TRABAJO PRACTICO NRO. 2

GIT Y GITHUB

Alumno: Thomas Reynoso

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

Aquí tienes las respuestas desarrolladas para cada pregunta sobre GitHub y Git:

• ¿Qué es GitHub?

GitHub es una **plataforma de desarrollo colaborativo basada en la web** que utiliza el sistema de control de versiones **Git**.[1] Funciona principalmente como un servicio de **alojamiento para repositorios Git**, pero va mucho más allá, ofreciendo herramientas para facilitar la colaboración, la revisión de código, el seguimiento de errores y tareas, la automatización de flujos de trabajo (CI/CD) y la publicación de documentación o sitios web sencillos.

En esencia, permite a los desarrolladores:

1. **Almacenar su código** de forma segura en la nube (en repositorios).
2. **Controlar las versiones** de su código usando Git, pudiendo rastrear cambios, revertir a versiones anteriores y explorar el historial del proyecto.
3. **Colaborar con otros** desarrolladores en el mismo proyecto, ya sea de forma pública (open source) o privada. Facilita la revisión de código mediante "Pull Requests" y la discusión mediante "Issues".
4. **Mostrar su trabajo**, ya que los perfiles de GitHub a menudo sirven como portafolios para los desarrolladores.
5. **Automatizar tareas** como pruebas y despliegues con GitHub Actions.

Es propiedad de Microsoft y es una de las plataformas más populares del mundo para el desarrollo de software, tanto para proyectos personales como para grandes empresas y la comunidad de código abierto.

• ¿Cómo crear un repositorio en GitHub?

Crear un repositorio en GitHub es un proceso que se realiza a través de su interfaz

web:

1. **Inicia sesión en GitHub:** Accede a tu cuenta en github.com.
2. **Haz clic en el botón "+" o "New":** En la esquina superior derecha de la página, encontrarás un icono "+" o un botón "New". Haz clic en él.
3. **Selecciona "New repository":** En el menú desplegable, elige la opción "New repository" (Nuevo repositorio).
4. **Completa el formulario:**
    - **Owner/Propietario:** Asegúrate de que esté seleccionado tu usuario (o una organización a la que pertenezcas, si aplica).
    - **Repository name/Nombre del repositorio:** Elige un nombre único y descriptivo para tu repositorio (ej. `mi-proyecto-web`).
    - **Description/Descripción (Opcional):** Añade una breve descripción de qué trata el repositorio.
    - **Public/Private:** Decide si el repositorio será visible para cualquiera (Público) o solo para ti y las personas a las que des acceso (Privado).
    - **Initialize this repository with:/Inicializar este repositorio con: (Opcional pero recomendado para empezar):**
        - **Add a README file:** Crea un archivo `README.md` inicial donde puedes describir tu proyecto más detalladamente.
        - **Add .gitignore:** Selecciona una plantilla de `.gitignore` según el lenguaje o framework de tu proyecto. Este archivo le dice a Git qué archivos o carpetas ignorar (no incluir en el control de versiones).
        - **Choose a license:** Selecciona una licencia de código abierto si tu proyecto es público y quieres definir cómo otros pueden usarlo.
5. **Haz clic en "Create repository":** Una vez completado el formulario, presiona el botón verde "Create repository" (Crear repositorio).

¡Listo! Tu repositorio ha sido creado en GitHub y puedes empezar a trabajar con él.

• ¿Cómo crear una rama en Git?

Una rama (branch) en Git es una línea independiente de desarrollo. Permite trabajar en nuevas características o correcciones de errores sin afectar la línea principal (comúnmente llamada `main` o `master`). Se crea usando comandos de Git en tu terminal local:

Método 1: Crear la rama sin cambiar a ella:

Bash

```
git branch <nombre-nueva-rama>
```

- **Ejemplo:** `git branch feature-login`
- **Explicación:** Este comando crea una nueva rama llamada `feature-login` que apunta al mismo commit en el que te encuentras actualmente. No te cambia automáticamente a la nueva rama.

Método 2: Crear la rama y cambiar a ella en un solo paso (recomendado):

Usando `git checkout -b` (más antiguo):

```
Bash

    git checkout -b <nombre-nueva-rama>
```

- **Ejemplo:** `git checkout -b fix-bug-123`

Usando `git switch -c` (más moderno, introducido para separar funcionalidades de `checkout`):

```
Bash

    git switch -c <nombre-nueva-rama>
```

- **Ejemplo:** `git switch -c documentation-update`

- **Explicación:** Ambos comandos (`checkout -b` y `switch -c`) crean la nueva rama especificada y, acto seguido, te cambian a ella para que puedas empezar a trabajar directamente en esa nueva línea de desarrollo.

• ¿Cómo cambiar a una rama en Git?

Si una rama ya existe y quieres empezar a trabajar en ella o ver su contenido, necesitas cambiar tu "posición" actual (HEAD) a esa rama.

Método 1: Usando `git checkout` (más antiguo):

```
Bash

    git checkout <nombre-rama-existente>
```

- **Ejemplo:** `git checkout main`
- **Explicación:** Este comando mueve el HEAD a la rama especificada y actualiza los archivos de tu directorio de trabajo para que coincidan con el último commit de esa rama.

Método 2: Usando `git switch` (más moderno):

```
Bash

    git switch <nombre-rama-existente>

```

- **Ejemplo:** `git switch feature-login`
- **Explicación:** Funciona igual que `git checkout` para cambiar entre ramas existentes, pero su nombre es más específico para esta acción.

Importante: Antes de cambiar de rama, es recomendable que no tengas cambios sin confirmar (`commit`) en tu directorio de trabajo, o que uses `git stash` para guardarlos temporalmente, para evitar conflictos o perder trabajo.

• ¿Cómo fusionar ramas en Git?

Fusionar (merge) es el proceso de integrar los cambios de una rama en otra. Por ejemplo, una vez que terminas de trabajar en una rama de característica (`feature-branch`), querrás fusionar esos cambios de vuelta a la rama principal (`main`).

El proceso general es:

1. **Cambia a la rama receptora:** Primero, sitúate en la rama que va a *recibir* los cambios. Generalmente, esta es la rama principal.
2. ```
   Bash

       git switch main # O git checkout main

   ```
3.
4.
5. **Ejecuta el comando `git merge`:** Indica la rama cuyos cambios quieres incorporar.
6. ```
   Bash

       git merge <nombre-rama-a-fusionar>

   ```
7.

- **Ejemplo:** `git merge feature-login`

¿Qué puede pasar durante la fusión?

- **Fast-forward merge:** Si la rama receptora (`main`) no ha tenido nuevos commits desde que se creó la rama a fusionar (`feature-login`), Git simplemente moverá el puntero de `main` hacia adelante para que apunte al último commit de `feature-login`. Es una fusión limpia y directa.
- **Merge commit:** Si ambas ramas han tenido commits desde que se separaron, Git creará un nuevo commit especial llamado "merge commit". Este commit tiene dos "padres" (el último commit de cada rama) y une los historiales de ambas. Git intentará combinar los cambios automáticamente.
- **Conflictos:** Si Git detecta que ambas ramas modificaron la misma parte del mismo archivo de formas diferentes, no podrá resolverlo automáticamente y marcará un **conflicto**. Deberás editar manualmente los archivos conflictivos para decidir cómo combinar los cambios, luego añadir los archivos resueltos (`git add .`) y finalizar la fusión con `git commit`.

• ¿Cómo crear un commit en Git?

Un "commit" es una instantánea (snapshot) de los cambios de tu proyecto guardada en el historial del repositorio local. Crear un commit es un proceso de dos pasos:

1. **Preparar los cambios (Staging):** Primero, debes indicar a Git qué cambios específicos quieres incluir en el próximo commit. Esto se hace con el comando `git add`.
    - Para añadir un archivo específico: `git add nombre_del_archivo.txt`
    - Para añadir todos los cambios en el directorio actual (archivos nuevos, modificados, eliminados): `git add .`
2. **Crear el commit:** Una vez que los cambios están en el área de "staging", usas el comando `git commit` para guardarlos permanentemente en el historial local. Es **fundamental** incluir un mensaje descriptivo que explique *qué* cambios hiciste.
3. Bash

    ```
    git commit -m "Tu mensaje descriptivo aquí"
    ```

4.
    - **Ejemplo:** `git commit -m "Añadir función de login de usuario"`
    - **Explicación:** La opción `-m` permite escribir el mensaje directamente en la línea de comandos. Si omites `-m`, Git abrirá un editor de texto para que escribas un mensaje más largo.

Un buen commit representa una unidad lógica de trabajo.

• ¿Cómo enviar un commit a GitHub?

Enviar un commit (o una serie de commits) a GitHub significa subir los cambios que has guardado en tu repositorio *local* al repositorio *remoto* alojado en GitHub. Esto se hace con el comando `git push`.

Pre-requisitos:

- Debes haber creado commits localmente (`git commit`).
- Tu repositorio local debe estar conectado a un repositorio remoto en GitHub (normalmente llamado `origin`). Si clonaste el repositorio desde GitHub, esto ya está configurado. Si lo creaste localmente, necesitas añadir el remoto (`git remote add origin <URL>`).

Comando:

```
Bash

    git push <nombre-remoto> <nombre-rama-local>

```

- `<nombre-remoto>`**:** Es el alias del repositorio remoto. Por convención, suele ser `origin`.
- `<nombre-rama-local>`**:** Es el nombre de la rama en tu repositorio local cuyos commits quieres enviar.

Ejemplo común:

```
Bash

    git push origin main

```

- **Explicación:** Este comando envía todos los commits que están en tu rama local `main` pero que aún no están en la rama `main` del repositorio remoto `origin` (en GitHub).

Si estás empujando una rama por primera vez, Git podría sugerirte un comando más largo como `git push --set-upstream origin <nombre-rama-local>` para establecer la relación de seguimiento entre tu rama local y la remota.

• ¿Qué es un repositorio remoto?

Un repositorio remoto es una versión de tu repositorio Git que está **alojada en un servidor diferente a tu máquina local**. Generalmente se encuentra en internet (como en GitHub, GitLab, Bitbucket) o en un servidor de red interno.

Sus propósitos principales son:

1. **Colaboración:** Sirve como un punto central donde varios desarrolladores pueden compartir sus cambios y obtener los cambios de los demás.
2. **Respaldo (Backup):** Mantiene una copia de tu repositorio fuera de tu máquina local.
3. **Accesibilidad:** Permite acceder a tu código desde diferentes computadoras.

Interactúas con los repositorios remotos usando comandos como `git clone`, `git fetch`, `git pull`, y `git push`.

• ¿Cómo agregar un repositorio remoto a Git?

Si iniciaste un repositorio Git localmente (`git init`) y quieres conectarlo a un repositorio que ya creaste en GitHub (que debería estar vacío o recién inicializado sin commits conflictivos), necesitas agregar GitHub como un repositorio remoto.

Comando:

```Bash
git remote add <nombre-remoto> <URL-del-repositorio-remoto>
```

- `<nombre-remoto>`**:** Es un alias o nombre corto que le darás a esa URL remota. La convención más común es usar `origin`.
- `<URL-del-repositorio-remoto>`**:** Es la URL del repositorio en GitHub. La encuentras en la página principal de tu repositorio en GitHub, usualmente bajo el botón "Code". Puede ser una URL HTTPS (ej. `https://github.com/tu-usuario/tu-repo.git`) o SSH (ej. `git@github.com:tu-usuario/tu-repo.git`).

Ejemplo:

```Bash
git remote add origin https://github.com/tu-usuario/mi-proyecto.git
```

Después de ejecutar este comando, puedes verificar que se agregó correctamente con `git remote -v`. Ahora podrás usar `origin` en comandos como `git push origin main`.

• ¿Cómo empujar cambios a un repositorio remoto?

"Empujar" (push) cambios es el acto de enviar tus commits locales a un repositorio remoto. Es exactamente lo mismo que "enviar un commit a GitHub" (pregunta anterior).

Comando:

```
Bash

    git push <nombre-remoto> <nombre-rama-local>

```

Ejemplo:

```
Bash

    git push origin develop

```

- **Explicación:** Envía los commits de tu rama local `develop` a la rama `develop` del repositorio remoto llamado `origin`. Si la rama `develop` no existe en el remoto, Git normalmente la creará.

• ¿Cómo tirar de cambios de un repositorio remoto?

"Tirar" (pull) de cambios significa traer los cambios más recientes desde un repositorio remoto a tu repositorio local y fusionarlos con tu rama actual. Es útil para actualizar tu copia local con el trabajo que otros han subido o que tú mismo subiste desde otra máquina.

Comando:

```
Bash

    git pull <nombre-remoto> <nombre-rama-remota>

```

- `<nombre-remoto>`**:** El alias del repositorio remoto (ej. `origin`).
- `<nombre-rama-remota>`**:** El nombre de la rama en el repositorio remoto de la

cual quieres traer los cambios (ej. `main`).

Ejemplo común (estando en la rama `main` local):

```
Bash

    git pull origin main

```

- **Explicación:** Este comando hace dos cosas:
    1. `git fetch origin`: Descarga los objetos y referencias (commits, ramas remotas) más recientes del repositorio remoto `origin`, pero *no* los integra todavía en tus ramas locales. Actualiza tus "ramas de seguimiento remoto" (como `origin/main`).
    2. `git merge origin/main`: Fusiona los cambios descargados de la rama remota (`origin/main`) en tu rama local actual (que en este ejemplo asumimos es `main`).

Si hay conflictos durante la fusión, deberás resolverlos como en cualquier otro `git merge`.

Alternativamente, puedes usar `git fetch origin` para solo descargar, y luego decidir manualmente si hacer `git merge origin/main` o `git rebase origin/main` para integrar los cambios.

• ¿Qué es un fork de repositorio?

Un "fork" es una **copia personal de un repositorio existente** que realizas en una plataforma como GitHub. Cuando haces un fork de un repositorio (especialmente uno que no te pertenece), creas una versión de ese repositorio bajo tu propio usuario o organización.

El propósito principal de un fork es:

1. **Experimentar libremente:** Puedes hacer cambios en tu copia (el fork) sin afectar el repositorio original.
2. **Contribuir a proyectos:** Es el mecanismo estándar para proponer cambios a proyectos de código abierto. Haces los cambios en tu fork, y luego envías una "Pull Request" desde tu fork al repositorio original para que los mantenedores revisen y potencialmente incorporen tus contribuciones.

Tu fork mantiene una conexión con el repositorio original (llamado "upstream"), lo que te permite mantener tu copia actualizada con los cambios que ocurran en el

original.

• ¿Cómo crear un fork de un repositorio?

Crear un fork se hace directamente en la interfaz web de la plataforma donde está alojado el repositorio (como GitHub):

1. **Navega al repositorio original:** Ve a la página principal del repositorio que deseas copiar en GitHub.
2. **Encuentra y haz clic en el botón "Fork":** Este botón suele estar en la esquina superior derecha de la página del repositorio.
3. **(Opcional) Selecciona el propietario:** Si perteneces a organizaciones en GitHub, te podría preguntar dónde quieres crear el fork (bajo tu usuario personal o en una organización). Selecciona la opción deseada.
4. **Espera a que GitHub copie el repositorio:** GitHub creará una copia completa del repositorio, incluyendo todo su historial, bajo tu cuenta. Esto puede tardar unos segundos o minutos dependiendo del tamaño del repositorio.

Una vez completado, serás redirigido a la página de *tu* copia (el fork) del repositorio. Ahora puedes clonar *tu* fork a tu máquina local (`git clone <URL-de-tu-fork>`) para empezar a trabajar en él.

Aquí tienes las respuestas desarrolladas para la segunda parte de tus preguntas sobre GitHub y Git:

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

Una Solicitud de Extracción (Pull Request o PR) es la forma estándar en GitHub (y plataformas similares) de proponer cambios desde tu repositorio (o una rama específica) para que sean incorporados (fusionados) en otro repositorio o rama (generalmente el repositorio original o la rama principal).

Pasos Generales (Asumiendo que has hecho cambios en tu fork o en una rama):

1. **Empuja tus cambios a GitHub:** Asegúrate de que los commits con los cambios que quieres proponer estén en tu repositorio remoto en GitHub (en tu fork o en la rama correspondiente del repositorio principal).
2. Bash

```
# Estando en tu rama local con los cambios y commits hechos

git push origin <tu-rama>
```

3.
4.
5. **Inicia la Pull Request en GitHub:**
    - Ve a la página del repositorio *original* (al que quieres contribuir) o a la página de *tu fork* en GitHub.
    - GitHub a menudo detectará que has empujado una nueva rama recientemente y mostrará un banner con un botón "Compare & pull request". Haz clic en él.
    - Alternativamente, ve a la pestaña "Pull requests" del repositorio *original* y haz clic en "New pull request".
6. **Configura las Ramas:**
    - **Base Repository / Base:** Selecciona el repositorio y la rama *a la que quieres que se fusionen tus cambios* (ej: `usuario-original/repo-original` y rama `main`).
    - **Head Repository / Compare:** Selecciona tu repositorio (fork) y la rama *que contiene tus cambios* (ej: `tu-usuario/repo-original` y rama `feature-nueva`).
7. **Revisa los Cambios:** GitHub mostrará una vista comparativa (diff) de los cambios que estás proponiendo. Asegúrate de que sean los correctos.
8. **Escribe Título y Descripción:**
    - **Título:** Claro y conciso, resumiendo el propósito de la PR.
    - **Descripción:** Detallada, explicando *qué* hace el cambio, *por qué* es necesario, y cómo probarlo. Puedes enlazar a "Issues" existentes (ej: `Closes #123`).
9. **(Opcional) Configura Adicionales:** Puedes solicitar revisores específicos, asignarte a ti mismo o a otros, añadir etiquetas (labels), o asociarlo a un proyecto (project board).
10. **Crea la Pull Request:** Haz clic en el botón "Create pull request".

Una vez creada, los mantenedores del repositorio original serán notificados y podrán revisar tu propuesta, dejar comentarios, solicitar cambios o, finalmente, aceptarla y fusionarla.

• ¿Cómo aceptar una solicitud de extracción?

Aceptar (o fusionar) una Pull Request es el acto de incorporar los cambios propuestos en la rama base del repositorio. Esto generalmente lo hacen los mantenedores o colaboradores con permisos de escritura.

Pasos (en la interfaz de GitHub):

1. **Navega a la Pull Request:** Ve a la pestaña "Pull requests" del repositorio y selecciona la PR que quieres aceptar.
2. **Revisa la PR:** Lee la descripción, revisa los comentarios, y examina los

cambios en la pestaña "Files changed". Verifica que las pruebas automatizadas (si las hay) hayan pasado.
3. **Verifica Conflictos:** GitHub indicará si existen conflictos de fusión entre la rama de la PR y la rama base. Si hay conflictos, deben resolverse antes de poder fusionar. Esto puede hacerse localmente o, para conflictos sencillos, a veces directamente en la interfaz de GitHub.
4. **Haz clic en "Merge pull request":** Si todo está en orden y no hay conflictos (o se han resuelto), aparecerá un botón verde "Merge pull request". Haz clic en él.
5. **(Opcional) Elige Estrategia de Fusión y Edita Mensaje:**
    - Puedes elegir entre "Create a merge commit" (opción por defecto, mantiene todo el historial), "Squash and merge" (combina todos los commits de la PR en uno solo en la rama base), o "Rebase and merge" (reaplica los commits de la PR sobre la rama base, creando un historial lineal).
    - Puedes editar el mensaje del commit de fusión si es necesario.
6. **Confirma la Fusión:** Haz clic en "Confirm merge" (o el botón correspondiente según la estrategia).
7. **(Opcional) Elimina la Rama:** GitHub te ofrecerá eliminar la rama de origen de la PR, ya que sus cambios han sido incorporados. Generalmente es seguro hacerlo para mantener limpio el repositorio.

• ¿Qué es un etiqueta en Git?

Una etiqueta (tag) en Git es una **marca o referencia a un punto específico en el historial de commits**. Se usan comúnmente para marcar versiones de lanzamiento (releases) del software, como `v1.0`, `v2.1.3`, etc.

A diferencia de una rama (que es un puntero que se mueve a medida que se añaden nuevos commits), una etiqueta, una vez creada, generalmente apunta siempre al mismo commit, sirviendo como un marcador permanente para ese estado particular del código.

Existen dos tipos principales de etiquetas:

1. **Lightweight (Ligeras):** Son simplemente un puntero a un commit específico. No almacenan información adicional más allá del nombre de la etiqueta y el commit al que apuntan.
2. **Annotated (Anotadas):** Son objetos completos en la base de datos de Git. Almacenan información adicional como el nombre del "etiquetador" (tagger), su email, la fecha de creación, un mensaje de etiquetado (distinto al mensaje del commit), y opcionalmente pueden ser firmadas con GPG para verificación. **Son el tipo recomendado para marcar releases.**

• ¿Cómo crear una etiqueta en Git?

Las etiquetas se crean usando el comando `git tag` en tu repositorio local.

Para crear una etiqueta ligera:

Apunta al commit actual:

```
Bash

git tag <nombre-etiqueta>

# Ejemplo: git tag v0.5-alpha
```

Apunta a un commit específico anterior (necesitas el hash del commit):

```
Bash

git tag <nombre-etiqueta> <hash-del-commit>

# Ejemplo: git tag v0.4 abc1234
```

Para crear una etiqueta anotada (recomendado para releases):

Apunta al commit actual:

```
Bash

git tag -a <nombre-etiqueta> -m "Mensaje descriptivo de la etiqueta"

# Ejemplo: git tag -a v1.0 -m "Primera versión estable lanzada"
```

Apunta a un commit específico anterior:

```
Bash

git tag -a <nombre-etiqueta> -m "Mensaje descriptivo" <hash-del-commit>

# Ejemplo: git tag -a v0.9.1 -m "Corrección de bug crítico" abc1234
```

Puedes listar todas las etiquetas existentes con `git tag` y ver los detalles de una etiqueta específica con `git show <nombre-etiqueta>`.

• ¿Cómo enviar una etiqueta a GitHub?

Por defecto, el comando `git push` **no envía las etiquetas** al repositorio remoto. Debes indicarle explícitamente que las envíe.

Para enviar una etiqueta específica:

```
Bash

git push <nombre-remoto> <nombre-etiqueta>
```

- **Ejemplo:** `git push origin v1.0`

Para enviar todas las etiquetas locales que no estén en el remoto:

```
Bash

git push <nombre-remoto> --tags
```

- **Ejemplo:** `git push origin --tags`

Una vez enviadas, las etiquetas aparecerán en la sección "Releases" o "Tags" de tu repositorio en GitHub. A menudo se usan las etiquetas anotadas para crear "Releases" formales en GitHub, donde puedes adjuntar archivos binarios y notas de la versión.

• ¿Qué es un historial de Git?

El historial de Git es el **registro completo de todos los commits** (instantáneas del proyecto) que se han realizado en un repositorio desde su creación. No es simplemente una lista lineal, sino un **Grafo Acíclico Dirigido (DAG)**, ya que las ramas crean líneas de desarrollo paralelas y las fusiones (merges) las unen, haciendo que un commit de fusión tenga más de un "padre".

Cada commit en el historial contiene:

- Un identificador único (hash SHA-1).
- Una referencia a su commit(s) padre(s).
- Una instantánea del contenido de todos los archivos del proyecto en ese momento.
- Metadatos: autor, fecha, y el mensaje del commit.

El historial permite entender cómo ha evolucionado el proyecto, quién hizo qué cambios, cuándo se introdujeron, y facilita volver a estados anteriores del código o investigar cuándo apareció un bug (`git bisect`).

• ¿Cómo ver el historial de Git?

El comando principal para ver el historial de commits en Git es `git log`. Tiene muchas opciones para personalizar la salida:

- `git log`: Muestra el historial completo (en orden cronológico inverso), con hash del commit, autor, fecha y mensaje completo. Navegas con las flechas y sales con `q`.
- `git log --oneline`: Versión muy concisa, muestra solo el hash corto y la primera línea del mensaje de cada commit.
- `git log --graph`: Muestra una representación gráfica ASCII de las ramas y fusiones. Muy útil combinado con otras opciones: `git log --graph --oneline --decorate --all` (muestra todas las ramas y etiquetas de forma gráfica y concisa).
- `git log -p` o `git log --patch`: Muestra los cambios específicos (el "diff" o parche) introducidos en cada commit.
- `git log --stat`: Muestra estadísticas de los archivos modificados en cada commit (cuántos archivos cambiaron, cuántas líneas se añadieron/eliminaron).
- `git log -- <ruta/al/archivo>`: Muestra solo los commits que afectaron a un archivo o directorio específico.
- `git log -N` (donde N es un número): Muestra solo los últimos N commits (ej: `git log -5`).

También existen muchas herramientas gráficas (GUIs como GitKraken, SourceTree, integradas en IDEs como VS Code) que ofrecen interfaces visuales para explorar el historial de manera más interactiva.

• ¿Cómo buscar en el historial de Git?

Puedes usar opciones del comando `git log` para buscar commits específicos basados en diferentes criterios:

- **Buscar por mensaje del commit:**
- Bash

```
git log --grep="palabra clave en mensaje"
```

-
-

- **Buscar por autor:**
- Bash

```
git log --author="Nombre del autor o email"
```

-
-
- **Buscar por cambios en el contenido (Pickaxe):** Encuentra commits que introdujeron o eliminaron una línea específica de código/texto.
- Bash

```
git log -S"texto específico en el código"
```

-
-
- **Buscar por rango de fechas:**
- Bash

```
git log --since="2024-01-01" --until="2024-12-31"

git log --since="3 weeks ago"
```

-
-
- **Buscar commits que afectaron un archivo:** (Mencionado antes)
- Bash

```
git log -- <ruta/al/archivo>
```

-
-
- **Combinar búsquedas:** Puedes usar varias de estas opciones juntas.

• ¿Cómo borrar el historial de Git?

Borrar o reescribir el historial de Git es una operación **avanzada y potencialmente destructiva**, especialmente si el historial ya ha sido compartido (empujado a un remoto). **Debe hacerse con extrema precaución.**

¡Advertencia! Reescribir el historial que otros colaboradores ya tienen puede causar serios problemas en sus repositorios locales al intentar sincronizar.

Generalmente, es mejor crear nuevos commits que reviertan cambios (`git revert`) en lugar de borrar historia compartida.

Dicho esto, algunas formas de alterar/borrar historial (principalmente para uso local o antes de compartir):

1. **Eliminar TODO el historial (¡Muy drástico!):** Borra la carpeta `.git` oculta en la raíz de tu proyecto. Esto elimina toda conexión con Git y convierte tu proyecto en archivos normales no versionados. Pierdes todo el historial.
2. Bash

```
# ¡¡¡CUIDADO!!! Esto no se puede deshacer fácilmente.

rm -rf .git
```

3.
4. Luego podrías iniciar un nuevo historial si quisieras: `git init`, `git add .`, `git commit -m "Commit inicial nuevo"`.
5. **Eliminar commits recientes (localmente):** Mueve el puntero de la rama hacia atrás, descartando commits.
6. Bash

```
# Descarta los últimos 3 commits en la rama actual

git reset --hard HEAD~3
```

7.
8. Si estos commits ya fueron empujados, necesitarás un `git push --force` para actualizar el remoto, lo cual es disruptivo para otros.
9. **Reescribir historial interactivo (local):** Para modificar, combinar (squash), eliminar o reordenar commits recientes *antes* de empujarlos.
10. Bash

```
# Abre un editor para manipular los últimos 5 commits

git rebase -i HEAD~5
```

11.
12.
13. **Eliminar archivos/datos sensibles de TODO el historial:** Herramientas más complejas como `git filter-repo` (recomendada sobre el antiguo `git filter-branch`) pueden usarse para eliminar un archivo o información de cada commit en la historia. Es complejo y requiere reescribir todo el historial y forzar el push.

En resumen: evita borrar historial compartido. Si necesitas deshacer cambios, prefiere `git revert`**.**

• ¿Qué es un repositorio privado en GitHub?

Un repositorio privado en GitHub es un repositorio cuyo contenido **no es visible públicamente**. Solo el propietario y los colaboradores a los que el propietario haya concedido acceso explícitamente pueden ver el código, el historial, las issues, las pull requests y otras características del repositorio.

Se utiliza para:

- Desarrollar software propietario o comercial.
- Proyectos personales que no se desean compartir públicamente.
- Colaboración interna dentro de un equipo u organización antes de hacer público un proyecto.

GitHub ofrece repositorios privados gratuitos con ciertas limitaciones en características avanzadas o recursos (como minutos de GitHub Actions) en sus planes gratuitos.

• ¿Cómo crear un repositorio privado en GitHub?

El proceso es idéntico al de crear cualquier repositorio en GitHub, pero seleccionando la opción de visibilidad adecuada:

1. Inicia sesión en GitHub.
2. Haz clic en "+" o "New" -> "New repository".
3. Completa el nombre, descripción, etc.
4. En la sección de visibilidad, **asegúrate de seleccionar la opción "Private"**.
5. (Opcional) Inicializa con README, .gitignore, licencia.
6. Haz clic en "Create repository".

El repositorio se creará y solo tú tendrás acceso inicialmente.

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

Para dar acceso a otras personas a tu repositorio privado, necesitas invitarlas como colaboradores:

1. Ve a la página principal de tu repositorio privado en GitHub.
2. Haz clic en la pestaña **"Settings"** (Configuración).
3. En el menú lateral izquierdo, haz clic en **"Collaborators"** (Colaboradores) o **"Manage access"** (Gestionar acceso).

4. Haz clic en el botón **"Add people"** (Añadir personas).
5. En el cuadro de búsqueda, escribe el **nombre de usuario, nombre completo o dirección de correo electrónico** de la persona asociada a su cuenta de GitHub. Selecciónala de la lista que aparece.
6. Elige el **nivel de permiso** que deseas otorgarle (por ejemplo: Read, Triage, Write, Maintain, Admin). Lee las descripciones para entender qué puede hacer cada rol.
7. Haz clic en **"Add [username] to this repository"** (Añadir [usuario] a este repositorio).

La persona invitada recibirá una notificación (por email y en GitHub) y deberá aceptar la invitación para poder acceder al repositorio.

• ¿Qué es un repositorio público en GitHub?

Un repositorio público en GitHub es un repositorio cuyo contenido, historial y actividad son **visibles para cualquier persona en Internet**. Cualquiera puede verlo, clonarlo (descargar una copia) y hacer un fork (crear su propia copia en GitHub).

Aunque cualquiera puede ver el código, solo los colaboradores con permisos explícitos pueden modificarlo directamente (hacer push). Las contribuciones de la comunidad se gestionan típicamente a través de forks y Pull Requests.

Son la base del desarrollo de software de código abierto (open source) y se usan ampliamente para compartir conocimiento, bibliotecas, herramientas y proyectos de todo tipo.

• ¿Cómo crear un repositorio público en GitHub?

El proceso es el mismo que para crear cualquier repositorio, asegurándote de elegir la visibilidad correcta:

1. Inicia sesión en GitHub.
2. Haz clic en "+" o "New" -> "New repository".
3. Completa el nombre, descripción, etc.
4. En la sección de visibilidad, **asegúrate de que esté seleccionada la opción "Public"** (suele ser la opción por defecto).
5. (Opcional pero muy recomendado para públicos) Inicializa con README, .gitignore y elige una licencia de código abierto.
6. Haz clic en "Create repository".

El repositorio será visible públicamente de inmediato.

• ¿Cómo compartir un repositorio público en GitHub?

Dado que un repositorio público ya es visible para todos, "compartirlo" se refiere más a darlo a conocer o facilitar que otros lo encuentren y usen:

1. **Compartir la URL:** La forma más directa es copiar la URL del repositorio (ej: `https://github.com/usuario/nombre-repo`) y pegarla en correos electrónicos, chats, documentación, redes sociales, etc.
2. **Añadir "Topics" (Temas):** En la página principal del repositorio, en la sección "About" (a la derecha), puedes añadir palabras clave relevantes ("topics"). Esto ayuda a que el repositorio aparezca en búsquedas temáticas dentro de GitHub.
3. **Buena Documentación (README):** Un archivo `README.md` claro y completo es esencial para que la gente entienda qué hace tu proyecto, cómo instalarlo y cómo usarlo.
4. **Licencia Clara:** Asegúrate de incluir un archivo `LICENSE` con una licencia de código abierto reconocida para que otros sepan cómo pueden usar legalmente tu código.
5. **Guías de Contribución:** Si quieres que otros contribuyan, crea archivos `CONTRIBUTING.md` (cómo contribuir) y `CODE_OF_CONDUCT.md` (código de conducta).
6. **Promoción Externa:** Menciona tu repositorio en blogs, foros relevantes, listas de correo, conferencias, etc.