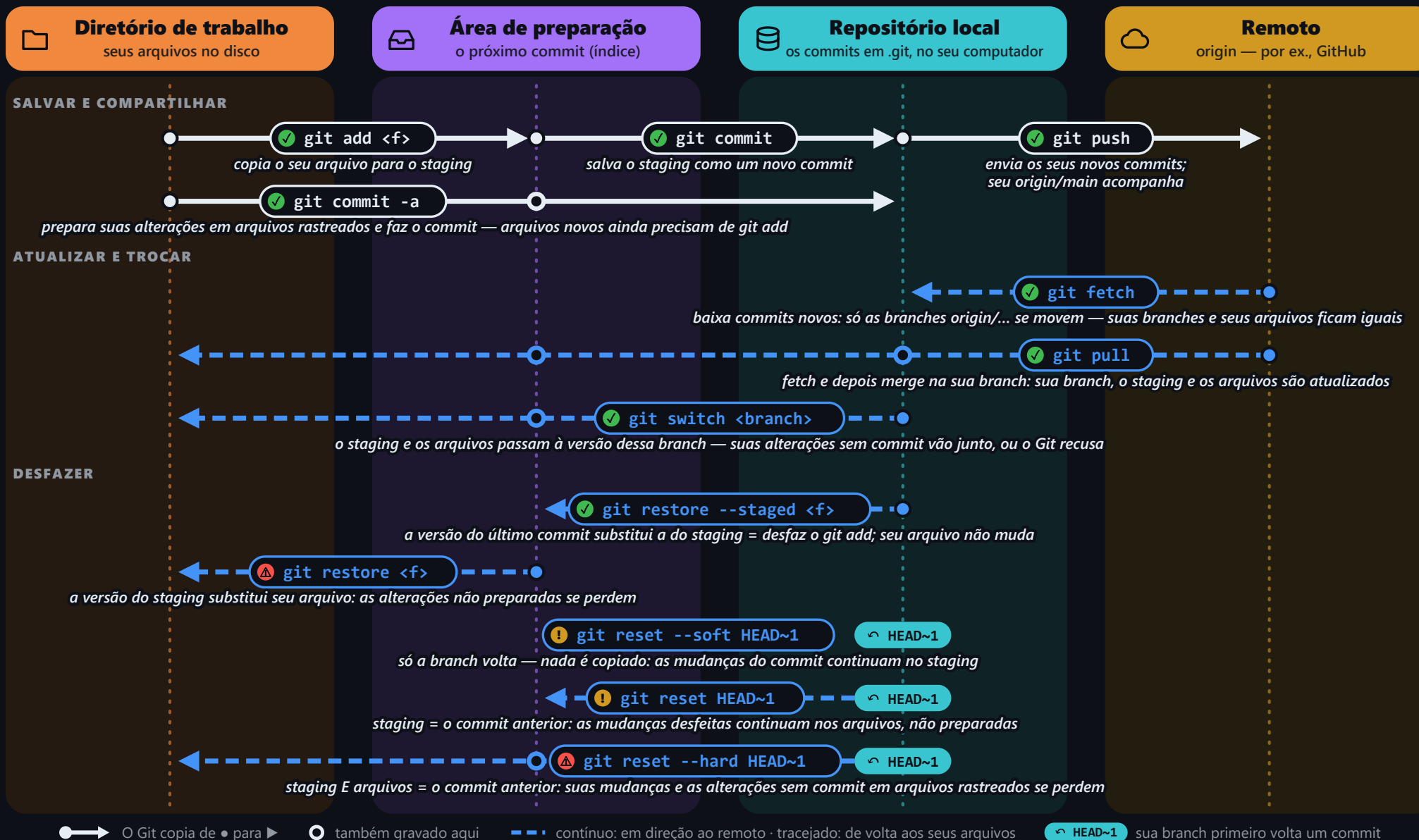


01 Git num relance

Os quatro lugares onde o Git guarda o seu trabalho — e o que cada comando copia exatamente entre eles

✓ Seguro ! Reescreve o histórico ⚠ Pode perder trabalho



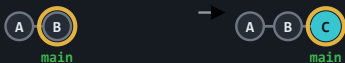
o **<f>** o caminho de um arquivo: digite **app.js**, sem os **<** **>**

o **HEAD~1** o commit antes daquele em que você está

o **git pull** se você e o remoto têm commits novos, o Git para até você escolher: **git config pull.rebase false** (merge) ou **true** (rebase)

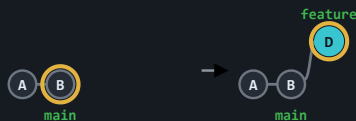
Antes → depois. Os círculos são commits; o anel amarelo é onde você está (HEAD). ● commit novo ○ nenhum branch aponta mais para ele

✓ git commit



Um commit novo com o que está no staging; seu branch avança para ele.

✓ git switch -c feature + commit



Um branch novo começa onde você está; seus próximos commits o movem, main fica.

✓ git merge feature (fast-forward)



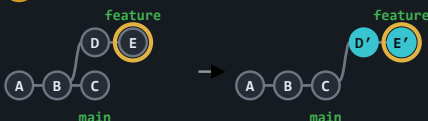
Na main, quando ela não tem commits novos: a main só avança até E — sem commit de merge.

✓ git merge feature



No main: junta feature com um commit de merge que tem dois pais.

! git rebase main



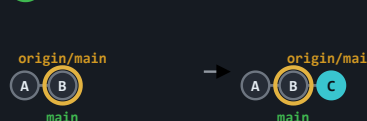
Na feature: reaplica D e E sobre a main como cópias com hashes novos (D' E'); os originais saem da branch.

✓ git cherry-pick D



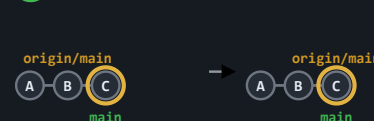
Copia um commit para o seu branch (D', hash novo); o original fica.

✓ git fetch



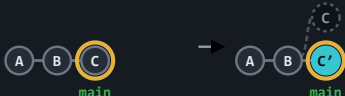
Baixa commits novos: só os rótulos origin/... se movem. Sua branch e seus arquivos ficam iguais.

✓ git push



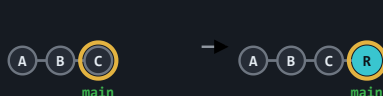
Envia C: a main no remoto — e o seu origin/main — passam para C. Seus arquivos não mudam.

! git commit --amend



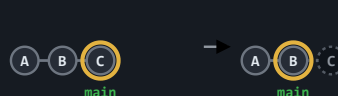
Substitui o último commit por um novo (C', hash novo); C sai da branch. Faça antes do push.

✓ git revert C



Adiciona um commit R que desfaz C. O histórico é mantido: seguro em branches compartilhados.

! git reset --soft HEAD~1



Leva sua branch de volta a B; as mudanças de C continuam no staging — faça commit de novo para refazê-lo.

⚠ git reset --hard HEAD~1



Faz sua branch voltar; o staging e os arquivos viram B: as mudanças de C somem (o git reflog ainda encontra C).

COMEÇAR

- `git config --global user.name "Your Name"`
Nome registrado nos seus commits (uma vez por computador)
- `git config --global user.email "you@example.com"`
E-mail registrado nos seus commits
- `git init`
Transformar a pasta atual em um repositório
- `git clone <url>`
Baixar um repositório com todo o histórico

DAR UMA OLHADA

- `git status`
O que mudou, o que está preparado, em qual branch você está
- `git diff`
Mudanças ainda não preparadas (diretório de trabalho vs. área de preparação)
- `git diff --staged`
O que o próximo commit vai conter
- `git log --oneline --graph --all`
Histórico de cada branch, uma linha por commit
- `git show <commit>`
Um commit: mensagem e mudanças

SALVAR O SEU TRABALHO

- `git add <file>`
Preparar um arquivo para o próximo commit
- `git add -p`
Preparar só algumas partes (hunks)
- `git commit -m "message"`
Registrar o que está preparado como um commit
- `git commit --amend`
 Refazer o último commit (mensagem ou conteúdo)
- `git restore <file>`
 Descartar as edições não preparadas de um arquivo
- `git restore --staged <file>`
Tirar da preparação (sua edição continua no disco)

COMPARTILHAR

- `git fetch`
Baixar commits novos; suas branches não se movem
- `git pull`
Fetch e depois merge na branch atual
- `git pull --rebase`
 Fetch e depois reaplica seus commits por cima (hashes novos)
- `git push`
Enviar seus commits para o upstream da branch
- `git push -u origin <branch>`
Primeiro push de uma branch nova (define o upstream)
- `git remote -v`
Com quais remotos este repositório conversa

- `HEAD` o commit em que você está (geralmente o último da sua branch atual)
- `origin` o nome padrão do remoto de onde você clonou
- `<file>` o caminho de um arquivo: digite `app.js`, sem os `< >`

■ CRIAR E TROCAR

- **git branch**
Listar branches locais (* = atual)
- **git branch -a**
Listar também as branches de rastreamento remoto
- **git switch <branch>**
Ir para uma branch (os arquivos acompanham)
- **git switch -c <branch>**
Criar uma branch aqui e trocar para ela
- **git switch -**
Voltar para a branch anterior
- **git branch -m <new-name>**
Renomear a branch atual

■ COMBINAR

- **git merge <branch>**
Trazer uma branch para a atual
- **git merge --no-ff <branch>**
Sempre registrar um commit de merge
- **git rebase <branch>**
! Reaplicar seus commits por cima de <branch> (hashes novos)
- **git rebase -i HEAD~3**
! Editar os 3 últimos commits: reordenar, juntar, reescrever, descartar
- **git cherry-pick <commit>**
Copiar um commit para a branch atual

■ CONFLITOS

- **git status**
Lista os arquivos em conflito (ex.: "both modified")
- **Editar o arquivo**
Mantenha o que quiser e apague as linhas
<<<<<< ===== >>>>>>
- **git add <file>**
Marcar o arquivo como resolvido
- **git merge --continue**
Concluir (ou: git rebase --continue / cherry-pick --continue)
- **git merge --abort**
Desistir e voltar a como estava (rebase --abort, ...)

■ FAXINA

- **git branch -d <branch>**
Excluir uma branch mesclada (recusa se não estiver mesclada)
- **git branch -D <branch>**
⚠ Excluir mesmo sem estar mesclada
- **git push origin --delete <branch>**
⚠ Apaga a branch no remoto (os commits dela sem merge vão junto)
- **git fetch --prune**
Apaga também as cópias origin/... de branches apagadas no remoto

■ TAGS

- **git tag**
Listar tags
- **git tag -a v1.0 -m "Release 1.0"**
Tag anotada no commit atual
- **git push origin v1.0**
Tags não são enviadas automaticamente
- **git push --follow-tags**
Enviar os commits com suas tags anotadas

■ LER O HISTÓRICO

- **git log --oneline**
Histórico compacto da branch atual
- **git log -p <file>**
Cada mudança feita em um arquivo
- **git log main..feature**
Commits de feature que o main não tem
- **git blame <file>**
Quem alterou cada linha por último

EU QUERO...	EXECUTAR	BOM SABER
❌ Descartar minhas edições em um arquivo	<code>git restore <file></code>	⚠ Edições não preparadas se perdem de vez
✅ Tirar um arquivo da preparação (manter minha edição)	<code>git restore --staged <file></code>	✓ Só a área de preparação muda
⚠ Corrigir a mensagem do último commit ou adicionar um arquivo esquecido	<code>git add <file> && git commit --amend</code>	! Hash novo — faça antes do push
⚠ Desfazer o último commit, mantendo as mudanças preparadas	<code>git reset --soft HEAD~1</code>	! Seu trabalho continua no staging, pronto para um novo commit — não depois do push
⚠ Desfazer o último commit, mantendo as mudanças nos arquivos	<code>git reset HEAD~1</code>	! --mixed por padrão: fora do staging mas no disco — não depois do push
✅ Desfazer um commit que já foi enviado	<code>git revert <commit></code>	✓ Adiciona um commit novo que o cancela — seguro em branches compartilhadas
✅ Guardar minhas mudanças por enquanto	<code>git stash push -m "wip"</code>	✓ Volte com <code>git stash pop</code> — arquivos novos não rastreados precisam de -u
❌ Descartar o último commit e suas mudanças	<code>git reset --hard HEAD~1</code>	⚠ Suas mudanças e as alterações sem commit em arquivos rastreados se perdem (o git reflog ainda encontra o commit)
✅ Encontrar um commit “perdido” (depois de reset, rebase, branch excluída)	<code>git reflog</code>	✓ Todos os lugares por onde o HEAD passou — depois <code>git branch rescue <hash></code>
✅ Parar um merge / rebase / cherry-pick em andamento	<code>git merge --abort</code>	✓ Ou <code>rebase --abort</code> , <code>cherry-pick --abort</code>
⚠ Substituir a branch remota depois de reescrever o histórico	<code>git push --force-with-lease</code>	! Recusa se a branch remota mudou desde o seu último fetch

STASH

git stash

Guardar mudanças sem commit

git stash -u

Incluir arquivos não rastreados (novos)

git stash list

Todos os stashes, o mais recente = `stash@{0}`

git stash pop

Reaplica o mais recente e o remove (fica guardado se houver conflito)

git stash apply stash@{1}

Reaplicar um e mantê-lo na lista

RESET — TRÊS INTENSIDADES

git reset --soft <commit>

! Move a branch; preparação e arquivos intactos

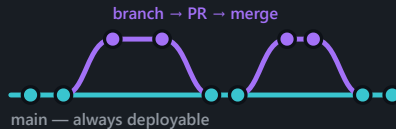
git reset <commit>

! Move a branch e redefine a preparação; arquivos intactos

git reset --hard <commit>

⚠ Move a branch, a preparação E os arquivos

GITHUB FLOW



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

Começar de um main atualizado

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

Uma branch de curta duração por mudança

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

Commit, push, abrir um pull request

Mesclar o pull request no GitHub

Depois da revisão; em seguida, excluir a branch

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

De volta à main; apaga também as cópias origin/... de branches apagadas

GIT FLOW



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

As funcionalidades começam a partir do develop

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

Concluir uma funcionalidade no develop

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

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

Congelar uma versão

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

...criar a tag e mesclar de volta no develop

TRUNK-BASED



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

Commits pequenos e completos no main

```
git pull --rebase
```

⚠ Reaplica os seus sobre os dos outros (hashes novos)

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

Depois de corrigir um conflito: marque como resolvido e continue

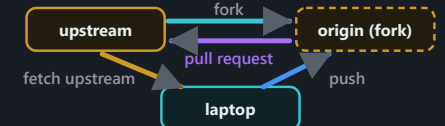
```
git push
```

Integrar pelo menos uma vez por dia

Entregar o trabalho inacabado desligado

Atrás de uma feature flag, por ex. NEW_CHECKOUT = false

FORKING



```
git remote add
upstream
```

```
<original-url>
```

Uma vez: o original como segundo remoto

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

Sincronizar o seu main com o original

```
git push
```

Atualizar o seu fork (origin)

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

Trabalhar em uma branch do seu fork

Abrir um pull request para o original

Mantenedores revisam e mesclam