

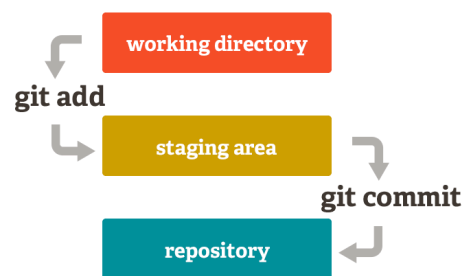


Git

Git es un sistema de control de versiones distribuido utilizado para el desarrollo de software y otros proyectos. Fue creado en 2005 por Linus Torvalds, el creador del sistema operativo Linux. Git permite a los desarrolladores trabajar juntos en proyectos, manteniendo un registro de todos los cambios realizados en el código fuente. Esto facilita la colaboración y permite a los desarrolladores revertir cambios en caso de errores.

Ciclo de trabajo

Un proyecto en Git tiene tres secciones principales:



- El **Working Directory** es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que puedan ser usados.
- El **Staging Area** es un área que almacena información acerca de lo que va a ir en la próxima confirmación. A veces se le denomina índice ("Index").
- En el **Repository** se almacenan los metadatos y la base de datos de objetos para tu proyecto. Es lo que se copia cuando clonas un repositorio desde otra computadora.

El flujo de trabajo básico puede resumirse como:

1. Modificar una serie de archivos en el directorio de trabajo.
2. Preparar los archivos, añadiéndolos a tu área de preparación (staging).
3. Confirmar los cambios (commit), lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en el directorio de Git.

Ramas

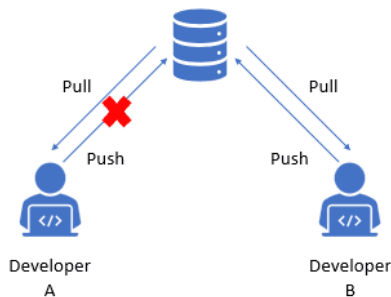
Una rama en Git es una versión independiente del código base que permite a los desarrolladores trabajar en diferentes partes de un proyecto de manera aislada. Cada rama puede contener sus propios cambios y desarrollos sin afectar el código principal.



Resolución de conflictos

En circunstancias normales Git resuelve las diferencias entre archivos automáticamente. Sin embargo, cuando las diferencias están en las mismas líneas de código, Git requiere la intervención humana para resolver estas diferencias. Este caso se denomina conflicto de fusión o, simplemente, conflicto.

Por ejemplo, digamos que hay dos desarrolladores, A y B, trabajando en la misma rama. Ambos desarrolladores extraen el mismo archivo del repositorio remoto y hacen cambios en él.



El desarrollador B completa su tarea antes y devuelve el archivo al servidor.

De manera similar, el desarrollador A completa su trabajo y procede a impulsarlo. Sin embargo, se enfrenta a un conflicto cuando intenta empujar el archivo. Este conflicto se debe a que el mismo archivo ya se reemplazó en el repositorio remoto con una versión actualizada del desarrollador B.

Git proporciona una ayuda diciéndonos que archivo tiene el conflicto, al abrir el archivo se nos muestra cuáles son los cambios tanto de una rama como de la otra.

Ahora tenemos la responsabilidad de elegir entre lo que está entre `<<<<<< HEAD` y `=====`, que es contenido que tenemos en la rama donde estamos haciendo el merge (main) o entre `=====` y `>>>>>> contenido`, donde están los cambios hechos en la rama que queremos unir (contenido).

```
<!DOCTYPE HTML>
<html>
  <head>
    <<<<<< HEAD
      <title>Nuevo Titulo</title>
    =====
      <title>Nuevo Titulo para la web</title>
    >>>>>> contenido
  </head>
  <body>
    <p>Contenido de la web</p>
    <p>Nuevo párrafo de la página</p>
  </body>
</html>
```

Una vez modificado el archivo con los cambios elegidos, guardamos, agregamos y hacemos commit de los cambios.

Otra manera para resolver los conflictos es que podemos indicarle de antemano a git que estrategia tomar cuando tiene que decidir un conflicto, esto con las opciones `ours` y `theirs`, de esta manera:

```
git merge -s recursive -X theirs rama-a-fusionar
```

```
git merge -s recursive -X ours rama-a-fusionar
```

Esto cuando queremos que git resuelva el conflicto usando los cambios de la rama a fusionar (theirs o suyos) y cuando queremos que tome los cambios de la rama donde se está fusionando (ours o nuestros)

GitHub

GitHub es una plataforma basada en la web que utiliza el sistema de control de versiones Git para gestionar y almacenar código fuente.

Aquí están algunas de las características y funcionalidades clave de GitHub:

1. **Repositorio de código:** GitHub aloja repositorios donde los desarrolladores pueden almacenar, gestionar y compartir sus proyectos de código fuente.
2. **Control de versiones:** Al integrar Git, GitHub permite a los desarrolladores realizar seguimiento de los cambios en el código, revertir a versiones anteriores y colaborar en el desarrollo simultáneo de múltiples versiones de un proyecto.
3. **Colaboración:** GitHub facilita la colaboración mediante herramientas como pull requests, que permiten a los desarrolladores proponer cambios y revisiones de código antes de fusionarlos con el proyecto principal.
4. **Documentación y wikis:** Los proyectos en GitHub pueden incluir documentación detallada y wikis para ayudar a los desarrolladores y usuarios a comprender y utilizar el código.
5. **Issues y gestión de proyectos:** GitHub proporciona una funcionalidad para rastrear errores (issues), sugerir mejoras y gestionar tareas, facilitando la planificación y el seguimiento del progreso de los proyectos.
6. **Integraciones y CI/CD:** GitHub se integra con muchas otras herramientas y servicios, incluyendo sistemas de integración continua y entrega continua (CI/CD) como GitHub Actions, permitiendo automatizar pruebas, despliegues y otras tareas.
7. **Comunidad y redes sociales:** GitHub no solo es una herramienta de desarrollo, sino también una comunidad donde los desarrolladores pueden seguir a otros, contribuir a proyectos de código abierto y participar en discusiones técnicas.

Comandos

A continuación se muestran los comandos básicos para poder trabajar con Git

Configuración

`git config --global user.name "name"` → Establece el nombre del usuario

`git config --global user.email "mail"` → Establece el correo del usuario

`git config --global init.defaultBranch <name>` → Establece el nombre de la rama default

`git config --global core.editor "code --wait"` → Establecer VS como editor

`git config --global alias.tree "log --graph"` → Crea Alias para simplificar líneas de instrucciones, Ahora se usaría `git tree`

Trabajo

`git init` → Inicia el repositorio

`git status` → Revisar el estado del fichero

`git add <File>` → Activa el seguimiento de un archivo

`git rm --cached <File>` → Desactivar el seguimiento de un archivo

`git commit -m <Mensaje>` → Realiza un commit con un mensaje asociado

`git commit --amend <Mensaje>` → Cambiar el mensaje del último commit

`git log` → Muestra el historial de commits

`git log --graph --pretty=oneline` → Muestra el historial de commits con formato

`git diff <Nombre de archivo>` → Ver las diferencias entre commits

`git checkout <HASH/Nombre de Rama>` → Navegar entre las ramas/versiones

`git reset --hard <HASH>` → Devuelve al commit especificado

`git branch` → Permite ver que ramas existen

`git branch <NAME>` → Crea una nueva rama

`git branch -m <VIEJO NOMBRE> <NUEVO NOMBRE>` → Cambia el nombre de la rama

`git branch -d <NOMBRE>` → Eliminar una Rama

`git merge <NOMBRE RAMA A FUSIONAR>` → Se realiza la fusión entre ramas, el merge se hace estando en la rama objetivo

`git merge --continue` → Termina el MERGE cuando hay un conflicto

Remoto

`git clone <HTTPS>` → Clonar un repositorio al local (también funciona con SSH)

`git remote add origin <HTTPS>` → Establecer un repositorio remoto (también funciona con SSH)

`git remote -v` → Visualizar el estado del repositorio remoto

`git pull origin main` → Trae la rama main del origin a mi main fusionando

`git fetch origin` → Trae la rama main del origin a mi main sin fusionar

`git push origin <Nombre de rama>` → Manda la rama al origin (se utiliza `--force` para forzar)

`git remote remove origin` → Elimina la dirección en origin
