

Práctico 2: Git y GitHub

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

- ¿Qué es GitHub?
 - Es una plataforma que ofrece alojamiento de repositorios de control de versiones, que permite a los desarrolladores almacenar y gestionar sus proyectos de software. Es una herramienta gratuita y de código abierto que permite el trabajo en equipo en proyectos, el intercambio de código y el trabajo conjunto de forma eficiente.
- ¿Cómo crear un repositorio en GitHub?
 - Primero debes crear una cuenta, puedes hacerlo en la página web de GitHub. Una vez que tengas una cuenta, puedes configurar tu perfil y empezar a crear repositorios. Si es la primera vez utilizando esta plataforma y queremos subir un repositorio a github debemos de clicar en New, una vez allí colocar el nombre, alguna que otra descripción de lo que vamos a subir, si queremos que sea público o privado, etc.
- ¿Cómo crear una rama en Git?
 - Para crear una nueva rama, se usará el comando git branch: `$ git branch nuevaRama` Git sabe en qué rama estás mediante un apuntador especial denominado HEAD. Que estemos creando una nueva rama con git branch no significa que estemos en dicha rama, estaríamos en la rama master.
- ¿Cómo cambiar a una rama en Git?
 - Para saltar de una rama a otra, tienes que utilizar el comando git checkout: `$ git checkout nuevaRama`
- ¿Cómo fusionar ramas en Git?

- Para fusionar ramas en Git, sigues un proceso que combina los cambios de una rama en otra. Primero, debes estar en la rama a la que quieres fusionar los cambios. Por lo general, es a la rama principal (master o main) o cualquier otra rama en la que estés integrando los cambios. \$ git checkout master. Una vez en la rama de destino, usa el comando git merge para fusionar la rama de origen en la rama actual: \$ git merge nuevaRama Este comando incorpora los cambios de nuevaRama en master.
- ¿Cómo crear un commit en Git?
 - Crear un commit en Git es el proceso mediante el cual se guardan los cambios realizados en el repositorio. Un commit captura el estado actual del código en un momento dado y lo almacena en el historial del proyecto. Pasos para hacer un commit: Primero realiza los cambios en los archivos de tu proyecto que desees guardar en el commit. Luego debes agregar los cambios al área de preparación. Esto se hace con el comando git add. Puedes agregar archivos específicos o todos los archivos modificados: \$ git add archivo1 (para agregar el archivo "archivo1") o \$git add . (para agregar todos los archivos) Una vez que los cambios están en el área de preparación, puede hacer el commit con el comando git commit. Debes proporcionar un mensaje de commit que describa los cambios realizados: \$ git commit -m "Mensaje de los cambios" Esto guarda el estado actual del código en el historial del repositorio con el mensaje correspondiente.
- ¿Cómo enviar un commit a GitHub?
 - Primero debemos clonar el repositorio de GitHub a nuestra máquina local: git clone https://github.com/tu_usuario/tu_repositorio.git cd tu_repositorio Haz cambios en los archivos del repositorio y agrégalos: git add nombre_del_archivo o git add . Crea un commit de tus cambios: git commit -m "Descripción de los cambios" Para enviar los commits al repositorio en GitHub, debemos empujarlos con: git push origin nombre_de_la_rama
- ¿Qué es un repositorio remoto?
 - Para poder colaborar en cualquier proyecto Git, necesitas saber cómo gestionar repositorios remotos. Los repositorios remotos son versiones de tu proyecto que están hospedadas en Internet o en cualquier otra red. Puedes tener varios de ellos, y en cada uno tendrás generalmente permisos de solo lectura o de lectura y escritura. Colaborar con otras personas implica gestionar estos repositorios remotos enviando y trayendo datos de ellos cada vez que necesites compartir tu trabajo. Gestionar repositorios remotos incluye saber cómo añadir un repositorio remoto,

eliminar los remotos que ya no son válidos, gestionar varias ramas remotas, definir si deben rastrearse o no y más.

- ¿Cómo agregar un repositorio remoto a Git?
 - Utiliza el comando `git remote add` para vincular tu repositorio local con un repositorio remoto y asignar un nombre a ese remoto: `$ git remote add [nombre] [url]` Ponerle un nombre te ayuda a referenciarlo de manera más fácil en lugar de usar la URL completa. Puedes usar el nombre en la línea de comandos. Para asegurarte de que el remoto se ha agregado correctamente y verificar el nombre que le has dado, usa el comando: `$ git remote -v` Esto mostrará una lista de todos los remotos configurados con sus URLs: `[nombre] [url]` Para traer toda la información del repositorio remoto que aún no tienes en tu repositorio local, usa el comando `git fetch` seguido del nombre del remoto: `$ git fetch [nombre]` Este comando descarga todos los cambios del repositorio remoto asociado con el nombre, pero no fusiona esos cambios con tu rama actual. Es útil para ver qué cambios están disponibles en el remoto.
- ¿Cómo empujar cambios a un repositorio remoto?
 - Antes de empujar tus cambios, es una buena práctica obtener los últimos cambios del repositorio remoto para evitar conflictos: `git pull origin nombre_de_la_rama`. Empuja tus cambios al repositorio remoto: `git push origin nombre_de_la_rama`
- ¿Cómo tirar de cambios de un repositorio remoto?
 - Como mencionamos en el punto anterior, para tirar los cambios del repositorio remoto Usa el comando `git pull` para descargar y fusionar los cambios del repositorio remoto con tu rama local: `git pull origin nombre_de_la_rama` Por ejemplo, si estás trabajando en la rama main: `git pull origin main`
- ¿Qué es un fork de repositorio?
 - En muchas ocasiones podemos ver en github repositorios que son de nuestro agrado y donde nosotros queremos tener una copia y hacerle algunas modificaciones, para ello github nos ofrece la implementación de fork. Buscamos algún repositorio que nos parezca interesante. Este puede ser cualquier proyecto de código abierto que quieras modificar o utilizar como base para tu propio trabajo. En la página del repositorio, busca el botón "Fork" ubicado en la parte superior derecha de la página. Haz clic en el botón "Fork". Esto creará una copia del repositorio en tu propia cuenta de GitHub. Esta copia será independiente del repositorio original. Cualquier cambio que realices en tu fork no afectará al repositorio original,

los cambios que hagamos no irán al repositorio original del autor de ese proyecto, si no a nuestra copia local. Entonces solo resta clonar ese repositorio que está en nuestro canal para poder trasladarlo a alguna carpeta deseada y seguir trabajando desde nuestro repositorio local.

- ¿Cómo crear un fork de un repositorio?
 - Para entender esto puedes por ejemplo crear una cuenta adicional en github para que puedas desde allí hacer un Fork de alguno de los repositorios de tu cuenta original, clonarlo en alguna carpeta y en algún archivo que nosotros notamos que podría mejorarse algo, procedemos a realizar algún cambio, lo agregamos al stage y lo commiteamos (todo esto desde nuestro repositorio local remoto) y luego mandamos los cambios a el forking que hicimos con un git push origin master. Ese cambio que hemos hecho queremos que sea visto por el creador del repositorio original (en este caso nosotros, desde nuestra cuenta original), debemos hacérselo notar puesto que en el repositorio original los cambios que hicimos previamente, no se verán reflejados.
- ¿Cómo enviar una solicitud de extracción (pull request) a un repositorio?
 - Para hacer el pull request nos dirigiremos a la solapa de Pull requests allí daremos click en new pull request, veremos una ventana a modo de resumen en donde se reflejarán los cambios que hemos hecho nosotros en comparación al repositorio original (el código original, mejor dicho). Daremos click en Create pull request donde veremos el asunto (colocamos algún mensaje global) y más abajo tenemos suficiente lugar para poder explayarnos en mencionar el porque ese cambio que hemos realizado nosotros, sería considerado como algo que a el repositorio original le vendrían bien agregarlo.
- ¿Cómo aceptar una solicitud de extracción?
 - El autor del repositorio verá en sus pull requests el mensaje que le hemos enviado, para que lo pueda observar y si lo considera realizar el cambio pertinente (además de poder responderle al usuario que le ha propuesto ese cambio). Lo bueno de todo esto es que si el usuario original considera que esta modificación es buena y no genera conflictos con la rama maestra de su repositorio local remoto, puede clickear en Merge pull request y de esta manera sumará a su repositorio los cambios que hizo un usuario (en modo de ayuda).
- ¿Qué es un etiqueta en Git?

- Como muchos VCS, Git tiene la posibilidad de etiquetar puntos específicos del historial como importantes. Esta funcionalidad se usa típicamente para marcar versiones de lanzamiento (v1.0, por ejemplo).
- ¿Cómo crear una etiqueta en Git?
 - Git utiliza dos tipos principales de etiquetas: ligeras y anotadas. Una etiqueta ligera es muy parecida a una rama que no cambia - simplemente es un puntero a un commit específico. Sin embargo, las etiquetas anotadas se guardan en la base de datos de Git como objetos enteros. Tienen un checksum; contienen el nombre del etiquetador, correo electrónico y fecha; tienen un mensaje asociado; y pueden ser firmadas y verificadas con GNU Privacy Guard (GPG). Normalmente se recomienda que crees etiquetas anotadas, de manera que tengas toda esta información; pero si quieres una etiqueta temporal o por alguna razón no estás interesado en esa información, entonces puedes usar las etiquetas ligeras.
- ¿Cómo enviar una etiqueta a GitHub?
 - Primero, debes crear una etiqueta en tu repositorio local. Puedes crear una etiqueta anotada (que incluye información adicional como el autor y la fecha) o una etiqueta ligera (simplemente un puntero a un commit): Ej. etiqueta ligera: `git tag v1.0` Puedes verificar las etiquetas que has creado localmente con:
`git tag`. Una vez que has creado la etiqueta en tu repositorio local, necesitas empujarla al repositorio remoto en GitHub. Puedes hacer esto con el siguiente comando: `git push origin v1.0` (origin es el nombre del repositorio remoto (por defecto suele ser origin) y v1.0 es el nombre de la etiqueta.) Para empujar todas las etiquetas creadas, usar: `git push origin --tags`
- ¿Qué es un historial de Git?
 - El historial de Git es una secuencia de todos los cambios realizados en un repositorio de Git. Cada cambio en el repositorio se guarda como un commit, y cada commit contiene información sobre el estado del proyecto en un momento específico, incluyendo: Identificador del commit Autor Fecha de realización Mensaje enviado
- ¿Cómo ver el historial de Git?
 - Esto lo conseguimos con el comando de Git: `git log` Con tipear este comando en el bash de Git podremos apreciar el histórico de commits, estando situados en la carpeta de nuestro proyecto. El listado de commits estará invertido, es decir, los últimos realizados aparecen los primeros. El comando `git log --oneline` es una forma compacta de visualizar el historial

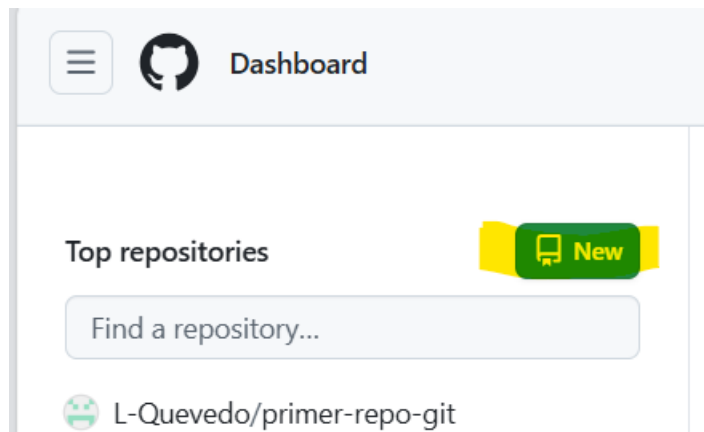
de commits en un repositorio Git. Muestra un resumen conciso de los commits recientes, con cada commit representado en una sola línea. Si tu proyecto ya tiene muchos commits, quizás no quieras mostrarlos todos, ya que generalmente no querrás ver cosas tan antiguas como el origen del repositorio. Para ver un número de logs determinado introducimos ese número como opción, con el signo "-" delante (-1, -8, -12...). Por ejemplo, esto muestra los últimos tres commits: `git log -3` Si queremos que el log también nos muestre los cambios en el código de cada commit podemos usar la opción -p. Esta opción genera una salida mucho más larga, por lo que seguramente nos tocará movernos en la salida con los cursores y usaremos CTRL + Z para salir. `git log -2 -p`

- ¿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?
 - El comando `git reset` se utiliza sobre todo para deshacer las cosas, como posiblemente puedes deducir por el verbo. Se mueve alrededor del puntero HEAD y opcionalmente cambia el index o área de ensayo y también puede cambiar opcionalmente el directorio de trabajo si se utiliza `- hard`. Esta última opción hace posible que este comando pueda perder tu trabajo si se usa incorrectamente, por lo que asegúrese de entenderlo antes de usarlo.
- ¿Qué es un repositorio privado en GitHub?
 - Un repositorio privado en GitHub es un tipo de repositorio en el que el contenido solo es accesible para usuarios específicos que han sido autorizados. A diferencia de los repositorios públicos, donde cualquier persona puede ver y clonar el contenido, un repositorio privado limita el acceso a los colaboradores que tú elijas. Esto es útil para proyectos que contienen información sensible o que aún están en desarrollo y no deseas que estén disponibles públicamente.
- ¿Cómo crear un repositorio privado en GitHub?

- Inicia sesión en GitHub Ingresa a la página de creación de repositorios: En la esquina superior derecha de la página principal, debes hacer clic en el botón “+” y seleccionar “New Repository” o hacer clic en “New” Completar la información del repositorio (nombre del repositorio, descripción) Seleccionar la configuración de privacidad: “Privada”. Esto asegura que el repositorio será privado y solo accesible para los colaboradores que tu elijas.
- ¿Cómo invitar a alguien a un repositorio privado en GitHub?
 - Invitar a alguien a un repositorio privado en GitHub es un proceso sencillo, pero requiere permisos adecuados.
Accede al repositorio, haz clic en la pestaña "Settings" del repositorio. Está en la parte superior del repositorio, junto a las pestañas como "Code" y "Issues".
Selecciona "Collaborators" en el menú de la izquierda. Esto te llevará a la página donde puedes administrar colaboradores. En la sección "Collaborators", haz clic en el botón "Add people" e ingresa el nombre de usuario de GitHub de la persona que deseas invitar. Selecciona el nivel de acceso que deseas otorgar: Read, Triage, Write, Maintain, o Admin.
Haz clic en el botón "Add" para enviar la invitación.
- ¿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?
 - Pasos: Inicia sesión en GitHub Ingresa a la página de creación de repositorios: En la esquina superior derecha de la página principal, debes hacer clic en el botón “+” y seleccionar “New Repository” o hacer clic en “New”
Completar la información del repositorio (nombre del repositorio, descripción) Seleccionar la configuración de privacidad: “Publico”
Esto asegura que el repositorio será público.
- ¿Cómo compartir un repositorio público en GitHub?
 - La forma más sencilla de compartir tu repositorio es proporcionar el enlace directo al mismo. Accede a tu repositorio, copia la URL de tu repositorio que se encuentra en un cuadro de texto que dice "<> Code":

2) Realizar la siguiente actividad:

- Crear un repositorio.
 - Dale un nombre al repositorio.



- Elije el repositorio sea público.

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 *

L-Quevedo

Repository name *

Nombre del repositorio

✓ Your new repository will be created as Nombre-del-repositorio.

The repository name can only contain ASCII letters, digits, and the characters ., -, and _.

Great repository names are short and memorable. Need inspiration? How about curly-happiness ?

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.

- 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).


```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1>git clone https://github.com/L-Quevedo/-UTN-TUPaD-P1-Lucas_Quevedo.git
Cloning into '-UTN-TUPaD-P1-Lucas_Quevedo'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 21 (delta 1), reused 0 (delta 0), pack-reused 19 (from 2)
Receiving objects: 100% (21/21), 4.43 KiB | 906.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1>
```

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1>git commit -m "Agregando mi-archivo.txt"
Auto packing the repository for optimum performance.
See "git help gc" for manual housekeeping.
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 32 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (19/19), done.
Total 19 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
|
```

L-Quevedo Delete README.MD	
01 Estructuras Secuenciales	😊 ¡Bienvenid@ a Programación 1!
02 Trabajo Colaborativo	😊 ¡Bienvenid@ a Programación 1!
03 Estructuras Condicionales	😊 ¡Bienvenid@ a Programación 1!
04 Estructuras Repetitivas	😊 ¡Bienvenid@ a Programación 1!
05 Funciones	😊 ¡Bienvenid@ a Programación 1!
06 Datos complejos	😊 ¡Bienvenid@ a Programación 1!
07 Manejo de errores	😊 ¡Bienvenid@ a Programación 1!
08 Test unitario	😊 ¡Bienvenid@ a Programación 1!
09 Análisis de algoritmos	😊 ¡Bienvenid@ a Programación 1!
10 Búsqueda y ordenamiento	😊 ¡Bienvenid@ a Programación 1!
11 Recursividad	😊 ¡Bienvenid@ a Programación 1!
12 Datos Avanzados	😊 ¡Bienvenid@ a Programación 1!
Readme.md	Create Readme.md

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>git push -u origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 32 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 677 bytes | 677.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/L-Quevedo/-UTN-TUPaD-P1-Lucas_Quevedo/pull/new/master
remote:
To https://github.com/L-Quevedo/-UTN-TUPaD-P1-Lucas_Quevedo.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>
```

- Creando Branchs

```
C:\Windows\System32\cmd.exe x + v
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>echo Esto es otra rama > archivo-feature.txt
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>git add .
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>git commit -m "Agregando archivo en la branch nueva-feature"
[nueva-feature 2cb7b58] Agregando archivo en la branch nueva-feature
1 file changed, 1 insertion(+)
create mode 100644 archivo-feature.txt
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>git push -u origin nueva-feature
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 32 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 327 bytes | 327.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'nueva-feature' on GitHub by visiting:
remote:   https://github.com/L-Quevedo/-UTN-TUPaD-P1-Lucas_Quevedo/pull/new/nueva-feature
remote:
To https://github.com/L-Quevedo/-UTN-TUPaD-P1-Lucas_Quevedo.git
 * [new branch]      nueva-feature -> nueva-feature
branch 'nueva-feature' set up to track 'origin/nueva-feature'.

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>
```

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

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`

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>cd
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>git clone https://github.com/L-Quevedo/conflict-exercise.git
Cloning into 'conflict-exercise'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git>cd conflict-exercise
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>cd
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\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"
```

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>cd
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git checkout -b feature-branch
Switched to a new branch 'feature-branch'
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git add README.md
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git commit -m "added a line in feature-bran
ch
[feature-branch ab334a0] added a line in feature-branch
1 file changed, 2 insertions(+), 1 deletion(-)
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>
```

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

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

`git checkout main`

- 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"

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>cd
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git add README.md

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git commit -m "added a line in main branch"

[main dd12ad1] added a line in main branch
1 file changed, 3 insertions(+), 1 deletion(-)

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 32 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 317 bytes | 317.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/L-Quevedo/conflict-exercise.git
 f14db9c..dd12ad1  main -> main

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>
```

Paso 5: Hacer un merge y generar un conflicto

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

`git merge feature-branch`

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

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\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.

C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\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"
```

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git commit -m "Resolved merge conflict"
[main f3598d4] Resolved merge conflict
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\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
```

```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git push origin main
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 32 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 1.08 KiB | 79.00 KiB/s, done.
Total 12 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/L-Quevedo/conflict-exercise.git
dd12ad1..f3598d4 main -> main
```


- También sube la feature-branch si deseas:




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


```
C:\Users\lmque\OneDrive\Desktop\TUP\1 - Programación 1\Git\conflict-exercise>git push origin feature-branch
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:   https://github.com/L-Quevedo/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/L-Quevedo/conflict-exercise.git
* [new branch]   feature-branch -> feature-branch
```


Paso 8: Verificar en GitHub




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

 **conflict-exercise** Public Pin Unwatch 1

 **main**  2 Branches  0 Tags t Add file Code

 **L-Quevedo** Resolved merge conflict f3598d4 · 5 minutes ago 6 Commits

 README.md Resolved merge conflict 5 minutes ago

 README  

conflict-exercise