Git_

Guía de atajos · control de versiones sin miedo

```
user@dajomar:~$ git status
```

Cómo piensa Git

```
Directorio de trabajo ──add──▶ Staging
Staging ──commit──▶ Repositorio local
Repositorio local ──push──▶ Remoto (GitHub)
Remoto ──pull / fetch──▶ Repositorio local
```

Working dir	Tus archivos tal como los editas
Staging	Lo que entrará en el próximo commit
Commit	Una foto del proyecto con mensaje e ID
Branch	Una línea de trabajo independiente
HEAD	Dónde estás parado ahora
origin	Nombre por defecto del remoto

Configuración inicial

```
git config --global user.name "David Maza"
git config --global user.email "tu@correo.com"
git config --global init.defaultBranch main
git config --global core.editor "code --wait"
git config --list
```

Empezar un proyecto

```
git init  # repo nuevo aquí
git clone https://github.com/user/repo.git
git clone git@github.com:user/repo.git  # SSH
```

Flujo diario

```
git status
git add archivo.txt
git add .  # todo lo modificado
git add -p  # elegir cambios por partes
git commit -m "feat: agrega login"
git commit -am "fix: corrige typo"  # add + commit
git push
```

Ver diferencias

```
git diff               # cambios sin agregar
git diff --staged      # lo que va al commit
git diff main..feature  # entre ramas
```

Historial

```
git log --oneline --graph --all
git log -p archivo.txt  # cambios de un archivo
git log --author="David" --since="2 weeks"
git show a1b2c3d        # detalle de un commit
git blame archivo.txt   # quién cambió cada línea
```

Ramas

```
git branch                  # listar locales
git branch -a               # incluye remotas
git switch -c feature/login  # crear y cambiar
git switch main
git merge feature/login
git branch -d feature/login  # borrar (ya unida)
git branch -D experimento    # forzar borrado
git push origin --delete feature/login
```

`git switch` reemplaza al viejo `git checkout` para cambiar de rama.

Remotos

```
git remote -v
git remote add origin git@github.com:user/repo.git
git push -u origin main  # primera vez
git fetch                # trae sin mezclar
git pull                 # fetch + merge
git pull --rebase        # historial más limpio
```

Pro Tip:
Haz `git pull` antes de empezar a trabajar y antes de cada `push`. Así evitas la mayoría de los conflictos.

Deshacer cambios

```
# descartar cambios de un archivo
git restore archivo.txt
# sacar un archivo del staging
git restore --staged archivo.txt
# corregir mensaje o contenido del último commit
git commit --amend
# deshacer un commit ya subido (crea uno nuevo)
git revert a1b2c3d
# recuperar commits o ramas "perdidos"
git reflog
```

git reset: mover la rama hacia atrás

Modo	Commit	Staging	Archivos
--soft	Se deshace	Se mantiene	Se mantienen
--mixed	Se deshace	Se vacía	Se mantienen
--hard ⚠	Se deshace	Se vacía	Se pierden

```
# deshace el último commit
git reset --soft HEAD~1
git reset --hard origin/main  # ⚠ igual al remoto
```

Regla: si el commit ya está en el remoto y otros lo usan, usa `revert`, no `reset`.

Stash: guardar para después

```
git stash  # guarda cambios sin commit
git stash -u  # incluye archivos nuevos
git stash list
git stash pop              # recupera y borra
git stash apply stash@{1}  # recupera sin borrar
```

Conflictos

```
<<<<<<< HEAD
tu versión
=======
la versión de la otra rama
>>>>>>> feature/login
```

```
git status         # ver archivos en conflicto
# edita, deja lo correcto y borra las marcas
git add archivo.txt
git commit         # o: git rebase --continue
git merge --abort  # cancelar el merge
```

Rebase y tags

```
git switch feature/login
git rebase main       # reaplica tus commits
git rebase -i HEAD~3  # unir o editar commits
git tag -a v1.0.0 -m "Primera versión"
git push origin --tags
```

Merge conserva la historia tal cual pasó. **Rebase** la deja lineal. Nunca hagas rebase de una rama compartida.

.gitignore

```
# dependencias y builds
node_modules/
dist/
# secretos y entorno
.env
*.pem
# sistema y editor
.DS_Store
.vscode/
```

```
git rm --cached .env  # dejar de seguirlo
```

Mensajes de commit

`feat:`	Nueva funcionalidad
`fix:`	Corrección de un error
`docs:`	Solo documentación
`refactor:`	Cambio de código sin cambiar comportamiento
`chore:`	Mantenimiento, dependencias, config
`test:`	Agrega o corrige pruebas

Formato **Conventional Commits**: en imperativo y corto.
`fix: corrige validación del correo`

Seguridad:
Si subiste una contraseña o API key, borrarla en un nuevo commit **no basta**: sigue en el historial. Revócala y genera una nueva de inmediato. Para prevenirlo, usa `.gitignore` y herramientas como `gitleaks`.

Pro Tip:
Crea tus propios atajos:

```
git config --global alias.lg \
  "log --oneline --graph --all"
git lg
```