1. **Conceitos do git**
   1. **O que é uma branch ?**

Uma **branch** é um "ramo" dentro do controle de versão (como o Git) que serve para você **trabalhar em uma linha separada do código** sem alterar a principal.

Exemplo:

* main ou master → código principal.
* Você cria uma **branch** → faz alterações/testes nela.
* Depois pode **juntar (merge)** de volta na principal.

Exemplos na videa real:

* Controle de versão de um codigo

**1.2 O que é um commit ?**

Um **commit** é como um **“salvamento” no histórico do Git**.

Quando você faz alterações no código (cria, edita ou remove arquivos) e dá um git commit, o Git:

* **Tira uma foto** do estado dos arquivos naquele momento.
* Guarda essa foto no histórico, junto com uma **mensagem** que você escreve para explicar o que mudou.
* Cada commit ganha um **ID único (hash)** que permite voltar nele quando quiser.

Exemplo prático:

1. Você altera um arquivo.
2. git add arquivo.txt (coloca a alteração na área de *staging*).
3. git commit -m "Adicionei função de login" (salva no histórico).

Resultado: você tem um **ponto no tempo** que pode ser consultado ou revertido depois.

Resumindo:  
 **Commit = checkpoint no desenvolvimento**.

1. **Comandos Git mais usados**

**2.1 Configuração inicial (só precisa fazer uma vez)**

**git config --global user.name "Seu Nome"**

**git config --global user.email "seuemail@exemplo.com"**

**2.2 Iniciar e clonar**

* **Criar repositório na pasta atual:**

**git init**

* Clonar repositório já existente:

**git clone url-do-repo**

**2.3 Status e histórico**

* Ver estado dos arquivos:

**git status**

* Ver histórico de commits:

**git log**

* Histórico resumido:

**git log --oneline**

**2.4 Adicionar e confirmar mudanças**

* **Adicionar um arquivo:**

**git add arquivo.txt**

* **Adicionar todos os arquivos modificados:**

**git add .**

* **Criar um commit:**

**git commit -m "Mensagem do commit"**

**2.5 Trabalhando com branches**

* Criar uma nova branch:

**git branch nome-da-branch**

* Trocar de branch:

**git checkout nome-da-branch**

* Criar e já trocar:

**git checkout -b nome-da-branch**

* Listar branches:

**git branch**

**2.6 Sincronizar com remoto**

* Enviar commits para o GitHub/GitLab:

git push origin nome-da-branch

* Baixar atualizações:

git pull origin nome-da-branch

**2.7 Mesclagem (merge)**

* Estando na branch principal (main ou master), juntar outra branch:

git merge nome-da-branch

**2.8 Comparando arquivos do que foi modificado**

**git diff**

O comando **git diff** mostra as diferenças entre versões de arquivos no repositório. Ele exibe **o que foi alterado no conteúdo** (linhas adicionadas, removidas ou modificadas).

Principais usos:

* **git diff** → mostra alterações feitas nos arquivos, mas que ainda **não foram adicionadas** com git add.
* **git diff –name-only ->** Mostra somente os arquivos alterados
* **git diff (nome do arquivo)** -> mostra somente as alterações nesse arquivo
* **git diff --staged** → mostra alterações que já foram adicionadas com git add, mas ainda **não foram commitadas**.
* **git diff <commit1> <commit2>** → compara as diferenças entre dois commits específicos.

📌 Exemplo:

git diff

Vai listar as mudanças linha por linha, usando - para remoções e + para adições.

1. **Como reverter modificações**

O comando git reset é usado para desfazer alterações no repositório Git. Ele pode atuar em três níveis:

* **git reset --soft <commit>** → Volta para um commit, mas mantém as alterações e arquivos.
* **git reset --mixed <commit>** (padrão) → Volta para um commit, remove da *staging area*, mas mantém os arquivos modificados no diretório.
* **git reset --hard <commit>** → Volta para um commit e apaga todas as modificações (tanto da *staging area* quanto do diretório de trabalho).

O <commit> pode ser o hash do commit ou algo como HEAD~1 (um commit antes do atual).

**⚠️ Atenção: --hard apaga mudanças que não foram salvas em commit.**