Conflictos en Git y Ramas

Guía práctica para usar en un Jupyter Notebook

Qué es un conflicto en Git

Los conflictos en Git son situaciones comunes que todo desarrollador encuentra. Comprender su naturaleza es el primer paso para manejarlos con confianza.

Definición

Un conflicto ocurre cuando dos cambios afectan exactamente la misma parte del código y Git no puede determinar automáticamente cuál debería prevalecer.

Cuándo sucede

Los conflictos aparecen durante operaciones como merge, rebase o pull, cuando diferentes versiones del mismo archivo intentan combinarse.

Resolución manual

Git no puede decidir qué versión es correcta por sí solo. El desarrollador debe intervenir manualmente para elegir o combinar los cambios.

Cómo aparece un conflicto en un archivo

Cuando Git detecta un conflicto, marca claramente las secciones problemáticas en tu archivo usando marcadores especiales. Reconocer esta estructura es fundamental para resolver conflictos eficientemente.

Estructura del conflicto

Los marcadores dividen el archivo en tres secciones claramente identificables:

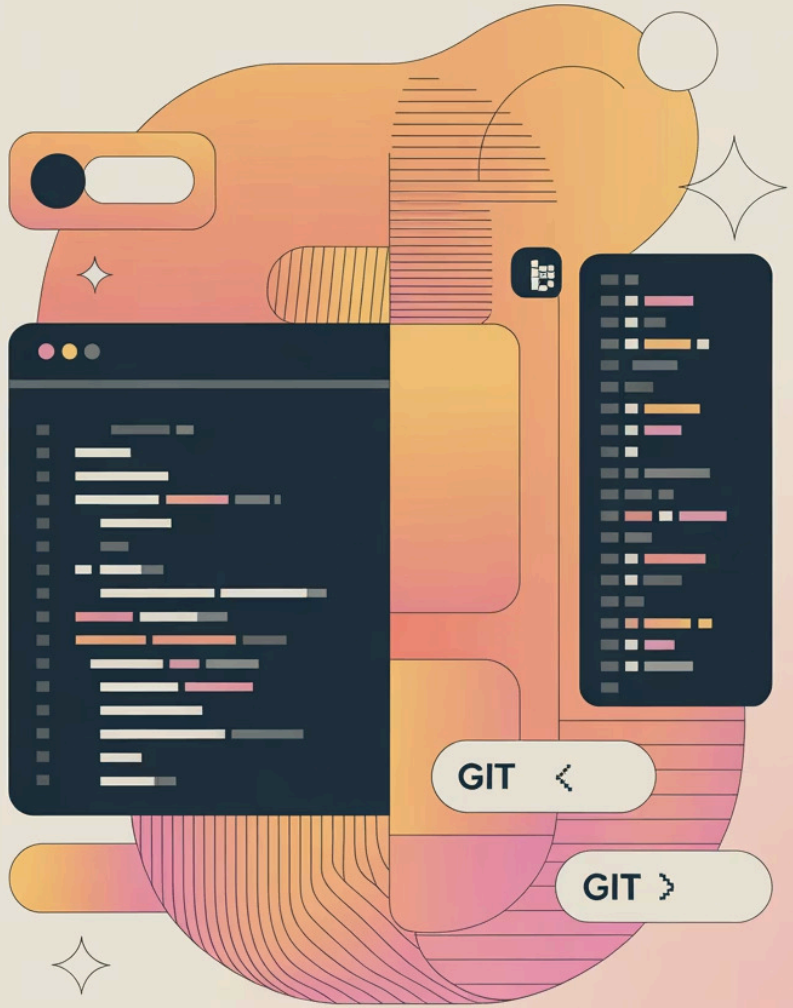
- **HEAD:** Tu versión actual del código
- **Separador (=):** Divide ambas versiones
- **Rama remota:** La versión entrante

Estos marcadores te permiten ver ambas versiones lado a lado y decidir cuál mantener.

Ejemplo visual

```
<<<<<<< HEAD
Código en tu rama
=======
Código en la otra rama
>>>>>>> rama-remota
```

Los símbolos < y > indican el inicio y fin de cada sección conflictiva.



Pasos para resolver conflictos

Resolver un conflicto en Git sigue un proceso sistemático y predecible. Aunque puede parecer intimidante al principio, estos pasos te guiarán de manera efectiva.

Abrir el archivo con conflicto

Localiza e identifica los archivos marcados con conflicto. Git te indicará exactamente cuáles archivos requieren tu atención.

Editar y dejar solo la versión correcta

Elimina los marcadores de conflicto y decide qué código mantener. Puedes elegir una versión, combinar ambas, o escribir algo completamente nuevo.

Guardar cambios

Una vez editado el archivo, guarda todos los cambios. Asegúrate de que no queden marcadores de conflicto en el código.

Ejecutar: git add .

Añade los archivos resueltos al área de preparación (staging area) para indicarle a Git que el conflicto ha sido solucionado.

Ejecutar: git commit

Completa el proceso con un commit que registra la resolución del conflicto en el historial del proyecto.

Ejemplo práctico en Jupyter Notebook

1	2	3
Simula cambios en dos ramas	**Haz un merge para provocar un conflicto**	**Muestra la resolución dentro del notebook**
Crea dos ramas diferentes en tu repositorio y realiza modificaciones distintas en el mismo notebook o archivo .py. Esto replicará el escenario típico de trabajo en equipo.	Intenta fusionar las ramas usando `git merge`. Git detectará las diferencias en el mismo archivo y generará un conflicto que podrás observar directamente.	Abre el notebook en Jupyter, observa los marcadores de conflicto, resuélvelo editando las celdas afectadas, y completa el proceso con git add y git commit.

Qué es una rama en Git

Las ramas son uno de los conceptos más poderosos de Git. Permiten un desarrollo paralelo y organizado, facilitando la colaboración entre equipos.

Línea independiente de desarrollo

Una rama es esencialmente una copia independiente de tu código donde puedes experimentar y desarrollar sin afectar el trabajo principal.

Protege la rama main

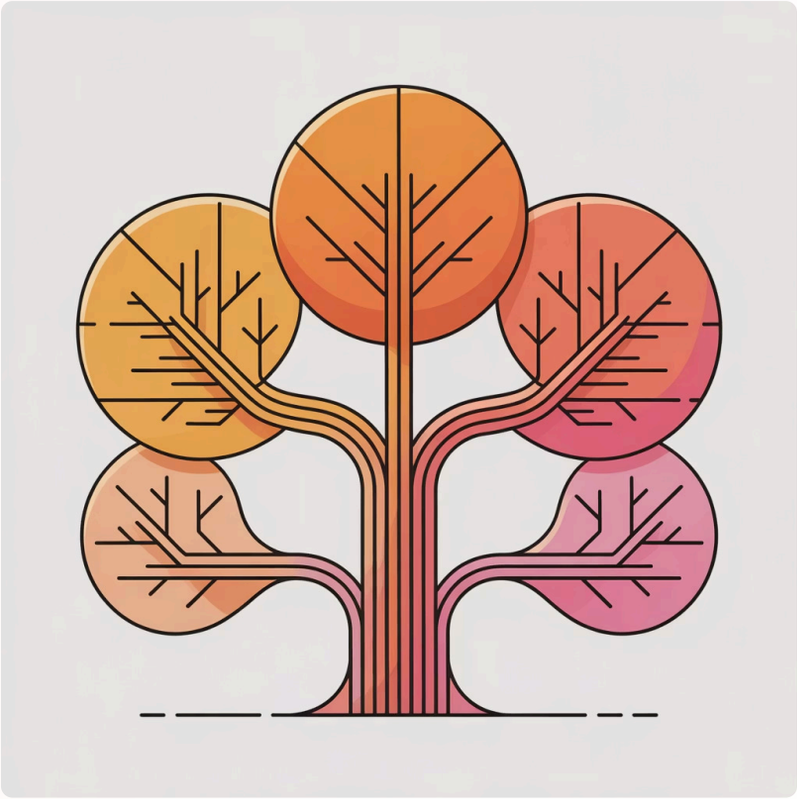
Permite trabajar en nuevas funcionalidades, correcciones de errores o experimentos sin tocar la rama principal (main o master).

Facilita la colaboración

Múltiples desarrolladores pueden trabajar simultáneamente en diferentes funcionalidades sin interferir entre sí.

Fusión controlada

Cuando el trabajo está completo y probado, se puede fusionar de vuelta a la rama principal de manera controlada.



Comandos básicos

Dominar estos comandos esenciales te permitirá gestionar ramas con eficiencia y fluidez en tu flujo de trabajo diario.

Crear una rama

```
git branch nombre-rama
```

Crea una nueva rama pero no cambia a ella automáticamente. La rama se crea desde tu posición actual.

Cambiar de rama

```
git checkout nombre-rama
```

Cambia tu directorio de trabajo a la rama especificada. Todos los cambios posteriores afectarán esta rama.

Crear y cambiar

```
git checkout -b nombre-rama
```

Atajo útil que combina los dos comandos anteriores: crea la rama y cambia a ella inmediatamente.

Fusionar

```
git merge nombre-rama
```

Integra los cambios de la rama especificada en tu rama actual. Aquí es donde pueden aparecer conflictos.

Consejo: Usa `git branch` sin argumentos para ver todas tus ramas locales. La rama actual estará marcada con un asterisco (*).

Flujo de trabajo recomendable

Adoptar un flujo de trabajo estructurado minimiza conflictos y mantiene tu proyecto organizado. Estas prácticas son estándares en la industria del desarrollo de software.

Crear una rama para cada feature

Cada nueva funcionalidad, corrección de bug o experimento debe tener su propia rama dedicada. Esto mantiene el trabajo aislado y organizado.

Hacer commits pequeños

Commits frecuentes y específicos facilitan el seguimiento de cambios, la depuración y la reversión si algo sale mal. Cada commit debe representar una unidad lógica de trabajo.

Hacer merge frecuente

Integra cambios de la rama principal regularmente en tu rama de trabajo. Esto reduce la divergencia y hace que los conflictos eventuales sean más pequeños y manejables.

Resolver conflictos cuanto antes

No pospongas la resolución de conflictos. Cuanto más tiempo pase, más compleja será la integración y mayor el riesgo de errores.