Práctico 2: Git y GitHub

Objetivo:

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

Resultados de aprendizaje:

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

Actividades

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

• ¿Qué es GitHub?

GitHub es una plataforma en línea para alojar y gestionar proyectos de desarrollo de software utilizando Git, que es un sistema de control de versiones. Sirve para varias cosas:

- Control de versiones: Permite llevar un historial de cambios en el código, facilitando volver a versiones anteriores si es necesario.
- Trabajo en equipo: Varios desarrolladores pueden colaborar en el mismo proyecto sin sobrescribir el trabajo de los demás.
- Alojamiento de código: Guarda y organiza el código de forma segura en la nube.
- Automatización y despliegue: Se pueden configurar procesos automáticos como pruebas de código o despliegues en servidores.

• ¿Cómo crear un repositorio en GitHub?

Existen dos formas de crear un repositorio en GitHub:

- Opción 1: desde la misma página web de GitHub:

1- Si aún no tenemos una cuenta, debemos crear una nueva.

2- Iniciamos sesión y hacemos clic en el botón "New" (Nuevo), o también podemos ir al signo + en la esquina superior derecha. Luego hacemos clic en “New repository”.

3- Debemos escribir un nombre para el repositorio (ej: "MiPrimerRepositorio").

4- Opcionalmente, Podemos añadir una descripción y elegir si será público o privado.

5- Podemos marcar la opción "Initialize this repository with a README" para que se cree un archivo de descripción.

6- Finalmente, hacemos clic en “Create repository”.

- Opción 2: usando Git en la terminal (si queremos trabajar desde nuestra PC):

1- Primero debemos asegurarnos de tener Git instalado en nuestra PC. Escribimos en la terminal:

```
git --version
```

2- Luego, tenemos que configurar nuestro usuario en Git (si es la primera vez que lo instalamos), escribiendo los siguientes comandos en la terminal:

```
git config --global user.name "TuNombre"
git config --global user.email nuestroemail@ejemplo.com
```

3- Creamos una carpeta y entramos en ella, con los siguientes comandos:

```
mkdir MiPrimerRepositorio
cd MiPrimerRepositorio
```

4- Inicializamos Git dentro de la carpeta creada, con el siguiente comando:

```
git init
```

5- Conectamos con el repositorio en GitHub, con el siguiente comando:

```
git remote add origin https://github.com/TuUsuario/MiPrimerRepositorio.git
```

6- Subimos archivos, con los siguientes comandos:

```
echo "# MiPrimerRepositorio" > README.md
git add .
git commit -m "Primer commit"
git push -u origin main
```

• ¿Cómo crear una rama en Git?

Tenemos que escribir el siguiente comando en la terminal:

```
git branch nombre-de-la-rama
```

Por ejemplo:

```
git branch nueva-funcionalidad
```

• ¿Cómo cambiar a una rama en Git?

Tenemos que escribir el siguiente comando en la terminal:

```
git checkout nombre-de-la-rama
```

Por ejemplo:

```
git checkout nueva-funcionalidad
```

• ¿Cómo fusionar ramas en Git?

- Primero tenemos que cambiar a la rama en la que queremos fusionar los cambios (generalmente es la rama main), con el siguiente comando:

```
git checkout main
```

- Luego tenemos que fusionar la otra rama en la actual, con el comando:

```
git merge nombre-de-la-rama
```

Por ejemplo:

```
git merge nueva-funcionalidad
```

• ¿Cómo crear un commit en Git?

1- Primero, debemos asegurarnos de que estamos en el directorio del repositorio:

Para ello, tenemos que navegar hasta la carpeta donde está nuestro repositorio local usando la terminal. Si no sabemos si estamos en la carpeta del repositorio, tenemos que usar el comando cd para cambiar de directorio:

```
cd ruta/al/repositorio
```

Por ejemplo:

```
cd Desktop/Programación-I/Practica-Git
```

En la terminal se nos va a mostrar la carpeta en donde estamos situados con un asterisco *.

2- Luego, debemos verificar el estado de los archivos:

Antes de hacer un commit, debemos ver qué archivos han sido modificados. Usamos el comando:

```
git status
```

Esto nos mostrará los archivos que están modificados o que no han sido añadidos al "staging area" (área de preparación).

3- Tenemos que añadir los cambios al área de preparación (staging area):

Para hacer un commit, primero debemos añadir los archivos que queremos guardar. Esto se hace con el comando git add:

- Para agregar todos los archivos modificados:

```
git add .
```

- Para agregar un archivo especifico:

```
git add nombre-del-archivo
```

4- Creamos el commit:

Después de añadir los cambios al área de preparación, podemos crear el commit con el comando:

```
git commit
```

Es recomendable que incluyamos un mensaje que describa los cambios realizados, usando la opción -m (que significa “message”), con el comando:

```
git commit -m "Mensaje descriptivo del commit"
```

Por ejemplo:

```
git commit -m "Corregido error en el archivo de configuración"
```

5- Para verificar los commits realizados, es decir, si queremos ver el historial de nuestros commits, usamos el comando:

```
git log
```

Esto nos mostrará los commits anteriores junto con sus identificadores únicos.

• ¿Cómo enviar un commit a GitHub?

1- Verificamos que estamos en la rama correcta:

Primero, tenemos que asegurarnos de que estamos trabajando en la rama en la que queremos enviar el commit. Usamos el siguiente comando para ver en qué rama nos encontramos:

```
git branch
```

Si necesitamos cambiar de rama, usamos:

```
git checkout nombre-de-la-rama
```

2- Verificamos el estado del repositorio:

Antes de hacer el push, podemos asegurarnos de que nuestro repositorio local esté sincronizado con el remoto usando el comando:

```
git status
```

3- Enviamos el commit a GitHub:

Para enviar nuestros cambios al repositorio remoto de GitHub, usamos el comando git push. Esto enviará nuestros commits a la rama en GitHub correspondiente a la rama local en la que estamos trabajando.

```
git push origin nombre-de-la-rama
```

Por ejemplo:

```
git push origin main
```

Esto enviará nuestros cambios a la rama main de nuestro repositorio en GitHub.

5- Verificamos en GitHub:

Si todo salió bien, podemos ir a nuestro repositorio en GitHub y ver los cambios reflejados en la rama correspondiente.

• ¿Qué es un repositorio remoto?

Un repositorio remoto es una versión del repositorio de Git que está alojada en un servidor externo, como GitHub, GitLab o Bitbucket. Mientras que un repositorio local reside en mi computadora, el repositorio remoto está en la nube, y permite que diferentes personas colaboren y sincronicen su trabajo.

¿Para qué sirve un repositorio remoto?

- Colaboración: Varios desarrolladores pueden trabajar en el mismo proyecto, subiendo y bajando cambios desde el repositorio remoto.
- Respaldo: Al estar en la nube, el repositorio remoto sirve como una copia de seguridad de tu código.
- Acceso desde cualquier lugar: Puedes acceder y trabajar en el proyecto desde cualquier computadora o dispositivo, siempre que tengas acceso al repositorio remoto.
- Facilita el uso de Git: El repositorio remoto es esencial para compartir tu trabajo y sincronizar los cambios entre tu repositorio local y el de otros colaboradores.

• ¿Cómo agregar un repositorio remoto a Git?

1- Primero, tenemos que ir a nuestro repositorio en GitHub (o la plataforma que usemos) y copiar la URL del repositorio remoto.

Por ejemplo: https://github.com/usuario/nombre-del-repositorio.git

2- Abrimos la terminal en nuestro proyecto local y navegamos hasta el directorio de nuestro repositorio local utilizando el comando cd:

```
cd ruta/a/mi/repositorio
```

3- Agregamos el repositorio remoto:

Usamos el comando git remote add para vincular nuestro repositorio local con el repositorio remoto en GitHub:

```
git remote add origin https://github.com/usuario/nombre-del-repositorio.git
```

En general, “origin” es el nombre predeterminado para el repositorio remoto. Si ya tenemos otro repositorio remoto llamado "origin", podemos cambiarlo por otro nombre.

4- Verificamos que se haya agregado correctamente:

Podemos comprobar que nuestro repositorio remoto ha sido agregado con el siguiente comando:

```
git remote -v
```

Esto nos mostrará la URL del repositorio remoto (tanto para fetch como para push).

• ¿Cómo empujar cambios a un repositorio remoto?

Una vez que tengamos el commit listo, usamos el siguiente comando para empujar los cambios a nuestro repositorio remoto:

```
git push origin nombre-de-la-rama
```

En general, “origin” es el nombre del repositorio remoto (aunque podemos modificarlo si deseamos).

Aquí, “nombre-de-la-rama” es la rama en la que estamos trabajando. Si estamos trabajando en la rama principal, probablemente se llame main o master.

Por ejemplo:

```
git push origin main
```

• ¿Cómo tirar de cambios de un repositorio remoto?

1- Verificamos la rama actual:

Antes de traer los cambios, debemos asegurarnos de que estamos en la rama correcta. Usamos el siguiente comando para verificar la rama en la que estamos trabajando:

```
git branch
```

Si necesitamos cambiar a otra rama, usamos:

```
git checkout nombre-de-la-rama
```

2- Obtenemos cambios del repositorio remoto:

El comando git pull obtiene los cambios más recientes del repositorio remoto y los fusiona con nuestra rama local actual:

```
git pull origin nombre-de-la-rama
```

Por ejemplo:

```
git pull origin main
```

• ¿Qué es un fork de repositorio?

Un fork de un repositorio es una copia independiente de un repositorio que existe en una plataforma como GitHub. Al hacer un fork, creamos una versión personal de ese repositorio en nuestra propia cuenta, lo que nos permite realizar cambios sin afectar el proyecto original. Esto es útil especialmente en proyectos de código abierto donde queremos contribuir sin modificar directamente el repositorio original.

• ¿Cómo crear un fork de un repositorio?

1- Primero, vamos al repositorio que deseamos forquear: entramos en GitHub y navegamos al repositorio del que deseamos hacer un fork.

2- Hacemos clic en el botón "Fork": en la esquina superior derecha de la página del repositorio, hacemos clic en el botón Fork.

3- Seleccionamos nuestra cuenta: si tenemos más de una cuenta en GitHub (por ejemplo, una personal y otra de una organización), seleccionamos en qué cuenta queremos crear el fork.

4- Al finalizar, GitHub creará una copia del repositorio en nuestra cuenta. Ahora podemos hacer cambios en nuestro fork sin afectar el repositorio original.

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

El Pull Request es una forma de proponer cambios a un proyecto en GitHub, generalmente a través de un fork. Subimos nuestros cambios, creamos la solicitud de extracción desde nuestro fork y los administradores repositorio original revisarán, comentarán y posiblemente fusionarán los cambios.

Los pasos a seguir son:

1- Realizamos cambios en nuestro fork:

- Clonamos el repositorio que hemos forkeado a nuestra máquina local:

```
git clone https://github.com/tu-usuario/nombre-del-repositorio.git
```

- Realizamos los cambios necesarios en nuestro proyecto y guardamos esos cambios.
- Hacemos commit de nuestros cambios:

```
git add .
git commit -m "Descripción clara de los cambios realizados"
```

2- Subimos tus cambios a nuestro fork en GitHub:

- Una vez que hayamos hecho los commits, subimos esos cambios a nuestro repositorio remoto (nuestro fork en GitHub):

```
git push origin nombre-de-tu-rama
```

3- Creamos el Pull Request en GitHub:

Ahora que nuestros cambios están en nuestro fork en GitHub, tenemos que seguir estos pasos:

1) Vamos a nuestro fork en GitHub: nos dirigimos a la página del repositorio que hemos forkeado en nuestra cuenta de GitHub.

2) Hacemos clic en "Compare & Pull Request": GitHub nos mostrará un botón de "Compare & Pull Request" (Comparar y hacer Pull Request) cuando subamos cambios a nuestro fork. Hacemos clic en ese botón.

3) Si no vemos el botón, podemos ir al repositorio original (el que hemos forkeado) y veremos un botón que dice "New Pull Request" o "Crear nuevo Pull Request".

4) Seleccionamos la base y la rama de comparación:
- Base: el repositorio y la rama en la que queremos que se integren nuestros cambios (por lo general, es la rama main o master del repositorio original).
- Compare: la rama de nuestro fork que contiene los cambios que hemos hecho.

5) Escribimos una descripción detallada: en el cuadro de texto, tenemos que describir claramente los cambios que realizamos, qué problemas resuelven, o cualquier otra información relevante.

6) Enviamos el Pull Request: hacemos clic en el botón "Create Pull Request" para enviarlo.

4- Revisión del Pull Request:

Después de enviar nuestro Pull Request, los administradores del repositorio original recibirán una notificación y podrán revisar nuestros cambios. Si todo está bien, lo fusionarán (merge) en el repositorio principal.
Si hay algún conflicto o problema con los cambios, los administradores pueden pedirnos que los resolvamos y actualicemos el Pull Request.

• ¿Cómo aceptar una solicitud de extracción?

Aceptar una solicitud de extracción (Pull Request) en GitHub es el proceso mediante el cual integras los cambios propuestos por otro colaborador en tu repositorio. Esto generalmente se hace cuando alguien ha enviado un Pull Request con modificaciones que desea que se fusionen en tu repositorio.

Los pasos a seguir son:

1- Revisamos el Pull Request:

- Accedemos a nuestro repositorio en GitHub: iniciamos sesión en nuestra cuenta de GitHub y nos dirigimos al repositorio en el que recibimos el Pull Request.

- Vamos a la sección "Pull Requests": en la parte superior del repositorio, veremos varias pestañas (como "Code", "Issues", "Pull Requests"). Hacemos clic en "Pull Requests".

- Seleccionamos el Pull Request: encontraremos una lista de solicitudes de extracción pendientes. Hacemos clic en la que deseamos revisar y aceptar.

2- Revisamos los cambios propuestos:

En la página del Pull Request, podremos ver los cambios propuestos. GitHub nos muestra las diferencias entre nuestra rama (base) y la rama de la solicitud (compare).
También podemos revisar los comentarios y la descripción del Pull Request. Es recomendable leer los comentarios de la persona que hizo el Pull Request para entender el propósito de los cambios.
Verificamos si los cambios pasan las pruebas (si tenemos configuradas pruebas automáticas como CI/CD).

3- Fusionamos el Pull Request:

Cuando estemos listos para aceptar los cambios, podemos fusionar el Pull Request de una de estas dos formas:

1) Hacemos clic en "Merge pull request":

- Después de revisar los cambios, hacemos clic en el botón verde que dice "Merge pull request" (Fusionar solicitud de extracción).
- Se abrirá una ventana donde podremos escribir un mensaje de confirmación del merge, aunque GitHub proporciona un mensaje por defecto.
- Hacemos clic en "Confirm merge" para finalizar la fusión.

2) Elegimos algún tipo de fusión ofrecida por GitHub:

- Merge commit (por defecto): funde los cambios y crea un commit de fusión en nuestra rama base.
- Squash and merge: combina todos los commits de la rama del Pull Request en un solo commit antes de fusionarlo.
- Rebase and merge: fusiona los cambios sin crear un commit de fusión, reescribiendo el historial.

4- Borramos la rama del Pull Request (opcional):

Después de fusionar el Pull Request, podemos eliminar la rama del Pull Request si ya no la necesitamos. GitHub ofrece un botón para eliminar la rama directamente desde la página del Pull Request después de haberlo fusionado.

Esto ayuda a mantener el repositorio limpio y organizado.

5- Verificamos la fusión:

Después de fusionar, podemos verificar que los cambios estén correctamente aplicados en nuestra rama principal.

Si es necesario, podemos hacer un git pull para traer los cambios a nuestra copia local, con el comando:

```
git pull origin main
```

• ¿Qué es una etiqueta en Git?

Una etiqueta (tag) en Git es un marcador que se usa para señalar un punto específico en la historia del repositorio. Generalmente, se utilizan para versionar lanzamientos (por ejemplo, v1.0, v2.0-beta), identificando versiones estables del código.

Las etiquetas son similares a los commits, pero no se mueven con el historial, es decir, quedan fijas en un commit específico.

• ¿Cómo crear una etiqueta en Git?

La creación varía según el tipo de etiqueta:

1- Crear una etiqueta ligera (Lightweight Tag):

Las etiquetas ligeras son simplemente referencias a un commit, sin información adicional.

Usamos el comando:

```
git tag nombre-del-tag
```

Por ejemplo:

```
git tag v1.0
```

Esto etiqueta el commit actual como v1.0.

2- Crear una etiqueta anotada (Annotated Tag)

Las etiquetas anotadas contienen información adicional como el autor, la fecha y un mensaje descriptivo.

Usamos el comando:

```
git tag -a nombre-del-tag -m "Mensaje de la etiqueta"
```

Por ejemplo:

```
git tag -a v1.0 -m "Versión 1.0 estable"
```

3- Crear una etiqueta en un commit específico:

Si quisiéramos etiquetar un commit que no es el actual, usamos su hash, con el comando:

```
git tag -a nombre-del-tag hash-del-commit -m "Mensaje"
```

Por ejemplo:

```
git tag -a v1.1 3f5e1a2 -m "Versión 1.1 con mejoras"
```

Para encontrar el hash del commit, podemos ejecutar:

```
git log --oneline
```

• ¿Cómo enviar una etiqueta a GitHub?

Para enviar una etiqueta (tag) a GitHub, seguimos estos pasos:

1- Verificamos las etiquetas en nuestro repositorio local:

Primero, nos aseguramos de que la etiqueta existe en nuestro repositorio local con:

```
git tag
```

Si no tenemos ninguna etiqueta, la podemos crear con:

```
git tag -a v1.0 -m "Versión 1.0 estable"
```

2- Enviamos una etiqueta a GitHub:

- Si solo queremos enviar una etiqueta en particular, usamos:

```
git push origin nombre-del-tag
```

Por ejemplo:

```
git push origin v1.0
```

- Si deseamos enviar todas las etiquetas creadas en nuestro repositorio local, usamos:

```
git push --tags
```

Esto subirá todas las etiquetas existentes en nuestro repositorio.

3- Verificamos que la etiqueta está en GitHub:

Una vez en nuestro repositorio en GitHub, hacemos clic en la pestaña "Releases" o en "Tags" para ver las etiquetas enviadas.

• ¿Qué es un historial de Git?

El historial de Git es el registro de todos los cambios (commits) realizados en un repositorio. Contiene información sobre:

- Commits realizados (quién hizo cada cambio y cuándo).
- Mensajes de commit (descripción de cada cambio).
- Ramas y fusiones (cómo se ha desarrollado el proyecto).
- Autores de los cambios (quién contribuyó en cada commit).

Cada commit tiene un hash único que lo identifica dentro del historial.

El historial de Git sirve para:

- Ver cambios previos y entender la evolución del código.
- Revertir errores regresando a un estado anterior.
- Auditar quién hizo qué cambios en un proyecto.
- Colaborar mejor en equipos de desarrollo.

• ¿Cómo ver el historial de Git?

Según el tipo de historial de cambios que queramos ver:

1- Para mostrar el historial básico, usamos:

```
git log
```

Esto mostrará una lista de commits con:

- Hash del commit

- Autor

- Fecha

- Mensaje del commit

Por ejemplo, nos saldrá:

```
commit 3f5e1a2d3b456789abcd1234567890abcdef
Author: Juan Pérez <juan@example.com>
Date:   Wed Mar 27 14:15:10 2024 +0000
    Agregado formulario de contacto
```

2- Para mostrar el historial en una sola línea (más compacto), usamos:

```
git log --oneline
```

Por ejemplo, nos saldrá:

```
3f5e1a2 Agregado formulario de contacto
b2c9d8f Corrección en el CSS del login
f1a7c2e Creada la página de inicio
```

Esto es útil para una vista rápida del historial.

3- Para ver historial con un gráfico de ramas, usamos:

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

Este comando muestra un gráfico que ayuda a visualizar cómo se conectan las ramas en el historial.

• ¿Cómo buscar en el historial de Git?

Git permite buscar en el historial de commits de varias maneras, según lo que necesitemos encontrar.

1- Buscar por mensaje de commit:

Si queremos encontrar un commit con una palabra clave en su mensaje, usamos:

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

Por ejemplo:

```
git log --grep="corrección"
```

Esto mostrará solo los commits cuyo mensaje contenga la palabra "corrección".

2- Buscar por autor:

Si necesitamos ver solo los commits hechos por una persona específica, usamos:

```
git log --author="Nombre o Email"
```

Por ejemplo:

```
git log --author="Juan Pérez"
```

Esto muestra solo los commits realizados por Juan Pérez.

3- Buscar por fecha:

- Para buscar commits en un rango de fechas, usamos:

```
git log --since="YYYY-MM-DD" --until="YYYY-MM-DD"
```

Por ejemplo:

```
git log --since="2024-03-01" --until="2024-03-27"
```

Esto lista los commits entre el 1 y el 27 de marzo de 2024.

- Si solo queremos ver los commits después de una fecha específica, usamos:

```
git log --since="2024-03-01"
```

4- Buscar por archivo modificado:

Si queremos ver los commits que afectaron un archivo específico, usamos:

```
git log -- filename
```

Por ejemplo:

```
git log -- index.html
```

Esto mostrará los commits en los que index.html fue modificado.

5- Buscar por contenido modificado (palabras dentro de archivos):

Si queremos encontrar commits que cambiaron una línea con una palabra clave, usamos:

```
git log -S "texto a buscar"
```

Por ejemplo:

```
git log -S "funcion login"
```

Esto mostrará los commits donde se agregó o eliminó la línea que contiene "funcion login".

6- Buscar un commit por su hash parcial:

Si recordamos una parte del hash del commit, podemos buscarlo con:

```
git log --oneline | grep "parte-del-hash"
```

Por ejemplo:

```
git log --oneline | grep "3f5e1a"
```

Esto mostrará solo los commits cuyo hash comienza con 3f5e1a.

7- Buscar en un rango de commits o ramas:

- Si solo queremos buscar en una rama o en ciertos commits, usamos:

```
git log rama1..rama2
```

Por ejemplo:

```
git log main..feature-nueva
```

Esto muestra los commits en feature-nueva que no están en main.

- Si solo queremos buscar en una cantidad específica de commits recientes, usamos:

```
git log -n 5
```

Esto mostrará solo los últimos 5 commits.

• ¿Cómo borrar el historial de Git?

1- Borrar todo el historial de Git y empezar desde cero:

Este método elimina completamente el historial del repositorio y crea uno nuevo.

- Elimina la carpeta .git (esto borra todo el historial):

```
rm -rf .git
```

- Reinicia Git en el directorio:

```
git init
```

- Agrega todos los archivos al nuevo historial:

```
git add .
```

- Crea un nuevo commit:

```
git commit -m "Reinicio del historial"
```

- Vincula nuevamente el repositorio remoto (si es necesario):

```
git remote add origin URL_DEL_REPOSITORIO
```

- Sube los cambios al repositorio remoto (esto sobrescribirá todo el historial anterior):

```
git push --force origin main
```

2- Borrar solo los commits recientes sin eliminar todo:

Si solo quieres borrar los últimos commits y no todo el historial:

- Borrar los últimos 3 commits (sin afectar archivos):

```
git reset --hard HEAD~3
```

- Borrar y forzar la actualización en GitHub:

```
git push --force origin main
```

Esto eliminará los commits de GitHub (solo usarlo si estamos seguros)

3- Borrar un commit específico del historial:

- Si cometimos un error en un commit antiguo y queremos eliminarlo del historial:

```
git rebase -i HEAD~N
```

(Donde N es la cantidad de commits que queremos revisar).

Luego, cambiamos pick por drop en la línea del commit que queremos borrar y guardamos los cambios.

- Sube los cambios forzadamente si es necesario:

```
git push --force origin main
```

• ¿Qué es un repositorio privado en GitHub?

Un repositorio privado en GitHub es un repositorio que solo nosotros y las personas que invitemos pueden ver y acceder. No es público en Internet y no aparecerá en los resultados de búsqueda.

Es útil cuando:

- Queremos trabajar en un proyecto sin que otros lo vean.
- Estamos desarrollando un software antes de lanzarlo.
- Guardamos información confidencial o código privado.

• ¿Cómo crear un repositorio privado en GitHub?

1- Iniciamos sesión en GitHub.

2- Hacemos clic en el botón "New" o ve a "Repositories" > "New".

3- Escribimos un nombre para el repositorio.

4- En la opción "Visibility", seleccionamos "Private".

5- Hacemos clic en "Create repository".

Ahora solo nosotros podemos verlo, a menos que invites a otros usuarios.

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

1- Vamos a nuestro repositorio y hacemos clic en "Settings".

2- En la barra lateral, seleccionamos "Collaborators".

3- Hacemos clic en "Add people" e ingresamos el nombre de usuario o email.

4- Asignamos permisos y enviamos la invitación.

5- La otra persona recibirá una notificación y podrá acceder al repositorio después de aceptarla.

• ¿Qué es un repositorio público en GitHub?

Un repositorio público en GitHub es un repositorio que cualquier persona en Internet puede ver, clonar y forkear (copiar a su cuenta). Es ideal para proyectos de código abierto, colaborativos o cuando queremos compartir nuestro trabajo con el mundo.

• ¿Cómo crear un repositorio público en GitHub?

1- Iniciamos sesión en GitHub.

2- Hacemos clic en el botón "New" o vamos a "Repositories" > "New".

3- Escribimos un nombre para el repositorio.

4- En la opción "Visibility", seleccionamos "Public".

5- Hacemos clic en "Create repository".

Ahora cualquiera podrá ver nuestro código y colaborar si lo permitimos.

• ¿Cómo compartir un repositorio público en GitHub?

Si tenemos un repositorio público en GitHub y queremos compartirlo con otros, hay varias formas de hacerlo:

1- Compartir el enlace del repositorio:

La forma más sencilla es enviar el enlace del repositorio.

1- Vamos a la página del repositorio en GitHub.

2- Copiamos la URL de la barra de direcciones (ejemplo: https://github.com/usuario/repositorio).

3- Compartimos ese enlace por email, redes sociales o mensajes.

2- Invitar colaboradores (para que puedan modificarlo):

Si queremos que alguien pueda hacer cambios en tu repositorio, debemos agregarlo como colaborador.

1- Vamos a nuestro repositorio en GitHub.

2- Entramos en "Settings" > "Collaborators".

3- Hacemos clic en "Add people" e ingresamos su nombre de usuario o email.

4- Asignamos permisos y enviamos la invitación.

Nota: Solo los dueños del repositorio pueden aceptar contribuyentes sin un fork.

3- Permitir contribuciones con forks y pull requests:

Si el repositorio es de código abierto y queremos que otros colaboren sin darles acceso directo:

- Pueden hacer un fork (copiar el repositorio a su cuenta).
- Pueden hacer cambios y enviarnos un pull request para que revisemos y aceptemos.

Solo tienen que ir a nuestro repositorio y hacer clic en "Fork". Luego, pueden modificar su copia y enviarnos una solicitud de extracción (pull request).

4- Compartir en redes sociales o páginas web:

Si queremos que más personas lo vean:

- Publicamos el enlace en Twitter, LinkedIn, Reddit o foros de programación.
- Lo agregamos en nuestro portafolio o blog personal.
- Lo compartimos en comunidades como GitHub Discussions, Dev.to o Stack Overflow.