**Estudos Git e Github**

Criando uma pasta direto pelo terminal GitBash

Comandos:

**1. mkdir git-estudo**

Cria uma nova pasta chamada git-estudo.  
📌 *Usado para organizar seus arquivos em um diretório.*

**2. cd git-estudo**

Entra na pasta git-estudo.  
📌 *Usado para navegar até o diretório onde você quer trabalhar com Git.*

**3. git init**

Inicializa um repositório Git vazio na pasta atual.  
📌 *Cria uma pasta oculta .git que guarda todo o histórico e configurações do Git.*

**4. git status**

Mostra o estado atual do repositório.  
📌 *Informa se há arquivos novos, modificados, ou prontos para commit.*

**5. git add nome-do-arquivo**

Adiciona o arquivo para a "área de preparação" (staging area).  
📌 *Diz ao Git que aquele arquivo deve ser incluído no próximo commit.*

**git add . –** adiciona todo conteúdo que está faltando ou foi alterado

**6. git commit -m "mensagem"**

Registra as alterações salvas no staging area como um ponto no histórico.  
📌 *Cria uma versão com mensagem explicando o que foi feito.*

Exemplo:

git commit -m "Adicionado o arquivo word contendo estudos git/github"

**7. git log**

Exibe o histórico dos commits realizados no repositório.  
📌 *Mostra os autores, mensagens de commit, datas e identificadores únicos.*

$ git log

commit 5c6c15a54ef8c4dc33e81f2648283dd675f8329c (**HEAD** -> **master**)

Author: kadubass2025 <kadubass@hotmail.com>

Date: Mon Apr 7 07:53:43 2025 -0300

Adicionado o arquivo word contendo estudos git/github

## 🧪 ETAPA 1.2 — Trabalhando com alterações

### 🎯 Objetivo:

* Ver como o Git detecta mudanças
* Usar git diff para ver o que foi alterado
* Fazer novos commits para salvar essas mudanças

### 8. git diff

Compara o conteúdo atual do(s) arquivo(s) com a última versão commitada.  
📌 Mostra **o que foi modificado**, **adicionado** ou **removido**, antes de fazer um novo commit.  
Se o arquivo tiver **espaços no nome**, coloque entre aspas:

git diff "Estudos Git e Github.docx"

## ✅ ETAPA 1.3 — **Remoção e recuperação de arquivos**

### 🎯 Objetivo:

Aprender como **remover arquivos com o Git**, e como **restaurar arquivos excluídos ou modificados** antes do commit. Isso vai te ajudar a não entrar em pânico quando “apagar algo sem querer” 😅

### 9. git rm "nome-do-arquivo"

Remove o arquivo do projeto e do controle de versão.  
📌 O arquivo é deletado fisicamente da pasta e o Git marca para remoção no próximo commit.

### 10. git restore "nome-do-arquivo"

Restaura um arquivo modificado ou deletado antes do commit.  
📌 O Git pega a **última versão salva** (do último commit) e recupera o arquivo.

### 11. git restore pode falhar se o arquivo ainda ****não foi commitado****

📌 Se o Git **ainda não salvou nenhuma versão** do arquivo, ele não poderá restaurar.  
📌 Sempre faça um commit ao menos **uma vez** antes de deletar arquivos importantes.

### 12. git checkout HEAD^ -- "nome-do-arquivo"

📌 Restaura a versão do arquivo a partir do **commit anterior ao atual**.  
💡 Útil quando você **já cometeu a remoção** de um arquivo e precisa recuperá-lo.

 HEAD = commit atual (onde o arquivo está **removido**)

 HEAD^ = commit anterior (onde o arquivo ainda **existe**)

git checkout HEAD^ -- "Estudos Git e Github.docx"

## ✅ ETAPA 1.4 — **Branches (ramificações)**

### 🎯 Objetivo:

Aprender a criar **ramificações (branches)** para testar novas ideias, corrigir bugs ou desenvolver recursos **sem mexer na versão principal do projeto**.

## 🧠 O que é uma branch?

Pense na **branch** como uma **cópia da sua linha de trabalho**.

Você pode:

✅ Criar uma branch nova  
✅ Testar ou desenvolver algo nela  
✅ E depois juntar de volta na principal (master ou main) quando estiver tudo ok

### 13. git branch

📌 Mostra todas as branches do projeto e qual está ativa no momento.

### 14. git branch nome-da-branch

📌 Cria uma nova branch com o nome indicado, baseada na atual.

### 15. git checkout nome-da-branch ou git switch nome-da-branch

📌 Muda para a branch especificada.

### 16. git commit -m "mensagem"

📌 Registra alterações na branch atual.