

# Git: Além do Pull, Commit e Push

Wall Berg Moraes

INSTITUTO FEDERAL DO PIAUÍ - CAMPUS FLORIANO  
VirteX Telecom

24 de Junho de 2025

# Quem sou eu

- Formado em Engenharia de Computação pela UTFPR
- +6 anos de experiencia em Desenvolvimento Web focado em Backend.
- Coordenador de Desenvolvimento da VirteX Telecom
- Engenheiro de Software na BRNX



- Revisão rápida de conceitos básicos (`clone`, `add`, `commit`, `push` e `pull`).
- Aprender a respeito do `git rebase`.
- Aprender a respeito do `git merge`.
- Aprender a respeito do `git cherry-pick`.
- Aprender a respeito do fluxo de trabalho com `git` entre equipes.
  - Criar nova branch para merge request (MR).
  - Deixar histórico limpo ao realizar o MR.
  - Editar commits.
  - Atualizar MR (com rebase).

# Clone do Projeto

## Repositório

<https://github.com/wallberg13/cais-tech-2025>

# Revisão rápida de conceitos básicos (clone, commit, push e pull)

```
git clone <url>
```

Realizar cópia local de um repositório Git remoto. É possível fazer clone tanto em `http`, como com `ssh`. Sempre recomendado com `ssh`.

```
git add <arquivo> ou git add .
```

Após fazer alterações em seus arquivos, você precisa informar ao Git quais dessas alterações você deseja incluir no próximo commit.

```
git commit -m "Mensagem do commit"
```

Depois de adicionar as alterações à staging area, o `git commit` as salva permanentemente no histórico do repositório local. A mensagem de commit (`-m`) precisa ser clara a respeito da justificativa do por que o commit existe.

# Revisão rápida de conceitos básicos (clone, commit, push e pull)

## `git push`

Este comando envia seus commits locais para o repositório remoto. É assim que suas alterações se tornam visíveis para outros membros da equipe.

## `git pull`

Usado para buscar e integrar as alterações do repositório remoto para o seu repositório local. É essencial para manter seu código atualizado com o trabalho de outros colaboradores.

# Introdução: Git Merge vs Git Rebase

- Em projetos com múltiplas pessoas ou branches simultâneas, é comum que diferentes partes do código sejam desenvolvidas em paralelo. Eventualmente, essas mudanças precisam ser integradas — e é aí que entram dois comandos essenciais do Git: `git merge` e `git rebase`.
- Ambos têm o mesmo objetivo: **unir ramificações distintas**, mas fazem isso de formas diferentes.
  - O `git merge` mantém o histórico como uma árvore, preservando a ordem original dos commits e criando um commit de merge para marcar o ponto de junção.
  - O `git rebase`, por outro lado, reescreve o histórico para criar uma linha do tempo linear, como se todos os commits tivessem sido feitos em sequência.
- Essa diferença de abordagem traz implicações importantes para a clareza do histórico, a resolução de conflitos e até a colaboração em equipe.

# Introdução: Git Merge vs Git Rebase

Basicamente, a diferença entre ambos:

## `git merge`

Resultado final de um Merge Request, realizar o merge do código que está sendo submetido para a branch principal.

## `git rebase`

Em uma equipe grande, várias pessoas fazem merge em branch principais, e a forma de manter a branch atualizada, é feito o rebase.



# Git Rebase

## Conceitos Fundamentais

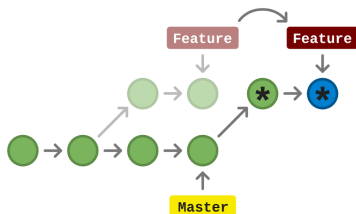
O que é Rebase?

O **git rebase** reescreve o histórico de *commits*, movendo ou combinando uma sequência de *commits* para uma nova base.

Resultado: histórico **linear e limpo**, sem *commits* de *merge* adicionais.

### CLI

```
git checkout minha-feature  
git rebase <branch-base>
```



# Git Rebase

## Interativo

### CLI

```
git checkout minha-feature  
git rebase -i <branch-base ou commit base>
```

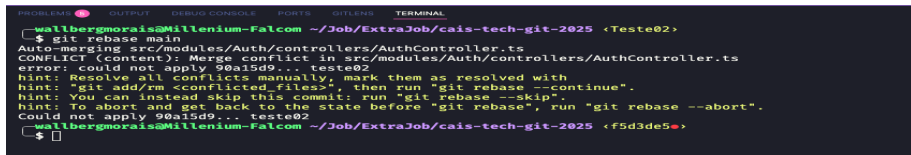
### Opções

- **p (pick)**: usar commit como está
- **r (reword)**: editar mensagem
- **e (edit)**: parar para editar
- **s (squash)**: mesclar com anterior
- **f (fixup)**: mesclar sem mensagem
- **d (drop)**: remove commit

# Git Rebase

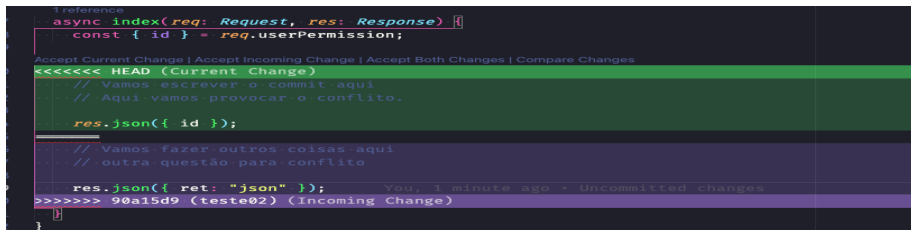
## Conflitos

- Fiz rebase e deu conflito, e agora?



```
wallbergmoraism@Millenium-Falcom ~/Job/ExtraJob/cais-tech-git-2025 <Teste02>
$ git rebase main
Auto-merging src/modules/Auth/controllers/AuthController.ts
CONFLICT (content): Merge conflict in src/modules/Auth/controllers/AuthController.ts
error: could not apply 90a15d9... teste02
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 90a15d9... teste02
wallbergmoraism@Millenium-Falcom ~/Job/ExtraJob/cais-tech-git-2025 <f5d3de5>
$
```

Figura 1: Mensagem no Terminal



```
1 reference
asynrc index(req: Request, res: Response) {
  const { id } = req.userPermission;

  <<<<<<< HEAD (Current Change)
  // Vamos escrever o commit aqui
  // Aqui vamos provocar o conflito.
  res.json({ id });

  // Vamos fazer outras coisas aqui
  // outra questão para conflito

  res.json({ ret: "json" });
>>>>>> 90a15d9 (teste02) (Incoming Change)
```

Figura 2: Conflitos no VSCode

- Edita-se o trecho de código foi realizado o rebase.



```
1 reference
  async index(req: Request, res: Response) {
    const { id } = req.userPermission;

    // Vamos fazer outras coisas aqui
    // outra questão para conflito

    res.json({ ret: "json" });
  }
You, 2 days ago · first commit
```

Figura 3: Código após rebase

- E executamos os seguintes comandos:

### CLI

```
git add -A
git rebase --continue
```

# Git Rebase

## Conflitos

```
Could not apply 90a15d9... teste02
wallbergmoraism@Millenium-Falcom ~/Job/ExtraJob/cais-tech-git-2025 <f5d3de5>
$ git status
interactive rebase in progress; onto f5d3de5
Last command done (1 command done):
  pick 90a15d9 teste02
No commands remaining.
You are currently rebasing branch 'Teste02' on 'f5d3de5'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
    both modified:   src/modules/Auth/controllers/AuthController.ts

no changes added to commit (use "git add" and/or "git commit -a")
wallbergmoraism@Millenium-Falcom ~/Job/ExtraJob/cais-tech-git-2025 <f5d3de5>
$ git add -A
wallbergmoraism@Millenium-Falcom ~/Job/ExtraJob/cais-tech-git-2025 <f5d3de5>
$ git rebase --continue
[detached HEAD d1ecc5e] teste02
1 file changed, 3 insertions(+), 3 deletions(-)
Successfully rebased and updated refs/heads/Teste02.
wallbergmoraism@Millenium-Falcom ~/Job/ExtraJob/cais-tech-git-2025 <Teste02>
$
```

Figura 4: Execução no Terminal

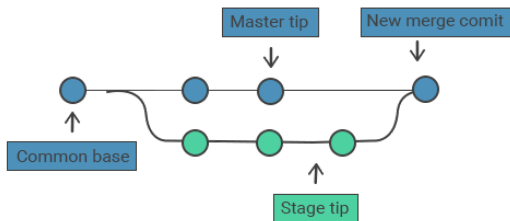
# Git Merge

O que é Merge?

O **git merge** unifica duas ou mais *branches* em uma só. É sempre utilizado para incorporar as mudanças de uma *branch* (como *feature*, *dev*, etc) para outra (como *main* ou *master*), preservando o histórico de *commits*.

## CLI

```
git checkout minha-feature  
git merge <branch-base>
```



# Git Merge

## Uso do Merge

Geralmente, no dia a dia, o `git merge` é pouco utilizado, se o time na qual você trabalha, sempre realiza merge de código em um *Pull Request* ou *Merge Requests*, o botão de “merge” roda esse comando implicitamente.

## Conflitos

Geralmente, os conflitos são feitos ao fazer o *rebase* da *branch* raiz, sem precisar resolver conflitos durante o *merge*. E o GitHub (ou GitLab), não deixa realizar *merge* com conflitos.



### This branch has conflicts that must be resolved

Use the [web editor](#) or the [command line](#) to resolve conflicts.

#### Conflicting files

content/issues/tracking-your-work-with-issues/linking-a-pull-request-to-an-issue.md

Resolve conflicts

# Git Cherry-Pick

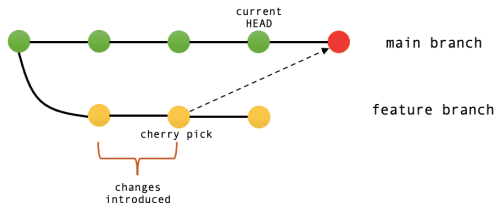
## Conceito

O que é Cherry-Pick?

O **git cherry-pick** permite selecionar um ou mais *commits* de uma branch e aplicá-los em outra branch. Muito utilizado para quando você precisa de um código que ainda está em desenvolvimento (não foi mergeado para a branch principal), mas que possuía uma funcionalidade necessária para sua task.

## CLI

```
git cherry-pick <commit hash>...<commit hash>
```





- **Hotfixes:** É o uso mais comum. Imagine um bug crítico encontrado em produção no seu software, e a correção foi desenvolvida em uma *feature branch* ou em uma *hotfix branch*. Em vez de mergear toda a branch, você pode utilizar o *cherry-pick* apenas para pegar o commit necessário.
- **Aplicar um commit específico em outra branch:** Suponha que um dev implementou uma melhoria ou correção não crítica em uma *feature branch*, mas que essa mudança também é relevante para outra *feature branch*. Em vez de reescrever o código ou mesclar *branches* inteiras, o *cherry-pick* permite você aplique somente o *commit* necessário.

# Git Cherry-Pick

## Observação

### Atenção

O *cherry-pick* realiza duplicação de commit / código. Use com cuidado, para sempre manter o histórico de commits limpo.

# Outros comandos

## Editando o último commit

```
git commit -amend
```

Muito útil quando você esquece de adicionar aquele arquivo no seu commit.

## Push Forçado

```
git push origin <branch> -f
```

**Muita atenção!** Push forçado reescreve o que está feito na branch. O uso mais comum é sempre depois de um *rebase*, ou depois de um *commit amend*.

## Reset

```
git reset <commit>
```

Serve para resetar algum commit, ou alguns commits. Os arquivos alterados entre os dois pontos, voltam como se tivessem em modificação.

## Reset hard

`git reset -hard <commit>` Volta, sem mostrar os arquivos salvos, para um commit de referência. Muito útil quando desejamos forçar que o nosso repositório esteja igual ao repositório remoto, por exemplo.

## Revert

`git revert <commit>`

Serve para rever algum commit. Sabe aquele commit que subiu em produção e agora você precisa desfazer o que ele fez? É um “ctrl+z” depois de subir um código em produção.



- LinkedIn: <https://www.linkedin.com/in/wallbergmoraais/>
- Instagram: @\_\_wallberg