git

parte 3: repositorios remotos

Repasamos parte 1

- git config: establecer nombre, email
- git init: inicia git en un directorio
- git add: marcar los ficheros para "commit"
- git status: indica los ficheros marcados para a "commit"
- git commit: guardo el estado. Confirmar cambios.
- git log: muestro el historial de commits

Repasamos parte 2

- git branch: crear y borrar rama
- git checkout: crear, mover puntero, quitar cambios
- git merge: fusionar ramas con commit de merge
- git rebase: fusionar ramas sin commit nuevo
- git reset: nunca lo utilizaremos
- git revert: eliminar un commit conservando la historia

Contenido para hoy

- Repositorio remoto: github
- Fork de un repositorio
- git remote
- git push
- git pull
- git fetch
- Ramas remotas (origin/main, origin/feature-branch)
- Pull Request: propuesta de fusión de código
- Modelos de ramas

¿Qué es un repositorio remoto?
Un repositorio remoto es una copia de tu repositorio en un servidor, accesible por Internet, donde múltiples colaboradores pueden contribuir al proyecto.

GitHub es una plataforma para gestionar repositorios remotos.

Creación de una cuenta en GitHub

https://github.com/signup

Crear un repositorio en GitHub

- Inicia sesión en tu cuenta de GitHub.
- Haz clic en el botón **New** para crear un nuevo repositorio.
- Introduce el nombre del proyecto y selecciona las opciones (público/privado).
- Puedes inicializar el repositorio con un archivo `README.md` o `.gitignore`.

Conectar un repositorio local con GitHub

- Clonar desde remoto:
  `git clone https://github.com/usuario/nombre_del_repo.git`

- Crear un repositorio local y enlazarlo a GitHub:
  `git init`
  `git add .`
  `git commit -m "Primer commit"`
  `git remote add origin <url>`
  `git push -u origin main`

Fork de un repositorio

Un **fork** en GitHub es una copia de un repositorio que se crea en tu propia cuenta de GitHub, a partir de un proyecto de otro usuario.
Permite que trabajes de forma independiente en ese proyecto sin afectar el repositorio original.

¿Para qué sirve un fork?

1. **Contribuir a proyectos de terceros**: Si quieres contribuir a un proyecto de código abierto, puedes hacer un fork para hacer tus modificaciones sin tocar el código original. Luego, puedes proponer tus cambios mediante un **pull request** al repositorio original.
2. **Personalizar proyectos**: Puedes usar un fork para hacer modificaciones o mejoras personalizadas que no necesariamente planeas compartir con el proyecto original.
3. **Explorar sin riesgos**: El fork te permite experimentar y hacer cambios en tu propia copia del proyecto, sin preocuparte por dañar el código del repositorio original.

Flujo típico de trabajo con un fork

1. **Accede al repositorio** en GitHub que deseas copiar.
2. En la parte superior derecha de la página del repositorio, haz clic en el botón **"Fork"**.
3. Selecciona la cuenta donde quieres guardar el fork.
4. Ahora tendrás una copia completa del repositorio en tu cuenta, donde puedes hacer cambios libremente.
5. **Clone**: Clonas el repositorio a tu máquina local.
6. **Modificar**: Trabajas en tu fork (añadiendo nuevas funcionalidades o corrigiendo errores).
7. **Push**: Subes tus cambios a tu fork en GitHub.
8. **Pull Request**: Si quieres contribuir al proyecto original, abres un **pull request** desde tu fork hacia el repositorio original, para que los mantenedores revisen y fusionen tus cambios si son aprobados.

Copiar vs Fork(ear)

- Un fork es una copia en GitHub que mantiene una relación con el repositorio original, permitiendo contribuciones.
- Podrías copiar un repo (por ejemplo, eliminando la carpeta git y subiendolo a un repo tuyo). Esta copia no tendría ninguna relación con el original.

```
git remote
```

El comando `git remote` se utiliza para gestionar las conexiones a repositorios remotos en Git.
Permite ver, agregar, modificar y eliminar referencias a repositorios remotos.

```
git remote
```

- Listar repositorios remotos:
- `git remote -v`
- Agregar un repositorio remoto:
  `git remote add <nombre-remoto> <url>`
  En general, si solo tienes un repositorio remoto, nombre-remoto = origin.
- Eliminar un repositorio remoto:
  `git remote remove <nombre-remoto>`

```
git fetch
```

El comando `git fetch` se utiliza para **descargar los últimos cambios del repositorio remoto**, pero a diferencia de `git pull`, no los fusiona automáticamente en tu rama local.

Esto es útil si quieres revisar los cambios primero antes de aplicarlos.

`git fetch <nombre-remoto>`

```
git pull
```

El comando `git pull` se utiliza para obtener los cambios más recientes del repositorio remoto **y fusionarlos automáticamente con tu rama local**.

Combina dos operaciones: `git fetch` y `git merge`.

`git pull <nombre-remoto> <nombre-rama>`
`git pull origin main`

```
git push
```

El comando `git push` envía los commits que has hecho en tu repositorio local hacia el repositorio remoto.
Si alguien ha subido cambios al remoto después de tu último `pull`, Git te pedirá que primero actualices tu repositorio local antes de poder hacer `push`.

`git push <nombre-remoto> <nombre-rama>`
`git push origin main`

```
git push
```

Primera vez (upstream)
Si es la primera vez que haces `push` de una rama, puedes usar `-u` para establecer la rama de seguimiento:
`git push -u origin main`

Esto establece una conexión entre la rama local y la rama remota, por lo que en futuras ocasiones podrás hacer simplemente `git push` sin especificar `origin` ni `main` (aunque no lo recomiendo).

```
git branch -r
```

Ramas Remotas

Las **ramas remotas** en Git son copias de las ramas locales que existen en un repositorio remoto como GitHub.

Estas ramas permiten que múltiples colaboradores puedan trabajar de forma simultánea en diferentes partes del proyecto y sincronizar sus cambios a través de un repositorio remoto.

```
git branch -r
```

Ramas Remotas

Las ramas remotas suelen tener la nomenclatura <nombre-remoto>/<nombre-rama>

- Por ejemplo:
  - `origin/main` : Rama main en el remoto origin .
  - `origin/feature-branch` : Una rama llamada feature-branch en el remoto origin.

```
git branch -r
```

Ramas Remotas
Nombres importantes:

- `origin` : Es el nombre por defecto que Git asigna al repositorio remoto principal cuando clonas o conectas tu repositorio local con uno remoto.
- `main` : Es la rama principal del proyecto, donde usualmente se encuentra el código más estable o listo para producción.
- `feature-branch` : Es una rama de desarrollo creada para trabajar en una nueva característica, corrección de errores o mejora antes de fusionarla con la rama principal (`main`).

```
git branch –r
```

Trabajar con Ramas Remotas

- ¿Cómo crear una rama Local y subirla?

git checkout -b feature-branch
git add .
git commit -m "Implementar nueva característica en feature-branch"
git push -u origin feature-branch
git branch -r

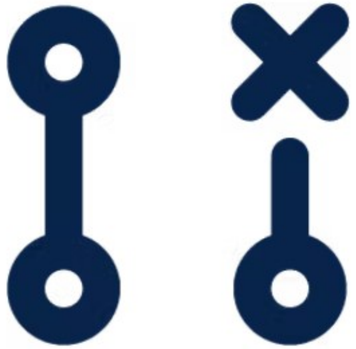
```
git branch -r
```

Trabajar con Ramas Remotas

- ¿Cómo sincronizar cambios desde el remoto?

Si otros colaboradores han hecho cambios en `origin/main` o en `origin/feature-branch`, puedes traer esos cambios a tu repositorio local con `git pull`.

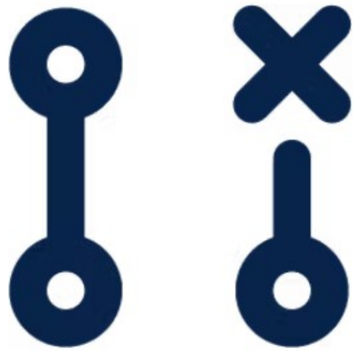
git pull origin main



Pull Request (PR) en GitHub o GitLab
Un Pull Request (PR) es una función que se utiliza en plataformas como GitHub para colaborar en el desarrollo de software.

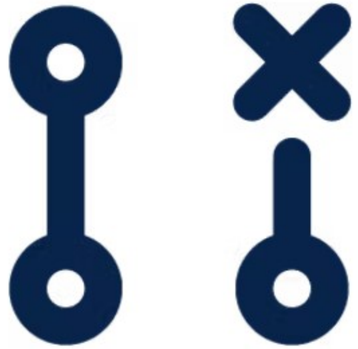
Permite a los desarrolladores proponer cambios **que luego son revisados por otros colaboradores antes de ser fusionados en la rama principal.**

> *El principio de la responsabilidad compartida establece que **todos** los miembros de un equipo son responsables de la calidad y el éxito del trabajo colaborativo.*



Crear el Pull Request en GitHub o GitLab

1. **Hacer clic en el botón "New Pull Request"** para abrir un PR desde tu rama hacia la rama `main`.
2. **Escribir un título y descripción** claros:
   - Título: Breve descripción del cambio, por ejemplo, "Añadir funcionalidad de login".
   - Descripción: Explica qué cambios has realizado, por qué son necesarios, e incluye cualquier detalle importante para la revisión.
3. **Asignar revisores**: Asigna a uno o más colaboradores para que revisen tu Pull Request. Puedes incluir etiquetas (labels) para indicar el tipo de PR (bug, enhancement, etc.).

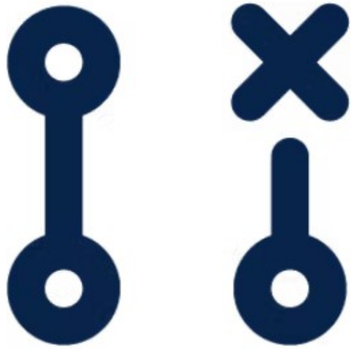


Modelos de ramas
Son enfoques estructurados para gestionar el flujo de trabajo en proyectos de software.
Definen cómo se crean, organizan y combinan las ramas para desarrollar nuevas funcionalidades, corregir errores y lanzar versiones.

Modelos populares:

- **GitFlow**
- **GitLab Flow**
- **Trunk-Based Development**

No hay un modelo mejor que otro. Dependiendo del proyecto y de la fase del mismo, puede convenir uno u otro.



GitFlow

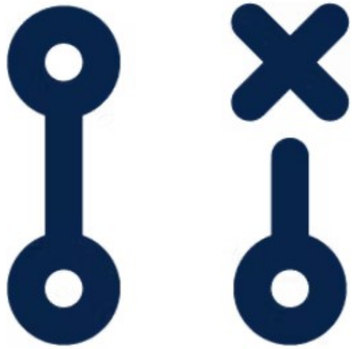
GitFlow es un modelo de ramificación enfocado en **desarrollos a largo plazo** y versiones de software bien definidas. Se organiza principalmente en dos ramas principales:

- `main` (o master): Contiene el código listo para producción.
- `develop` : Contiene el código en desarrollo y es la base para nuevas funcionalidades.

A partir de estas ramas se crean ramas temporales:

- **Feature branches**: Para nuevas características, se crean a partir de `develop` .
- **Release branches**: Preparan una versión para producción.
- **Hotfix branches**: Para arreglar errores urgentes en `main` .

Ejemplos: IntelliJ y proyectos de software empresarial.

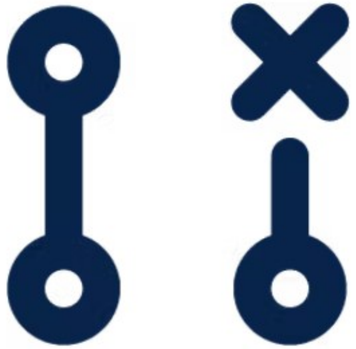


GitLab Flow

GitLab Flow es adecuado para sistemas sofware que no necesiten mantener abiertas ramas de release ni mantenimiento de diferentes versiones del código.

En GitLab Flow se establece una rama para cada entorno de desarrollo y producción. Con cada commit a cada una de estas ramas, se generará un deploy en el entorno. Es un modelo más orientado al CI/CD.

Ejemplos: GitLab.



Trunk-Based Development (TBD)
El **Trunk-Based Development** es un modelo de ramas simple donde los desarrolladores colaboran en una única rama principal llamada **"trunk"** (generalmente `main`).

A diferencia de modelos más complejos como GitFlow, no se crean ramas largas para desarrollar funcionalidades o versiones; en cambio, los cambios se integran frecuentemente en el "trunk". Es ideal para proyectos que usan integración continua y despliegue continuo (**CI/CD**).

Ejemplos: Kernel de Linux, Chrome, Facebook.

Mentimeter

¿Dudas?