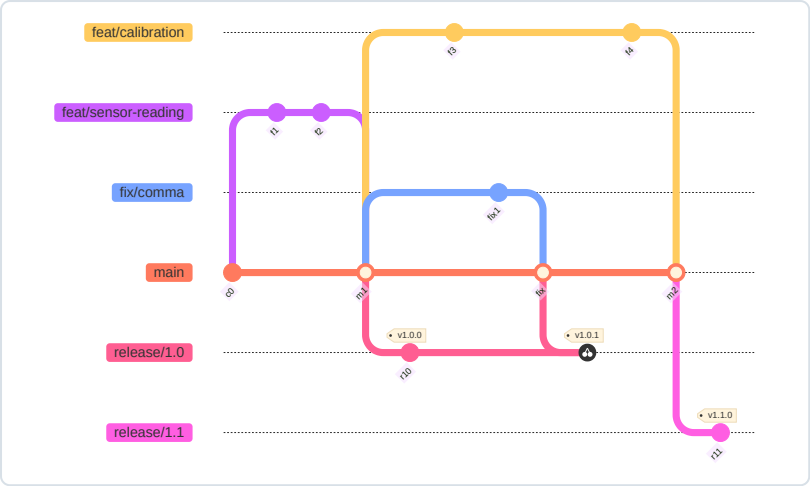


TRUNK-BASED DEVELOPMENT

Guía práctica de Git
Branches cortas · Integración frecuente · Releases controladas

La idea en 30 segundos

TBD mantiene `main` como línea de integración, con entregas pequeñas y branches cortas. **CI (Integración Continua)** compila, prueba y analiza el código en cada push o PR/MR. Las release branches son opcionales y solo estabilizan versiones mientras `main` avanza.

01 · Clonar un repositorio

Este es el flujo principal cuando el proyecto ya existe en un servidor Git. La URL puede usar SSH o HTTPS.

```
git clone <URL-DEL-REMOTO>
cd nombre-repo
git switch main
git pull --ff-only origin main
```

`origin` es el nombre convencional del remoto. `--ff-only` evita un merge automático y falla si hay divergencia.

02 · Crear una branch corta

Usa una branch para un cambio pequeño que necesite revisión o CI antes de llegar a la trunk. Termina en horas o pocos días.

Nombra la branch según el cambio: `feat/`, `fix/`, `docs/`, `test/`, `refactor/`, `build/` o `chore/`. El glosario explica estos prefijos.

```
git switch -c feat/lectura-sensor
git status
git diff -- src/sensor.c include/sensor.h
git add src/sensor.c include/sensor.h
git rm src/obsolete.c
git commit -m "feat: lee sensor por I2C"
```

Añade archivos explícitamente; `git rm` elimina el archivo del working tree y del índice.

Para corregir el último commit antes de publicar:

```
git add include/sensor.h
git commit --amend --no-edit
```

Buena práctica

Aunque el trabajo esté incompleto, haz un commit al final del día para no perder progreso. Si ese commit todavía no fue publicado, continúa al día siguiente y usa `--amend` para actualizarlo; esto cambia su SHA. Después de publicar, prefiere un commit nuevo antes que reescribir historial compartido.

Publica la branch para abrir el PR/MR. La integración en `main` ocurre después, en la plataforma remota:

```
git push -u origin feat/lectura-sensor
```

03 · Publicar e integrar un PR/MR

PR (GitHub) y MR (GitLab) son solicitudes de revisión en la plataforma; **abrir o aprobar no integra el código**. Tras CI y la aprobación requerida, haz el merge. Son opcionales en TBD: mantén branches y revisiones breves.
GitHub · Instalar `gh`

```
gh pr create --base main --fill
# después de aprobar
gh pr merge --squash --delete-branch
```

GitLab · Instalar `glab`

```
glab mr create --fill
# después de aprobar
glab mr merge --squash --remove-source-branch
```

Codeberg / Gitea · Instalar `tea`

```
tea pulls create \
  --base main \
  --head feat/ejemplo \
  --title "feat: agrega ejemplo"
# después de aprobar, haz el merge en la interfaz web
```

Merge local (integra en este clon, sin PR/MR):

```
git switch main
git merge --no-ff feat/ejemplo
```

Solo cambia el repositorio local; el servidor remoto no recibe este merge.

Actualizar × integrar

```
--ff-only  A---B---S  main
--no-ff    A---B---------M  main
                \       /
                 C-----D    feat/ejemplo
```

`pull --ff-only` actualiza `main`; `M` marca el merge `--no-ff`.

04 · Limpiar después del PR/MR

Merge local: `git branch -d`. Tras un *squash* remoto, confirma el PR/MR y usa `git branch -D`. La plataforma elimina la branch remota.

```
git switch main
git pull --ff-only
git branch -d feat/lectura-sensor
```

Marcio Bulla
From beginner to beginner

05 · Release, tag y actualización intermedia

Una release branch es opcional: créala cerca del lanzamiento, solo para estabilizar. La trunk continúa recibiendo features; la release no recibe funcionalidad nueva.

```
git switch main
git pull --ff-only origin main
git switch -c release/1.0
git push -u origin release/1.0
git tag -a v1.0.0 -m "Release 1.0.0"
git push origin v1.0.0
```

El tag debe apuntar al commit **exacto** aprobado. Para una actualización `1.0.1` nacida en la trunk, localiza el SHA, lleva la corrección a la release y crea el tag.

```
git switch main
git log --oneline --decorate -10
# corrige, prueba y crea el commit
git switch release/1.0
git cherry-pick <SHA>
git tag -a v1.0.1 -m "Release 1.0.1"
git push origin release/1.0 v1.0.1
```

Propagación controlada

`cherry-pick` crea otro commit, normalmente con otro SHA. En un conflicto, resuelve, usa `git add` y `git cherry-pick --continue`; cancela con `git cherry-pick --abort`.

06 · Leer el historial

Usa el log para localizar el commit aprobado, entender la topología y comparar trunk con release.

```
git log --oneline --decorate --graph --all
git show <SHA>
git log main..release/1.0
git diff main..release/1.0
```

07 · Recuperar después de un problema

Para recuperar un commit antiguo sin reescribir la trunk compartida, localízalo y crea una branch de recuperación.

```
git reflog
git switch -c recovery/old-state <SHA>
git restore --source=<SHA> -- src/sensor.c
git diff --stat
git add src/sensor.c
git commit -m "fix: restaura estado estable"
git push -u origin recovery/old-state
```

Si quieres deshacer un commit ya publicado, prefiere `revert`:

```
git switch main
git pull --ff-only origin main
git revert <SHA>
git push origin main
```

Cuidado

`git revert` crea un commit nuevo y conserva el historial. `git reset --hard` puede descartar cambios locales y no debe reescribir historial compartido. Conserva el trabajo con `git stash push -m "wip"`.

EXTRA · Crear un repositorio local

Usa este flujo cuando el proyecto todavía no es un repositorio Git. Después, publícalo en el remoto.

```
git init
git switch -c main
git add README.md
git commit -m "chore: inicia repositorio"
git remote add origin <URL-DEL-REMOTO>
git push -u origin main
```

Glosario rápido de comandos

Prefijos (branch: `/` · commit: `:`): `feat` funcionalidad · `fix` corrección · `docs` documentación · `test` pruebas · `refactor` estructura sin cambiar el comportamiento · `build` build/dependencias · `chore` mantenimiento y organización.

`git clone` — copia un repositorio remoto.
`git init` — inicia un repositorio Git local.
`git remote` — gestiona las URL de los repositorios remotos.
`git switch` — cambia de branch; `-c` crea una nueva.
`git status` — muestra el estado de los archivos.
`git diff` — compara cambios no confirmados.
`git add` — prepara cambios; no crea commit.
`git rm` — elimina y prepara la eliminación.
`git merge` — integra el historial de otra branch.
`git commit` — guarda en el historial lo preparado.
`git log` — lista commits; `--oneline` resume.
`git show` — muestra un commit y sus cambios.
`git pull` — busca e integra actualizaciones remotas.
`git push` — publica commits locales en el remoto.
`git fetch` — busca referencias sin cambiar tu branch.
`git tag` — marca un commit con una versión.
`git cherry-pick` — copia un commit a otra branch.
`git revert / restore / stash` — deshace, recupera o guarda.

Marcio Bulla
From beginner to beginner