

## TRABAJO PRÁCTICO UNIDAD 2

Alumno: Cheppi nicolas

### 1.¿Qué es GitHub?

GitHub es una plataforma basada en la nube donde puedes almacenar, compartir y trabajar junto con otros usuarios para escribir código.

Almacenar tu código en un "repositorio" en GitHub te permite lo siguiente:

- Presentar o compartir el trabajo.
- Seguir y administrar los cambios en el código a lo largo del tiempo.
- Dejar que otros usuarios revisen el código y realicen sugerencias para mejorarlo.
- Colaborar en un proyecto compartido, sin preocuparse de que los cambios afectarán al trabajo de los colaboradores antes de que esté listo para integrarlos.

### 2.¿Cómo crear un repositorio en GitHub?

Para crear un repositorio en GitHub, inicia sesión en tu cuenta, ve a la página principal y haz clic en "Nuevo repositorio"

### 3.¿Cómo crear una rama en Git?

Para crear una rama en Git se escribe: `git branch "nombre de la rama"`.

### 4.¿Cómo cambiar a una rama en Git?

Para cambiar de rama se escribe `git checkout "nombre de la rama"`.

### 5.¿Cómo fusionar ramas en Git?

Para fusionar las ramas se escribe `git merge "nombre de la rama a fusionar con la rama en la que te encuentras"`.

### 6.¿Cómo crear un commit en Git?

Para crear un commit en Git se escribe `git commit`; se recomienda escribir un mensaje descriptivo poniendo `-m "mensaje descriptivo"`

### 7.¿Cómo enviar un commit a GitHub?

Una vez que tienes commits locales y tu repositorio local está vinculado al remoto en GitHub, puedes enviar tus commits utilizando el comando `git push`.

Bash

`git push origin <nombre_de_la_rama>`

-`git push`: Es el comando para enviar commits a un repositorio remoto.

-`origin`: Es el nombre del remoto al que quieres enviar los commits (generalmente el repositorio en GitHub).

-`<nombre_de_la_rama>`: Es el nombre de la rama local que quieres enviar al remoto. La primera vez que envías una rama nueva, es común usar la opción `-u` o `--set-upstream` para establecer un seguimiento entre tu rama local y la rama remota correspondiente.

## 8.¿Qué es un repositorio remoto?

Un repositorio remoto es una versión de tu proyecto Git que está alojada en un servidor en línea, como GitHub, GitLab, Bitbucket o incluso un servidor privado.

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

Para agregar un repositorio remoto a tu repositorio local de Git, utilizas el comando `git remote add`

Bash

```
git remote add <nombre_del_remoto> <URL_del_repositorio>
```

-<nombre\_del\_remoto>: Es un nombre corto que le darás a la conexión con este repositorio remoto. La convención más común es usar `origin` para el repositorio remoto principal

-<URL\_del\_repositorio>: Es la URL del repositorio remoto. Esta URL puede ser HTTPS o SSH, dependiendo de cómo quieras autenticarte con el servidor remoto. Generalmente, la encontrarás en la página del repositorio en la plataforma que estés utilizando (por ejemplo, el botón "Code" en GitHub).

## 10.¿Cómo empujar cambios a un repositorio remoto?

Para empujar (enviar) tus commits locales a un repositorio remoto, utilizas el comando `git push`.

### Comando básico:

Bash

```
git push <nombre_del_remoto> <nombre_de_la_rama>
```

-<nombre\_del\_remoto>: Es el nombre del repositorio remoto al que quieres enviar tus cambios. La convención más común es `origin`.

-<nombre\_de\_la\_rama>: Es el nombre de la rama local que quieres enviar al repositorio remoto. Por ejemplo, `main`, `master`, `develop`, o alguna rama de funcionalidad que hayas creado.

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

Para traer (tirar) los cambios desde un repositorio remoto a tu repositorio local de Git, utilizas el comando `git pull`. Este comando descarga los commits y los archivos del repositorio remoto y los integra en tu rama local actual.

### Comando básico:

Bash

```
git pull <nombre_del_remoto> <nombre_de_la_rama_remota>
```

-<nombre\_del\_remoto>: Es el nombre del repositorio remoto del que quieres traer los cambios (normalmente `origin`).

-<nombre\_de\_la\_rama\_remota>: Es el nombre de la rama en el repositorio remoto de la que quieres traer los cambios (por ejemplo, `main`, `master`, `develop`).

## 12. ¿Qué es un fork de repositorio?

Un fork de un repositorio es esencialmente una copia personal y separada de un repositorio que reside en la plataforma del proveedor de Git remoto.

## 13. ¿Cómo crear un fork de un repositorio?

Crear un fork de un repositorio es un proceso sencillo que se realiza directamente en la plataforma del proveedor de Git remoto.

En GitHub:

1. Ve al repositorio que deseas forkar. Navega a la página principal del repositorio en GitHub.
2. Busca el botón "Fork". En la esquina superior derecha de la página del repositorio.
3. Haz clic en el botón "Fork". Al hacer clic en este botón, GitHub te redirigirá a una página donde puedes configurar tu fork.
4. Configura tu fork (opcional).
  - Owner: Por defecto, tu fork se creará bajo tu cuenta de usuario. Puedes elegir una organización a la que pertenezcas si tienes permisos para crear repositorios allí.
  - Repository name: Puedes mantener el mismo nombre del repositorio original o cambiarlo si lo deseas. Es recomendable mantenerlo igual para facilitar la comprensión de su origen.
  - Description (opcional): Puedes agregar una descripción para tu fork.
  - Copy only the default branch: Por defecto, GitHub copiará todo el historial de commits y todas las ramas del repositorio original. Si solo te interesa la rama principal actual, puedes marcar esta opción. Sin embargo, generalmente es mejor copiar todo el historial para tener un fork completo.
5. Haz clic en el botón "Create fork". Una vez que hayas configurado tu fork (o simplemente aceptado los valores predeterminados), haz clic en el botón verde que dice "Create fork".

GitHub tardará unos segundos en crear la copia del repositorio bajo tu cuenta. Una vez completado, serás redirigido a la página de tu nuevo fork. La URL de tu fork tendrá la siguiente estructura: [https://github.com/tu\\_usuario/nombre-del-repositorio-original](https://github.com/tu_usuario/nombre-del-repositorio-original).

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

Enviar una solicitud de extracción (Pull Request o PR) es el proceso fundamental para proponer tus cambios desde tu fork de un repositorio (o desde una rama en el mismo repositorio si tienes permisos de escritura) al repositorio principal.

**Flujo de trabajo general:**

1. **Haz un fork del repositorio (si no tienes permisos de escritura):** Si quieres contribuir a un repositorio al que no tienes acceso directo para escribir, primero necesitas crear un fork bajo tu cuenta.
2. **Clona tu fork localmente:** Descarga tu fork a tu computadora usando git clone <URL\_de\_tu\_fork>.
3. **Crea una rama para tus cambios:** Es una buena práctica crear una rama separada para cada conjunto de cambios que quieras proponer. Usa git checkout -b mi-nueva-funcionalidad.
4. **Realiza tus cambios y haz commits:** Edita los archivos, agrega los cambios al área de staging (git add . o archivos específicos), y crea commits descriptivos (git commit -m "Descripción detallada de mis cambios").
5. **Sube tus cambios a tu fork remoto:** Envía tu rama con tus commits a tu repositorio remoto en GitHub (git push origin mi-nueva-funcionalidad).
6. **Ve a la página de tu fork en GitHub:** Abre tu navegador y navega a la página de tu fork en GitHub ([https://github.com/tu\\_usuario/nombre-del-repositorio-original](https://github.com/tu_usuario/nombre-del-repositorio-original)).
7. **Encuentra el botón "Compare & pull request":** GitHub generalmente detecta tu reciente push y te mostrará un botón verde que dice "Compare & pull request". Haz clic en él. Si no lo ves, puedes ir a la pestaña "Pull requests" y hacer clic en el botón verde "New pull request".
8. **Selecciona las ramas para la comparación:**
  - **base repository:** Este es el repositorio principal al que quieres enviar tus cambios (generalmente el repositorio original del que hiciste el fork). Asegúrate de que esté correctamente seleccionado.
  - **base:** Esta es la rama del repositorio base a la que quieres fusionar tus cambios (comúnmente main, master, develop, etc.).
  - **head repository:** Este debería ser tu fork.
  - **compare:** Esta es la rama de tu fork que contiene los cambios que quieres proponer (la rama que subiste en el paso 5, por ejemplo, mi-nueva-funcionalidad). GitHub mostrará una comparación de los cambios entre estas dos ramas.
9. **Escribe una descripción para tu Pull Request:** Proporciona un título claro y conciso para tu Pull Request. En el cuerpo de la descripción, explica detalladamente los cambios que has realizado, por qué son necesarios, y cualquier otra información relevante para los mantenedores del repositorio. Sé claro y conciso.
10. **Revisa los commits y los cambios:** Antes de crear la Pull Request, revisa la lista de commits y los cambios en los archivos para asegurarte de que todo esté correcto.
11. **Haz clic en el botón "Create pull request":** Una vez que estés satisfecho con tu descripción y hayas revisado los cambios, haz clic en el botón verde "Create pull request".

## 15. ¿Cómo aceptar una solicitud de extracción?

Aceptar una solicitud de extracción (Pull Request o PR) es el proceso mediante el cual los mantenedores de un repositorio integran los cambios propuestos en una PR a la rama principal (o a otra rama designada) del repositorio.

## Pasos para aceptar una Pull Request en GitHub:

### 1. Revisa la Pull Request:

- **Lee la descripción:** Comprende el propósito de los cambios propuestos y el problema que intentan resolver.
- **Examina los commits:** Revisa el historial de commits para entender la progresión de los cambios. Los commits deben ser claros y concisos.
- **Analiza los cambios en los archivos:** Revisa cuidadosamente cada cambio realizado en los archivos afectados. Presta atención a la lógica, el estilo de código, las pruebas (si las hay) y cualquier posible efecto secundario.
- **Considera los comentarios y la discusión:** Lee cualquier comentario o discusión que haya tenido lugar en la PR. Aborda cualquier pregunta o inquietud planteada.
- **Verifica si hay conflictos:** GitHub te indicará si hay conflictos de fusión entre la rama de la PR y la rama base. Si hay conflictos, el autor de la PR deberá resolverlos antes de que puedas fusionarla fácilmente.

### 2. Realiza pruebas (si es necesario):

Si los cambios son significativos o afectan la funcionalidad, es recomendable probarlos localmente. Puedes hacer esto de varias maneras:

- **Hacer checkout de la rama de la PR localmente:** GitHub proporciona instrucciones para hacer esto en la página de la PR (generalmente bajo el título "Command line instructions").
- **Fusionar la rama de la PR en una rama local de prueba:** Puedes fusionar la rama remota de la PR en una rama local para probar los cambios sin afectar tu rama principal.

### 3. Decide si aceptas o rechazas la Pull Request:

Basándote en tu revisión y pruebas, decide si los cambios son apropiados para ser integrados en el repositorio.

### 4. Si decides aceptar (fusionar) la Pull Request:

- **Ve a la página de la Pull Request en GitHub.**
- **Busca el botón "Merge pull request".** Este botón suele estar ubicado en la parte inferior de la página de la PR (a veces en la parte superior).
- **Haz clic en el botón "Merge pull request".**
- **Confirma la fusión:** GitHub te presentará algunas opciones de fusión:
  - **Create a merge commit:** Esta es la opción predeterminada y crea un nuevo commit de "fusión" que registra la integración de la rama de la PR. Conserva el historial completo de la rama de la PR.
  - **Squash and merge:** Esta opción combina todos los commits de la rama de la PR en un único nuevo commit en la rama base. Esto puede crear un historial más limpio y lineal, pero pierde el historial individual de los commits de la PR. Es útil para PRs con muchos commits pequeños o "en progreso".

- **Rebase and merge:** Esta opción aplica los commits de la rama de la PR encima de la rama base actual y luego realiza una fusión rápida ("fast-forward"). Esto también crea un historial limpio y lineal, pero puede requerir que el autor de la PR rebase su rama si la rama base ha avanzado significativamente.
- **Elige la opción de fusión deseada y haz clic en el botón de confirmación** (por ejemplo, "Confirm squash and merge").
- **Opcionalmente, elimina la rama de la PR:** Una vez que la PR ha sido fusionada, puedes eliminar la rama de la PR en el repositorio remoto haciendo clic en el botón "Delete branch" que aparece después de la fusión. Esto ayuda a mantener limpio el listado de ramas.

16.¿Qué es un etiqueta en Git?

En Git, una **etiqueta (tag)** es una forma de marcar un punto específico en la historia de tu repositorio como importante. Piensa en ellas como marcadores permanentes que asignas a un commit en particular. A diferencia de las ramas, que están diseñadas para moverse a medida que se agregan nuevos commits, las etiquetas suelen apuntar a un commit específico y se espera que permanezcan allí.

17.¿Cómo crear una etiqueta en Git?

dos maneras principales: etiquetas ligeras y etiquetas anotadas.

### 1. Etiquetas Ligeras (Lightweight Tags)

Las etiquetas ligeras son como una rama que nunca se mueve. Son simplemente un puntero a un commit específico.

Para crear una etiqueta ligera, usa el siguiente comando:

```
Bash
git tag <nombre_de_la_etiqueta>
```

#### Características de las etiquetas ligeras:

- Simples y fáciles de crear.
- No almacenan ninguna información adicional como el autor o la fecha.
- Son solo una referencia al commit.

### 2. Etiquetas Anotadas (Annotated Tags)

Las etiquetas anotadas son objetos completos en la base de datos de Git. Almacenan información adicional como el nombre del etiquetador, el correo electrónico, la fecha y un mensaje. Se recomienda usar etiquetas anotadas para lanzamientos públicos.

Para crear una etiqueta anotada, usa el siguiente comando con la opción -a (annotated) y -m (message):

```
Bash
git tag -a <nombre_de_la_etiqueta> -m "<mensaje_de_la_etiqueta>"
```

### Características de las etiquetas anotadas:

- Almacenan metadatos importantes sobre la etiqueta.
- Se recomienda para lanzamientos y puntos importantes.
- Se pueden verificar criptográficamente (si están firmadas con GPG).

### 18.¿Cómo enviar una etiqueta a GitHub?

Para enviar una etiqueta que has creado localmente a tu repositorio en GitHub, necesitas usar el comando `git push` con algunas opciones específicas. Aquí te muestro las dos formas principales de hacerlo:

#### 1. Enviar una etiqueta específica:

Si solo quieres enviar una etiqueta en particular, utiliza el siguiente comando:

```
Bash
git push origin <nombre_de_la_etiqueta>
```

#### 2. Enviar todas las etiquetas:

Si has creado varias etiquetas localmente y quieres enviarlas todas a GitHub de una vez, puedes usar la opción `-tags` con el comando `git push`:

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

Este comando tomará todas las etiquetas que existen en tu repositorio local y las subirá al repositorio remoto origin.

### 19.¿Qué es un historial de Git?

es un registro cronológico y detallado de todos los cambios que se han realizado en un repositorio a lo largo del tiempo. Imagínalo como un libro de contabilidad completo de tu proyecto, donde cada entrada (commit) representa una modificación específica.

### ¿Qué información contiene el historial de Git?

Cada entrada en el historial de Git, conocida como **commit**, contiene la siguiente información esencial:

- **SHA-1 Hash:** Un código alfanumérico único de 40 caracteres que identifica de forma inequívoca cada commit. Es como la "huella digital" del commit.
- **Autor:** El nombre y la dirección de correo electrónico de la persona que realizó el commit.
- **Fecha y Hora:** El momento exacto en que se realizó el commit.
- **Mensaje del Commit:** Una breve descripción del cambio realizado en ese commit. Es crucial escribir mensajes claros y concisos para entender el propósito de cada modificación.
- **Padre(s):** Una referencia al commit o commits que precedieron a este commit. Para commits lineales, generalmente hay un solo padre. En el caso de merges (cuando se combinan ramas), un commit puede tener múltiples padres.

- **Los cambios reales:** Aunque no se muestran directamente en la vista resumida del historial, cada commit almacena las diferencias exactas entre el estado del repositorio antes y después de la modificación.

20.¿Cómo ver el historial de Git?

El comando principal para ver el historial de Git es git log.

### 1. Historial Básico:

Simplemente ejecutar git log en tu terminal mostrará el historial de commits en orden cronológico inverso (el commit más reciente aparece primero). Para cada commit, verás:

- El SHA-1 hash completo del commit.
- El nombre del autor y su dirección de correo electrónico.
- La fecha y hora en que se realizó el commit.
- El mensaje del commit.

```
Bash
git log
```

### 2. Historial en una Línea:

Para una vista más compacta, donde cada commit se muestra en una sola línea, puedes usar la opción --oneline:

```
Bash
git log --oneline
```

Esto muestra el SHA-1 hash abreviado y el mensaje del commit.

### 3. Historial con Gráfico de Ramas y Merges:

Si tu repositorio tiene múltiples ramas y merges, puede ser útil ver una representación gráfica del historial. Usa la opción --graph:

```
Bash
git log --graph
```

Puedes combinarlo con --oneline y --decorate para obtener una vista más informativa:

```
Bash
git log --all --graph --oneline --decorate
```

--all: Muestra commits de todas las ramas.

--graph: Dibuja un gráfico de la historia de los commits a la izquierda de los mensajes del commit.

--oneline: Muestra cada commit en una sola línea.

--decorate: Muestra las referencias (ramas, etiquetas, HEAD, etc.) que apuntan a los commits.



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

Varias formas de hacerlo con el comando `git log`:

### 1. Buscar por Mensaje del Commit:

La forma más común es buscar commits que contengan ciertas palabras o frases en su mensaje. Usa la opción `--grep`:

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

La opción `--regex-ignore-case` puede ser útil para hacer la búsqueda insensible a mayúsculas y minúsculas:

```
Bash
git log --grep="error" --regex-ignore-case
```

### 2. Buscar por Autor:

Puedes filtrar commits por el autor utilizando la opción `--author`:

```
Bash
git log --author="Nombre del Autor"
```

También puedes usar expresiones regulares para buscar patrones en los nombres de los autores.

### 3. Buscar por Committer:

El committer es la persona que realmente aplicó el commit. En muchos casos es la misma que el autor, pero puede ser diferente (por ejemplo, en el caso de aplicar un parche enviado por otra persona). Usa la opción `--committer`:

```
Bash
git log --committer="Nombre del Committer"
```

### 4. Buscar por Fechas:

Ya vimos cómo filtrar por rangos de fechas con `--since` y `--until`. Puedes combinarlos con otras opciones de búsqueda:

```
Bash
git log --since="ayer" --until="hoy" --author="Usuario Específico"
```

### 5. Buscar por Contenido del Commit (Diff):

Si quieres encontrar commits que introdujeron o eliminaron una línea de código específica, puedes usar la opción `-S` (con mayúscula). Esto buscará commits cuya diferencia introduce o elimina líneas que coinciden con la cadena especificada:

```
Bash
git log -S"función importante()"
```

## 6. Combinar Criterios de Búsqueda:

22.¿Cómo borrar el historial de Git?

### Escenario 1: Eliminar el historial de commits locales (sin afectar el remoto)

Si quieres limpiar tu historial local pero aún no has subido tus cambios a un repositorio remoto, puedes usar git reset. **¡Ten mucho cuidado con este comando, especialmente con la opción --hard!**

**git reset --soft <commit>**: Mueve el puntero HEAD y la rama actual al commit especificado, pero deja los cambios en el área de staging y en tu directorio de trabajo. Es como si hubieras deshecho los commits, pero tus cambios siguen ahí listos para ser recommiteados.

Bash

```
git reset --soft <SHA-1 del commit al que quieres volver>
```

**git reset --mixed <commit> (opción por defecto)**: Similar a --soft, pero además de mover los punteros, también quita los cambios del área de staging. Los cambios permanecen en tu directorio de trabajo, pero como cambios no rastreados.

Bash

```
git reset --mixed <SHA-1 del commit al que quieres volver>
```

**git reset --hard <commit>**: **¡CUIDADO!** Esta opción mueve los punteros y además elimina todos los cambios en el área de staging y en tu directorio de trabajo desde el commit especificado. **Perderás datos si no los has commiteado o staged.**

Bash

```
git reset --hard <SHA-1 del commit al que quieres volver>
```

### Escenario 2: Eliminar commits del historial remoto (reescribiendo el historial)

**Esta es la opción más peligrosa y solo se recomienda en circunstancias muy específicas, como cuando has subido información sensible por error y nadie más ha basado su trabajo en esos commits.** Reescribir el historial remoto puede causar problemas graves a otros colaboradores.

El comando principal para reescribir el historial es git push --force-with-lease (o simplemente git push --force, aunque la primera es más segura). Antes de usarlo, necesitas manipular tu historial local.

Un método común es usar git rebase -i (interactive rebase):

#### 1.Inicia el rebase interactivo:

Bash

```
git rebase -i <commit_padre_del_primer_commit_que_quieres_modificar>
```

Por ejemplo, si quieres modificar los últimos 3 commits, podrías usar git rebase -i HEAD~3

**2. Edita el archivo que se abre:** Aparecerá un archivo en tu editor de texto con una lista de tus commits. Para eliminar una línea (y por o tanto, el commit), simplemente **borra la línea completa** del commit que quieres eliminar. Guarda y cierra el archivo.

**3. Resuelve posibles conflictos:** Si la eliminación de commits causa conflictos, Git te pedirá que los resuelvas.

**4. Fuerza la subida de los cambios (¡con precaución!):**

Bash

```
git push origin --force-with-lease <tu_rama>
```

Reemplaza <tu\_rama> con el nombre de tu rama (por ejemplo, main o master).

--force-with-lease es una opción más segura que --force ya que verifica que tu rama remota coincida con lo que esperas antes de forzar la subida.

**Alternativas a borrar el historial:**

En muchos casos, en lugar de borrar el historial, existen alternativas más seguras:

**-git revert <commit>:** Crea un nuevo commit que deshace los cambios introducidos por el commit especificado. Esto mantiene el historial intacto y es una forma segura de deshacer cambios ya publicados.

**-git cherry-pick <commit>:** Aplica los cambios de un commit específico a tu rama actual. Puedes usar esto para seleccionar solo los cambios que quieres conservar.

**En resumen:**

**-Para cambios locales no subidos:** Usa git reset con cuidado.

**-Para cambios remotos (¡con extrema precaución!):** Usa git rebase -i seguido de git push --force-with-lease. **Entiende los riesgos.**

**-Alternativas más seguras:** Considera git revert o git cherry-pick

23. ¿Qué es un repositorio privado en GitHub?

Un **repositorio privado en GitHub** es un espacio para alojar tu código y archivos que **solo es visible para ti y las personas a las que explícitamente les das acceso**. Es lo opuesto a un repositorio público, que es visible para cualquier persona en internet.

**Características Clave:**

**-Visibilidad Restringida:** Solo tú (el propietario) y los colaboradores que invites pueden ver el contenido del repositorio, incluyendo el código, los issues, los pull requests, los wikis y la configuración.

**-Control de Acceso Granular:** Puedes invitar a colaboradores con diferentes niveles de permiso (por ejemplo, solo lectura, escritura, o administración).

**-Ideal para Información Sensible:** Es la opción adecuada para proyectos que contienen código propietario, información confidencial, datos privados o cualquier cosa que no quieras hacer pública.

**-Colaboración Segura:** Permite trabajar en equipo en proyectos privados sin exponer el código al público.

**-Funcionalidades Completas:** Los repositorios privados disfrutan de todas las funcionalidades de GitHub, como el control de versiones con Git, el seguimiento de issues, las pull requests, las Actions, los Projects, etc.

24.¿Cómo crear un repositorio privado en GitHub?

**Desde la interfaz web de GitHub (github.com):**

1. **Inicia sesión en tu cuenta de GitHub.**
2. **Ve a la página de tus repositorios.** Puedes hacerlo de varias maneras:
  - Haz clic en el botón **"+" (Crear nuevo...)** en la esquina superior derecha de cualquier página y selecciona **"Nuevo repositorio"**.
  - En tu página de perfil, busca la pestaña **"Repositorios"** y luego haz clic en el botón verde **"Nuevo"**.
3. **Completa la información del nuevo repositorio:**
  - **Nombre del repositorio:** Elige un nombre descriptivo para tu proyecto.
  - **Descripción (opcional):** Puedes añadir una breve descripción de tu repositorio.
  - **Privacidad: Aquí es donde eliges la privacidad.** Selecciona la opción **"Privado"**. Verás una indicación de que solo tú y las personas con las que lo compartas podrán ver este repositorio.
  - **Inicializar este repositorio con:** Opcionalmente, puedes marcar las casillas para inicializar tu repositorio con:
    - **Un archivo README:** Es una buena práctica incluir un archivo README con información sobre tu proyecto.
    - **.gitignore:** Selecciona una plantilla para ignorar archivos específicos según el lenguaje de programación o el entorno de desarrollo que utilices.
    - **Una licencia:** Elige una licencia de código abierto si planeas hacerlo público en el futuro o si aplica a tu proyecto. Para un repositorio privado inicial, esto es opcional.
4. **Haz clic en el botón verde "Crear repositorio".**

25.¿Cómo invitar a alguien a un repositorio privado en GitHub?

- 1.**Ve a tu repositorio privado en GitHub.** Abre tu navegador y navega hasta la página de tu repositorio privado.
- 2.**Haz clic en la pestaña "Settings".** Esta pestaña se encuentra en la barra de menú superior de la página de tu repositorio, generalmente a la derecha de "Code", "Issues", "Pull requests", etc.
- 3.**En la barra lateral izquierda, haz clic en "Collaborators".** Dependiendo de la organización a la que pertenezca el repositorio, es posible que veas una sección llamada "Manage access" y dentro de ella la opción "Collaborators".
- 4.**Haz clic en el botón verde "Add people".** Este botón se encuentra en la parte superior de la sección "Collaborators".
- 5.**Busca a la persona que quieres invitar.** En el campo de búsqueda que aparece, puedes escribir el nombre de usuario de GitHub, el nombre completo o la dirección de correo

electrónico de la persona que deseas invitar. GitHub te mostrará sugerencias a medida que escribes.

**6. Selecciona a la persona correcta de los resultados de búsqueda.** Haz clic en el nombre de la persona que quieres invitar.

**7. Revisa los permisos (opcional).** Antes de añadir a la persona, GitHub te mostrará el nivel de permiso que se le otorgará por defecto (generalmente "Write"). Si necesitas otorgar permisos diferentes (por ejemplo, solo lectura), puedes hacerlo en este paso (aunque a menudo se gestionan permisos más granulares a través de equipos en organizaciones).

**8. Haz clic en el botón verde "Add <nombre\_de\_usuario> to this repository".** Esto enviará una invitación a la persona que seleccionaste.

26. ¿Qué es un repositorio público en GitHub?

Un **repositorio público en GitHub** es un espacio para alojar tu código y archivos que es **visible y accesible para cualquier persona en internet**. Es la forma predeterminada y la esencia de la colaboración abierta en la plataforma.

#### **Características Clave:**

**-Visibilidad Pública:** Cualquier persona puede ver el contenido del repositorio, incluyendo el código fuente, el historial de commits, los issues, los pull requests, los wikis y la configuración. No se requiere tener una cuenta de GitHub para verlos.

**-Lectura Universal:** Cualquier usuario de GitHub (o incluso sin cuenta) puede clonar (copiar) el repositorio a su máquina local para examinar el código.

**-Colaboración Abierta:** Otros desarrolladores pueden proponer cambios a través de pull requests (solicitudes de fusión). El propietario del repositorio tiene la decisión de aceptar o rechazar estas contribuciones.

**-Forking:** Otros usuarios pueden "forkear" (bifurcar) tu repositorio, creando una copia completa en su propia cuenta. Esto les permite realizar cambios de forma independiente sin afectar tu repositorio original. Si lo desean, pueden luego enviar pull requests con sus modificaciones.

**-Issues Públicos:** El sistema de seguimiento de issues permite a cualquier persona reportar errores, sugerir mejoras o iniciar discusiones sobre el proyecto.

**-Transparencia:** Todo el desarrollo y la discusión son transparentes y accesibles para la comunidad.

**-Ideal para Código de Fuente Abierta:** Es la opción predeterminada y recomendada para proyectos de software de código abierto (open source), donde se fomenta la colaboración y la transparencia.

**-Portafolio Público:** Los repositorios públicos pueden servir como un portafolio para mostrar tus habilidades y proyectos a otros desarrolladores y potenciales empleadores.

27. ¿Cómo crear un repositorio público en GitHub?

**Desde la interfaz web de GitHub (github.com):**

**1-Inicia sesión en tu cuenta de GitHub. Ve a la página de tus repositorios.** Puedes hacerlo de varias maneras:

-Haz clic en el botón "+" (**Crear nuevo...**) en la esquina superior derecha de cualquier página y selecciona **"Nuevo repositorio"**.

-En tu página de perfil, busca la pestaña "**Repositorios**" y luego haz clic en el botón verde "**Nuevo**".

## 2-Completa la información del nuevo repositorio:

-**Nombre del repositorio:** Elige un nombre descriptivo para tu proyecto.

-**Descripción (opcional):** Puedes añadir una breve descripción de tu repositorio.

-**Privacidad: Aquí es donde eliges la privacidad.** Asegúrate de que la opción "**Público**" esté seleccionada. Verás una indicación de que cualquier persona en internet podrá ver este repositorio.

-**Inicializar este repositorio con:** Opcionalmente, puedes marcar las casillas para inicializar tu repositorio con:

-**Un archivo README:** Es una buena práctica incluir un archivo README con información sobre tu proyecto.

-**.gitignore:** Selecciona una plantilla para ignorar archivos específicos según el lenguaje de programación o el entorno de desarrollo que utilices.

-**Una licencia:** Elige una licencia de código abierto para definir cómo otros pueden usar tu código. Esto es especialmente importante para proyectos públicos.

## 3-Haz clic en el botón verde "Crear repositorio".

28.¿Cómo compartir un repositorio público en GitHub?

La forma más directa y común de compartir tu repositorio público:

- **Desde la página del repositorio en GitHub:**

1. Ve a la página de tu repositorio público en <https://github.com/>.
2. En la parte superior de la página, debajo del nombre del repositorio, verás la **URL del repositorio**. Generalmente tiene el formato `https://github.com/<tu_nombre_de_usuario>/<nombre_del_repositorio>`.
3. **Copia esta URL** y compártela por el medio que prefieras: correo electrónico, redes sociales, mensajes, documentación, etc.

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

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
```

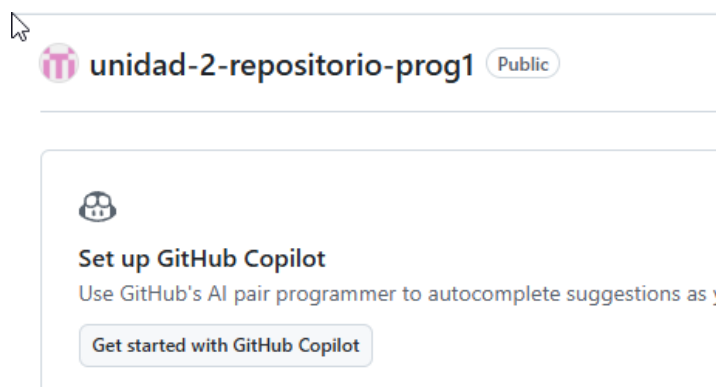
- También sube la feature-branch si deseas:

```
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.

Resolución:



The screenshot shows the GitHub interface for a repository named 'unidad-2-repositorio-prog1', which is marked as 'Public'. Below the repository name, there is a section for 'Set up GitHub Copilot', which includes the text 'Use GitHub's AI pair programmer to autocomplete suggestions as you type' and a button labeled 'Get started with GitHub Copilot'.

```
Nicolás@DESKTOP-6E07K3D MINGW64 ~/OneDrive/Desktop/progra (master)
$ git add miarchivo.txt
```

```
Nicolás@DESKTOP-6E07K3D MINGW64 ~/OneDrive/Desktop/progra (master)
$ git remote add origin https://github.com/yunga44/unidad-2-repositorio-progl.git
```

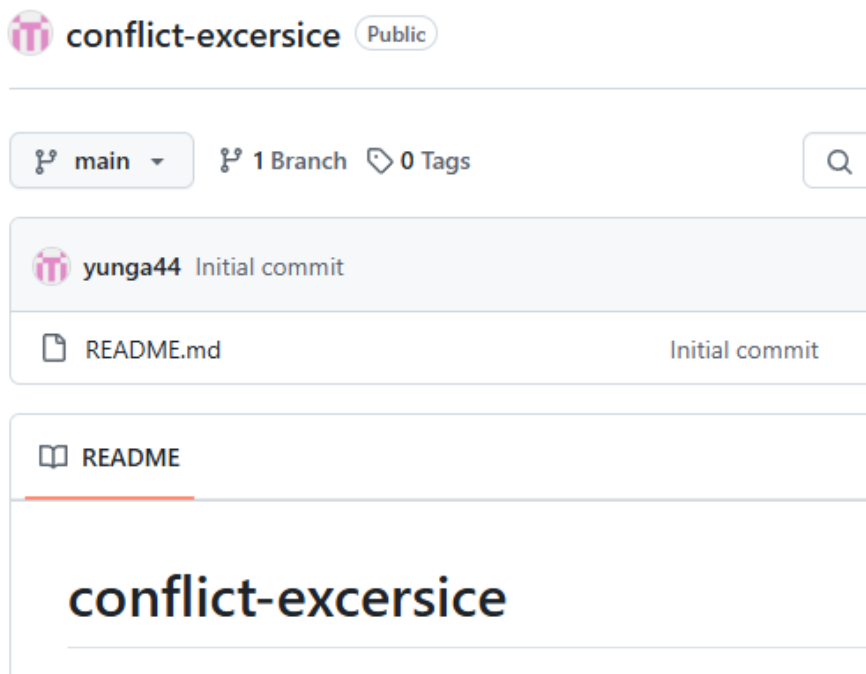
```
Nicolás@DESKTOP-6E07K3D MINGW64 ~/OneDrive/Desktop/progra (master)
$ git push -u origin master
fatal: the origin does not appear to be a git repository
```

```
Nicolás@DESKTOP-6E07K3D MINGW64 ~/OneDrive/Desktop/progra (master)
$ git branch nuevarama
```

Pase a trabajar en VSC

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra> git checkout nuevarama
Switched to branch 'nuevarama'
```

3-



```
PS C:\Users\Nicolás\OneDrive\Desktop\progra> git clone https://github.com/yunga44/conflict-excersice.git
Cloning into 'conflict-excersice'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
PS C:\Users\Nicolás\OneDrive\Desktop\progra>
```

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git checkout -b feature-branch
Switched to a new branch 'feature-branch'
```

```
fatal: pathspec 'feature-branch' did not match any files
```

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra> cd conflict-excersice
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git add README.md
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>
```

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git commit -m "Linea agregada en conflict-excersice"
[main ff8eb27] Linea agregada en conflict-excersice
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>
```

#### Paso 4

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git checkout main
Already on 'main'
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>
```

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git add README.md
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git commit -m "Added a line in main branch"
[main 599e64b] Added a line in main branch
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>
```

#### Paso 5

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git merge feature-branch
merge: feature-branch - not something we can merge
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>
```

#### Paso 6

```
1 # conflict-excersice
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2 <<<<<< HEAD (Current Change)
3 cambio en main
4
5 =====
6 cambio feature branchhh
7 >>>>>> feature-branch (Incoming Change)
8
```

#### Paso 7

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
cambio en main y feature
=====
>>>>>> feature-branch (Incoming Change)
```

```
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git add README.md
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git commit -m "Resolved merge conflict in README.md"
[main b8a41f0] Resolved merge conflict in README.md
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>
```

```

PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> git push origin main
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (21/21), 1.79 KiB | 366.00 KiB/s, done.
Total 21 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/yunga44/conflict-excersice.git
   9c2627a..b8a41f0  main -> main
PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice>

PS C:\Users\Nicolás\OneDrive\Desktop\progra\conflict-excersice> 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/yunga44/conflict-excersice/pull/new/feature-branch
remote:
To https://github.com/yunga44/conflict-excersice.git
 * [new branch]      feature-branch -> feature-branch

```

## conflict-excersice

---

<<<<<< HEAD cambio en main y feature

=====

| | | | | | | feature-branch