**Merge e Rebase na prática**

Tanto o **Merge** quanto o **Rebase** são formas de integrar mudanças de uma branch para outra no Git, mas funcionam de maneiras diferentes. Vamos analisar ambos na prática:

**1. Merge**

O **Merge** combina o histórico de commits de duas branches criando um novo commit chamado **"merge commit"**, que preserva o histórico original das duas.

**Cenário:**

* Você tem uma branch main e outra feature.
* Na feature, fez alterações que quer integrar na main.

**Comandos:**

# Na branch main

git checkout main

# Faz o merge da branch feature na main

git merge feature

**Resultado:**

* O histórico da main terá os commits da feature **sem modificar os commits originais da feature**.
* Será criado um **commit de merge**, representando o ponto onde as branches se encontraram.

**Exemplo de histórico após o merge:**

\* Merge commit (main)

|\

| \* Commit na feature

| \* Outro commit na feature

\* | Commit na main

\* | Outro commit na main

**Vantagens:**

* Preserva todo o histórico.
* Ideal para projetos colaborativos, onde se deseja manter um histórico completo de como as mudanças foram integradas.

**Desvantagens:**

* Pode deixar o histórico "poluído" com muitos commits de merge, especialmente em projetos grandes.

**2. Rebase**

O **Rebase** move os commits de uma branch para outro ponto no histórico, reescrevendo os commits como se tivessem sido feitos em uma linha contínua.

**Cenário:**

* Mesma situação: você tem uma branch main e outra feature.
* Quer integrar feature na main.

**Comandos:**

# Na branch feature

git checkout feature

# Rebase a branch feature no topo da main

git rebase main

**Resultado:**

* Os commits da feature serão reaplicados **por cima** dos commits da main.
* Não cria um merge commit, o histórico parecerá linear.

**Exemplo de histórico após o rebase:**

\* Commit na feature (reescrito)

\* Outro commit na feature (reescrito)

\* Commit na main

\* Outro commit na main

**Vantagens:**

* Histórico limpo e linear.
* Ideal para projetos onde o histórico limpo é uma prioridade.

**Desvantagens:**

* Reescreve o histórico, o que pode causar problemas se a branch já tiver sido compartilhada com outros desenvolvedores.

**Merge ou Rebase: Qual usar?**

* **Use Merge** quando:
  + Quer preservar o histórico completo de todas as branches.
  + Está trabalhando em equipe e precisa evitar problemas de reescrita de histórico.
* **Use Rebase** quando:
  + Quer um histórico limpo e linear.
  + Está trabalhando sozinho ou em uma branch local não compartilhada.

**Na prática em projetos colaborativos:**

1. Trabalhe na branch feature normalmente.
2. Antes de fazer merge na branch principal, use rebase para manter a branch atualizada:
3. git checkout feature
4. git pull --rebase origin main
5. Resolva conflitos, se houverem, e finalize o rebase:
6. git rebase --continue
7. Depois, faça o merge na branch principal sem problemas:
8. git checkout main
9. git merge feature

Essas práticas ajudam a evitar conflitos grandes e mantêm o histórico organizado.