Git en Entornos Colaborativos

August 22, 2025

Introducción

- Git es el sistema de control de versiones más utilizado en desarrollo de software.
- Facilita el trabajo colaborativo, incluso en equipos distribuidos.
- Aporta orden, trazabilidad y seguridad al proceso de programación.
- Conocer comandos intermedios y avanzados optimiza el flujo de trabajo.

Importancia en entornos colaborativos

- Coordina equipos grandes sin sobrescribir cambios.
- Permite ramas paralelas para nuevas características o pruebas.
- Registra cada modificación con autor, fecha y motivo.
- Compatible con metodologías ágiles, CI/CD y despliegue automatizado.

Comandos de Git (I)

- `git stash`
  Guarda cambios temporales sin hacer commit.
- `git log --oneline --graph --decorate`
  Visualiza historial de commits resumido.
- `git reset {commit}`
  Regresa a un estado anterior descartando commits.

Comandos de Git (II)

- `git checkout -b feature/login-page`
  Crea y mueve el HEAD a una nueva rama llamada `feature/login-page`.
- `git cherry-pick {commit}`
  Copia un commit específico desde otra rama.

Comandos de Git (III)

- `git reflog`
  Registra todos los movimientos de HEAD.
- `git blame archivo`
  Muestra quién modificó cada línea de un archivo.

¿Qué es un rebase?

- El `git rebase` se utiliza para integrar los cambios de una rama en otra, pero a diferencia del `merge`, mantiene un historial más limpio y lineal.
- Durante un rebase, Git "reaplica" tus commits sobre otro commit base (por ejemplo, la rama `main`).
- El uso del `-i` en `git rebase -i` te permite realizar un rebase interactivo, donde puedes reordenar, modificar, o combinar commits (squash).

Rebase interactivo

- El rebase interactivo es útil cuando quieres limpiar tu historial antes de hacer un push a un repositorio compartido.
- Los comandos más comunes en un rebase interactivo son:
  - `pick`: Mantiene el commit tal cual está.
  - `squash`: Combina el commit con el anterior.
  - `edit`: Modifica el commit.
  - `drop`: Elimina el commit.
- Ejemplo de rebase interactivo:
  - `git rebase -i HEAD 3`
    Inicia el rebase interactivo de los últimos tres commits.

Ejemplo: guardar cambios temporales

- `git stash`
  Guarda los cambios no confirmados de forma temporal.
- `git checkout main`
  Cambia a la rama principal `main`.
- `git pull origin main`
  Obtiene los últimos cambios de `main`.
- `git stash pop`
  Aplica los cambios guardados previamente con `git stash`.
- Uso: Cuando surge una tarea urgente en otra rama.
- Beneficio: Cambio de contexto rápido sin ensuciar historial.

Ejemplo: revertir y limpiar historial

- `git reset --hard abc123`
  Revierte a un commit específico, descartando cambios posteriores.
- `git rebase -i HEAD 3`
  Realiza una rebase interactiva para modificar los últimos tres commits.
- Uso: Volver a un estado anterior o reorganizar commits.
- Beneficio: Historial limpio y entendible.

Ejemplo: integración y rebase

- `git checkout main`
  Cambia a la rama principal `main`.
- `git pull origin main`
  Obtiene los últimos cambios de `main`.
- `git checkout feature/login-page`
  Cambia a la rama `feature/login-page`.
- `git rebase main`
  Integra los cambios de `main` en la rama `feature/login-page`.
- `git rebase --continue`
  Continúa el rebase después de resolver conflictos.
- `git push origin feature/login-page`
  Empuja la rama `feature/login-page` al repositorio remoto.
- Uso: Integrar rama con la versión más reciente de `main`.
- Beneficio: Historial lineal y claro.

¿Qué es un cherry-pick?

- El `git cherry-pick` permite aplicar cambios específicos de un commit en otra rama.
- A diferencia de un `merge` o un `rebase`, el `cherry-pick` no integra toda la rama, sino un solo commit individual.
- Es útil cuando necesitas aplicar una corrección o una mejora de una rama a otra sin mover todo el historial de commits.

¿Cuándo usar cherry-pick?

- Cuando necesitas traer un commit de una rama a otra sin fusionar ramas completas.
- Ideal para situaciones en las que se desea corregir errores o añadir características de una rama a otra sin alterar su historial.
- Usado frecuentemente para trasladar correcciones de errores de ramas de desarrollo a ramas de producción.

Ejemplo de cherry-pick

- Comando básico:
  - `git cherry-pick {commit}`: Aplica el commit especificado desde otra rama.
- Ejemplo práctico:
  - Supón que quieres aplicar un commit con ID `abc123` desde la rama `feature/login-page` a la rama `main`.
  - Primero, cambia a la rama `main`:
    - `git checkout main`
  - Luego, aplica el commit:
    - `git cherry-pick abc123`

Conclusión

- Git es esencial para equipos de desarrollo modernos.
- Los comandos intermedios y avanzados brindan control total sobre el código.
- Bien usados, mejoran productividad y reducen errores.
- Dominar Git = colaborar con seguridad y eficiencia.