# Introducción a Git y Sistemas de Control de Versiones

En el desarrollo de software, mantener un registro claro y organizado de los cambios en el código es fundamental. Los sistemas de control de versiones (SCV) son herramientas diseñadas para cumplir con esta necesidad. Permiten registrar, rastrear, comparar y recuperar versiones anteriores de un proyecto, lo que facilita enormemente la colaboración y el mantenimiento del software.

Cuando varios desarrolladores trabajan sobre un mismo proyecto, es habitual que cada uno implemente nuevas funcionalidades, resuelva errores o actualice documentación. Sin un sistema de control de versiones, coordinar estas tareas se volvería caótico. Git, uno de los SCV más utilizados en la actualidad, ayuda a gestionar estos cambios de manera eficiente, segura y colaborativa.

## ☑ Ventajas de usar un sistema de control de versiones

1. **Historial completo de cambios**: Cada modificación queda registrada junto con el autor, la fecha y un mensaje explicativo. Esto permite auditar los cambios, rastrear errores o incluso deshacer modificaciones si es necesario.
2. **Ramas y fusión**: Facilita el trabajo en paralelo. Se pueden crear "ramas" para trabajar en nuevas características, experimentar o corregir errores sin afectar el código principal. Luego, estas ramas se pueden fusionar al proyecto principal de manera controlada.
3. **Trazabilidad**: Los cambios pueden asociarse con tareas, tickets o bugs específicos. Esto mejora la organización y el seguimiento del progreso del proyecto.
4. **Recuperación ante errores**: Si algo sale mal, siempre se puede volver a un estado anterior del código sin perder el trabajo previo.

## 🗂️ Tipos de sistemas de control de versiones

- **Centralizados (CVCS)**: Todo el proyecto se encuentra en un servidor central. Los desarrolladores descargan archivos para trabajar localmente y luego los suben al servidor. Si el servidor falla, se puede perder información importante.
- **Distribuidos (DVCS)**: Cada usuario tiene una copia completa del repositorio, incluyendo todo el historial. Esto permite trabajar sin conexión y recuperar el proyecto si el servidor deja de funcionar. Git pertenece a este tipo.

## 🧠 ¿Qué es Git?

Git es un sistema de control de versiones distribuido, de código abierto, creado por Linus Torvalds en 2005. Desde entonces, se ha convertido en el estándar de facto en la industria del software.

Con Git, cada desarrollador trabaja con una copia completa del repositorio, lo que permite una gran flexibilidad y autonomía. A través de comandos, se pueden realizar tareas como guardar versiones del proyecto, colaborar con otros desarrolladores, revisar el historial o resolver conflictos.

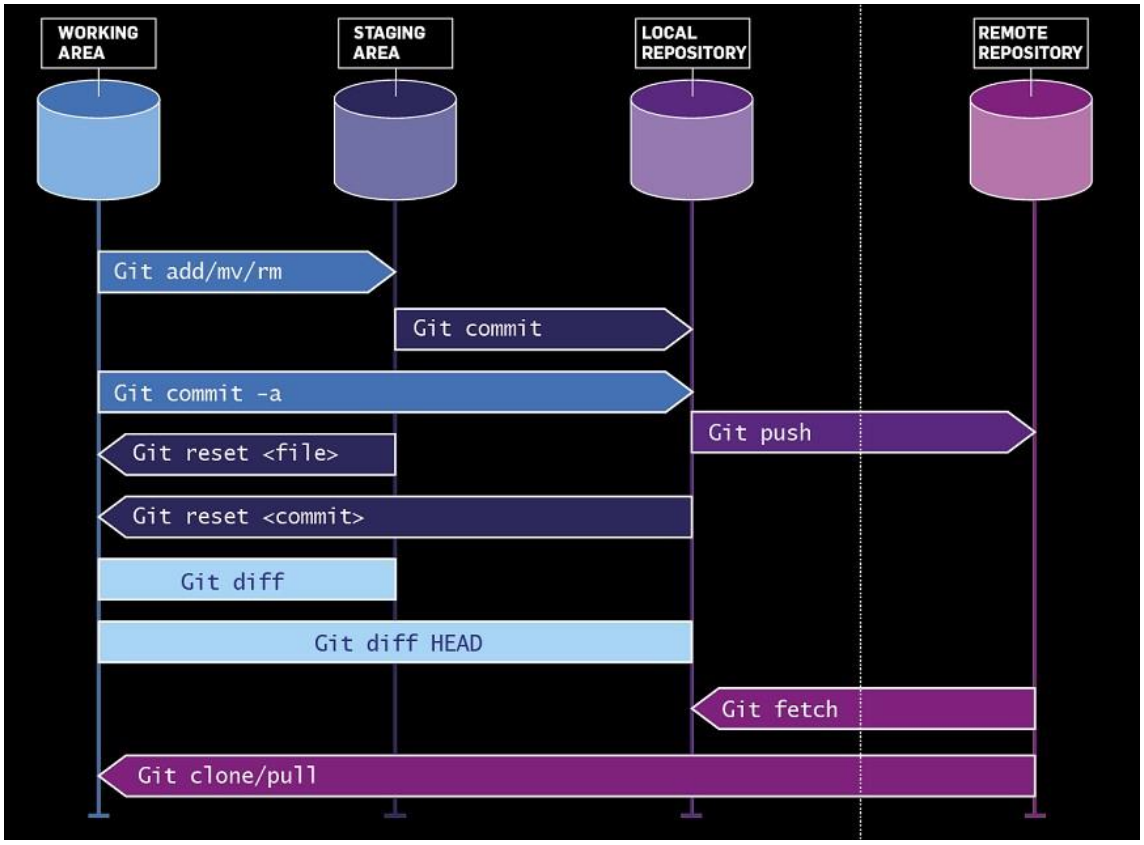
## ✨ Características clave de Git

- **Rendimiento**: Git es extremadamente rápido para registrar cambios, comparar versiones o fusionar ramas, incluso en proyectos grandes.
- **Seguridad**: Utiliza un sistema de hash criptográfico (SHA-1) que garantiza la integridad del historial. Cada cambio queda firmemente vinculado a su versión anterior.
- **Flexibilidad**: Soporta múltiples flujos de trabajo (centralizado, colaborativo, por funcionalidades, etc.) y se adapta a todo tipo de proyectos, desde pequeños scripts hasta grandes aplicaciones corporativas.

## 🧱 Áreas y estados en Git

Cuando trabajamos con Git, los archivos del proyecto pueden encontrarse en distintos "estados" dentro de distintas "áreas". Comprender este flujo es clave para utilizar Git correctamente.

- **Working Directory (directorio de trabajo)**: es el área donde se crean o modifican archivos. Son los archivos visibles en tu carpeta de proyecto.
- **Staging Area (área de preparación)**: aquí se colocan los archivos que están listos para ser guardados (commiteados). Es como una lista de espera.
- **Repositorio local**: contiene todo el historial de commits del proyecto. Cada vez que hacés `git commit`, los cambios se almacenan aquí.
- **Repositorio remoto**: generalmente alojado en plataformas como GitHub o GitLab. Permite sincronizar el trabajo con otros desarrolladores.




# 🛠️ Comandos básicos de Git

- `git init`: Crea un repositorio nuevo en tu proyecto.
- `git add <archivo>`: Mueve archivos modificados al área de preparación.
- `git status`: Muestra qué archivos cambiaron y cuál es su estado.
- `git commit -m "mensaje"`: Guarda los archivos preparados como una nueva versión del proyecto.
- `git push`: Sube los commits al repositorio remoto.
- `git pull`: Descarga los cambios desde el repositorio remoto y los fusiona con tu versión local.
- `git clone <url>`: Crea una copia local de un repositorio remoto.

Estos comandos componen el flujo básico de trabajo con Git.

# 🌐 GitHub y el trabajo colaborativo

GitHub es una plataforma en línea que permite almacenar proyectos que usan el sistema de control de versiones Git. Es el servicio más popular del mundo para compartir y colaborar en proyectos de software, tanto de código abierto como privados. Almacena los repositorios en la nube, lo que facilita el trabajo colaborativo entre desarrolladores ubicados en diferentes partes del mundo.

Además de servir como un "espacio remoto" donde subir tu código, GitHub incluye herramientas para organizar, discutir, revisar y aprobar cambios, lo que lo convierte en una solución integral para el desarrollo de software moderno.

## Funcionalidades principales:

- **Issues (Incidencias)**: Registrar errores, tareas, ideas o mejoras.
- **Milestones (Hitos)**: Agrupar issues para organizar entregas o versiones.
- **Labels (Etiquetas)**: Clasificar las issues (bug, enhancement, help wanted, etc.).
- **Projects**: Usar tableros tipo Kanban para organizar visualmente el trabajo.
- **Pull requests**: Proponer cambios a proyectos. Permiten revisiones, comentarios y aprobaciones antes de fusionar el código.

GitHub también ofrece acciones automatizadas (GitHub Actions), integración con herramientas externas, estadísticas de contribución y perfiles colaborativos.

# 🌿 Ramas en Git

Una **rama** es una versión paralela del proyecto. Se usa para desarrollar nuevas funcionalidades sin afectar la rama principal (`main` o `master`).

### Ventajas de usar ramas:

- Evitás modificar código estable hasta que la nueva funcionalidad esté lista.
- Podés experimentar sin comprometer el trabajo de otros.
- Facilita la revisión y el control de calidad antes de fusionar cambios.

## 🔧 Comandos útiles para ramas

- `git branch`: Lista las ramas locales.
- `git branch <nombre>`: Crea una nueva rama.
- `git checkout <nombre>`: Cambia a otra rama.
- `git checkout -b <nombre>`: Crea y cambia a la nueva rama en un solo paso.
- `git merge <rama>`: Fusiona una rama con la actual.
- `git branch -d <nombre>`: Elimina una rama local.
- `git push origin --delete <nombre>`: Elimina una rama remota.

# Ejemplo práctico de flujo con ramas

Una vez que ya tenés un repositorio iniciado (es decir, que ejecutaste `git init` o clonaste uno existente), podés comenzar a experimentar con el uso de ramas. Este ejemplo te muestra cómo crear una rama nueva, hacer cambios sobre ella y luego fusionarlos con la rama principal.

1. Crear y cambiar de rama:

```
git checkout -b nueva-funcion
```

2. Hacer cambios, agregarlos y confirmarlos:

```
git add archivo.js
git commit -m "Agrega nueva funcionalidad"
```

3. Cambiar a la rama principal y fusionar:

```
git checkout main
git merge nueva-funcion
```

4. Eliminar la rama si ya no es necesaria:

```
git branch -d nueva-funcion
```