TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

Práctico 2: Git y GitHub

Alumno: Perez Leandro Matias

Comisión: 19

Actividades

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

- **¿Qué es GitHub**?

  Es una plataforma de desarrollo colaborativo que funciona como un sistema de control de versiones y repositorio de código fuente, es una herramienta esencial para los desarrolladores de software, ya que facilita la colaboración, el control de versiones y el almacenamiento de código.

- **¿Cómo crear un repositorio en GitHub?**

1. Crea una cuenta: Si no tienes una cuenta de GitHub, ve a “github.com” y regístrate.
2. Inicia sesión: Accede con tu usuario y contraseña.
3. Accede al Dashboard: Una vez dentro, haz clic en el botón verde que dice New o "Crear repositorio" en la página principal.
4. Rellena los detalles del repositorio:
   - Nombre del repositorio: Dale un nombre único y descriptivo.
   - Descripción (opcional): Agrega una breve descripción sobre el propósito del proyecto.
   - Visibilidad: Escoge entre "Público" (visible para todos) o "Privado" (solo tú y quienes invites tendrán acceso).
5. Configura opciones iniciales:
   - Marca la casilla de "Initialize this repository with a README" si deseas incluir un archivo README, que describe el proyecto.
   - También puedes agregar un archivo `.gitignore` o una licencia, dependiendo de tus necesidades.
6. Crea el repositorio: Haz clic en el botón "Create repository".

- **¿Cómo crear una rama en Git?**

1. Abre tu terminal y navega al directorio del repositorio de Git en el que deseas trabajar.
2. Verifica las ramas existentes con el comando:

```
git branch
```

Esto te mostrará una lista de las ramas disponibles y destacará en cuál estás actualmente.

3. Crea una nueva rama usando:

```
git branch nombre_de_rama
```

Sustituye `nombre_de_rama` con el nombre que deseas para la nueva rama.

- **¿Cómo cambiar a una rama en Git?**

Con el comando:

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

Reemplaza “nombre-de-la-rama” con el nombre de la rama a la que deseas cambiar

- **¿Cómo fusionar ramas en Git?**

Con el comando:

```
git merge nombre-rama-origen
```

Reemplaza “nombre-de-la-rama” con la rama que deseas fusionar. Git combinará los cambios de esta rama en la rama actual.

- **¿Cómo crear un commit en Git?**

Con el comando:

```
git commit -m "mensaje"
```

- **¿Cómo enviar un commit a GitHub?**

Con el comando:

```
git push origin nombre-rama-origen
```

- **¿Qué es un repositorio remoto?**

Un repositorio remoto en Git es una copia de tu repositorio que se encuentra alojada en un servidor o plataforma en línea, como GitHub, GitLab o Bitbucket. Estos repositorios permiten colaborar con otras personas, compartir proyectos y hacer un seguimiento centralizado de los cambios realizados en el código.

- **¿Cómo agregar un repositorio remoto a Git?**

Con el comando:

```
git remote add origin dirección_url
```

- **¿Cómo empujar cambios a un repositorio remoto?**

Con el comando:

```
git push -u origin mater
```

- **¿Cómo tirar de cambios de un repositorio remoto?**

Con el comando:

```
git pull origin master
```

O el comando:

`git pull (Si usamos -u en el push)`

- ## ¿Qué es un fork de repositorio?

Un FORK de un repositorio es una copia completa de un repositorio existente, generalmente alojado en una plataforma como GitHub, que se crea en tu propio espacio (cuenta de usuario u organización). Es independiente del repositorio original, lo que significa que puedes trabajar en tu fork sin afectar al proyecto original. Sin embargo, puedes enviar cambios de vuelta al repositorio original mediante _pull requests_, si el propietario original lo permite.

El propósito principal de un fork es facilitar la colaboración en proyectos open source o grandes, permitiendo a los desarrolladores:

- Hacer cambios y experimentos libremente en su copia.

- Proponer mejoras o correcciones a los proyectos originales sin comprometer la estabilidad de estos.

- Usar el proyecto como base para crear algo nuevo.

- ## ¿Cómo crear un fork de un repositorio?

Un fork es una copia de un repositorio que se guarda en tu propia cuenta de GitHub. Esto te permite hacer cambios en el código sin afectar el repositorio original.
Pasos para crear un fork:

1. Encuentra el repositorio que quieres forkar: Ve a la página del repositorio en GitHub.
2. Haz clic en el botón "Fork": Este botón se encuentra en la esquina superior derecha de la página.
3. Elige dónde quieres crear el fork: Si tienes varias organizaciones en GitHub, puedes elegir dónde quieres guardar el fork.

Después de crear un fork:
Puedes clonar el fork en tu computadora para hacer cambios en el código.
Puedes enviar solicitudes de extracción (pull requests) al repositorio original para proponer tus cambios.

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

Una solicitud de extracción es una forma de proponer cambios a un repositorio de GitHub. Cuando haces un fork de un repositorio y haces cambios en tu copia, puedes enviar una solicitud de extracción para pedirle al propietario del repositorio original que incorpore tus cambios.

Pasos para enviar una solicitud de extracción:

1. Crea una solicitud de extracción: Ve a la página del repositorio original en GitHub y haz clic en el botón " **pull request** ", veremos una ventana a modo de resumen en donde se reflejarán los cambios que hemos hecho nosotros en comparación al repositorio original.
2. Elige las ramas: Elige la rama de tu fork que contiene tus cambios y la rama del repositorio original a la que quieres incorporar tus cambios.
3. Revisa tus cambios: Revisa tus cambios para asegurarte de que son correctos.
4. Envía la solicitud de extracción: Haz clic en el botón "Crear solicitud de extracción" para enviar tu solicitud de extracción.

Después de enviar una solicitud de extracción:

- ✓ El propietario del repositorio original revisará tus cambios.
- ✓ Si el propietario aprueba tus cambios, los incorporará al repositorio original.
- ✓ Si el propietario solicita cambios, tendrás que hacer los cambios y volver a enviar la solicitud de extracción.

- ¿Cómo aceptar una solicitud de extracción?

Aceptar una solicitud de extracción en **GitHub** e sigues estos pasos:

1. **Accede al repositorio**:
    - Ve al repositorio donde se ha enviado la solicitud de extracción.
2. **Navega a la pestaña "Pull Requests"**:
    - En el menú superior del repositorio, selecciona la opción "Pull Requests".
3. **Selecciona la solicitud de extracción**:
    - Busca la solicitud específica que deseas revisar y haz clic en ella.
4. **Revisa los cambios**:
    - Observa los cambios propuestos y los archivos modificados. Esto se muestra en la pestaña "Files changed" dentro de la solicitud.
    - Lee los comentarios o descripciones proporcionados por el autor.
5. **Deja comentarios (opcional)**:
    - Si es necesario, deja tus observaciones o pide modificaciones al autor.
6. **Aprueba la solicitud**:
    - Si estás conforme, haz clic en el botón **"Review changes"** y

selecciona "Approve". Luego, escribe un comentario si lo consideras oportuno y confirma.

7. **Fusiona la solicitud**:
    - Una vez aprobada, haz clic en el botón **"Merge pull request"**.
    - Confirma la fusión haciendo clic en "Confirm merge".
    - (Opcional) Elimina la rama relacionada con los cambios si ya no es necesaria.
8. **Verifica el repositorio**:
    - Tras la fusión, confirma que todo funcione correctamente y que los cambios estén integrados como esperabas.

- ## ¿Qué es un etiqueta en Git?

Una etiqueta (en Git es una referencia que se utiliza para marcar puntos específicos en el historial de un repositorio. Es comúnmente empleada para señalar versiones importantes, como lanzamientos de software (releases), de forma que sea más fácil identificarlas y acceder a ellas en el futuro.

- ## ¿Cómo crear una etiqueta en Git?

Git utiliza dos tipos principales de etiquetas: ligeras y anotadas.

Etiqueta ligera (lightweight): Esta es una etiqueta básica que simplemente apunta a un commit. Usa el siguiente comando:

```
git tag nombre-de-la-etiqueta
```

Etiqueta anotada (annotated): Si necesitas agregar información como un mensaje descriptivo, usa una etiqueta anotada con el siguiente comando:

```
git tag -a nombre-de-la-etiqueta -m "Mensaje de la etiqueta"
```

- ## ¿Cómo enviar una etiqueta a GitHub?

Para enviar una etiqueta (tag) a GitHub, sigue estos pasos:

1. **Crea una etiqueta localmente**:
   Abre tu terminal y navega al repositorio local del proyecto.

   Usa el siguiente comando para crear una etiqueta:
   ```
   git tag -a v1.0.0 -m "Primera versión"
   ```

1. **Envía la etiqueta al repositorio remoto**:
   Después de crear la etiqueta, envíala al repositorio remoto usando:
   ```
   git push origin v1.0.0
   ```

   Esto enviará la etiqueta específica.
   Si deseas enviar todas las etiquetas creadas, usa:

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

1. **Verifica en GitHub**:
   Ve a la página del repositorio en GitHub y verifica que la etiqueta aparezca en la sección "Tags" o "Releases".

- ## ¿Qué es un historial de Git?

El historial de Git es una secuencia de todos los cambios realizados en un repositorio de Gi. Este historial está asociado a las confirmaciones (commits) que se han hecho, donde cada commit representa un conjunto de cambios específicos en el código o los archivos del proyecto.

El historial de Git incluye información como:

- Autor
- Fecha de realización
- Mensaje enviado

- ## ¿Cómo ver el historial de Git?

Puedes ver el historial de Git utilizando comando En la terminal :

Ver el historial básico:

```
git log
```

Este comando muestra todos los commits realizados en el repositorio, incluyendo detalles como el autor, la fecha y el mensaje del commit.

Historial en formato resumido (una línea por commit):

```
git log --oneline
```

Es ideal para tener una vista rápida y compacta del historial.

Historial con ramas y gráficas:

```
git log --graph --oneline --all
```

Este comando visualiza el historial con un gráfico que muestra cómo las ramas se bifurcan y se fusionan.

Historial de un archivo específico: Si quieres saber los cambios hechos en un archivo particular, usa:

```
git log -- <nombre_del_archivo>
```

Historial con diferencias (detalles de los cambios):

```
git log -p
```

Esto incluye las diferencias (diffs) de cada commit.

Historial limitado: Si necesitas solo los últimos n commits, usa:

```
git log -n
```

Por ejemplo, `git log -5` mostrará los últimos 5 commits.

- ## ¿Cómo buscar en el historial de Git?

Para buscar en el historial de commits de Git, puedes utilizar varios comandos y

opciones que te permiten filtrar y localizar commits específicos.

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

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

Para buscar commits en un rango de fechas específico, usa las opciones --since y --until: git log --since="2024-01-01" --until="2024-01-31"

Para encontrar commits hechos por un autor específico, usa --author: git log --author="Nombre del Autor"

- ## ¿Cómo borrar el historial de Git?

Para Borrar el historial de Git se lo puede realizar de varias formas:

- git reset -> Quita del stage todos los archivos y carpetas del proyecto.
- git reset nombreArchivo -> Quita del stage el archivo indicado.
- git reset nombreCarpeta/ -> Quita del stage todos los archivos de esa carpeta.
- git reset nombreCarpeta/nombreArchivo -> Quita ese archivo del stage (que a la vez está dentro de una carpeta).
- git reset nombreCarpeta/*.extensión -> Quita todos los archivos que cumplan con la condición indicada previamente dentro de esa carpeta del stage.

- ## ¿Qué es un repositorio privado en GitHub?

Un repositorio privado en GitHub es un espacio donde puedes almacenar tu código y archivos, al igual que en un repositorio público, pero con la diferencia clave de que su contenido no está visible para todos. Solo las personas a las que les otorgues acceso podrán ver o colaborar en él.

Esto es especialmente útil para proyectos en los que trabajas en equipo y necesitas privacidad, como código propietario, experimentos, trabajo previo al lanzamiento público, entre otros. Además, GitHub ofrece herramientas para gestionar quién tiene acceso y qué permisos específicos tienen (lectura, escritura, etc.).

- ## ¿Cómo crear un repositorio privado en GitHub?

Para Crear un repositorio privado en GitHub se lo puede realizar de la siguente forma:

1. **Inicia sesión en GitHub:** Accede a tu cuenta en github.com.
2. **Accede a la opción de crear un repositorio:** En la esquina superior derecha, haz clic en el botón **+** y selecciona **New repository** (Nuevo repositorio).

3. **Configura el repositorio:**

a. En el campo **Repository name** (Nombre del repositorio), escribe el nombre que deseas para tu proyecto.
b. Opcionalmente, añade una descripción en el campo correspondiente.
c. Marca la opción **Private** (Privado) en la sección de visibilidad para asegurarte de que solo las personas con acceso puedan verlo.

4. **Crea el repositorio:** Haz clic en el botón verde **Create repository** para finalizar.

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

Para Invitar a alguien a un repositorio privado en GitHub se lo puede realizar realizado los siguiente pasos:

1. Accede a tu repositorio privado: Ve a GitHub, inicia sesión y selecciona el repositorio privado al que deseas invitar a alguien.
2. Abre la configuración del repositorio: En la parte superior del repositorio, haz clic en la pestaña Settings (Configuración).
   Gestiona los colaboradores:
   - En el menú lateral, busca y selecciona Collaborators and teams (Colaboradores y equipos) bajo la sección "Access" o "Manage access" (dependiendo de la interfaz que veas).
   - Haz clic en el botón Invite a collaborator (Invitar a un colaborador).
3. Invita a la persona:
   Escribe el nombre de usuario de GitHub, correo electrónico o el nombre de la persona que quieres invitar.

   Selecciónalo de la lista que aparece.
4. Otorga permisos (opcional): Antes de enviar la invitación, puedes decidir qué nivel de acceso tendrá el colaborador: Read (leer), Write (escribir) o Admin (administrador).
5. Envía la invitación: Haz clic en el botón Send invitation (Enviar invitación). La persona recibirá una notificación en su cuenta de GitHub.
6. Espera a la aceptación: Una vez que la persona acepte la invitación, tendrá acceso al repositorio según los permisos que configuraste.

- ## ¿Qué es un repositorio público en GitHub?

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

- ## ¿Cómo crear un repositorio público en GitHub?

Para Crear un repositorio público en GitHub de la siguiente forma:

1. Inicia sesión en tu cuenta de GitHub

2. Accede a la opción de crear un repositorio:

- Haz clic en el botón **+** en la esquina superior derecha de la página.

- Selecciona **New repository** (Nuevo repositorio) en el menú desplegable.

3. Completa los detalles del repositorio:

- Repository name: Escribe el nombre que deseas para tu repositorio.

- Description (opcional): Agrega una breve descripción de tu proyecto.

- Visibilidad: Elige **Public** (Público) para que el repositorio sea accesible a cualquier persona.

4. Crea el repositorio: Haz clic en el botón verde **Create repository** para finalizar.

- **¿Cómo compartir un repositorio público en GitHub?**

Compartir un repositorio público en GitHub es muy sencillo y puedes hacerlo de varias formas. Aquí tienes los pasos básicos:

1. Obtén el enlace del repositorio:

- Ve al repositorio público que deseas compartir en GitHub.

- En la barra de direcciones de tu navegador, copia la URL del repositorio. Por ejemplo, algo como: https://github.com/usuario/nombre-del-repositorio.

2. Comparte la URL:

- Puedes enviar el enlace por correo electrónico, mensaje, redes sociales, o incluso publicarlo en una página web o foro si deseas que alcance a más personas.

3. Usa el botón "Share":

- Algunos proyectos cuentan con opciones de compartir directamente desde GitHub, pero esto varía según las integraciones que tengas.

2) Realizar la siguiente actividad:

- Crear un repositorio.

  - Dale un nombre al repositorio.
  - Elije el repositorio sea público.

- Inicializa el repositorio con un archivo.

- Agregando un Archivo

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

- Creando Branchs

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

```
PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Traba
jo Practico
$ git clone https://github.com/matiasplm/Actividad-2-tp2.git
Cloning into 'Actividad-2-tp2'...
warning: You appear to have cloned an empty repository.

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Traba
jo Practico
$ cd /d/UTN/Programacion\ I/02\ Trabajos\ Colaborativos/Trabajo\ Practico/Actividad-2-tp2

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (main)
$ echo "mi archivo" > mi-archivo.txt

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (main)
$ git add .
warning: in the working copy of 'mi-archivo.txt', LF will be replaced by CRLF the next time Git touches it

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (main)
$ git commit -m "agregado mi-archivo.txt"
[main (root-commit) 0f16b6a] agregado mi-archivo.txt
 1 file changed, 1 insertion(+)
 create mode 100644 mi-archivo.txt

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 233 bytes | 233.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/matiasplm/Actividad-2-tp2.git
 * [new branch]      main -> main

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (main)
$ git branch mi-rama

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (main)
$ git checkout mi-rama
Switched to branch 'mi-rama'

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (mi-rama)
$ echo "mi archivo 2" > mi-archivo2.txt

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (mi-rama)
$ git add .
warning: in the working copy of 'mi-archivo2.txt', LF will be replaced by CRLF the next time Git touches it

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (mi-rama)
$ git commit -m "agragado mi archivo2.txt a mi-rama"
[mi-rama ee68428] agragado mi archivo2.txt a mi-rama
 1 file changed, 1 insertion(+)
 create mode 100644 mi-archivo2.txt

PC@DESKTOP-727JP4S MINGW64 /d/UTN/Programacion I/02 Trabajos Colaborativos/Trabajo Practico/Actividad-2-tp2 (mi-rama)
$ git push origin mi-rama
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
```

Link de la actividad:

https://github.com/matiasplm/Actividad-2-tp2.git

3) Realizar la siguiente actividad:

Paso 1: Crear un repositorio en GitHub

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

Paso 2: Clonar el repositorio a tu máquina local

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

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

- Entra en el directorio del repositorio:

```
cd conflict-exercise
```

Paso 3: Crear una nueva rama y editar un archivo

- Crea una nueva rama llamada feature-branch:

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

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

Este es un cambio en la feature branch.

- Guarda los cambios y haz un commit:

```
git add README.md
```

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

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

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

```
git checkout main
```

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

Este es un cambio en la main branch.

- Guarda los cambios y haz un commit:

```
git add README.md
```

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

Paso 5: Hacer un merge y generar un conflicto

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

```
git merge feature-branch
```

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

Paso 6: Resolver el conflicto

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

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

Este es un cambio en la main branch.

```
=======
```

Este es un cambio en la feature branch.

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

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

```
git add README.md
```

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

Paso 7: Subir los cambios a GitHub

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

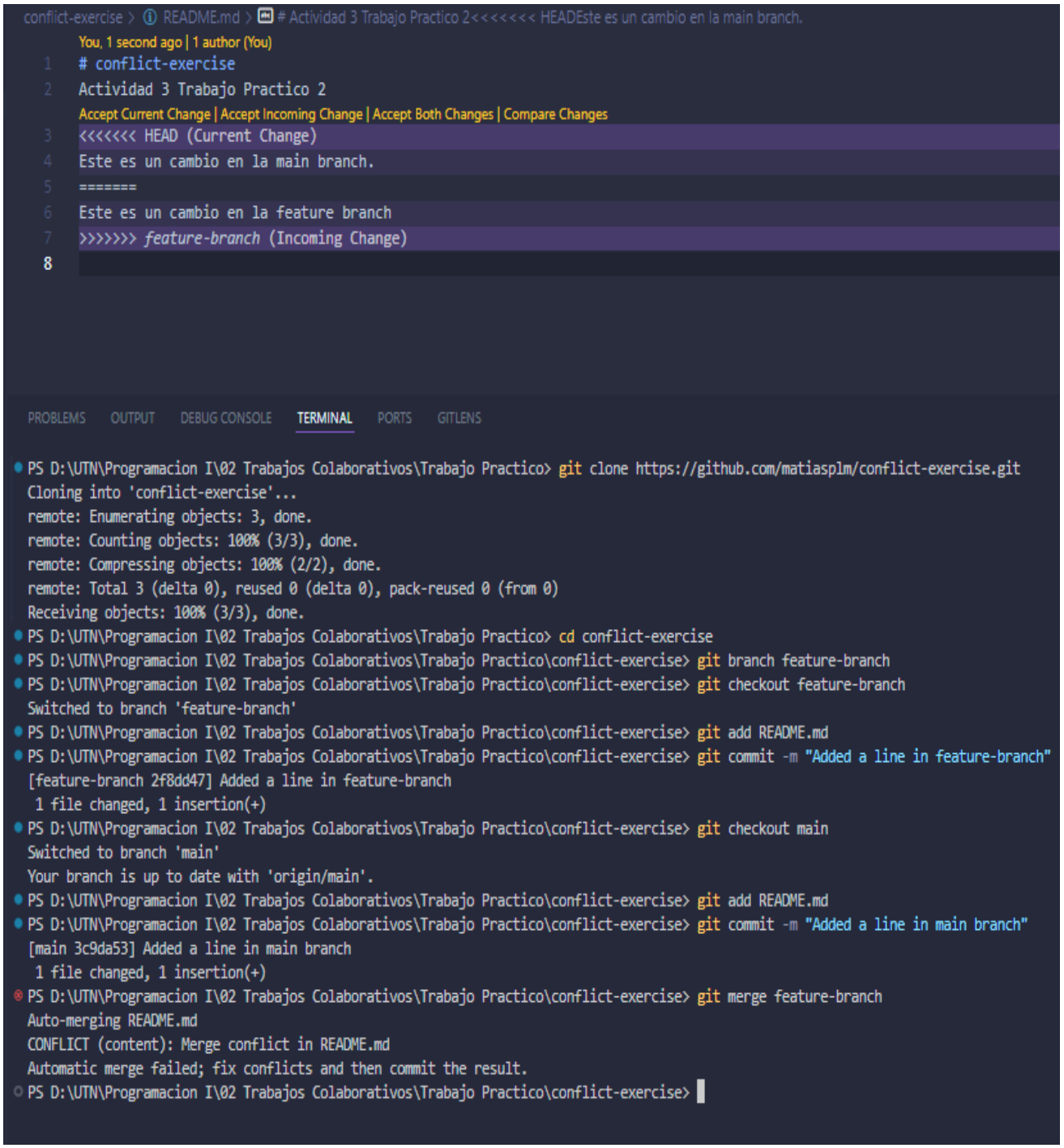
```
git push origin main
```

- También sube la feature-branch si deseas:

git push origin feature-branch

Paso 8: Verificar en GitHub

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



README.md

conflict-exercise > README.md > # conflict-exercise

You, 1 second ago | 1 author (You)

```
1  # conflict-exercise
2  Actividad 3 Trabajo Practico 2
3  Este es un cambio en la main branch.
4  Este es un cambio en la feature branch      You, 1 second ago • Uncommitted changes
5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico> git clone https://github.com/matiasplm/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 D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico> cd conflict-exercise
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git branch feature-branch
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git checkout feature-branch
Switched to branch 'feature-branch'
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git add README.md
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git commit -m "Added a line in feature-branch"
[feature-branch 2f8dd47] Added a line in feature-branch
 1 file changed, 1 insertion(+)
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git add README.md
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git commit -m "Added a line in main branch"
[main 3c9da53] Added a line in main branch
 1 file changed, 1 insertion(+)
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\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 D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git add README.md
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise> git commit -m "Resolved merge conflict"
[main 652dffb] Resolved merge conflict
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\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), 747 bytes | 373.00 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/matiasplm/conflict-exercise.git
   2ceb5cd..652dffb  main -> main
PS D:\UTN\Programacion I\02 Trabajos Colaborativos\Trabajo Practico\conflict-exercise>
```

Link de la actividad 3:

https://github.com/matiasplm/conflict-exercise.git