# TP 2: Git y GitHub

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

### ¿Qué es GitHub?

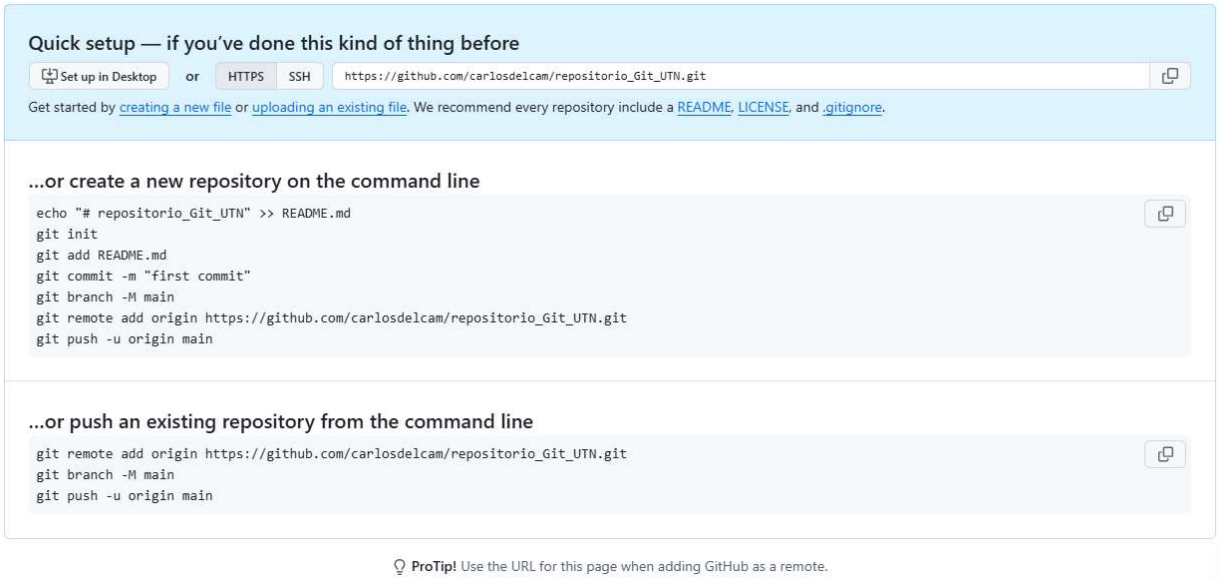
GitHub es una plataforma en línea, gratuita y de código abierto, que ofrece alojamiento de repositorios, basados en Git, permitiendo el control de versiones, donde los desarrolladores pueden centralizar, almacenar, compartir, gestionar y colaborar en proyectos -trabajando de forma colaborativa- de desarrollo de software de forma eficiente. Funciona como una red social para desarrolladores, donde se pueden subir códigos fuente, hacer cambios y trabajar en equipo usando Git, que es un sistema de control de versiones

### ¿Cómo crear un repositorio en GitHub?

Puedo crear localmente en Git con el comando **git init**
O en la plataforma github.com, se debe disponer de una cuenta registrada. Una vez iniciada sesión, se permite configurar el perfil y empezar a crear repositorios; para ello, se hace clic en el botón "***New"*** o "**Crear repositorio"**, colocándole un nombre para identificar dicho repositorio, alguna descripción explicativa si se desea, eligiendo si será público o privado, y se puede agregar un archivo README. Después de esto, solo hay que darle a ***"Create repository"*** . De esta forma se dispone de un repositorio creado para empezar a subir archivos o trabajar en equipo.

Si hacemos clic dentro del repositorio, mostrara la URL del repositorio y líneas de comando para:

***"…or create a new repository on the command line"*** nos proporciona los pasos para enlazar el repositorio local con el nuevo repositorio remoto ó "***…or push an existing repository from the command line"*** para enlazar lo almacenado en el repositorio existente desde la línea de comandos.




### ¿Cómo crear una rama en Git?

Para crear una rama en Git, primero debemos asegurarnos de estar dentro de un repositorio. Luego, en la terminal, se usa el comando:

**git branch** ***nombre_rama***

Por defecto, cuando creamos un repositorio en Git con **git init** , crea una rama "principal" o "master"

### ¿Cómo cambiar a una rama en Git?

Para movernos o cambiar de rama de inmediato, en la terminal usamos:

**git checkout** ***nombre_rama_acambiar***

O una forma más rápida con un solo comando:

**git checkout -b *nombre-de-la-rama*** de esta forma ya podemos trabajar en esa rama sin afectar el código de la principal.

### ¿Cómo fusionar ramas en Git?

Para fusionar ramas en Git, (hacer Merge), primero, debemos estar en la rama a la que deseamos 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, usamos el comando:

**git merge *nombre--rama-a- fusionar*** este comando incorpora los cambios de la rama a fusionar 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, actualiando los cambios del repositorio actual en el remoto y haciendo un seguimiento en el historial del proyecto; para esto se debe emplear los siguientes comandos:

**git add .** para agregar todos los archivos al repositorio virtual

**git commit -m "*comentario de los cambios realizados*"** Esto guarda el estado actual del código en el historial del repositorio con el mensaje correspondiente.

### ¿Cómo enviar un commit a GitHub?

Para enviar un commit a GitHub, debemos seguir estos pasos:

**git checkout *nombre-de-la-rama*** para asegurarnos de estar en la rama correcta

**git add .** preparar para agrega los cambios al área de preparación

**git commit -m "*Descripción del cambio*"** guardado de los cambios actuales con un mensaje descriptivo

**git push origin *nombre-de-la-rama*** sube los cambios al repositorio remoto en GitHub:

Si estamos en la rama main o master, usamos **git push origin main** o **git push origin master**.

De esta forma, nuestros cambios estarán disponibles en GitHub para compartir o seguir trabajando

### ¿Qué es un repositorio remoto?

Un repositorio remoto es una versión en línea de nuestro repositorio local en GitHub, donde guardamos el proyecto en la nube para poder acceder a él desde cualquier lugar y/o permitirnos colaborar con otros desarrolladores del equipo. Se conecta con nuestro proyecto local usando Git y permite subir (push) o descargar (pull) cambios fácilmente.

**git remote –v** Permite ver los repositorios remotos en un proyecto

**git remote add origin *URL-del-repositorio*** para conectar un repositorio local con GitHub

### ¿Cómo agregar un repositorio remoto a Git?

**git init** Inicializamos Git, en caso de no estarlo, en la carpeta del proyecto

**git remote add origin *URL-del-repositorio*** Agregamos el repositorio remoto con la URL de GitHub

**git remote –v** Verificamos que se agregó correctamente:

Luego de esto, ya podemos subir (push) y descargar (pull) los cambios entre nuestro proyecto local y el remoto en GitHub.

### ¿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***

**git push origin *nombre_de_la_rama*** Empuja tus cambios al repositorio remoto

### ¿Cómo tirar de cambios de un repositorio remoto?

Para actualizar un repositorio local con los cambios más recientes de un repositorio remoto en Git, debemos utilizar el comando **git pull** ; este comando combina dos acciones: primero, descarga el contenido del repositorio remoto y, luego, fusiona esos cambios en la rama local actual

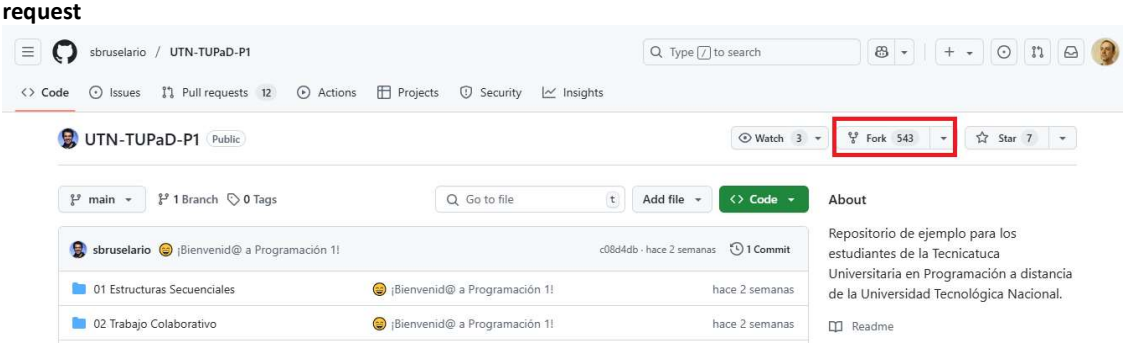
Pasos para realizar un **git pull**, una veas abierta la terminal, en el repositorio local:

**cd** ***ruta/al/repositorio*** navegamos al directorio del repositorio local
**git pull origin** ***rama***

***origin*** es el nombre del repositorio remoto (por defecto, suele llamarse origin).
***rama*** es el nombre de la rama que deseamos actualizar (por ejemplo, main o master).

### ¿Qué es un fork de repositorio?

Un fork en GitHub es una copia de un repositorio externo en nuestra propia cuenta. Sirve para modificar un proyecto sin afectar el original, lo que es útil cuando queremos contribuir a proyectos propiedad de otras personas.
Para hacer un fork, solo hay que ir al repositorio en GitHub y hacer clic en el botón "**Fork**". Después, podemos clonarlo, hacer cambios y, si queremos que esos cambios se integren en el proyecto original, enviamos un **pull request**




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

Para enviar una solicitud de extracción (**pull request**) en GitHub, seguimos estos pasos:
1.Hacemos un fork del repositorio (si no es nuestro) y clonamos el proyecto en nuestra computadora:
**git clone** ***URL-del-repositorio***
2.Creamos una nueva rama para hacer cambios:
**git checkout -b** ***nombre-de-la-rama***
3.Realizamos los cambios, los agregamos y hacemos un commit:
**git add .**
**git commit -m "*Descripción de los cambios*"**
4.Subimos la rama a nuestro repositorio en GitHub:
**git push origin** ***nombre-de-la-rama***
5.En GitHub, vamos al repositorio original, hacemos clic en "**Pull Requests**" y luego en "**New Pull Request**". Seleccionamos nuestra rama con los cambios y enviamos la solicitud; si el dueño del repositorio acepta nuestro pull request, los cambios se integrarán en el proyecto original.

### ¿Cómo aceptar una solicitud de extracción?

Para aceptar una solicitud de extracción (**pull request**) :
1. En el repositorio en GitHub, hacer clic en la pestaña "**Pull Requests**".
2. Seleccionar la solicitud de extracción que queremos revisar.
3. Revisar los cambios propuestos. Podemos ver los archivos modificados y dejar comentarios si es necesario.
4. Si todo está bien, hacemos clic en el botón "**Merge pull request**".
5. Confirmamos la fusión haciendo clic en "**Confirm merge**".
6. Opcionalmente, eliminamos la rama que se fusionó para mantener limpio el repositorio.
De esta forma los cambios de la solicitud se integran en la rama principal del proyecto.

### ¿Qué es un etiqueta en Git?

En Git, una etiqueta o tag es como una marca que ponemos en un punto específico de la historia de nuestro proyecto para identificar versiones importantes, como una versión 1.0 o una actualización significativa. Son útiles para señalar lanzamientos o momentos clave en el desarrollo, y nos ayudan a llevar un control claro de las versiones y facilitar el acceso a puntos específicos en la historia del proyecto.

Existen dos tipos principales de etiquetas en Git:

- **Etiquetas ligeras**: Son más simples y solo apuntan a una confirmación específica sin información extra.
  **git tag v1.0-lw**

- **Etiquetas anotadas**: almacenan información adicional, como el nombre del creador de la etiqueta, la fecha y un mensaje descriptivo.
  **git tag -a v1.0 -m "*Versión 1.0: Primera versión estable*"**

Para ver todas las etiquetas en un repositorio, usamos: **git tag**
Si deseamos compartir nuestras etiquetas con el repositorio remoto en GitHub, las enviamos con: **git push origin --tags**

### ¿Cómo crear una etiqueta en Git?

Para crear una etiqueta en Git, debemnos estar en el repositorio correcto y en el commit que queremos etiquetar.

- **Etiqueta ligera:** son simples punteros a una confirmación específica y no almacenan información adicional.
  **git tag *nombre-de-la-etiqueta***
- **Etiqueta anotada:** almacenan información adicional, como el nombre del creador, la fecha y un mensaje descriptivo. Se recomienda su uso para marcar versiones oficiales del proyecto.
  **git tag -a *nombre-de-la-etiqueta* -m "*Mensaje descriptivo de la etiqueta*"**

### ¿Cómo enviar una etiqueta a GitHub?

- Enviar una etiqueta específica: Para subir una sola etiqueta:
  **git push origin *nombre-de-la-etiqueta*** ejemplo: git push origin v1.0

- Enviar todas las etiquetas de una vez y subirlas todas al mismo tiempo:
  **git push origin –tags** enviará todas las etiquetas locales que aún no estén en el repositorio remoto.

### ¿Qué es un historial de Git?

En Git, el historial es el registro de todas las confirmaciones (commits) que se han realizado en un repositorio. Cada confirmación representa una instantánea del proyecto en un momento específico e incluye información como el autor, la fecha, el mensaje descriptivo y un identificador único (hash). Este historial permite rastrear cómo ha evolucionado el proyecto a lo largo del tiempo y facilita la colaboración entre desarrolladores.

### ¿Cómo ver el historial de Git?

**git log** muestra una lista de todas las confirmaciones realizadas en el repositorio, ordenadas desde la más reciente hasta la más antigua; nos ayudan a entender mejor la evolución del proyecto y a identificar cambios específicos en el código.
Cada entrada incluye el identificador único del commit (hash), el autor, la fecha y el mensaje descriptivo.
Opciones para personalizar la visualización del historial:

- **git log --stat:** Muestra estadísticas sobre los archivos modificados en cada commit, indicando cuántas líneas se añadieron o eliminaron.
- **git log -p:** Muestra las diferencias introducidas en cada commit, permitiendo ver qué cambios se realizaron en el código.
- **git log --pretty=format:"%h - %an, %ar : %s":** Personaliza la salida del historial, mostrando el hash abreviado, el autor, el tiempo relativo y el mensaje del commit en una sola línea.

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

Para buscar en el historial de Git, podemos utilizar varios comandos que nos permiten filtrar y localizar commits específicos según diferentes criterios. A continuación, se presentan algunas de las opciones más útiles:

Buscar por mensaje de commit:

- **git log --grep="*error*"** mostrará todos los commits cuyo mensaje incluya la palabra "error".

Buscar por autor:

- **git log --author="*Juan Pérez*"** queremos ver los commits realizados por un autor en particular

Buscar por rango de fechas:

- **git log --since="*2023-04-01*" --until="*2023-04-30*"** filtrar commits dentro de un intervalo de tiempo específico.

Buscar por cambios en el código:

- **git log -S "texto"** identificar los commits donde se realizaron cambios (añadió o eliminó una línea) que contiene una cadena específica, relacionados con esa cadena especifica.

Buscar por expresiones regulares en diferencias:

- **git log -G "función"** localizar commits que introdujeron cambios coincidentes con una expresión regular prporcionada. Por ejemplo, para buscar commits con cambios en líneas que contienen "función":

Se pueden combinar estos parámetros, ayudándonos a afinar aún más nuestras búsquedas.

### ¿Cómo borrar el historial de Git?

En Git, el historial es el registro de todos los cambios realizados en un proyecto. Aunque generalmente es importante mantener este historial para rastrear modificaciones y colaborar con otros, hay situaciones en las que podríamos querer eliminar o modificar el historial, como cuando se han cometido errores o se han incluido datos sensibles por accidente.

**Formas de borrar o modificar el historial de Git:**

**1. Eliminar todo el historial localmente:** eliminando la carpeta oculta .git en el directorio del proyecto local en la pc. Esta carpeta contiene todo el historial y la configuración del repositorio. De esta forma se eliminará todo el historial de versiones y el proyecto dejará de ser un repositorio de Git hasta que se vuelva a inicializar con **git init.**

**2. Eliminar commits específicos del historial:** con el comando **git reset**.

**git reset --hard HEAD~1** para eliminar el último commit; este comando retrocede un commit y elimina los cambios asociados. Si ya has subido estos commits a un repositorio remoto, necesitarás forzar la actualización:

**git push origin main --force**

Sin embargo, usar --force puede sobrescribir cambios en el repositorio remoto, por lo que debe usarse con precaución.

**3. Eliminar un archivo específico del historial:** Si se ha añadido accidentalmente un archivo con información sensible y deseamos eliminarlo del historial, usar el comando **git filter-branch:**

**git filter-branch --force --index-filter "git rm --cached --ignore-unmatch *RUTA/DEL/ARCHIVO*" --prune-empty --tag-name-filter cat -- --all**

Después, se deberá forzar la actualización al repositorio remoto:

**git push origin --force --all**

Este proceso reescribe el historial y elimina el archivo especificado de todos los commits.

Reescribir el historial de Git puede afectar a otros colaboradores y causar conflictos si no se maneja adecuadamente. Es esencial coordinar con el equipo antes de realizar cambios significativos en el historial.

Siempre realiza copias de seguridad antes de modificar el historial, ya que estos cambios pueden ser irreversibles.

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

En GitHub, un repositorio privado es un espacio donde se puede almacenar y gestionar el código de forma confidencial. A diferencia de los repositorios públicos, que son accesibles para cualquier persona en internet,

los repositorios privados solo pueden ser vistos y modificados por el administrador y por las personas a las que se les otorguen acceso explícito. Esto significa que se dispone de un control total sobre quién puede ver y colaborar en el proyecto; controlando

Características principales de un repositorio privado:

- **Control de acceso**: Solo tú y las personas que invites pueden acceder al contenido del repositorio. Esto es útil para proyectos que aún no están listos para ser compartidos públicamente o que contienen información sensible.
- **Colaboración segura**: Puedes trabajar en equipo de manera segura, asegurándote de que solo los colaboradores autorizados puedan ver y modificar el código.
- **Confidencialidad garantizada**: GitHub considera el contenido de los repositorios privados como confidencial y toma medidas para protegerlo contra accesos no autorizados.

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

Para crear un repositorio privado en GitHub, accedemos a la opción para crear un nuevo repositorio, en la esquina superior derecha de cualquier página de GitHub, y hacemos clic en el ícono "+" y seleccionamos "*Nuevo* repositorio". Colocamos un nombre único y descriptivo para el repositorio, una descripción (opcional) que proporciona una breve descripción del propósito o contenido del repositorio y en Visibilidad, seleccionamos la opción "Privado" para asegurarte de que solo tú y las personas a las que invites puedan acceder al repositorio.

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

Para invitar a alguien a colaborar en un repositorio privado de GitHub, debemos en la esquina superior derecha, hacer clic en la foto de perfil y seleccionar "Tus repositorios". Luego, eligimos el repositorio privado al que deseamos agregar un colaborador.

Dentro del repositorio, hacdemos clic en la pestaña "Configuración" (Settings) ubicada en la parte superior de la página y en el menú lateral izquierdo, dentro de la sección "Administración de acceso" (Access), cliqueamos en "Colaboradores" (Collaborators); luego en el botón "Añadir personas" (Add people); en el campo de búsqueda, escribimos el nombre de usuario, nombre completo o dirección de correo electrónico de la persona que deseamos invitar. Aparecerá una lista de coincidencias; y seleccionamos a la persona correcta.

Después de seleccionar al colaborador, hacemos clic en "Añadir NOMBRE al repositorio" (Add NAME to REPOSITORY).

La persona invitada recibirá un correo electrónico con la invitación. Deberá aceptar la invitación para obtener acceso al repositorio.

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

En GitHub, un repositorio público es un espacio donde puedes almacenar y gestionar el código o proyectos, permitiendo que cualquier persona en internet pueda ver, descargar y contribuir a ellos. Esto es especialmente útil para proyectos de código abierto, ya que facilita la colaboración y el aprendizaje compartido.

- **Accesibilidad:** Cualquier usuario, tenga o no una cuenta en GitHub, puede acceder al contenido del repositorio, visualizar el código, clonar el proyecto y, si lo desea, proponer cambios mediante solicitudes de extracción (pull requests).
- **Colaboración abierta:** Al ser público, otros desarrolladores pueden contribuir al proyecto, reportar problemas (issues), sugerir mejoras y participar activamente en el desarrollo del mismo.
- **Visibilidad:** Los repositorios públicos pueden ser indexados por motores de búsqueda, lo que aumenta la visibilidad del proyecto y puede atraer a más colaboradores y usuarios interesados.

"Es importante tener en cuenta que, al hacer público un repositorio, toda la información contenida en él será accesible para cualquier persona. Por lo tanto, se debe asegurar de no incluir datos sensibles o información privada en estos repositorios."

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

Para crear un repositorio público en GitHub, sigue estos pasos:"

1. Inicia sesión en GitHub:
   *Accede a tu cuenta de GitHub en github.com.*
2. Accede a la opción para crear un nuevo repositorio:
   *En la esquina superior derecha de cualquier página, haz clic en el ícono "+" y selecciona "Nuevo repositorio".*
3. Completa la información del repositorio:

- Nombre del repositorio: *Escribe un nombre único y descriptivo para tu repositorio.*
- Descripción (opcional): *Proporciona una breve descripción del propósito o contenido del repositorio.*
- Visibilidad: *Selecciona la opción "Público" para que cualquier persona pueda ver tu repositorio.*

4. Inicializa el repositorio (opcional):
   *Puedes optar por añadir archivos iniciales como:*
   - README: *Un archivo que describe tu proyecto.*
   - .gitignore: *Especifica archivos que Git debe ignorar.*
   - Licencia: *Define los términos bajo los cuales se distribuye tu código.*
5. Crea el repositorio:
   *Una vez completados los campos anteriores, haz clic en el botón "Crear repositorio".*

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

Para compartir un repositorio público en GitHub, se puede compartir con otros la URL, que podemos extraer en el repositorio que deseamos compartir, haciendo clic en el botón verde "Code" ubicado en la parte superior derecha de la página del repositorio; el que tiene el siguiente formato:

**https://github.com/cuenta-usuario/nombre-del-repositorio.git.**

Una vez copiada la URL del repositorio, podemos compartirla con otros a través de correo electrónico, mensajería instantánea o cualquier otro medio de comunicación.

Al ser un repositorio público, cualquier persona con el enlace podrá acceder al contenido, clonarlo y, si lo desea, contribuir al proyecto mediante solicitudes de extracción (pull requests).

"Recuerdemos que, al ser público, el repositorio es accesible para todos. No debemos incluir información sensible o privada en él."

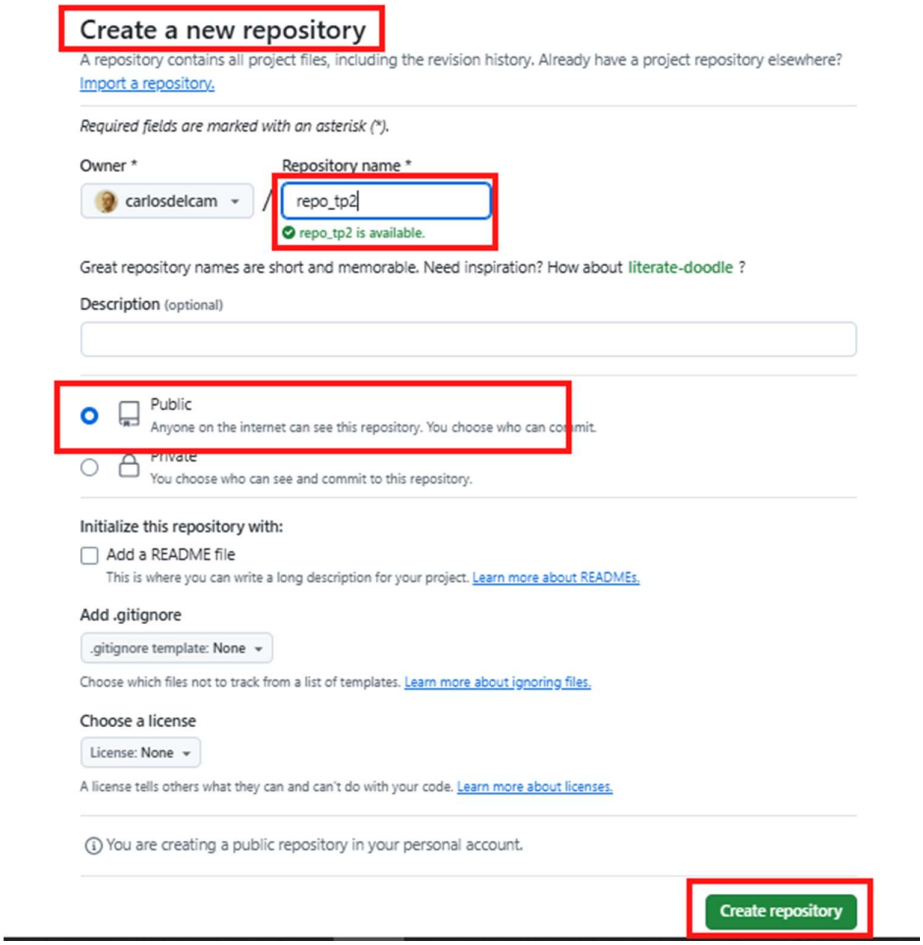
**2. Realizar la siguiente actividad:**

✓ Crear un repositorio, publico

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

1. **Iniciamos sesión en GitHub:** *accede a la cuenta en GitHub.*
2. **Creamos un nuevo repositorio** *en la esquina superior derecha de la página, hacemos clic en el ícono "+" y seleccionamos de la lista deplegable "Nuevo repositorio".*
3. **Completamos la información del repositorio:**
   - **Nombre del repositorio:** *es un nombre único y descriptivo para el repositorio.*
   - **Descripción (opcional):** *Proporciona una breve descripción del propósito o contenido del repositorio.*
   - **Visibilidad:** *Selecciona la opción "Público" para que cualquier persona pueda ver tu repositorio.*
   - **Inicializar el repositorio con un README:** *Marca la casilla "Initialize this repository with a README". Esto creará un archivo README.md que puedes editar posteriormente para describir tu proyecto.*

4. **Hacemos clic en el botón "Crear repositorio".**

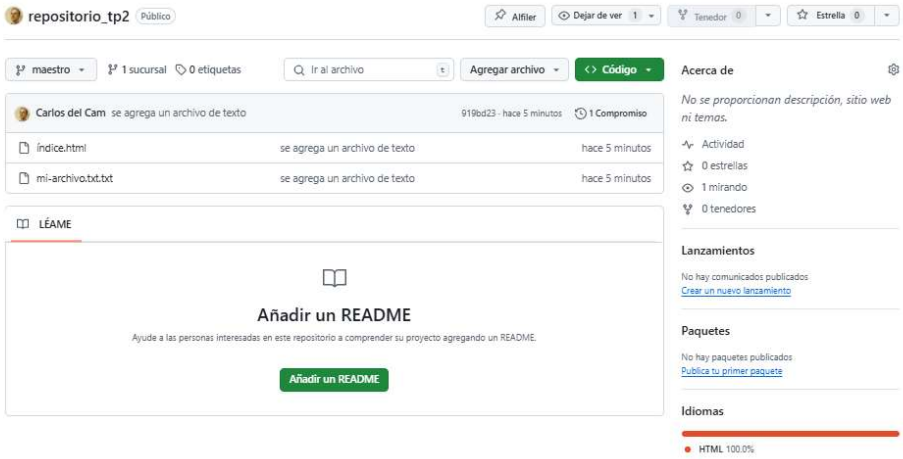



Ahora estamos en condiciones de comenzar a añadir archivos y/o colaborar con otros en el proyecto.

✓ 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).

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git remote add origin https://github.com/carlosdelcam/repo_tp2.git

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 491 bytes | 491.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/carlosdelcam/repo_tp2.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git add .

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git commit -m "se agrega un archivo de texto"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git push -u origin master
branch 'master' set up to track 'origin/master'.
Everything up-to-date

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$
```




✓ Creando Branchs

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

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git branch
* master

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git branch nueva_semana

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git branch
* master
  nueva_semana
```

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git checkout nueva_semana
Switched to branch 'nueva_semana'

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (nueva_semana
)
$ git branch
  master
* nueva_semana
```

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (nueva_semana
)
$ git add .

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (nueva_semana
)
$ git commit -m "se sube nuevo archivo a la rama nueva_semana"
[nueva_semana a5c94d0] se sube nuevo archivo a la rama nueva_semana
 1 file changed, 12 insertions(+)
 create mode 100644 index2.html

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (nueva_semana
)
$ git push -u origin nueva_semana
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 522 bytes | 174.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'nueva_semana' on GitHub by visiting:
remote:      https://github.com/carlosdelcam/repo_tp2/pull/new/nueva_semana
remote:
To https://github.com/carlosdelcam/repo_tp2.git
 * [new branch]      nueva_semana -> nueva_semana
branch 'nueva_semana' set up to track 'origin/nueva_semana'.

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (nueva_semana
)
$ dir
index.html  index2.html  mi-archivo.txt.txt

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (nueva_semana
)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git branch
* master
  nueva_semana

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$ git merge nueva_semana
Updating 919bd23..a5c94d0
Fast-forward
 index2.html | 12 ++++++++++++
 1 file changed, 12 insertions(+)
 create mode 100644 index2.html

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador (master)
$
```

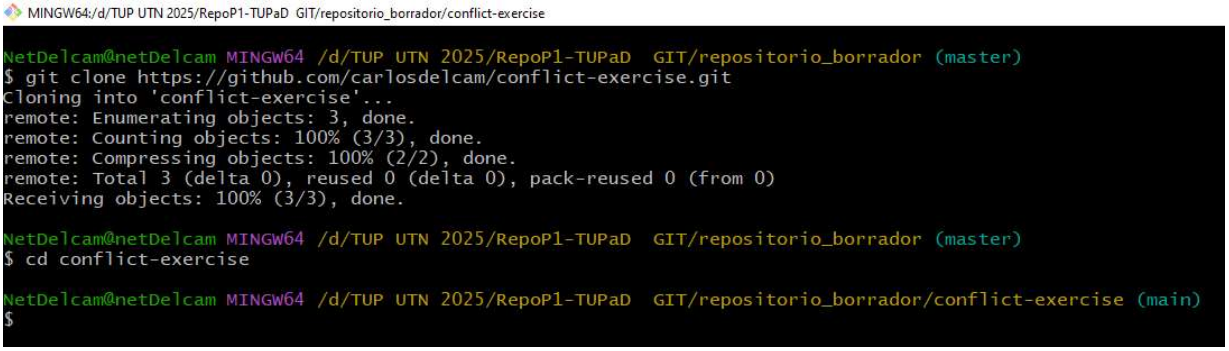
**3) Realizar la siguiente  actividad:**

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".
-

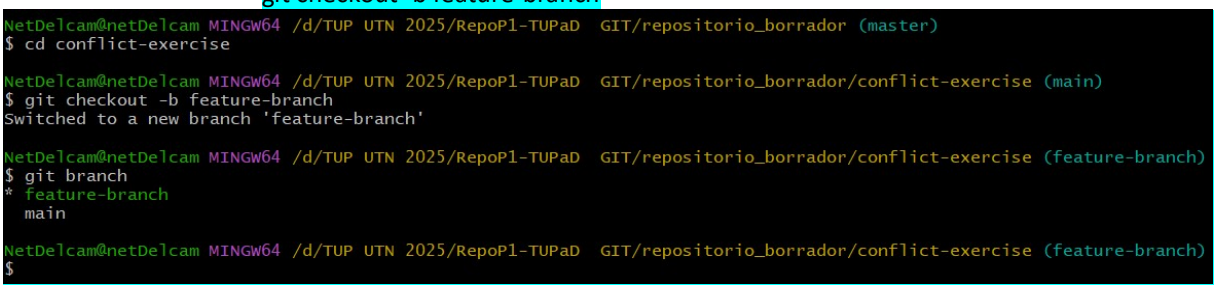
Paso 2: Clonar el repositorio a tu máquina local

- Copia la URL del repositorio:
  https://github.com/carlosdelcam/conflict-exercise
- Abre la terminal o línea de comandos en tu máquina.
- Clona el repositorio usando el comando:
  git clone https://github.com/carlosdelcam/conflict-exercise.git
- Entra en el directorio del repositorio: cd conflict-exercise

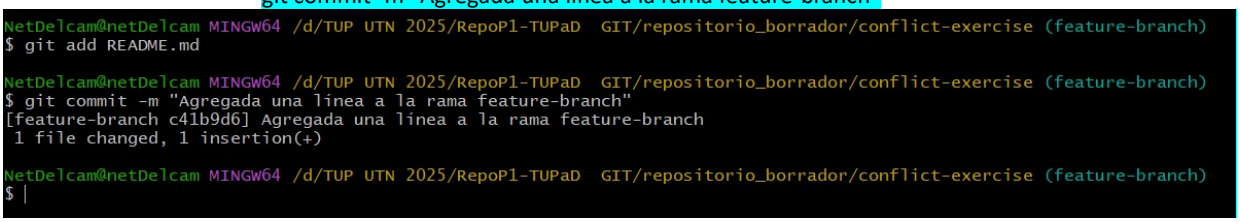



Paso 3: Crear una nueva rama y editar un archivo

- 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 "Agregada una línea a la rama 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 nuevo cambio escrito en Readme y en la rama main.*

- Guarda los cambios y haz un commit:
- git add README.md
- git commit -m "Agregada una nueva línea en el Readme y en la rama main"

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ git add README.md

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ git commit -m "Agregada una nueva linea en el Readme y en la rama main"
[main d106f80] Agregada una nueva linea en el Readme y en la rama main
 1 file changed, 3 insertions(+)

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (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

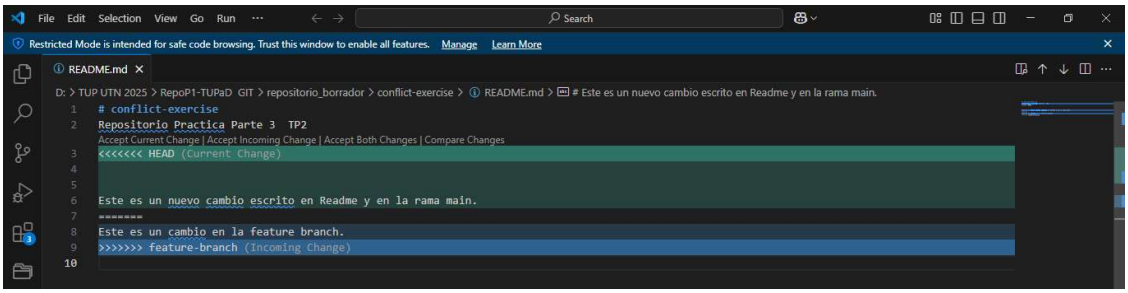
- Se generará un conflicto porque ambos cambios afectan la misma línea del archivo README.md.

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ 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.

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main|MERGING)
$
```

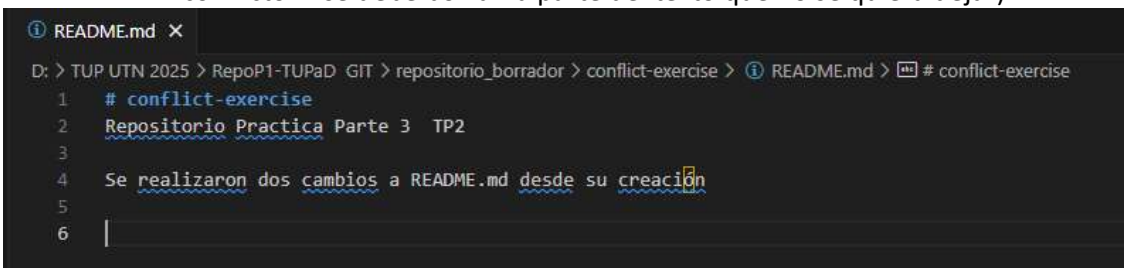
Paso 6: Resolver el conflicto

- 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 estés 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 "Conflicto al realizar el merge resuelto "

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ git add README.md

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ git commit -m "Conflicto al realizar el merge resuelto "
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$
```

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

```
NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ 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), 1020 bytes | 204.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/carlosdelcam/conflict-exercise.git
   04c5073..339ab01  main -> main

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$ 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/carlosdelcam/conflict-exercise/pull/new/feature-branch
remote:
To https://github.com/carlosdelcam/conflict-exercise.git
 * [new branch]      feature-branch -> feature-branch

NetDelcam@netDelcam MINGW64 /d/TUP UTN 2025/RepoP1-TUPaD  GIT/repositorio_borrador/conflict-exercise (main)
$
```

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.