**3 – Criar pasta de controle .git**

Essa aula foi toda prática. A seguir vou colar os passos dados para criação da pasta .git com meu primeiro commit.

**1) git add 'Aula 01-git-gitlab-github.docx'**

* **Antes** desse comando, o arquivo estava como **untracked** (o Git via o arquivo na pasta, mas ainda não o controlava).
* O git add pega o conteúdo **exato daquele instante** e coloca no **staging area** (índice).
* Internamente, o Git:
  + Calcula um **hash** do conteúdo e cria um **blob** (objeto de conteúdo) dentro de .git/objects/…
  + Anota no **índice** (arquivo .git/index) que esse caminho de arquivo deve entrar no próximo commit.

👉 Resultado prático: o arquivo passou de “untracked” → **staged** (pronto para commit).

**2) git commit -S -m "meu primeiro commit assinado"**

A saída que você mostrou:

[Principal (root-commit) e14295e] meu primeiro commit assinado

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 Aula 01-git-gitlab-github.docx

Significa:

* **Principal**: é o nome da sua **branch** (equivalente a main).
* **root-commit**: é o **primeiro commit** do repositório (não tem pai).
* **e14295e**: é o **ID (hash)** do commit (os 7 primeiros dígitos do SHA).
* **1 file changed… create mode 100644 …**: um arquivo foi criado e passou a ser rastreado; 100644 indica arquivo normal (não executável).
* Você usou -S (S maiúsculo) ⇒ o commit foi **assinado por GPG**. Em GitHub/GitLab ele aparecerá com selo **Verified** depois que você fizer git push.

👉 Resultado prático: o Git tirou a “foto” do estado dos arquivos **staged** e gravou no histórico. A partir daqui, esse .docx passou a ser **tracked** (versionado).

**O que é a pasta .git (e por que ela é sagrada)**

Essa pasta transforma a sua pasta normal em um **repositório Git**. Não edite nada ali manualmente. Principais itens que você viu:

* **HEAD**: aponta para a branch atual (ex.: refs/heads/Principal).
* **config**: configurações **do repositório** (remotos, opções locais, etc.).
* **index**: o **staging area** (lista do que vai para o próximo commit).
* **objects/**: onde o Git guarda **todo o conteúdo** (blobs dos arquivos, **trees** das pastas e **commits**), endereçados pelo hash.
* **refs/**: ponteiros para commits:
  + refs/heads/Principal → a ponta da sua branch
  + refs/tags/ → tags, se existirem
* **logs/**: **reflogs** (histórico de movimentos de branches/HEAD, útil para recuperar coisas).
* **hooks/**: scripts opcionais que rodam em eventos (pre-commit, post-commit, etc.).
* **info/**: inclui exclude, um “gitignore local” só desse repo.

Sem essa pasta, a sua árvore de arquivos vira apenas uma pasta comum; **com** ela, o Git sabe histórico, branches, objetos, etc.

**Como conferir o que ficou gravado**

No seu repo, rode:

git status

# Deve dizer "nothing to commit, working tree clean"

git log --oneline

# Deve mostrar: e14295e meu primeiro commit assinado

git log --show-signature -1

# Mostra os detalhes da assinatura GPG desse commit

git ls-files

# Lista os arquivos atualmente rastreados (inclui seu .docx)

**O que é uma *branch* no Git?**

* Uma **branch** é uma **ramificação do histórico** do repositório.
* Pense nela como uma **linha do tempo independente** onde você pode desenvolver funcionalidades, corrigir bugs ou testar coisas **sem mexer na linha principal**.

**📌 Analogia simples**

Imagine que o projeto é um livro:

* A **branch main** é a versão oficial publicada.
* Você quer escrever um novo capítulo → cria uma branch chamada feature-capitulo-novo.
* Trabalha ali, faz commits.
* Quando terminar, faz um **merge** (fusão) dessa branch de volta na main.

Assim o livro “oficial” só recebe o capítulo novo quando estiver pronto.

**🔑 Características das branches**

* **Leves**: no Git, branch nada mais é que um **ponteiro** para um commit.
* **Paralelas**: várias podem existir ao mesmo tempo.
* **Isoladas**: alterações em uma branch não afetam as outras, até que você faça merge.

**🛠️ Comandos úteis**

* Ver a branch atual:
* git branch

A branch atual aparece com um \*.

* Criar uma nova branch:
* git checkout -b minha-nova-branch
* Voltar para a main:
* git checkout main
* Fundir uma branch na main:
* git merge minha-nova-branch

Estabelecemos o repositório “treinamento” como o remoto, em seguida demos o push do main branch pra lá, dos arquivos commited e signeds.

Texto

O conteúdo gerado por IA pode estar incorreto.

Resultado 😁

Tela de celular com aplicativo aberto

O conteúdo gerado por IA pode estar incorreto.