

Git Básico

Arturo Silvelo

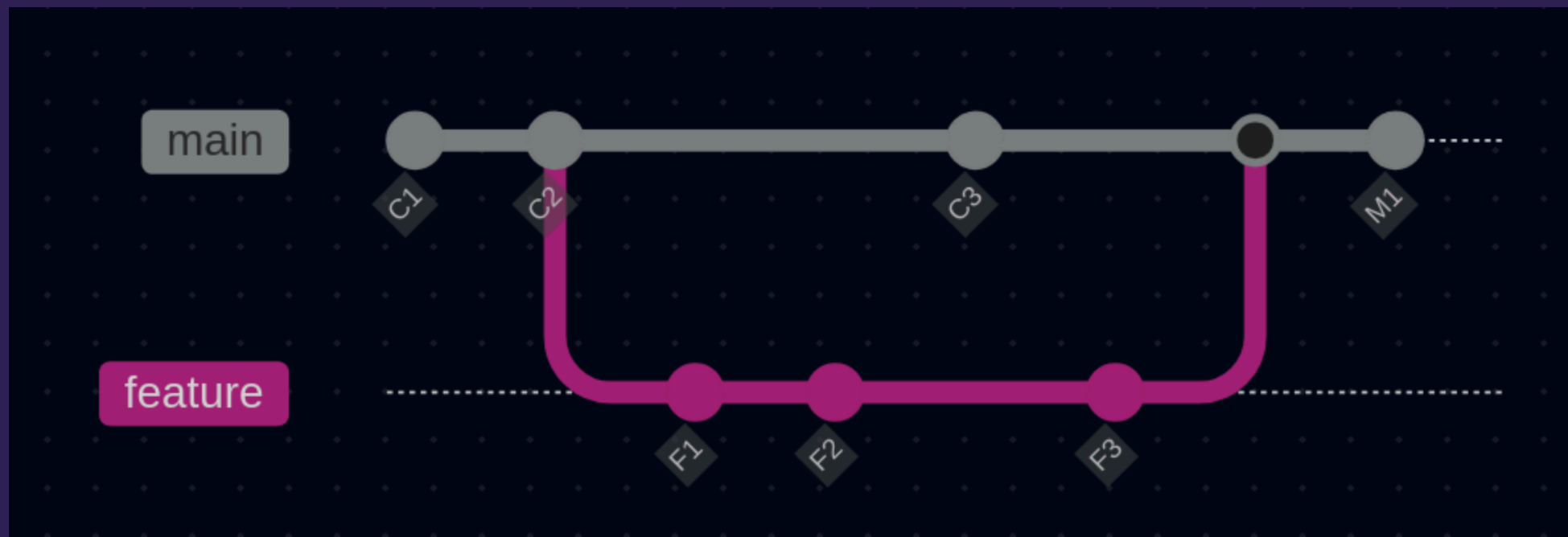
Try New Roads

Ramas

¿Qué es una rama?

Una **rama** (branch) en Git es una línea de desarrollo independiente que permite:

- Trabajar en nuevas características sin afectar el código principal
- Experimentar con cambios de forma segura
- Colaborar en paralelo con otros desarrolladores
- Mantener un historial limpio y organizado



Ventajas de usar ramas

- **Aislamiento:** Los cambios en una rama no afectan otras ramas
- **Experimentación:** Puedes probar ideas sin riesgo
- **Colaboración:** Múltiples desarrolladores pueden trabajar simultáneamente
- **Organización:** Cada característica o bug fix tiene su propia rama
- **Historial limpio:** Facilita el seguimiento de cambios

Crear una rama

Para crear una nueva rama se pueden usar diferentes comandos

- Crear solo la rama (sin cambiar a ella)

```
git branch feature
```

Para cambiar a la rama creada:

```
git switch feature  
# ó  
git checkout feature
```

- Crear la rama y cambiar a ella:

```
git switch -c feature  
git checkout -b feature
```

- Crear rama desde un commit específico:

```
git branch feature abc1234  
git switch -c feature abc1234
```

Listar ramas

- Ver ramas locales:

```
git branch
```

La rama actual tendrá un asterisco al inicio.

- Ver todas las ramas (locales y remotas):

```
git branch -a
```


- Ordenar por fecha de modificación:

```
git branch --sort=-committerdate
```

- Ver información detallada:

```
git branch -v
```

Trabajando Con Ramas

Escenario inicial

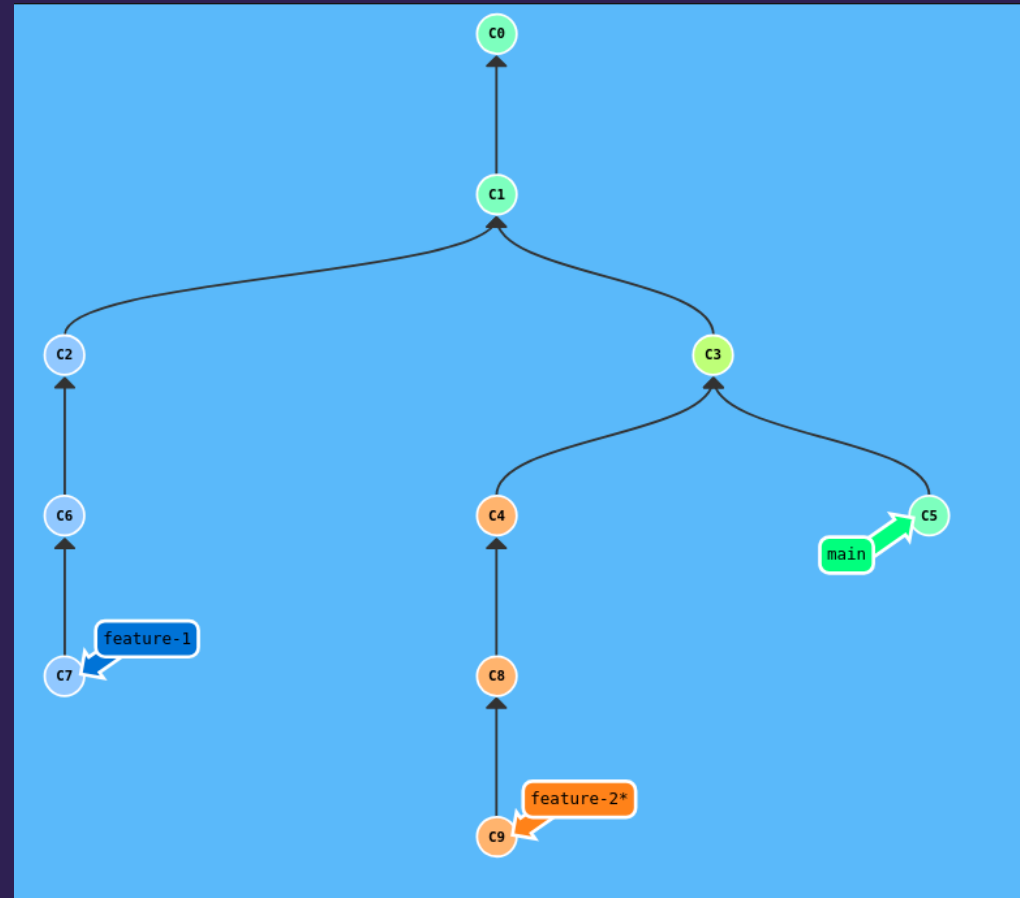
Tu equipo está trabajando en un proyecto y necesitan manejar múltiples tareas simultáneamente.

Tareas a realizar:

1. **Nueva característica:** Te asignan desarrollar `feature-1`
2. **Bug crítico:** Durante el desarrollo encuentras un error que necesita solución inmediata
3. **Desarrollo paralelo:** Un compañero debe crear `feature-2` al mismo tiempo
4. **Segundo bug:** Se reporta otro error en la aplicación principal mientras todos trabajan

Cargar el nivel

1. **Learn Git Branching**
2. Escribimos `import level`
3. Copiamos el fichero `ejercicios_1.json` y lo pegamos.



Fusionar Ramas

¿Qué es fusionar (merge)?

Fusionar es el proceso de **integrar los cambios** de una rama en otra rama.

- Los commits de la rama origen se incorporan a la rama destino
- Se crea un historial unificado
- Permite combinar el trabajo de diferentes desarrolladores
- Mantiene la trazabilidad de los cambios

- Comando básico de fusión

```
git merge <rama-origen>
```

Ejemplo:

```
# 1. Cambiar a la rama destino  
git switch main  
  
# 2. Fusionar la rama feature  
git merge feature-1
```

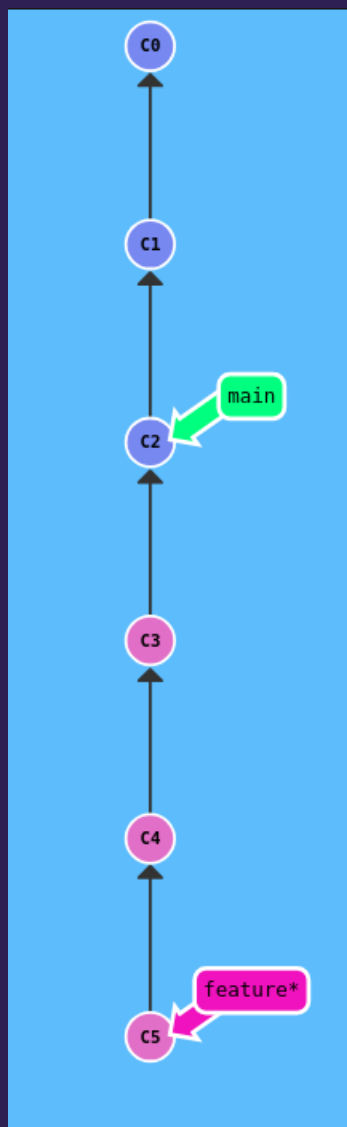
 **Importante:** Siempre debes estar en la rama destino antes de hacer merge

Tipos de merge

Fast-forward

- Cuando no hay commits nuevos en la rama destino
- Simplemente mueve el puntero hacia adelante

```
git merge --ff-only my-branch
```



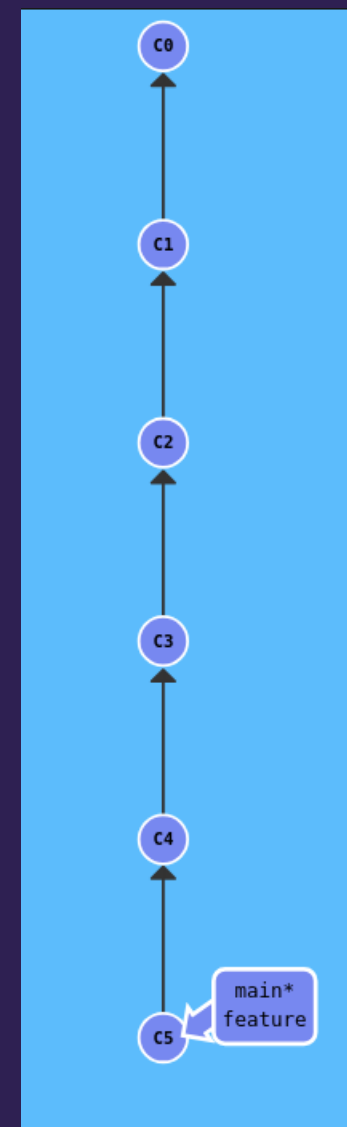
```
# Creamos la nueva rama
git switch -c feature

# Añadimos cambios
git commit
git commit
git commit

# Volvemos a la rama principal
git switch main

# Fusionamos las ramas
git merge --ff-only feature

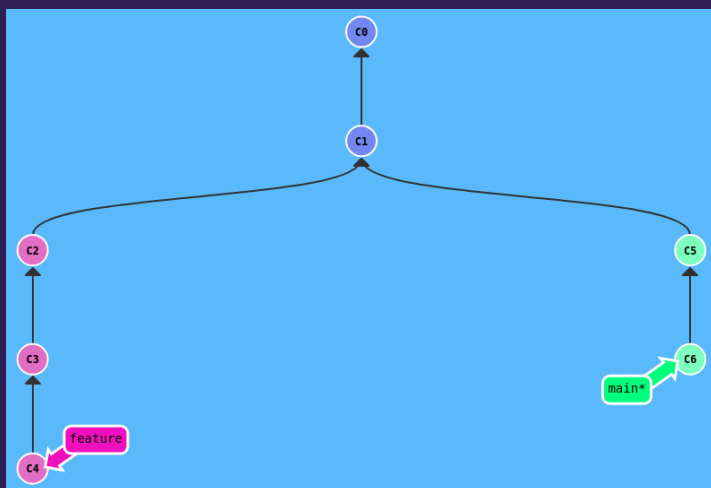
# ó
git merge feature
```



No Fast-forward

- Cuando ambas ramas tienen commits nuevos
- Crea un commit de fusión que une ambas historias
- Preserva la estructura de ramas en el historial

```
git merge --no-ff my-branch
```



```
# Creamos la nueva rama  
git switch -c feature
```

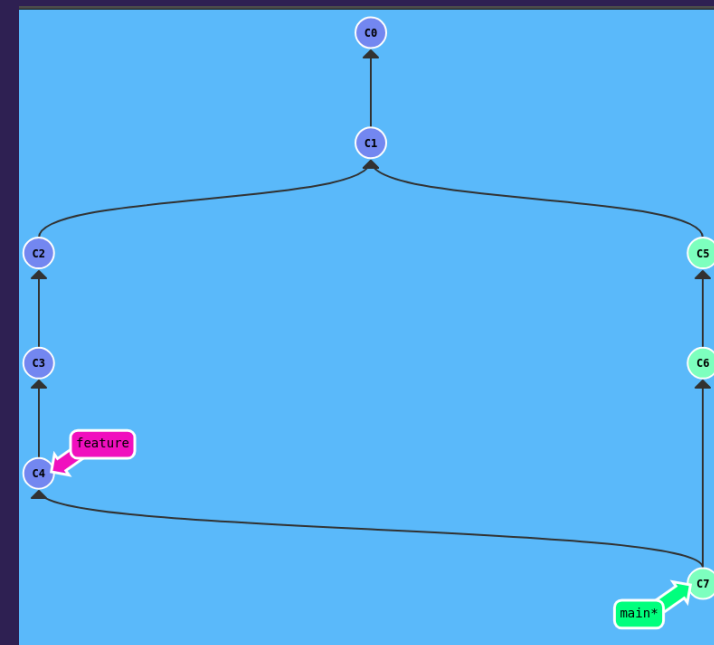
```
# Añadimos cambios  
git commit  
git commit  
git commit
```

```
# Volvemos a la rama principal  
git switch main
```

```
# Añadimos cambios  
git commit  
git commit  
git commit
```

```
# Fusionamos las ramas  
git merge --no-ff feature
```

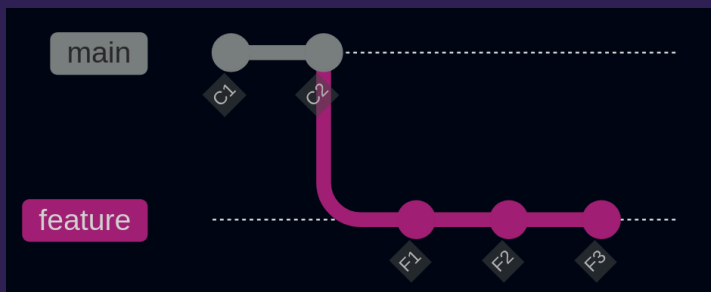
```
# ó  
git merge feature
```



Squash Merge

- Combina todos los commits de una rama en un solo commit
- Crea un historial más limpio y lineal
- Útil cuando la rama tiene muchos commits pequeños o experimentales

```
git merge --squash feature  
git commit -m "Add complete feature X"
```

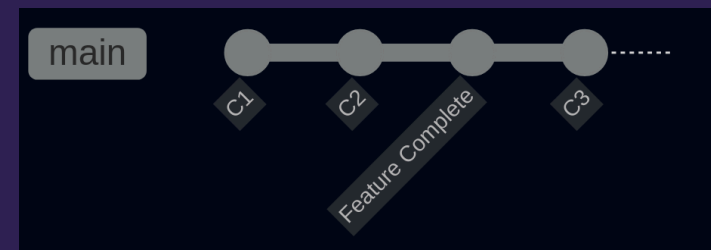


```
# Creamos la nueva rama
git switch -c feature

# Añadimos múltiples commits
git commit -m "F1"
git commit -m "F2"
git commit -m "F3"

# Volvemos a main
git switch main

# Squash merge: combina todos en uno
git merge --squash feature
git commit -m "Feature Complete"
git commit -m "C3"
```



Personalizar mensajes de merge

Puedes modificar el mensaje de un merge usando `--edit` y `--no-commit`:



- `--edit`: Abre el editor para modificar el mensaje automático
- `--no-commit`: Realiza el merge sin crear el commit automáticamente

Comportamiento por tipo de merge

Fast-forward merge:

```
git merge --edit feature      # ✗ No funciona (no hay commit de merge)
git merge --no-commit feature # ✔ Evita fast-forward, deja cambios en staging
```

No Fast-forward merge:

```
git merge --edit --no-ff feature      #  Abre editor para mensaje  
git merge --no-commit --no-ff feature #  Prepara merge sin commitear
```

Squash merge:

```
git merge --squash feature          # ☒ Siempre requiere commit manual  
git merge --squash --edit feature  # ☒ Redundante (ya controlas el mensaje)
```

Ejemplos prácticos

- git merge --edit --no-ff feature

```
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 master > git status -s
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 master > git switch -c feature
Switched to a new branch 'feature'
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 feature > touch feature
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 feature > git add feature
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 feature > git commit -m 'Add feature'
[feature b2f33c0] Add feature
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 feature > git switch main
fatal: invalid reference: main
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 feature > git switch master
Switched to branch 'master'
silvelo@silvelo-Inspiron-7559 ~/.../course-example 📁 master > git merge --edit --no-ff feature
Merge made by the 'ort' strategy.
 feature | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature
```

GNU nano 7.2

```
Mi merge personalizado
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

- `git merge --no-commit --no-ff feature`

```
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git switch -c feature
Switched to a new branch 'feature'
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > touch feature-file
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > git add feature-file
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > git commit -m 'My feature file'
[feature 4872d76] My feature file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 feature-file
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > git switch master
Switched to branch 'master'
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git status -s
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git merge --no-co
--no-commit      --no-continue
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git merge --no-commit --no-ff feature
Automatic merge went well; stopped before committing as requested
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git status -s
A   feature-file
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git commit -m 'My merge'
[master a4da5ca] My merge
```

- git merge --squash feature

```
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git switch -c feature
Switched to a new branch 'feature'
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > touch feature-45
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > git add feature-45
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > git commit -m 'feature-45'
[feature d6f5943] feature-45
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature-45
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 feature > git switch master
D   feature
Switched to branch 'master'
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git merge --squash feature
Updating a4da5ca..d6f5943
Fast-forward
Squash commit -- not updating HEAD
 feature-45 | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature-45
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git status -s
D feature
A feature-45
silvelo@silvelo-Inspiron-7559 ~/..📁../course-example 📁 master > git commit -m 'Complete Features'
[master 305e9f5] Complete Features
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 feature-45
```

```
commit 305e9f52d42aba00ed2441a30aa0ab5885bc21fd (HEAD -> master)
Author: silvelo <arturo.silvelo@gmail.com>
Date: Mon Sep 22 12:57:01 2025 +0200
```

Complete Features

```
commit a4da5ca3a65915ad9984394e5d58052bff59e1bc
Merge: ec218dc 4872d76
Author: silvelo <arturo.silvelo@gmail.com>
Date: Mon Sep 22 12:53:13 2025 +0200
```

My merge

```
commit 4872d7696e238be1ea5edacea1cd66cafd9985f6
Author: silvelo <arturo.silvelo@gmail.com>
Date: Mon Sep 22 12:52:16 2025 +0200
```

My feature file

```
commit ec218dccbf796b019623e57e0ce545eff296a7fb
Merge: 309800d b2f33c0
Author: silvelo <arturo.silvelo@gmail.com>
Date: Mon Sep 22 12:48:06 2025 +0200
```

Mi merge personalizado

```
commit b2f33c01d2ca03c44c90aef992291eb83c9ef3d6
Author: silvelo <arturo.silvelo@gmail.com>
Date: Mon Sep 22 12:44:39 2025 +0200
```

Conflictos de Merge

¿Qué es un conflicto?

Un **conflicto** ocurre cuando Git no puede fusionar automáticamente los cambios porque:

- Dos ramas modificaron las **mismas líneas** en el mismo archivo
- Una rama modificó un archivo que otra rama eliminó
- Ambas ramas crearon archivos con el **mismo nombre**

Git necesita **tu ayuda** para decidir qué cambios mantener.

¿Cuándo ocurren los conflictos?

Escenarios comunes:

- Dos desarrolladores editan la misma función
- Cambios en la misma línea de código
- Modificaciones simultáneas en archivos de configuración
- Refactorización que afecta las mismas secciones

Git puede fusionar automáticamente:

- Cambios en líneas diferentes del mismo archivo
- Cambios en archivos diferentes
- Adición de nuevas líneas sin solapamiento

Identificar un conflicto

Cuando ocurre un conflicto, Git te muestra:

```
$ git merge feature
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Verificar estado:

```
git status
# On branch main
# You have unmerged paths.
# Unmerged paths:
#   both modified:   file.txt
```

Anatomía de un conflicto

Git marca los conflictos en el archivo con marcadores especiales:

```
<<<<<<< HEAD
Código de la rama actual (main)
=====
Código de la rama que se está fusionando (feature)
>>>>>>> feature
```

Ejemplo real:

```
function saludar() {  
  <<<<<< HEAD  
    return "Hola Mundo";  
  =====  
    return "Hello World";  
>>>>>> feature  
}
```

Abortar un merge

Si decidimos **no continuar** con el merge después de encontrar conflictos:

```
# Abortar el merge y volver al estado anterior  
git merge --abort
```

¿Cuándo usar merge --abort?

- Los conflictos son demasiado complejos
- Te das cuenta de que no era el momento adecuado para el merge
- Prefieres resolver los conflictos de otra manera
- Necesitas consultar con el equipo antes de decidir

Resultado:

- Vuelve al estado anterior al merge
- No se pierde ningún trabajo
- Puedes intentar el merge más tarde

Comandos útiles durante conflictos

```
# Ver estado actual del merge
git status

# Ver qué archivos tienen conflictos
git diff --name-only --diff-filter=U

# Abortar el merge completamente
git merge --abort

# Ver diferencias entre las versiones
git diff HEAD..feature
git diff HEAD..HEAD~1

# Después de resolver conflictos
git add archivo-resuelto.txt
git commit # Completa el merge
```

Resolver conflictos automáticamente

Git ofrece estrategias para resolver conflictos automáticamente:

Opción `-X ours`: Favorece cambios de la rama actual

```
git merge -X ours feature
```

Opción `-X theirs`: Favorece cambios de la rama que se fusiona

```
git merge -X theirs feature
```


Resolver conflictos durante el merge

Si ya estamos en medio de un conflicto, puedes resolverlos automáticamente:

```
# Resolver TODOS los conflictos favoreciendo nuestra rama
git checkout --ours .
git add .
git commit

# Resolver TODOS los conflictos favoreciendo la otra rama
git checkout --theirs .
git add .
git commit
```

Resolver por archivo específico

```
# Durante un conflicto activo:  
git status # Ver archivos en conflicto  
  
# Elegir versión por archivo  
git checkout --ours archivo1.txt  
git checkout --theirs archivo2.txt  
  
# Completar el merge  
git add .  
git commit
```

Borrar Ramas

¿Por qué borrar ramas?

Ventajas de borrar ramas fusionadas:

- Mantener el repositorio limpio y organizado
- Evitar confusión con ramas obsoletas
- Reducir la lista de ramas al hacer `git branch`
- Liberar espacio (aunque Git es eficiente)

¿Cuándo borrar?

- Después de fusionar exitosamente una feature
- Cuando una rama experimental ya no es necesaria
- Al finalizar un hotfix integrado

Borrar ramas locales

Borrado seguro (solo ramas fusionadas):

```
git branch -d feature-completada
```

Borrado forzado (cualquier rama):


```
git branch -D feature-experimental
```

Diferencias entre -d y -D

`git branch -d` (delete):


- Solo borra ramas **completamente fusionadas**
- Git te protege de perder trabajo
- Recomendado para uso normal

`git branch -D` (Delete forzado):



- Borra **cualquier rama** sin verificar
- Útil para ramas experimentales o squash merge
-  **Cuidado:** Puedes perder trabajo no fusionado

Ejemplos por tipo de merge

Después de Fast-forward o No Fast-forward:

```
git merge feature-login  
git branch -d feature-login #  Funciona sin problemas
```

Después de Squash merge:

```
git merge --squash feature-payment  
git commit -m "Add payment system"  
git branch -d feature-payment #  Error: no está fusionada  
git branch -D feature-payment #  Funciona (forzado)
```

```
silvelo@silvelo-Inspiron-7559 ~/..📁../git-merge-practice 📁 main > git branch --delete feature/fast-forward
Deleted branch feature/fast-forward (was b238b59).
silvelo@silvelo-Inspiron-7559 ~/..📁../git-merge-practice 📁 main > git branch --delete feature/squash-merge
error: the branch 'feature/squash-merge' is not fully merged.
If you are sure you want to delete it, run 'git branch -D feature/squash-merge'
silvelo@silvelo-Inspiron-7559 ~/..📁../git-merge-practice 📁 main > 
```


Borrar múltiples ramas

```
# Borrar varias ramas específicas
git branch -d feature-1 feature-2 hotfix-bug

# Borrar todas las ramas fusionadas (excepto main)
git branch --merged | grep -v main | xargs git branch -d

# Ver ramas no fusionadas antes de borrar
git branch --no-merged
```

Recreando ramas borradas

Si borraste una rama por error, puedes recuperarla:

```
# Encontrar el commit de la rama borrada
git reflog

# Recrear la rama desde el commit
git branch rama-recuperada abc1234

# O directamente hacer checkout
git checkout -b rama-recuperada abc1234
```



Consejo: Git mantiene los commits durante ~30 días por defecto

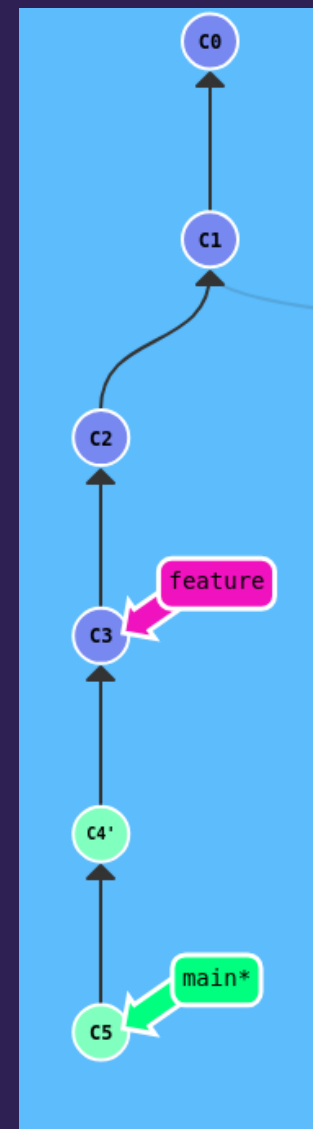
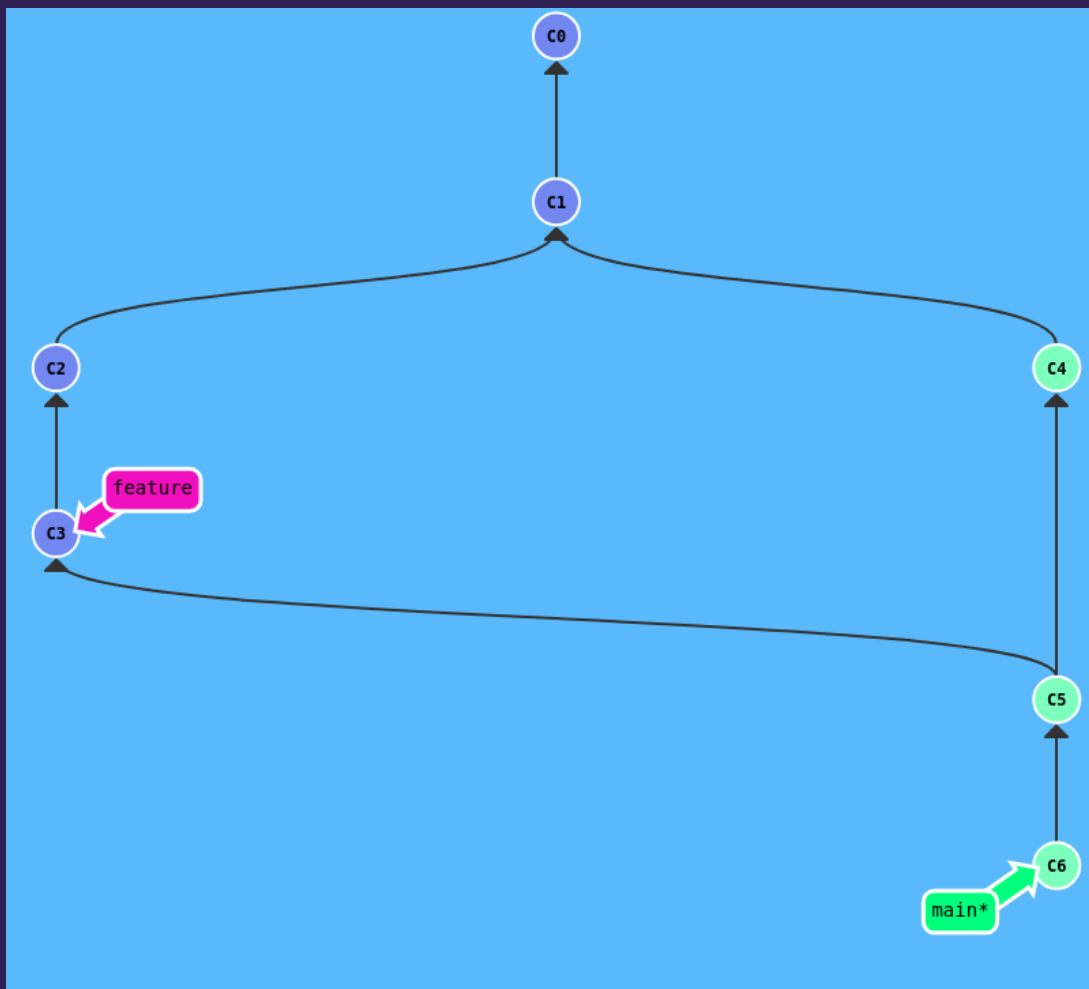
Rebase

¿Qué es rebase?

Rebase es una alternativa a merge que **reescribe el historial** para crear una línea de desarrollo más limpia y lineal.

- **Reaplica commits** de una rama sobre otra
- **Cambia la base** de donde se originó la rama
- **Crea historial lineal** sin commits de merge
- **Reescribe hashes** de los commits movidos

Rebase vs Merge



Comando básico de rebase

```
# Cambiar a la rama que quieres rebasar  
git switch feature  
  
# Rebasar sobre main  
git rebase main
```

Resultado: Los commits de `feature` se reaplican sobre el último commit de `main`

Ventajas

- **Historial lineal y limpio** - fácil de seguir
- **Sin commits de merge** - menos "ruido" en el log
- **Fast-forward siempre posible** después del rebase
- **Mejor para revisión de código** en PRs
- **Commits agrupados** por feature

Desventajas

- **Reescribe historial** - puede causar problemas en colaboración
- **Más complejo** de resolver conflictos
- **Pierde contexto** de cuándo se hizo el merge

Rebase interactivo

El rebase interactivo te permite **modificar commits** durante el proceso:

```
# Rebase interactivo de los últimos 3 commits
git rebase -i HEAD~3

# Rebase interactivo sobre main
git rebase -i main
```

Opciones del rebase interactivo

```
pick f7f3f6d F1: Add login form
pick 310154e F2: Add validation
pick a5f4a0d F3: Fix typo
```

```
# Cambiar 'pick' por:
# pick      = usar commit tal como está
# reword    = cambiar mensaje del commit
# edit      = pausar para editar el commit
# squash    = combinar con commit anterior
# drop      = eliminar commit
```

Ejemplo de rebase interactivo

Antes:

```
git log --oneline
a5f4a0d F3: Fix typo
310154e F2: Add validation
f7f3f6d F1: Add login form
c2e8f9a C2: Main work
```

Configuración del rebase:

```
pick f7f3f6d F1: Add login form  
squash 310154e F2: Add validation  
drop a5f4a0d F3: Fix typo
```

Resultado:

```
git log --oneline  
b4d2c1e F1: Add login form with validation  
c2e8f9a C2: Main work
```

Conflictos durante rebase

Si hay conflictos durante el rebase:

```
# Git pausará el rebase
git status # Ver archivos en conflicto

# Resolver conflictos manualmente
# Editar archivos...

# Continuar el rebase
git add archivo-resuelto.txt
git rebase --continue

# O abortar si es necesario
git rebase --abort
```

Rebase vs Merge - ¿Cuándo usar cada uno?

- Usar Rebase cuando
 - Quieres historial lineal y limpio
 - Trabajas en feature branches personales
 - Antes de hacer merge a main
 - Para limpiar commits antes de PR
- Usar Merge cuando
 - Trabajas en equipo en la misma rama
 - Quieres preservar contexto temporal
 - En ramas públicas/compartidas
 - Para mantener trazabilidad de merges

Comandos útiles para rebase

```
# Rebase simple
git rebase main
# Rebase interactivo
git rebase -i HEAD~3
# Continuar después de resolver conflictos
git rebase --continue
# Saltar commit problemático
git rebase --skip
# Abortar rebase
git rebase --abort
# Ver estado del rebase
git status
git log --oneline --graph
```