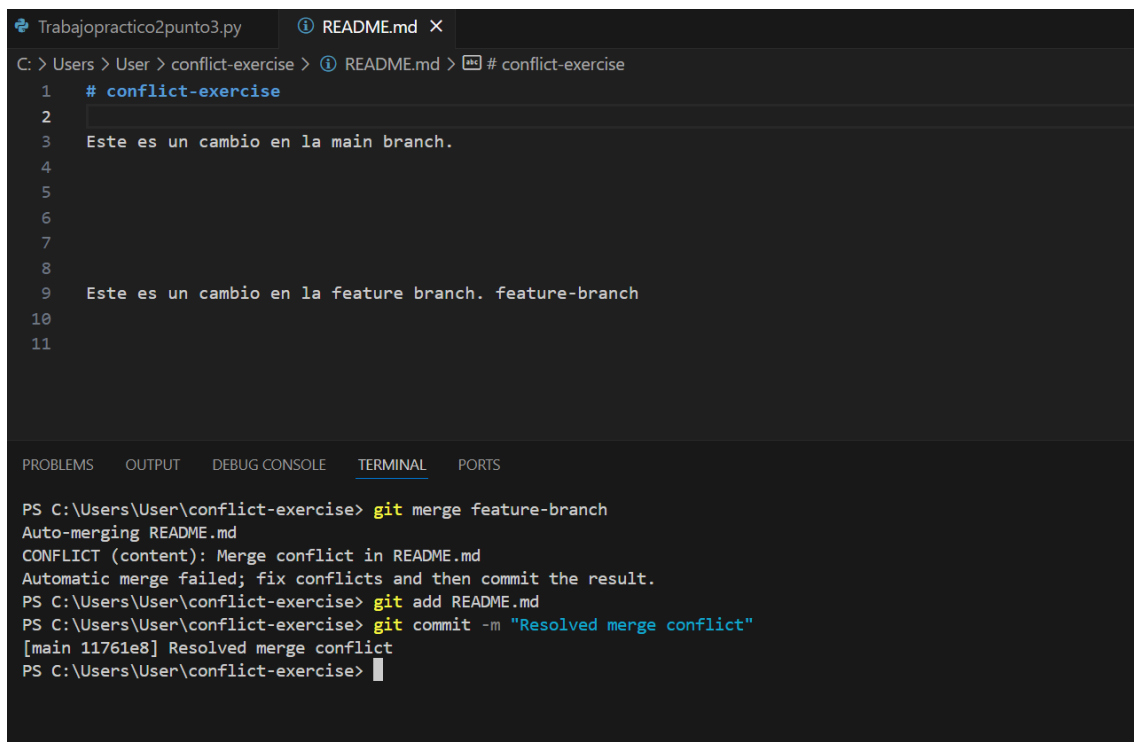


```
Trabajopractico2punto3.py  README.md X
C: > Users > User > conflict-exercise > README.md > # conflict-exercise
1  # conflict-exercise
2  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
3  <<<<<< HEAD (Current Change)
4  Este es un cambio en la main branch.
5
6
7
8  =====
9  Este es un cambio en la feature branch.
10 >>>>>> feature-branch (Incoming Change)
11
12

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\User\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\User\conflict-exercise>
```



```
C: > Users > User > conflict-exercise > README.md > # conflict-exercise
1  # conflict-exercise
2
3  Este es un cambio en la main branch.
4
5
6
7
8
9  Este es un cambio en la feature branch. feature-branch
10
11
```



```
Trabajopractico2punto3.py  README.md X
C: > Users > User > conflict-exercise > README.md > # conflict-exercise
1  # conflict-exercise
2
3  Este es un cambio en la main branch.
4
5
6
7
8
9  Este es un cambio en la feature branch. feature-branch
10
11

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\User\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\User\conflict-exercise> git add README.md
PS C:\Users\User\conflict-exercise> git commit -m "Resolved merge conflict"
[main 11761e8] Resolved merge conflict
PS C:\Users\User\conflict-exercise>
```

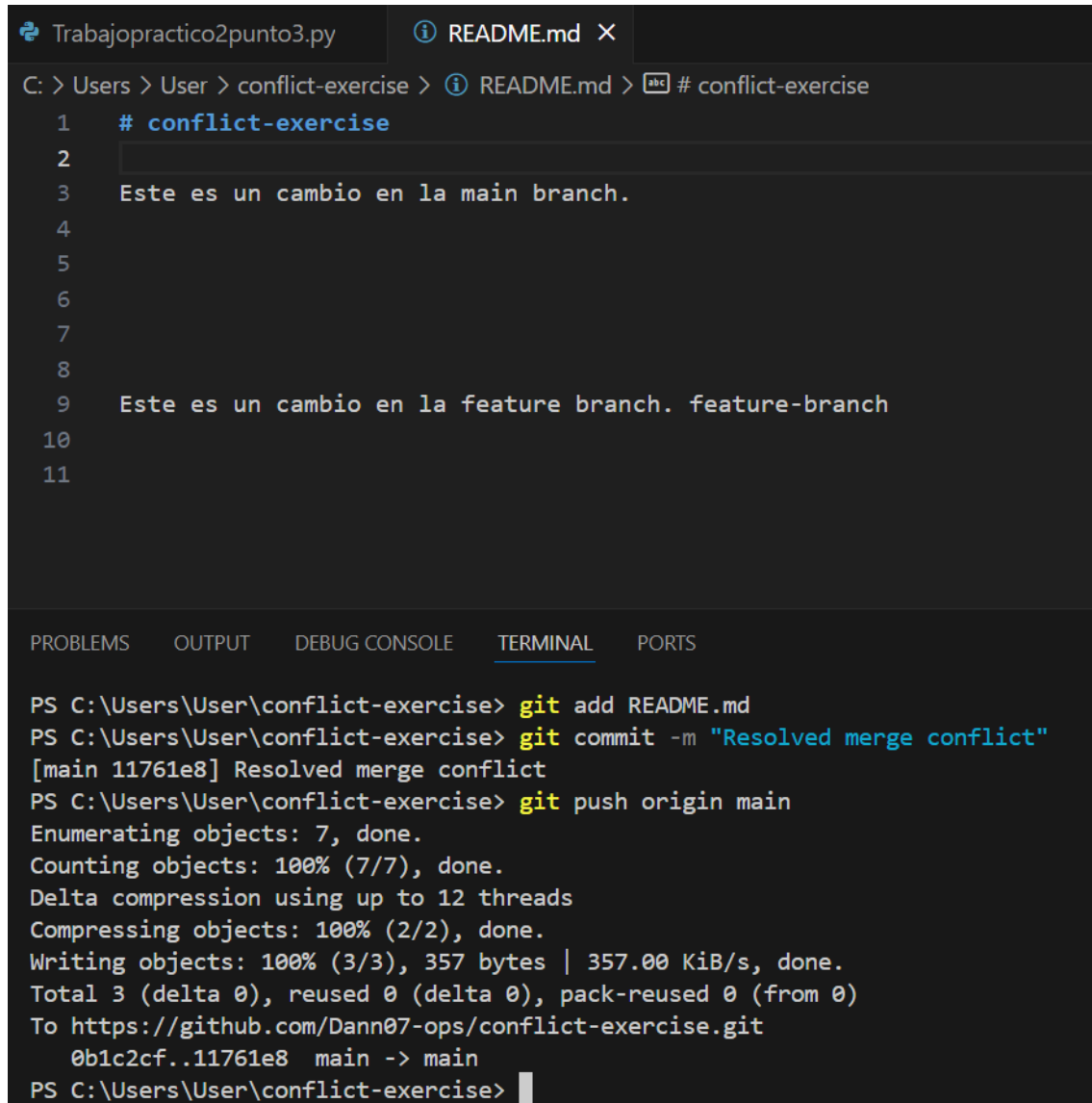
Paso 7: Subir los cambios a GitHub

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

```
git push origin main
```

- También sube la feature-branch si deseas:

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



The image shows a VS Code editor window with two tabs: 'Trabajopractico2punto3.py' and 'README.md'. The 'README.md' tab is active, showing a file with 11 lines. Line 1 is a comment '# conflict-exercise'. Line 2 is empty. Line 3 contains the text 'Este es un cambio en la main branch.'. Line 4 is empty. Line 5 is empty. Line 6 is empty. Line 7 is empty. Line 8 is empty. Line 9 contains the text 'Este es un cambio en la feature branch. feature-branch'. Line 10 is empty. Line 11 is empty. Below the editor, the 'TERMINAL' tab is active, showing the output of the following commands:

```
PS C:\Users\User\conflict-exercise> git add README.md
PS C:\Users\User\conflict-exercise> git commit -m "Resolved merge conflict"
[main 11761e8] Resolved merge conflict
PS C:\Users\User\conflict-exercise> git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 357 bytes | 357.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Dann07-ops/conflict-exercise.git
    0b1c2cf..11761e8  main -> main
PS C:\Users\User\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.
- Puedes revisar el historial de commits para ver el conflicto y su resolución.

conflict-exercisePublic

PinUnwatch1Fork0Star0

main2 Branches0 TagsGo to fileAdd fileCode

Dann07-opsResolved merge conflict11761e8 · 2 minutes ago7 Commits

README.mdResolved merge conflict2 minutes ago

README

# conflict-exercise

Este es un cambio en la main branch.

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

About

Ejercicio 3 del trabajo practico 2 de programacion 1

ReadmeActivity0 stars1 watching0 forks

Releases

No releases published

Create a new release

Packages

Dann07-ops / conflict-exerciseType to search

CodeIssuesPull requestsActionsProjectsWikiSecurityInsightsSettings

mainconflict-exercise / README.mdGo to file

Dann07-opsResolved merge conflict11761e8 · 3 minutes agoHistory

PreviewCodeBlame10 lines (3 loc) · 119 BytesCode 55% faster with GitHub CopilotRawDownloadEdit

# conflict-exercise

Este es un cambio en la main branch.

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

Resolved merge conflict

main2 parents 0b1c2cf + 5b801e2 commit 11761e8

Filter files...README.md

1 file changed+3 -0 lines changedSearch within code

README.md+3 -0

@@ -1,7 +1,10 @@

11# conflict-exercise

2+

23Este es un cambio en la main branch.

34

45

56

67

78

9+ Este es un cambio en la feature branch. feature-branch

10+

Comments0

Activar WindowsLock conversation

## Commits

mainAll usersAll time

Commits on May 5, 2025

Resolved merge conflict11761e8

Dann07-ops committed 7 minutes ago

Added a line in main branch0b1c2cf

Dann07-ops committed 23 minutes ago

Added a line in feature-branch5b801e2

Dann07-ops committed 40 minutes ago