Práctico 2: Git y GitHub

Objetivo:

El estudiante desarrollará competencias para trabajar con Git y GitHub, aplicando conceptos fundamentales de control de versiones, colaboración en proyectos y resolución de conflictos, en un entorno simulado y guiado.

Resultados de aprendizaje:

1. Comprender los conceptos básicos de Git y GitHub: Identificar y explicar los principales términos y procesos asociados con Git y GitHub, como repositorios, ramas, commits, forks, etiquetas y repositorios remotos.
2. Manejar comandos esenciales de Git: Ejecutar comandos básicos para crear, modificar, fusionar y gestionar ramas, commits y repositorios, tanto en local como en remoto.
3. Aplicar técnicas de colaboración en GitHub: Configurar y utilizar repositorios remotos, realizar forks, y gestionar pull requests para facilitar el trabajo colaborativo.
4. Resolver conflictos en un entorno de control de versiones: Identificar, analizar y solucionar conflictos de merge generados en un flujo de trabajo con múltiples ramas.

Actividades

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

- ### ¿Qué es GitHub?

GitHub es una plataforma basada en la web que permite a los desarrolladores almacenar, gestionar y colaborar en proyectos de software. Se basa en **Git**, un sistema de control de versiones distribuido, que permite a los programadores llevar un registro de los cambios en su código a lo largo del tiempo y colaborar con otros de manera eficiente.

Características clave de GitHub:

1. **Control de versiones**: GitHub permite llevar un seguimiento de todos los cambios realizados en un proyecto, lo que facilita volver a versiones anteriores si es necesario, y ver quién hizo qué cambios y cuándo.
2. **Colaboración**: Los desarrolladores pueden trabajar en un mismo proyecto de forma simultánea sin interferir entre sí. GitHub permite gestionar **ramas** (branches) para que las personas puedan trabajar en diferentes partes del proyecto sin afectar la versión principal.
3. **Repositorios**: Un **repositorio** (repo) es donde se almacenan los archivos de un proyecto. Pueden ser públicos (cualquiera puede verlos y contribuir) o privados (solo los colaboradores autorizados pueden acceder).

4. **Pull Requests**: Una de las características más importantes en GitHub es la opción de hacer **pull requests**. Esto permite que un desarrollador envíe sus cambios a otro para su revisión antes de fusionarlos en el proyecto principal.
5. **GitHub Actions**: Herramientas integradas para la automatización de tareas, como pruebas, despliegues y más.
6. **Documentación y seguimiento**: Los usuarios pueden crear **issues** para rastrear errores o tareas pendientes, y utilizar **wikis** o **README.md** para documentar el proyecto.

GitHub es ampliamente utilizado tanto en **proyectos de código abierto** como en **entornos empresariales**, siendo una de las plataformas más populares para el desarrollo de software colaborativo.

- ## ¿Cómo crear un repositorio en GitHub?

1. **Inicia sesión en GitHub**: Ve a github.com y entra con tu cuenta.
2. **Haz clic en "New repository"**: En la página de inicio de tu cuenta, haz clic en el botón verde que dice **"New"** o "New repository" (generalmente en la parte superior derecha).
3. **Configura tu repositorio**:
   - **Repository name**: Escribe un nombre único para tu repositorio.
   - **Description** (opcional): Añade una breve descripción del repositorio.
   - **Public o Private**: Decide si quieres que el repositorio sea público o privado.
   - **Initialize this repository with a README** (opcional): Si deseas, puedes agregar un archivo README inicial.
4. **Haz clic en "Create repository"**: Después de configurar tu repositorio, haz clic en el botón verde para crear el repositorio.
5.

- ## ¿Cómo crear una rama en Git?

Crear la rama: git branch nombre-de-la-rama

Cambiar a la nueva rama: git checkout nombre-de-la-rama
O, para crear y cambiar a la rama en un solo paso:
git checkout -b nombre-de-la-rama

- ## ¿Cómo cambiar a una rama en Git?

Para cambiar a una rama en **Git**, utiliza el siguiente comando:
git checkout nombre-de-la-rama

Si estás usando una versión más reciente de Git, también puedes usar:
git switch nombre-de-la-rama

Ambos comandos hacen lo mismo: cambian a la rama especificada.

• ¿Cómo fusionar ramas en Git?

1. **Cambia a la rama de destino** (la rama en la que quieres fusionar los cambios):
   git checkout rama-de-destino

   O si estás usando una versión reciente de Git:
   git switch rama-de-destino

2. **Fusiona la rama** que quieres incorporar:
   git merge rama-origen

   Reemplaza **rama-origen** con el nombre de la rama que quieres fusionar en la rama actual.

• ¿Cómo crear un commit en Git?

1. **Añadir los cambios al área de preparación** (staging):
   git add archivo

   O para añadir todos los archivos modificados:
   git add .

2. **Crear el commit** con un mensaje descriptivo:
   git commit -m "Mensaje del commit"

• ¿Cómo enviar un commit a GitHub?

1. **Asegúrate de estar en la rama correcta**:
   git checkout nombre-de-la-rama

2. **Enlaza tu repositorio local con el repositorio remoto en GitHub** (si aún no lo has hecho):
   git remote add origin https://github.com/usuario/repositorio.git

3. **Envía el commit a GitHub** (sube los cambios al repositorio remoto):
   git push origin nombre-de-la-rama

¿Qué es un repositorio remoto?

Un **repositorio remoto** es una **versión del repositorio Git** que está almacenada en un servidor externo, como **GitHub**, **GitLab**, o **Bitbucket**, en lugar de estar solo en tu computadora local.

Características:

- **Accesible desde cualquier lugar**: Puedes acceder a él desde diferentes dispositivos y compartirlo con otros colaboradores.
- **Sincronización**: Permite **subir** (push) y **bajar** (pull) cambios entre tu repositorio local y el remoto.
- **Colaboración**: Los repositorios remotos facilitan el trabajo en equipo, permitiendo que varias personas trabajen en el mismo proyecto.

¿Cómo agregar un repositorio remoto a Git?

1. **Navega al directorio de tu repositorio local**:
   cd ruta/a/tu/repositorio
2. **Agrega el repositorio remoto** utilizando el comando git remote add:
   git remote add origin https://github.com/usuario/repositorio.git
3. **Verifica que el repositorio remoto se haya agregado correctamente**:
   git remote –v
   Esto mostrará la URL de tu repositorio remoto (tanto para **fetch** como para **push**).

¿Cómo empujar cambios a un repositorio remoto?

1. **Asegúrate de estar en la rama correcta**:
   git checkout nombre-de-la-rama
2. **Añadir los cambios al área de preparación** (si aún no lo has hecho):
   git add .
3. **Crear un commit** con tus cambios:
   git commit -m "Mensaje descriptivo del commit"
4. **Empujar los cambios al repositorio remoto**:
   git push origin nombre-de-la-rama

origin: Es el nombre del repositorio remoto (es el valor predeterminado).

nombre-de-la-rama: El nombre de la rama en la que estás trabajando (por ejemplo, main o master).

¿Cómo tirar de cambios de un repositorio remoto?

1. **Asegúrate de estar en la rama correcta**:
   git checkout nombre-de-la-rama
2. **Tirar los cambios del repositorio remoto**:
   git pull origin nombre-de-la-rama

¿Qué es un fork de repositorio?

Un **fork** de repositorio es una **copia personal** de un repositorio que se crea en tu cuenta de GitHub (u otra plataforma similar) para que puedas **modificarlo de manera independiente** sin afectar el repositorio original.

Características de un fork:

- **Copia independiente**: Un fork te permite tener una copia completa del repositorio original en tu propia cuenta, donde puedes hacer cambios sin afectar el proyecto original.
- **Contribuciones**: Usualmente, se hace un fork cuando se quiere contribuir a un proyecto de código abierto. Después de realizar cambios en el fork, puedes enviar un **pull request** para sugerir que esos cambios sean integrados al repositorio original.
- **Sincronización**: Puedes sincronizar tu fork con el repositorio original para obtener las actualizaciones más recientes.

¿Cómo crear un fork de un repositorio?

a) **Ve al repositorio** que deseas bifurcar (fork) en GitHub.
b) Accede al repositorio original en GitHub.
c) **Haz clic en el botón "Fork"**:
d) Este botón se encuentra en la esquina superior derecha de la página del repositorio.
e) **Selecciona tu cuenta**:
f) Después de hacer clic en "Fork", GitHub te pedirá que selecciones dónde crear el fork. Generalmente, esto será en tu cuenta personal o en una organización si perteneces a una.
g) **Fork completo**:

GitHub creará una copia completa del repositorio en tu cuenta, y podrás ver el nuevo repositorio en tu perfil de GitHub.

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

1. **Haz un fork del repositorio** (si aún no lo has hecho).

   Si aún no tienes un fork del repositorio al que deseas contribuir, haz uno siguiendo los pasos anteriores.

2. **Realiza los cambios en tu fork**:
   Clona el repositorio a tu máquina local o trabaja directamente en GitHub.
   Realiza los cambios que deseas en tu fork.

3. **Haz commit y push de tus cambios**:
   Después de realizar los cambios, haz un commit y empuja los cambios a tu repositorio remoto en GitHub:

   ```
   git add .
   git commit -m "Descripción de los cambios"
   git push origin nombre-de-la-rama
   ```

4. **Ve a la página de tu fork en GitHub**:
   Navega a la página del repositorio en tu cuenta de GitHub.

5. **Haz clic en el botón "Compare & pull request"**:

   Verás un mensaje en tu repositorio que dice algo como "This branch is X commits ahead of [nombre del repositorio original]". Haz clic en el botón **"Compare & pull request"**.

6. **Escribe una descripción para tu pull request**:

   Escribe un título claro y una descripción detallada sobre qué cambios has realizado y por qué.

7. **Envía el pull request**:

   Haz clic en el botón **"Create pull request"** para enviarlo.

- ## ¿Cómo aceptar una solicitud de extracción?

Pasos para aceptar un pull request:

1. **Ve al repositorio original**:
   Dirígete a la página del repositorio al que se ha enviado la solicitud de extracción (pull request).

2. **Haz clic en la pestaña "Pull requests"**:

   En la parte superior del repositorio, haz clic en la pestaña **"Pull requests"** para ver las solicitudes de extracción abiertas.

3. **Selecciona la solicitud de extracción (pull request)** que deseas aceptar:
   Haz clic en la solicitud de extracción (pull request) que quieres revisar y aceptar.

4. **Revisa los cambios**:

   GitHub te mostrará las diferencias entre la rama de la solicitud de extracción y la rama principal (por lo general, main o master). Puedes revisar los cambios, los comentarios, y las conversaciones que se hayan generado alrededor de la solicitud.

5. **Haz clic en el botón "Merge pull request"**:

   Una vez que hayas revisado y aprobado los cambios, haz clic en el botón **"Merge pull request"**. Esto fusionará los cambios de la solicitud en tu rama principal.

6. **Confirma la fusión**:

   Después de hacer clic en **"Merge pull request"**, GitHub te pedirá que confirmes la fusión. Haz clic en **"Confirm merge"** para completar el proceso.

7. **Opcional: Elimina la rama de la solicitud de extracción**:

   Una vez que la solicitud se haya fusionado, GitHub te dará la opción de eliminar la rama en la que se realizaron los cambios. Si ya no necesitas esa rama, puedes eliminarla para mantener limpio el repositorio.

- ## ¿Qué es un etiqueta en Git?

En **Git**, una **etiqueta** (o **tag**) es una **marca** o **punto de referencia** en el historial del repositorio, que generalmente se utiliza para identificar versiones importantes o hitos específicos de un proyecto, como una **versión de lanzamiento** (por ejemplo, v1.0, v2.0).

Características de una etiqueta:

- **Inmutable**: Una vez que se crea, una etiqueta no cambia. Es un marcador fijo en un commit específico.
- **Versiones y lanzamientos**: Las etiquetas son comúnmente usadas para marcar puntos de lanzamiento o versiones importantes del proyecto.
- **No cambia con las actualizaciones**: A diferencia de las ramas, que pueden cambiar y evolucionar con los commits, las etiquetas permanecen estáticas en el commit que representan.

Tipos de etiquetas:

1. **Etiquetas ligeras** (lightweight tags): Son simplemente un marcador en un commit, sin más información asociada.
2. **Etiquetas anotadas** (annotated tags): Son etiquetas completas que se almacenan como objetos en Git, incluyendo información como el nombre del creador, la fecha y un mensaje opcional.

- ## ¿Cómo crear una etiqueta en Git?

1. **Crear una etiqueta ligera**:
   git tag nombre-etiqueta

2. **Crear una etiqueta anotada**:
   git tag -a nombre-etiqueta -m "Mensaje descriptivo"

- ## ¿Cómo enviar una etiqueta a GitHub?

1. **Crear una etiqueta en tu repositorio local** (si aún no lo has hecho):

Para una etiqueta **ligera** (sin mensaje):
   git tag nombre-etiqueta

Para una etiqueta **anotada** (con mensaje descriptivo):
   git tag -a nombre-etiqueta -m "Descripción de la etiqueta"

2. **Enviar (empujar) la etiqueta a GitHub**:

Para **enviar una etiqueta específica**:
   git push origin nombre-etiqueta

Para **enviar todas las etiquetas** de tu repositorio local a GitHub:
git push --tags

- ## ¿Qué es un historial de Git?

El **historial de Git** es un registro completo de todos los **commits** realizados en un repositorio, que guarda un seguimiento detallado de cada cambio en los archivos, incluyendo:

1. **Fecha y hora de los cambios**.
2. **Autor de los cambios**.
3. **Descripción de los cambios** (mensaje del commit).
4. **Identificador único del commit** (hash SHA-1).

Este historial permite realizar un seguimiento completo de cómo ha evolucionado el proyecto a lo largo del tiempo y facilita la **restauración de versiones anteriores**, la **revisión de cambios** y la **colaboración** entre desarrolladores.

Características clave del historial de Git:

- **Versionado completo**: Cada commit guarda un "snapshot" de los archivos en un momento específico, lo que permite retroceder a cualquier estado anterior del proyecto.
- **Integridad**: Git utiliza hashes únicos para identificar cada commit, lo que asegura la integridad del historial.
- **Deshacer cambios**: Puedes usar el historial para revertir cambios a estados anteriores del repositorio si es necesario.

• ¿Cómo ver el historial de Git?

Puedes usar el comando:
git log

Este comando mostrará los detalles de todos los commits realizados en el repositorio, con información como el identificador del commit (hash), el autor, la fecha y el mensaje de commit.

• ¿Cómo buscar en el historial de Git?

1. **Buscar por mensaje de commit:**
   Si deseas buscar un término específico dentro de los mensajes de commit, puedes usar el siguiente comando:
   git log - -grep="término de búsqueda"

2. **Buscar por un autor específico:**
   Si quieres encontrar todos los commits realizados por un **autor específico**, usa el siguiente comando:
   git log - -author="Nombre del Autor"
   Esto filtrará los commits realizados por la persona cuyo nombre aparece en los commits.

3. **Buscar cambios en un archivo específico:**
   Si deseas ver el historial de un archivo específico, puedes usar:
   git log - - <ruta/del/archivo>
   Esto mostrará todos los commits que afectaron a ese archivo en particular.

4. Buscar por un rango de fechas:
Puedes buscar commits realizados en un rango de fechas utilizando las opciones `--since` y `--until`.
Ejemplo:
git log --since="2023-01-01" --until="2023-03-01"
Esto mostrará los commits realizados entre el 1 de enero y el 1 de marzo de 2023.

5. Buscar cambios en el contenido de los archivos:
Si deseas buscar cambios en el contenido de los archivos (por ejemplo, una palabra específica que se haya añadido o modificado), puedes usar:
git log -S "palabra clave"
Este comando buscará los commits donde la palabra clave haya sido añadida o eliminada en los archivos del repositorio.

6. Filtrar por ramas:
Si solo deseas ver el historial de una rama en particular, puedes usar:
git log nombre-de-la-rama
Esto limitará la búsqueda al historial de esa rama.

- ## ¿Cómo borrar el historial de Git?

Eliminar el historial de commits (reescribir el historial)

Opción 1: Usar git reset (cambiar a un estado limpio)

1. **Restablecer el repositorio** a un commit inicial (por ejemplo, el primer commit o un commit específico):
   git reset --hard <commit-id>
   Esto resetea el repositorio al commit especificado, **eliminando todos los commits posteriores** a ese punto. El historial ya no será accesible.

2. **Eliminar todos los commits (iniciar desde un nuevo estado):**
   git reset - -hard <hash del primer commit>
   Luego, para eliminar los archivos rastreados de Git:
   rm -rf .git

Opción 2: Usar git rebase (eliminar un rango de commits)

Si deseas eliminar un rango específico de commits (por ejemplo, los últimos N commits), puedes hacer un rebase interactivo:

1. Inicia el rebase interactivo con:
   git rebase -i HEAD~N

   Esto abrirá un editor de texto donde puedes elegir los commits que quieres eliminar. Cambia `pick` por `drop` para eliminar un commit específico.

2. Guarda y cierra el editor. Git reescribirá el historial, eliminando los commits seleccionados.

Eliminar un archivo o una carpeta específica del historial

git filter-branch --tree-filter 'rm -f archivo' --prune-empty HEAD
Esto eliminará `archivo` de todo el historial del repositorio.

bfg - -delete-files archivo
Es una herramienta más rápida y eficiente para limpiar archivos de grandes repositorios.
Esto eliminará el archivo del historial y hará que Git reescriba los commits.

Eliminar el historial de GitHub (repositorio remoto)

git push origin - -force
Advertencia: El uso de `--force` puede sobrescribir el historial en el repositorio remoto, lo que afectará a otros colaboradores. Es importante coordinar con nuestro equipo antes de hacer esto.

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

Un **repositorio privado en GitHub** es un repositorio de código cuyo **acceso está restringido**. Solo las personas o equipos que el propietario del repositorio **invita explícitamente** pueden ver, clonar, o colaborar en él. A diferencia de los repositorios públicos, que son accesibles por cualquier persona en la web, los repositorios privados están protegidos para que solo los usuarios autorizados puedan interactuar con ellos.

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

- Accede a tu cuenta de GitHub.

- Haz clic en el botón **"New repository"** (Nuevo repositorio).

- En el formulario de creación del repositorio, selecciona la opción **"Private"** en lugar de "Public".

- Completa los detalles del repositorio y haz clic en **"Create repository"**.

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

- Puedes invitar colaboradores y asignarles permisos (lectura, escritura, administración) en la sección **"Settings"** de tu repositorio privado.

• ¿Qué es un repositorio público en GitHub?

Un **repositorio público en GitHub** es un repositorio de código cuyo **acceso es libre y abierto para todos**. Cualquier persona, incluso si no tiene cuenta en GitHub, puede ver, clonar, bifurcar (hacer un *fork*), y colaborar en el repositorio (dependiendo de los permisos que otorgue el propietario).

Características de un repositorio público:

1. **Acceso libre**: Cualquier persona puede ver el contenido del repositorio, descargarlo, y clonar el código.
2. **Visibilidad en motores de búsqueda**: Un repositorio público puede ser indexado por motores de búsqueda como Google, lo que facilita que otros lo encuentren.
3. **Colaboración abierta**: Otros usuarios pueden hacer un **fork** del repositorio, sugerir cambios mediante **pull requests** y colaborar en el desarrollo, según los permisos establecidos por el propietario.
4. **Ideal para proyectos de código abierto**: Los repositorios públicos son utilizados frecuentemente para proyectos de **código abierto**, donde se desea que la comunidad contribuya y utilice el código de manera libre.

• ¿Cómo crear un repositorio público en GitHub?

1. **En GitHub**:
   - Accede a tu cuenta de GitHub.
   - Haz clic en el botón **"New repository"** (Nuevo repositorio).
   - En el formulario de creación del repositorio, selecciona la opción **"Public"** (Público).
   - Completa los detalles del repositorio (nombre, descripción) y haz clic en **"Create repository"**.

2. **Colaboración y contribuciones**:
   - Puedes permitir que otros colaboren en el proyecto mediante **pull requests**. Deberás configurar los permisos y guías para los contribuyentes si quieres que sigan un flujo de trabajo organizado.

• ¿Cómo compartir un repositorio público en GitHub?

1. **Accede a tu repositorio**:
   - Inicia sesión en tu cuenta de **GitHub**.
   - Navega hasta el repositorio que deseas compartir.

2. **Obtener la URL del repositorio**:
   - Una vez dentro de tu repositorio, encontrarás la URL en la barra de direcciones del navegador. La URL tiene la siguiente forma:

   https://github.com/usuario/nombre-del-repositorio

Donde **usuario** es tu nombre de usuario en GitHub y **nombre-del-repositorio** es el nombre de tu repositorio.

3. **Copiar y compartir el enlace**:
   - Simplemente **copia** esa URL de la barra de direcciones.
   - Puedes **enviarla** por correo electrónico, compartirla en redes sociales, incluirla en tu sitio web, o compartirla de cualquier otra forma que desees.

Ejemplo de URL:

Si tu nombre de usuario es juan123 y el nombre de tu repositorio es mi-proyecto, la URL del repositorio sería:

https://github.com/juan123/mi-proyecto

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

ACTIVIDAD RESUELTA:

https://github.com/daiVelasquez/repo-de-prueba.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, g.
- 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.

ACTIVIDAD RESUELTA:

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