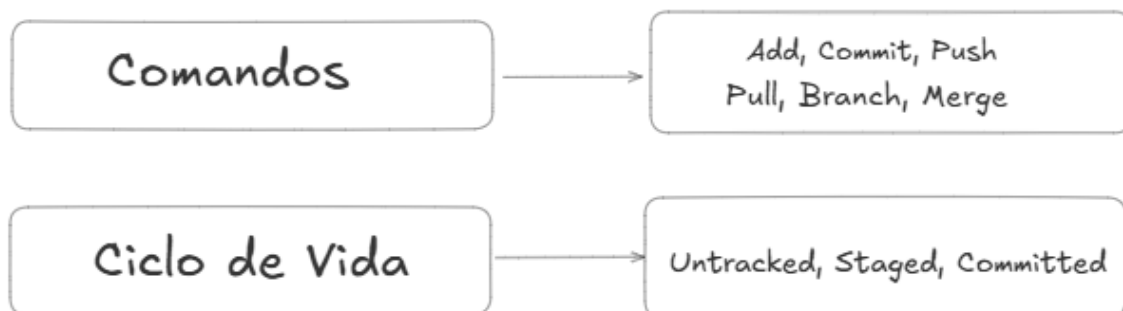


26 - Prática: Git e Github para Iniciantes (III)

Higor Miller Grassi

Descrição da Atividade: Nesta atividade nós aprendemos desde a instalação do git até a manipulação dele com o cmd, criando repositórios e modificando o mesmo aprendendo conceitos fundamentais de versionamento e controle de código. Também vimos a importância dele em um sistema corporativo, já que facilita o trabalho em grupo, diminuindo a quantidade de conflitos.

História do Git: Uma empresa chamada BitKeeper versionar o código do Linux, porém com a quebra da mesma empresa, ela não deixou isento da taxa para armazenar o sistema operacional. Linus Torvalds(criador), não quis pagar pelo sistema, e decidiu criar o Git com as melhorias e deficiências que ele percebia, no caso de velocidade, design mais simples, um suporte a desenvolvimento não linear para suportar melhor o trabalho em equipe, totalmente distribuído e capaz de lidar com projetos grandes



Git: O Git é um sistema de controle de versão distribuído usado para gerenciar e acompanhar mudanças em arquivos, especialmente no desenvolvimento de software. permitindo que vários desenvolvedores trabalhem simultaneamente no mesmo projeto, rastreando o histórico de alterações, facilitando a colaboração, o controle de diferentes versões do código e a integração de novas funcionalidades de forma segura e organizada.

GitHub: O GitHub é uma plataforma de hospedagem de código-fonte que utiliza o Git para controle de versão, sendo um serviço de web compartilhado. Ele permite que desenvolvedores armazenem, compartilhem e colaborem em projetos de software de forma online, e além de tudo isso, oferece recursos para controle de issues, revisão de código, integração contínua e documentação, facilitando o trabalho em equipe e o desenvolvimento de projetos open source ou privados.

Configuração do Sistema:

Define o nome do user e o seu email:

```
C:\Users\Higor>git config --global user.name "Higor Miller Grassi"
C:\Users\Higor>git config --global user.email "higormiller55@gmail.com"
```

Define o Vscode como padrão:

```
C:\Users\Higor>git config --global core.editor "\"C:\\Users\\Higor\\AppData\\Local\\Programs\\Microsoft VS Code\\bin\\code\"" --wait
C:\Users\Higor>git config --global core.editor
"C:\\Users\\Higor\\AppData\\Local\\Programs\\Microsoft VS Code\\bin\\code" --wait
```

Criei o repositório e entrei no mesmo:

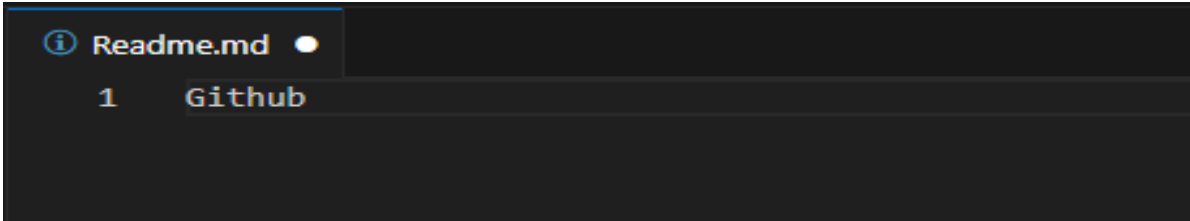
```
C:\Users\Higor>mkdir git-course
C:\Users\Higor>cd git-course
```

Inicializei o git:

```
C:\Users\Higor\git-course>git init
Initialized empty Git repository in C:/Users/Higor/git-course/.git/
```

Como estou utilizando o Vscode como padrão, utilizo este código para abrir o Readme.md:

```
C:\Users\Higor\git-course>code Readme.md
C:\Users\Higor\git-course>_
```



O ciclo de vida do Git: Ele é composto pelas seguintes fases:

- Untracked: Arquivos que foram adicionados ao diretório de trabalho, mas que o Git ainda não está monitorando.
- Unmodified: Arquivo já rastreado pelo Git, mas sem alterações desde o último commit.
- Modified: Arquivo que foi alterado, mas ainda não foi adicionado à área de preparação.
- Staged: Arquivos que foram modificados e preparados para serem incluídos no próximo commit.
- Committed: Arquivos que foram registrados no repositório local através do comando git commit, sendo salvo de forma permanente no repositório Git.

Serve para verificar qual o status do git até o momento(sem comit):

```

C:\Users\Higor\git-course>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
C:\Users\Higor\git-course>
```

Neste caso, modifiquei o readme.md, ele apareceu Untracked

```

C:\Users\Higor\git-course>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  Readme.md

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\Higor\git-course>
```

Utilizando o add, temos o arquivo pronto para ser comitado, estando no momento Staged:

```

C:\Users\Higor\git-course>git add Readme.md

C:\Users\Higor\git-course>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Readme.md

C:\Users\Higor\git-course>_

```

Caso eu modifique algo após dar o add no readme.md, ele aparecerá no estágio de Modified(tendo que dar o add novamente no arquivo para assim poder comitar):

```

C:\Users\Higor\git-course>code Readme.md

C:\Users\Higor\git-course>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Readme.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   Readme.md

C:\Users\Higor\git-course>

```

Criou um master no diretório atual, com a hash e comentário a seguir, nunca tendo uma versão e números iguais, assim sendo bom para caso precise voltar para alguma versão, apenas precise pegar o número correto

```

C:\Users\Higor\git-course>git commit -m "Add Readme.me"
[master (root-commit) ea6eab3] Add Readme.me
 1 file changed, 3 insertions(+)
 create mode 100644 Readme.md

C:\Users\Higor\git-course>

```

Com o git log conseguimos adquirir a hash, quem foi o autor e a modificacao feita, a data e a descrição do comit

```
C:\Users\Higor\git-course>git log
commit ea6eab3c8f7aa39961236d23d243f60379c3d3ba (HEAD -> master)
Author: Higor Miller Grassi <higormiller55@gmail.com>
Date:   Sun Feb 2 15:50:27 2025 -0300

    Add Readme.me

C:\Users\Higor\git-course>
```

Com o git log --decorate mostra as informações por exemplo de qual branch para qual branch, se houve um merge etc

```
C:\Users\Higor\git-course>git log --decorate
commit ea6eab3c8f7aa39961236d23d243f60379c3d3ba (HEAD -> master)
Author: Higor Miller Grassi <higormiller55@gmail.com>
Date:   Sun Feb 2 15:50:27 2025 -0300

    Add Readme.me

C:\Users\Higor\git-course>
```

Tem como filtrar por autor:

```
C:\Users\Higor\git-course>git log --author="Higor"
commit ea6eab3c8f7aa39961236d23d243f60379c3d3ba (HEAD -> master)
Author: Higor Miller Grassi <higormiller55@gmail.com>
Date:   Sun Feb 2 15:50:27 2025 -0300

    Add Readme.me

C:\Users\Higor\git-course>
```

Com o shortlog podemos ver em ordem alfabética os autores e quantos commits fizeram:

```
C:\Users\Higor\git-course>git shortlog
Higor Miller Grassi (1):
    Add Readme.me

C:\Users\Higor\git-course>
```

Mostra em forma gráfica mais detalhado:

```
C:\Users\Higor\git-course>git log --graph
* commit ea6eab3c8f7aa39961236d23d243f60379c3d3ba (HEAD -> master)
   Author: Higor Miller Grassi <higormiller55@gmail.com>
   Date:   Sun Feb 2 15:50:27 2025 -0300

       Add Readme.me

C:\Users\Higor\git-course>
```

Com o git diff, conseguimos ver o q foi modificado antes de dar o commit, tendo várias maneiras de utilizar, por exemplo com o git diff --name-only podemos ver apenas o nome do arquivo que foi modificado:

```
C:\Users\Higor\git-course>git diff
diff --git a/Readme.md b/Readme.md
index 88116ed..80f92e2 100644
--- a/Readme.md
+++ b/Readme.md
@@ -1,3 +1,5 @@
   Github - coruse

- Este e um repositorio teste
\ No newline at end of file
+ Este e um repositorio teste
+
+ Gostou do curso?
\ No newline at end of file

C:\Users\Higor\git-course>
```

Com o git checkout conseguimos voltar para antes da atualização indesejada, e com o git reset HEAD conseguimos voltar mesmo se anteriormente demos add no repositório:

```
C:\Users\Higor> Prompt de Comando

C:\Users\Higor\git-course>git checkout Readme.md
Updated 1 path from the index

C:\Users\Higor\git-course>git diff

C:\Users\Higor\git-course>
```

O git reset --soft(apenas volta o commit para o estágio staged), mixed(matar o commit no estágio modified) e hard(ignorar a existência do commit e matar tudo):

```
C:\Users\Higor\git-course>git reset --soft 841264d985145f2d15095a436cced75b428ca951

C:\Users\Higor\git-course>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Readme.md

C:\Users\Higor\git-course>
```

```
C:\Users\Higor\git-course>git reset --mixed 841264d985145f2d15095a436cced75b428ca951
Unstaged changes after reset:
M       Readme.md

C:\Users\Higor\git-course>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Readme.md

no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\Higor\git-course>_
```

```
C:\Users\Higor\git-course>git reset --hard 841264d985145f2d15095a436cced75b428ca951
HEAD is now at 841264d Edit Readme.md
```

Criando e add uma chave SSH, e com essa chave, basta add no github

```
C:\Users\Higor\git-course>ssh-keygen -t rsa -b 4096 -C "higormiller55@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\Higor/.ssh/id_rsa):
Created directory 'C:\Users\Higor/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\Higor/.ssh/id_rsa
Your public key has been saved in C:\Users\Higor/.ssh/id_rsa.pub
The key's fingerprint is:
SHA256:PGDOqsuQNE0/bEbrscG0Zw+8bWzLZpFx/d5PT7VMiTk higormiller55@gmail.com
The key's randomart image is:
+---[RSA 4096]-----+
|
|  .oo
| o *++o . . .o .
| o . %o=S +   E.o
| ... +.B *+   +.o
| o . o . * . .o+
| o .   +o   +o
| +.   oo   +
+---[SHA256]-----+

C:\Users\Higor\git-course>_
```

Dando push para o repositório remoto:

```
C:\Users\Higor> cd C:\Users\Higor\git-course

C:\Users\Higor\git-course>git remote add origin git@github.com:HigorMillerGrassi/github-course.git

C:\Users\Higor\git-course>git remote -v
origin  git@github.com:HigorMillerGrassi/github-course.git (fetch)
origin  git@github.com:HigorMillerGrassi/github-course.git (push)
```

```

C:\Users\Higor\git-course>git remote set-url origin https://github.com/higormilleer/github-course
C:\Users\Higor\git-course>git push -u origin master
info: please complete authentication in your browser...
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 24 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 549 bytes | 549.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/higormilleer/github-course/pull/new/master
remote:
To https://github.com/higormilleer/github-course
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

```

Para transitar entre os branches, e para excluir basta escrever `git branch -D <nome_do_branch>`:

```

C:\Users\Higor\git-course>git branch test
C:\Users\Higor\git-course>git branch
* master
  test

C:\Users\Higor\git-course>git checkout test
Switched to branch 'test'

C:\Users\Higor\git-course>git branch
  master
* test

```

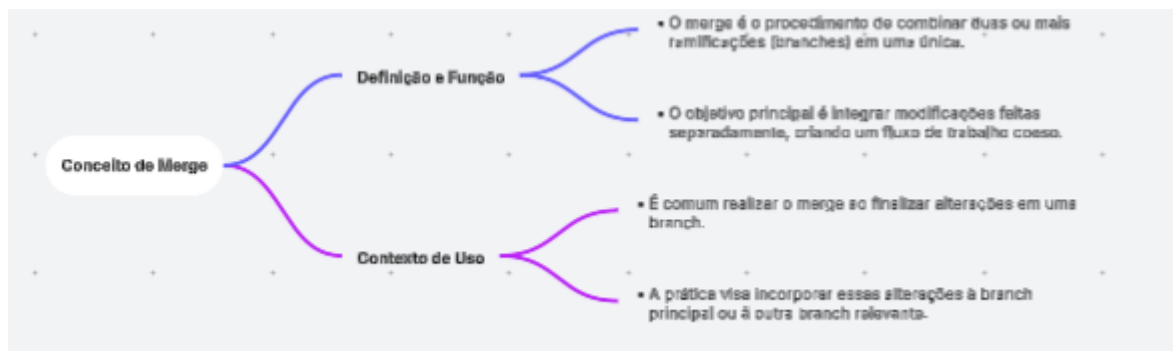
Repositórios remotos: são cópias de um repositório Git armazenadas em servidores externos, acessíveis pela internet ou por uma rede interna, possibilitando que várias pessoas colaborem simultaneamente em um projeto, compartilhando e sincronizando suas edições. O repositório remoto funciona como um ponto central de referência, permitindo que usuários enviem (push) e recebam (pull) atualizações.

Branch: trata-se de uma trilha independente de desenvolvimento que possibilita a criação de diferentes versões de um projeto ao mesmo tempo. Cada branch mantém seu próprio histórico de commits, permitindo que novas funcionalidades sejam testadas, problemas sejam corrigidos e ideias inovadoras sejam exploradas sem modificar a versão principal do projeto.

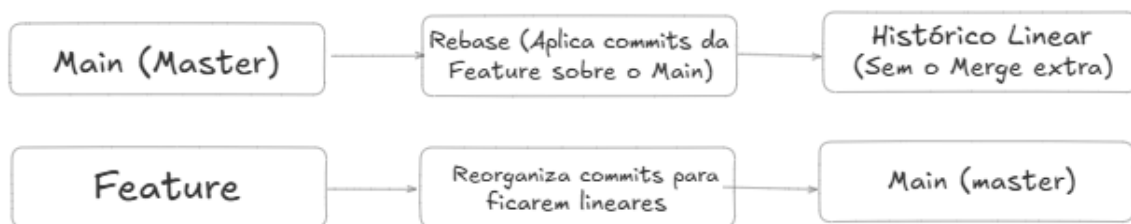
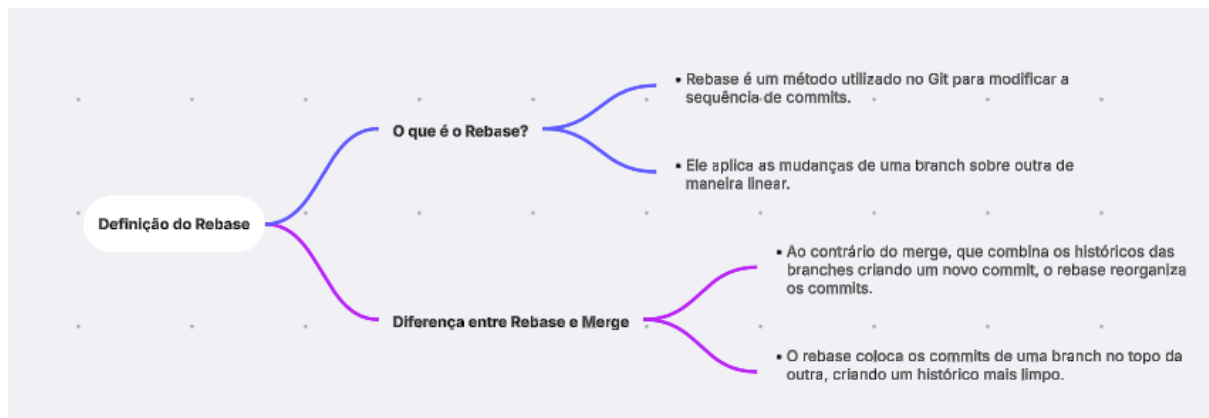




Merge: Procedimento de unir duas ou mais ramificações em uma só, junta as modificações feitas separadamente em diferentes branches, deixando em um único fluxo de trabalho. O merge costuma ser realizado ao finalizar as alterações em uma branch e ao querer incorporá-las a outra, como a principal.



Rebase: Método utilizado para modificar a sequência de commits, aplicando as mudanças de uma branch sobre outra de maneira linear. Ao contrário do merge, que simplesmente junta os históricos das branches criando um novo commit, o rebase reorganiza os commits de uma branch, colocando-os no topo da outra, fazendo com que o histórico pareça ter sido construído de forma sequencial.



Fork: Cópia de um repositório no GitHub, permitindo que você faça alterações de forma independente sem afetar o projeto original. É comum para contribuir em projetos de código aberto, onde você faz alterações no seu fork e, se desejar, enviar essas mudanças de volta ao repositório original através de um pull request, diferentemente do clone que podemos realizar apenas para o nosso repositórios.



Conceito	O que é / faz	Aplicação / Importância
Git	Sistema de controle de versão distribuído que rastreia alterações no código e gerencia histórico.	Permite colaboração simultânea e segura entre desenvolvedores.
GitHub	Plataforma online que hospeda repositórios Git, adicionando recursos como pull requests, issues, etc.	Ideal para colaboração remota, código aberto e integração com outras ferramentas.
Ciclo de vida do Git	Etapas de um arquivo: Untracked → Staged → Committed.	Ajuda a entender o estado atual dos arquivos em relação ao versionamento.
Commit	Registro permanente de alterações no repositório local.	Guarda versões específicas do projeto, podendo ser recuperadas.
Branch	Linha de desenvolvimento independente.	Usado para testar novas funcionalidades sem afetar a versão principal.
Merge	Junta alterações de duas branches.	Unifica desenvolvimentos paralelos em um único fluxo.
Rebase	Reorganiza os commits para manter o histórico linear.	Mantém o histórico de alterações mais limpo e sequencial.
Fork	Cópia independente de um repositório remoto.	Permite contribuir com projetos sem afetar diretamente o original (ex: projetos open source).
Reset	Reverte alterações no repositório local (soft, mixed, hard).	Corrige erros sem afetar o repositório remoto.
Push / Pull	Envia (push) ou recebe (pull) alterações do repositório remoto.	Sincroniza o trabalho local com o repositório compartilhado.

Conclusão: Entender Git é fundamental para empresas porque ele facilita o controle de versões, permite colaboração eficiente entre equipes e garante o rastreamento de mudanças no código. Com Git, múltiplos desenvolvedores podem trabalhar simultaneamente sem conflitos, além de ser possível reverter alterações e manter o histórico de progresso de projetos, aumentando a produtividade e a organização.