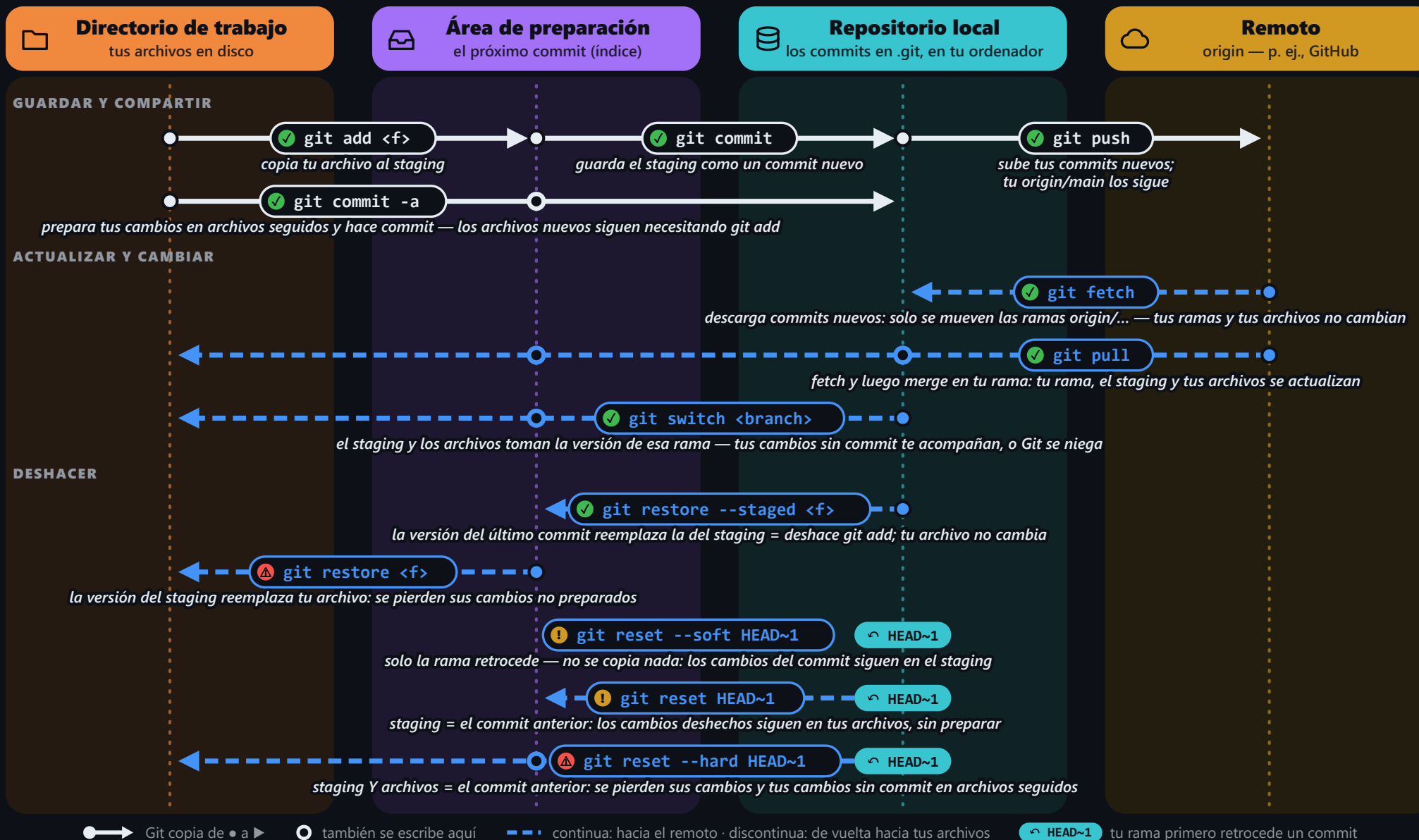


01 Git de un vistazo

Los cuatro lugares donde Git guarda tu trabajo — y lo que cada comando copia exactamente entre ellos

✓ Seguro ! Reescribe el historial ⚠ Puede perder trabajo



o `<f>` la ruta de un archivo: escribe `app.js`, sin los `<`

o `HEAD~1` el commit anterior a aquel en el que estás

o `git pull` si tú y el remoto tenéis commits nuevos, Git se detiene hasta que elijas: `git config pull.rebase false` (merge) o `true` (rebase)

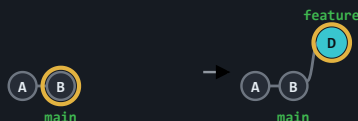
Antes → después. Los círculos son commits; el anillo amarillo es donde estás (HEAD). ● commit nuevo ○ ya no apunta ninguna rama a él

✓ git commit



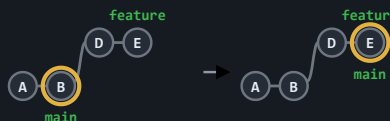
Un commit nuevo con lo preparado; tu rama avanza hasta él.

✓ git switch -c feature + commit



Una rama nueva empieza donde estás; tus próximos commits la mueven, main se queda.

✓ git merge feature (fast-forward)



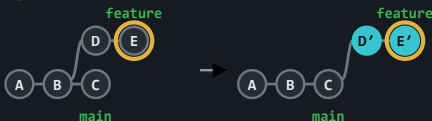
En main, cuando main no tiene commits nuevos: main simplemente avanza hasta E — sin commit de merge.

✓ git merge feature



En main: una feature con un commit de fusión que tiene dos padres.

! git rebase main



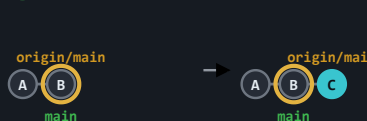
En feature: reaplica D y E sobre main como copias con hashes nuevos (D' E'); los originales salen de la rama.

✓ git cherry-pick D



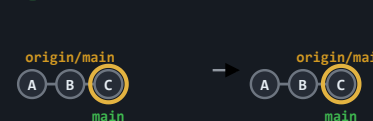
Copia un commit en tu rama (D', hash nuevo); el original se queda.

✓ git fetch



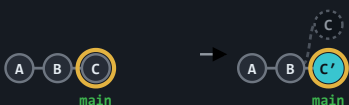
Descarga commits nuevos: solo se mueven las etiquetas origin/... Tu rama y tus archivos no cambian.

✓ git push



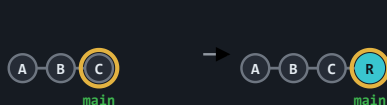
Sube C: main en el remoto — y tu origin/main — pasan a C. Tus archivos no cambian.

! git commit --amend



Reemplaza el último commit por uno nuevo (C', hash nuevo); C sale de la rama. Hazlo antes de hacer push.

✓ git revert C



Añade un commit R que deshace C. El historial se conserva: seguro en ramas compartidas.

! git reset --soft HEAD~1



Lleva tu rama de vuelta a B; los cambios de C siguen en el staging — haz commit otra vez para rehacerlo.

⚠ git reset --hard HEAD~1



Hace retroceder tu rama; el staging y los archivos pasan a ser B: los cambios de C desaparecen (git reflog aún encuentra C).

EMPEZAR

- git config --global user.name "Your Name"**
Nombre que figura en tus commits (una vez por ordenador)
- git config --global user.email "you@example.com"**
Correo que figura en tus commits
- git init**
Convertir la carpeta actual en un repositorio
- git clone <url>**
Descargar un repositorio con todo su historial

ECHAR UN VISTAZO

- git status**
Qué cambió, qué está preparado y en qué rama estás
- git diff**
Cambios aún sin preparar (directorio de trabajo frente a área de preparación)
- git diff --staged**
Lo que contendrá el próximo commit
- git log --oneline --graph --all**
El historial de cada rama, una línea por commit
- git show <commit>**
Un commit: mensaje y cambios

GUARDAR TU TRABAJO

- git add <file>**
Preparar un archivo para el próximo commit
- git add -p**
Preparar solo algunas partes (hunks)
- git commit -m "message"**
Registrar lo preparado como un commit
- git commit --amend**
 Rehacer el último commit (mensaje o contenido)
- git restore <file>**
 Descartar los cambios no preparados de un archivo
- git restore --staged <file>**
Quitar de la preparación (tu cambio sigue en el disco)

COMPARTIR

- git fetch**
Descargar commits nuevos; tus ramas no se mueven
- git pull**
Fetch y luego merge en la rama actual
- git pull --rebase**
 Fetch y luego reapplica tus commits encima (hashes nuevos)
- git push**
Subir tus commits al upstream de la rama
- git push -u origin <branch>**
Primer push de una rama nueva (define el upstream)
- git remote -v**
Con qué remotos se comunica este repositorio

- o **HEAD** el commit en el que estás (normalmente el último de tu rama actual)
- o **origin** el nombre por defecto del remoto desde el que clonaste
- o **<file>** la ruta de un archivo: escribe **app.js**, sin los **< >**

CREAR Y CAMBIAR

- git branch**
Listar las ramas locales (* = actual)
- git branch -a**
Listar también las ramas de seguimiento remoto
- git switch <branch>**
Ir a una rama (los archivos la siguen)
- git switch -c <branch>**
Crear una rama aquí y cambiar a ella
- git switch -**
Volver a la rama anterior
- git branch -m <new-name>**
Renombrar la rama actual

COMBINAR

- git merge <branch>**
Traer una rama a la actual
- git merge --no-ff <branch>**
Registrar siempre un commit de merge
- git rebase <branch>**
! Reaplicar tus commits encima de <branch> (hashes nuevos)
- git rebase -i HEAD~3**
! Editar los 3 últimos commits: reordenar, combinar, reformular, eliminar
- git cherry-pick <commit>**
Copiar un commit en la rama actual

CONFLICTOS

- git status**
Lista los archivos en conflicto (p. ej. «both modified»)
- Editar el archivo**
Conserva lo que quieras y borra las líneas
<<<<<< ===== >>>>>>
- git add <file>**
Marcar el archivo como resuelto
- git merge --continue**
Terminar (o: git rebase --continue / cherry-pick --continue)
- git merge --abort**
Rendirse y volver a como estaba (rebase --abort, ...)

LIMPIAR

- git branch -d <branch>**
Borrar una rama fusionada (se niega si no está fusionada)
- git branch -D <branch>**
⚠ Borrar aunque no esté fusionada
- git push origin --delete <branch>**
⚠ Borra la rama en el remoto (sus commits sin fusionar se van con ella)
- git fetch --prune**
Borra también las copias origin/... de ramas eliminadas en el remoto

ETIQUETAS

- git tag**
Listar etiquetas
- git tag -a v1.0 -m "Release 1.0"**
Etiqueta anotada en el commit actual
- git push origin v1.0**
Las etiquetas no se suben automáticamente
- git push --follow-tags**
Subir los commits junto con sus etiquetas anotadas

LEER EL HISTORIAL

- git log --oneline**
Historial compacto de la rama actual
- git log -p <file>**
Cada cambio hecho en un archivo
- git log main..feature**
Commits de feature que main no tiene
- git blame <file>**
Quién cambió cada línea por última vez

QUIERO...	EJECUTAR	CONVIENE SABER
❌ Descartar mis cambios en un archivo	<code>git restore <file></code>	⚠ Los cambios no preparados se pierden para siempre
✅ Quitar un archivo de la preparación (conservar mi cambio)	<code>git restore --staged <file></code>	✓ Solo cambia el área de preparación
⚠ Corregir el mensaje del último commit o añadir un archivo olvidado	<code>git add <file> && git commit --amend</code>	! Hash nuevo: hazlo antes de hacer push
⚠ Deshacer el último commit y dejar los cambios preparados	<code>git reset --soft HEAD~1</code>	! Tu trabajo sigue en el staging, listo para un nuevo commit — no después de hacer push
⚠ Deshacer el último commit y dejar los cambios en los archivos	<code>git reset HEAD~1</code>	! --mixed por defecto: fuera del staging pero en disco — no después de hacer push
✅ Deshacer un commit que ya subiste	<code>git revert <commit></code>	✓ Añade un commit nuevo que lo anula: seguro en ramas compartidas
✅ Apartar mis cambios por ahora	<code>git stash push -m "wip"</code>	✓ Se recuperan con <code>git stash pop</code> — los archivos nuevos sin seguimiento necesitan -u
❌ Descartar el último commit y sus cambios	<code>git reset --hard HEAD~1</code>	⚠ Se pierden sus cambios y tus cambios sin commit en archivos seguidos (<code>git reflog</code> aún encuentra el commit)
✅ Encontrar un commit «perdido» (tras reset, rebase o borrar una rama)	<code>git reflog</code>	✓ Todos los sitios por los que ha pasado HEAD; luego <code>git branch rescue <hash></code>
✅ Detener un merge / rebase / cherry-pick en curso	<code>git merge --abort</code>	✓ O <code>rebase --abort</code> , <code>cherry-pick --abort</code>
⚠ Sustituir la rama remota tras reescribir el historial	<code>git push --force-with-lease</code>	! Se niega si la rama remota cambió desde tu último fetch

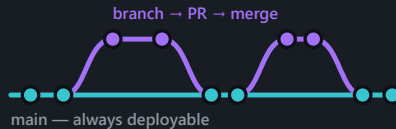
STASH

- `git stash`
Apartar los cambios sin commit
- `git stash -u`
Incluir archivos sin seguimiento (nuevos)
- `git stash list`
Todos los stashes, el más reciente = `stash@{0}`
- `git stash pop`
Reaplica el más reciente y lo elimina (se conserva si hay conflicto)
- `git stash apply stash@{1}`
Reaplicar uno y conservarlo en la lista

RESET: TRES INTENSIDADES

- ⚠ `git reset --soft <commit>`
! Mueve la rama; preparación y archivos intactos
- ⚠ `git reset <commit>`
! Mueve la rama y restablece la preparación; archivos intactos
- ❌ `git reset --hard <commit>`
⚠ Mueve la rama, la preparación Y los archivos

GITHUB FLOW



```
git switch main &&
git pull
```

Empezar desde un main actualizado

```
git switch -c
<feature>
```

Una rama de corta duración por cambio

```
git commit -am "... " &&
git push -u origin
<feature>
```

Commit, push y abrir una pull request

Integrar el pull request en GitHub

Tras la revisión; luego borrar la rama

```
git switch main &&
git pull --prune
```

Vuelta a main; borra también las copias origin/... de ramas eliminadas

GIT FLOW



```
git checkout -b
feature/x develop
```

Las funcionalidades parten de develop

```
git checkout develop
&& git merge --no-ff
feature/x
```

Terminar una funcionalidad en develop

```
git checkout -b
release-1.1 develop
```

Congelar una versión

```
git checkout main &&
git merge --no-ff
release-1.1
```

Publicar en main...

```
git tag -a 1.1 -m
"Release 1.1"
```

...etiquetarla y fusionarla de vuelta en develop

TRUNK-BASED



```
git commit -am "small
change"
```

Commits pequeños y completos en main

```
git pull --rebase
```

! Reaplica los tuyos sobre los de los demás (hashes nuevos)

```
git add <file> && git
rebase --continue
```

Tras arreglar un conflicto: márcalo como resuelto y sigue

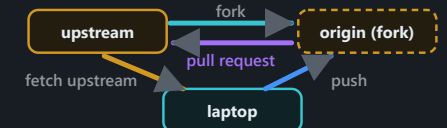
```
git push
```

Integrar al menos una vez al día

Publicar el trabajo sin terminar desactivado

Tras un feature flag, p. ej. NEW_CHECKOUT = false

FORKING



```
git remote add
upstream
```

```
<original-url>
```

Una vez: el original como segundo remoto

```
git fetch upstream &&
git merge
upstream/main
```

Sincronizar tu main con el original

```
git push
```

Actualizar tu fork (origin)

```
git switch -c <fix> ...
git push -u origin
<fix>
```

Trabajar en una rama de tu fork

Abrir un pull request hacia el original

Los mantenedores revisan y fusionan