git

GitHub

¿Por qué usar ramas?

- Permiten trabajar en múltiples tareas al mismo tiempo.
- Evitan conflictos entre desarrolladores.
- Mantienen el branch principal (main) estable y listo para ser implementado.

Tipos de Ramas

a. Main (o master)

- Es la rama principal y debe contener el código más estable y listo para producción.

b. Feature branches (ramas de características)

- Se crean para desarrollar nuevas características o funcionalidades.
- **Ejemplo:** feature/jwt-auth, feature/user-profile.
- Una vez completadas y probadas, se fusionan con main.

c. Hotfix branches (ramas de correcciones urgentes)

- Se utilizan para solucionar errores críticos en producción.
- **Ejemplo:** hotfix/fix-login-bug.

d. Release branches (ramas de versiones)

- Preparan una versión específica para producción.
- Permiten corregir pequeños errores y realizar ajustes finales.
- **Ejemplo:** release/1.0.0.

e. Develop (opcional)

- Una rama intermedia entre main y las ramas de características.
- A menudo utilizada en proyectos grandes para probar cambios antes de moverlos a producción.

Estructura general de la nomenclatura de ramas

<tipo>/<id-issue>-<descripción-corta>

<tipo>

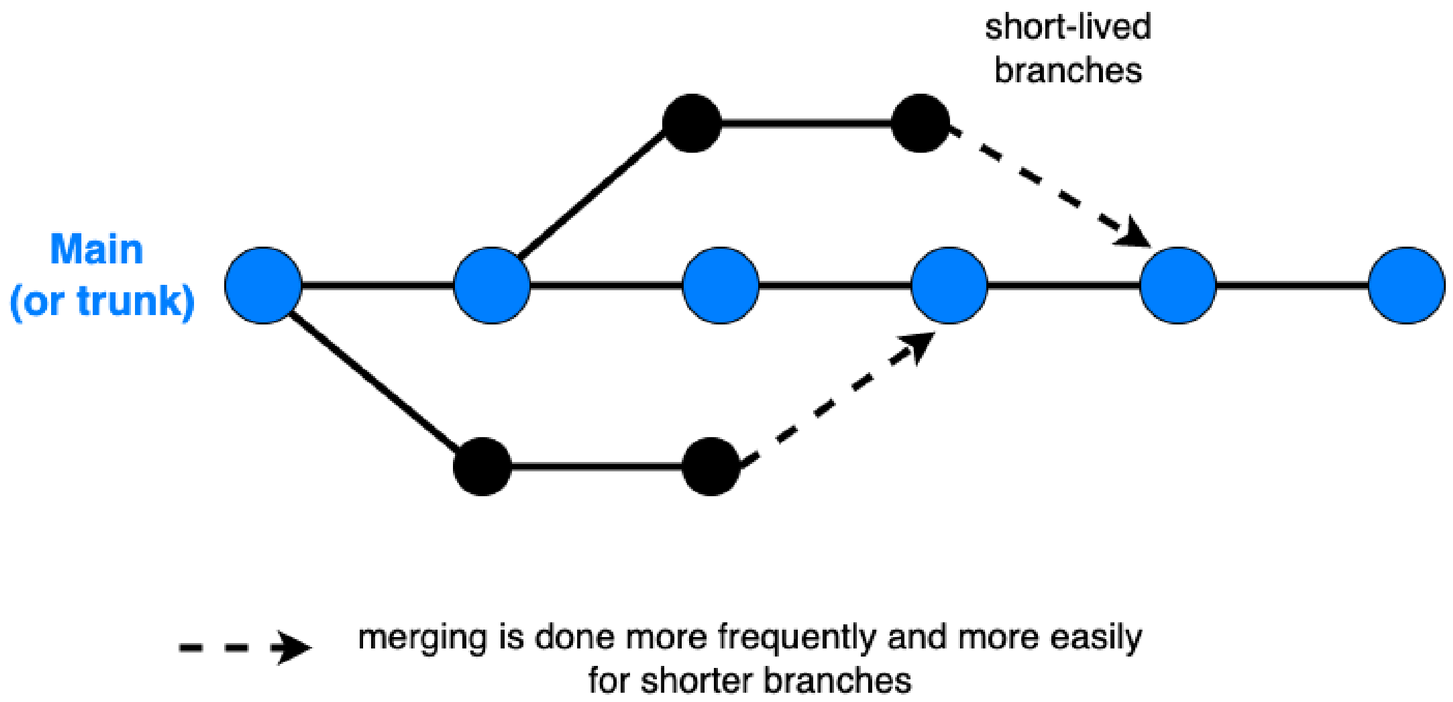
Representa el propósito de la rama. Utiliza los mismos prefijos que en los commits para mantener consistencia.

- **feat:** Nueva funcionalidad o característica.
- **fix:** Corrección de errores.
- **docs:** Cambios en documentación.
- **refactor:** Refactorización del código.
- **chore:** Tareas menores o de mantenimiento.
- **test:** Pruebas.
- **build:** Cambios relacionados al sistema de compilación.
- **perf:** Mejoras de rendimiento.

Ejemplos:

- fix/12-login-dont-allow-retry-bug
- refactor/16-reorganizar-vistas-en-carpetas-por-modulo
- perf/19-corregir-consultas-Qn+1
- feat/33-implementar-funcion-para-convertir-word-a-pdf
- docs/31-proceso-para-levantar-proyecto-con-docker-readme

Trunk Based Development

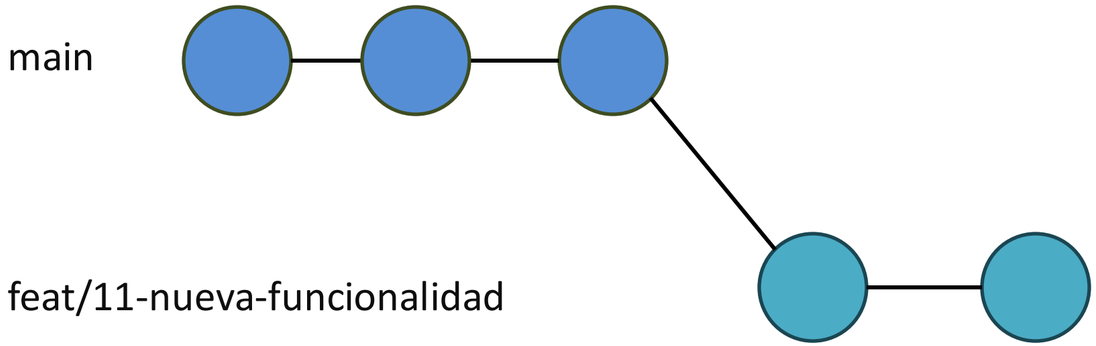


Crear una rama:

git branch nombre_rama

Analogía: Imagina que tienes un libro donde escribes tu historia principal (la rama main).

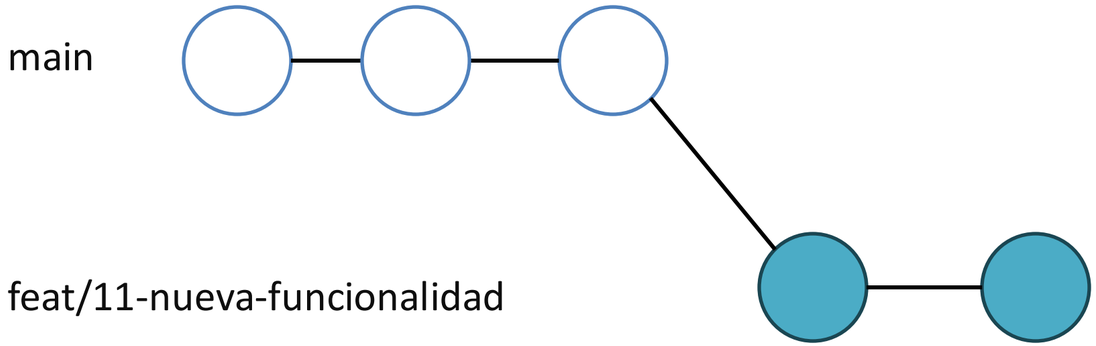
Crear una rama es como tomar una hoja nueva para escribir una versión alternativa o un capítulo diferente de la historia. Esa hoja nueva no afecta el contenido original hasta que decides integrarla.



Cambiar de rama:

git switch nombre_rama

Analogía: Cambiar de rama es como cambiar de carpeta en tu computadora. Cada rama es una carpeta con una versión diferente del proyecto. Si estás trabajando en una carpeta distinta, no puedes ver lo que hay en la otra hasta que vuelvas a cambiar.



Trabajar en una rama:

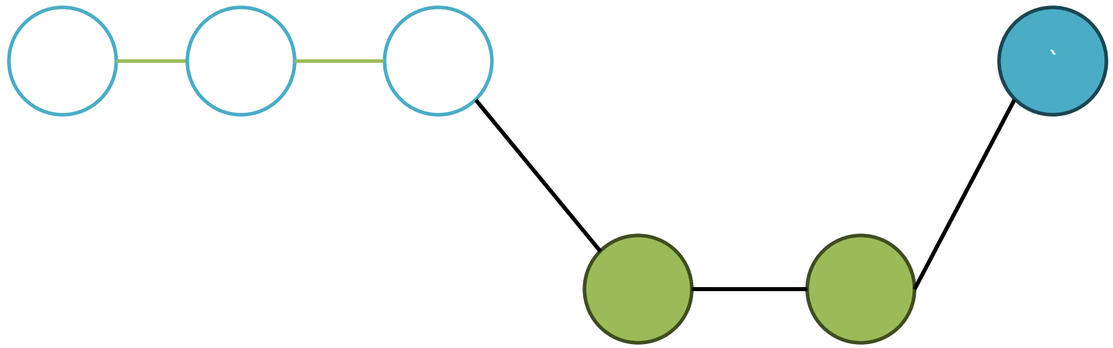
git switch nombre_rama

1. Realiza cambios en los archivos.
2. Usa git add archivo para preparar los cambios.
3. Usa git commit -m "mensaje" para guardar los cambios en el historial de esa rama.

4. Usa git pull para traer algun cambio realizado sobre esa rama.

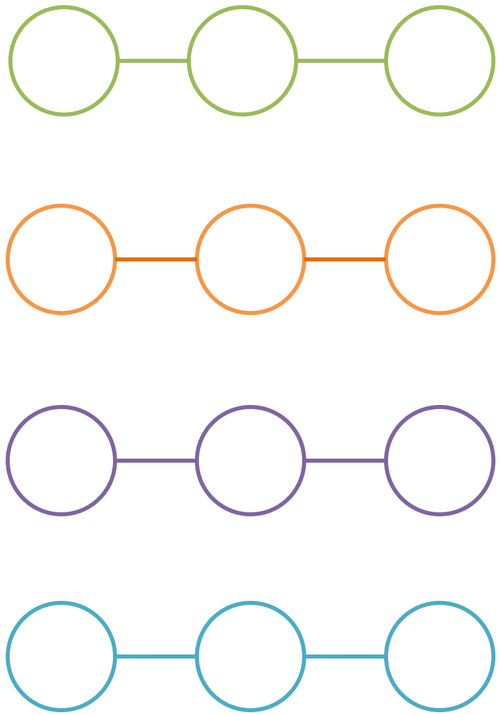
   git pull origin nombre-rama

5. Usa git push para subir los cambios



Visualizar ramas existentes

git branch



Estructura general del mensaje de commit

<tipo>/<descripción-corta>

Tipo

- **feat:** Para nuevas funcionalidades o características.
- **fix:** Para corrección de errores (bugs).
- **docs** Cambios en documentación (README, comentarios, etc.).
- **style** Cambios de formato o estilo (espacios, puntos y comas, etc., sin modificar lógica).
- **refactor** Refactorización de código sin cambios en funcionalidad o comportamiento.
- **test** Agregar o modificar pruebas.
- **chore** Tareas menores (actualización de dependencias, cambios de configuración).
- **perf** Mejoras en el rendimiento o optimización.
- **build** Cambios en el sistema de compilación o herramientas de desarrollo.
- **ci** Cambios en integración continua o scripts relacionados con el pipeline.
- **revert** Revertir un commit previo.

Rebase Interactivo

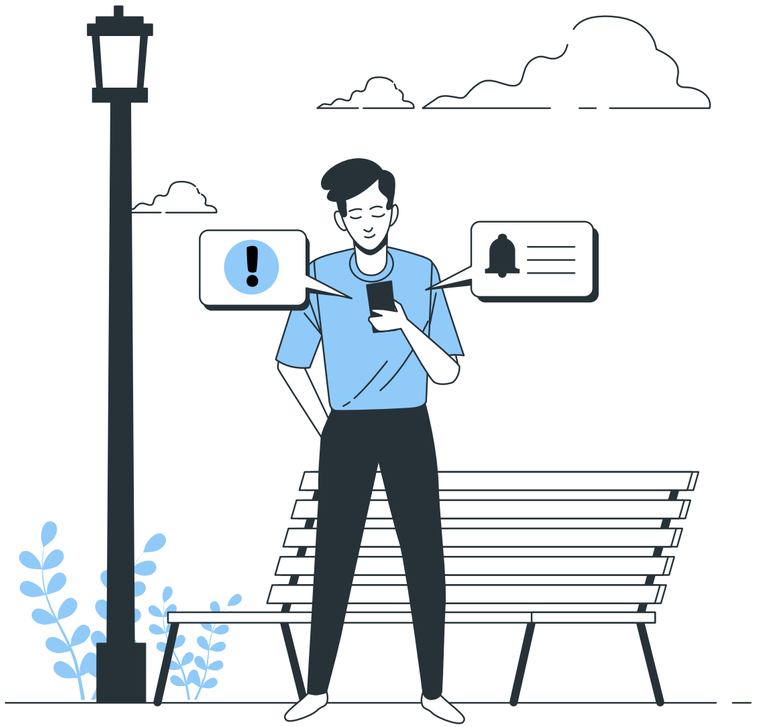
git rebase -i HEAD~3

El rebase interactivo (git rebase -i) es una herramienta poderosa en Git que te permite reorganizar, editar, combinar, eliminar o reescribir mensajes de commits en un historial. Es útil para limpiar el historial de commits antes de compartir tu trabajo.

- **pick** Mantener el commit tal como está.
- **reword** Cambiar el mensaje del commit, manteniendo los cambios de código.
- **edit** Pausar para modificar el commit (*puedes cambiar el contenido o el mensaje*).
- **squash** Combinar este commit con el anterior.
- **fixup** Similar a squash, pero descarta el mensaje del commit actual.
- **drop** Eliminar este commit del historial.

¿Qué es un Pull Request?

Un **Pull Request (PR)** es una funcionalidad en plataformas de control de versiones como GitHub, GitLab o Bitbucket, que permite a los desarrolladores notificar a otros miembros del equipo que han completado cambios en una rama y solicitan que esos cambios se revisen e integren (o “fusionen”) en una rama principal, como main o develop.



¿Para qué sirve un Pull Request?

1. **Revisión de código colaborativa**:
    - Permite que otros revisen tu código antes de integrarlo en el proyecto principal.
    - Ayuda a detectar errores, inconsistencias o mejoras posibles.
2. **Facilitar la integración de cambios**:
    - Consolida el trabajo de diferentes desarrolladores.
    - Garantiza que los cambios sean compatibles con el resto del código.
3. **Trazabilidad**:
    - Registra qué cambios se han hecho, por qué y quién los hizo.
    - Facilita la auditoría y el mantenimiento del código.
4. **Integración con pruebas automáticas**:
    - Ejecuta pipelines de CI/CD (pruebas, compilación, etc.) antes de aprobar el PR.