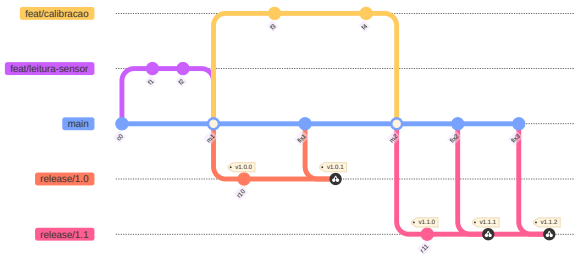
TRUNK-BASED DEVELOPMENT

Guia prático de Git
Branches curtas · Integração frequente · Releases controladas



GitGraph com a main como trunk, branches curtas, releases e correções selecionadas.
main · features curtas · releases

A ideia em 30 segundos

TBD mantém `main` como linha de integração, com entregas pequenas e branches curtas. **CI (Integração Contínua)** compila, testa e analisa o código a cada push ou PR/MR. Release branches são opcionais e apenas estabilizam versões enquanto `main` evolui.

01 · Clonar repositório

Este é o fluxo principal quando o projeto já existe em um servidor Git. A URL pode usar SSH ou HTTPS.

```
git clone <URL-DO-REMOTO>
cd firmware-sensor
git switch main
git pull --ff-only origin main
```

`origin` é o nome convencional do remoto. `--ff-only` evita criar merge automático e falha se houver divergência.

02 · Criar uma feature curta

Use uma branch para uma mudança pequena que precisa de revisão ou CI antes de chegar à trunk. Termine em horas ou poucos dias.

```
git switch -c feat/leitura-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: lê sensor por I2C"
git push -u origin feat/leitura-sensor
```

Adicione arquivos explicitamente; `git rm` remove o arquivo do working tree e do índice.

Para corrigir o último commit antes de publicar:

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

Boa prática

Mesmo com a linha pela metade, faça um commit ao fim do dia para não perder progresso. No dia seguinte, continue com `--amend` e mantenha o histórico limpo. `--amend` corrige o último commit e troca o SHA; use antes de publicar a branch. No PR/MR, **Squash** une os commits de trabalho em um commit limpo.

03 · Publicar e integrar PR/MR

Antes de integrar, atualize sua branch sobre a trunk. `rebase` reescreve commits; evite-o em branch compartilhada.

```
git fetch origin
git rebase origin/main
# depois de editar e salvar o arquivo em conflito:
git add src/sensor.c
git rebase --continue
# para cancelar o rebase
git rebase --abort
git push -u origin feat/leitura-sensor
```

`git add` coloca a versão corrigida na área de preparação (*staging*); ele não cria o commit nem corrige o arquivo sozinho. Depois de salvar a resolução, use `git diff` para conferir o conteúdo e só então rode `git add <arquivo>`.

Abra uma Pull Request ou Merge Request. A CI executa build e testes, a revisão aprova e a proteção da trunk controla o merge.

Alerta

Evite `git push --force` em `main` e em branches compartilhadas. Rebase e force push exigem coordenação.

04 · Limpeza após o PR/MR

Depois de a integração ser concluída, atualize a cópia local e remova a branch curta.

```
git switch main
git pull --ff-only origin main
git branch -d feat/leitura-sensor
git push origin --delete feat/leitura-sensor
```

Boa prática

Branch curta não é uma segunda trunk. Mantenha `main` íntegra, testada e pronta para integração.

Marcio Bulla
From beginner to beginner

05 · Release, tag e update intermediário

Release branch é opcional: crie perto do lançamento, somente para estabilizar. A trunk continua recebendo features; a release não recebe funcionalidades novas.

```
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
```

A tag deve apontar para o commit **exato** aprovado. Para um update `1.0.1` nascido na trunk, localize o SHA, leve a correção à release e só então crie a tag.

```
git switch main
git log --oneline --decorate -10
# corrija, teste e commite
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
```

Propagação controlada

`cherry-pick` cria outro commit, normalmente com outro SHA. Em conflito, resolva, use `git add` e `git cherry-pick --continue`; cancele com `git cherry-pick --abort`.

06 · Ler o histórico

Use o log para localizar o commit aprovado, entender a topologia e comparar trunk com release.

```
git log --oneline --decorate --graph --all -20
git show <SHA>
git log main..release/1.0
git diff main..release/1.0
```

Regra

Anote o SHA antes de propagar uma correção ou criar uma tag.

07 · Restaurar após um problema

Para recuperar um commit antigo sem reescrever a trunk compartilhada, localize-o e crie uma branch de recuperação.

```
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 estável"
git push -u origin recovery/old-state
```

Se o objetivo é desfazer um commit já publicado, prefira `revert`:

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

Cuidado

`git revert` cria um novo commit e preserva o histórico. `git reset --hard` pode descartar mudanças locais e não deve reescrever histórico compartilhado. Antes de trocar de estado, preserve trabalho com `git stash push -m "wip"`.

Glossário rápido de comandos

`git clone` — copia um repositório remoto.
`git switch` — muda de branch; `-c` cria uma nova.
`git status` — mostra o estado dos arquivos.
`git diff` — compara alterações ainda não commitadas.
`git add` — prepara alterações; não cria commit.
`git rm` — remove e prepara a remoção de um arquivo.
`git rebase` — reaplica commits sobre outra base; reescreve SHAs.
`git merge` — integra o histórico de outra branch.
`git commit` — grava no histórico o que foi preparado.
`git log` — lista commits; `--oneline` resume.
`git show` — exibe um commit e suas alterações.
`git pull` — busca e integra atualizações remotas.
`git push` — publica commits locais no remoto.
`git fetch` — busca referências sem alterar sua branch.
`git tag` — marca um commit com uma versão.
`git cherry-pick` — copia um commit para outra branch.
`git revert / restore / stash` — desfaz, recupera ou guarda.

Marcio Bulla

From beginner to beginner