

# INTRODUÇÃO

AO

GIT



O Grupo de Estudos ENIAC é formado por quatro estudantes de Ciência da Computação da UFAL - Arapiraca. Acreditamos que o conhecimento é a chave para um futuro mais igualitário e tecnológico. Promovemos minicursos, estudos coletivos e atividades práticas para compartilhar aprendizado e mostrar que a tecnologia pode ser fascinante e acessível a todos.

[estudos.eniac](mailto:estudos.eniac)

# MINISTRANTES



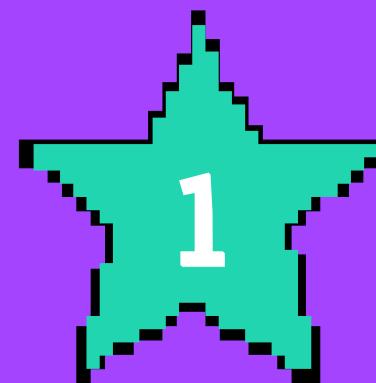
**EUELLYN rodrigues**

- Desenvolvedora frontend no Sonoria e no PlanUrbi;
- Coordenadora do projeto de extensão Eniac;
- Estudante do 7º período de Ciência da Computação na UFAL - Campus Arapiraca

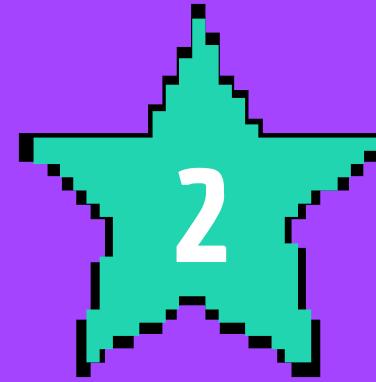


**JAMILY barbosa**

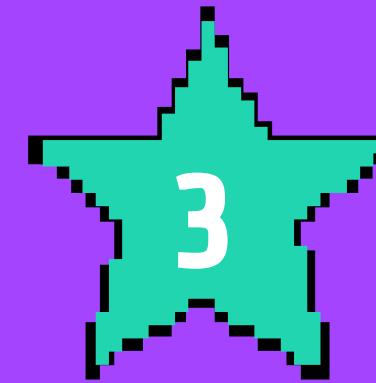
- Estudante do 3º de Ciência da Computação na UFAL - Campus Arapiraca;
- Coordenadora de Marketing do projeto de Extensão Eniac;
- Bolsista CNPq no projeto {Katie}, de pesquisa e extensão



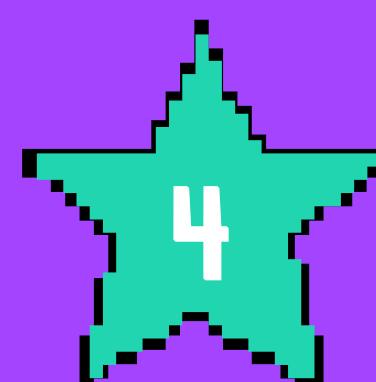
Versionamento  
de código



Surgimento do  
GIT



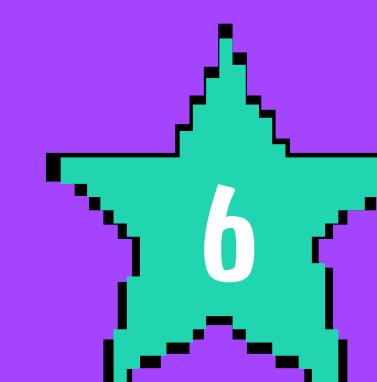
Git X Github



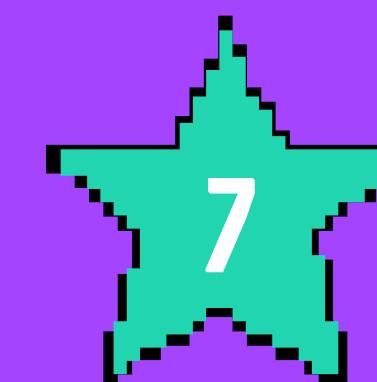
Estrutura do GIT



Principais  
Comandos



Pull Requests



Padrão de commits e  
branches

## PERGUNTA

Você está desenvolvendo um projeto e em um determinado momento você percebe que o jeito que estava fazendo anteriormente era melhor

O que você pode fazer levando em consideração que jeito anterior teve muitas alterações?



## RESPOSTA

Você pode usar o `git`, em que através de simples comandos você pode voltar para qualquer versão anterior sem nenhum problema e de forma rápida.

# VERSIONAMENTO DE CÓDIGO

- Criar versões do seu código;
- Manter registro das versões criadas



v1



v2

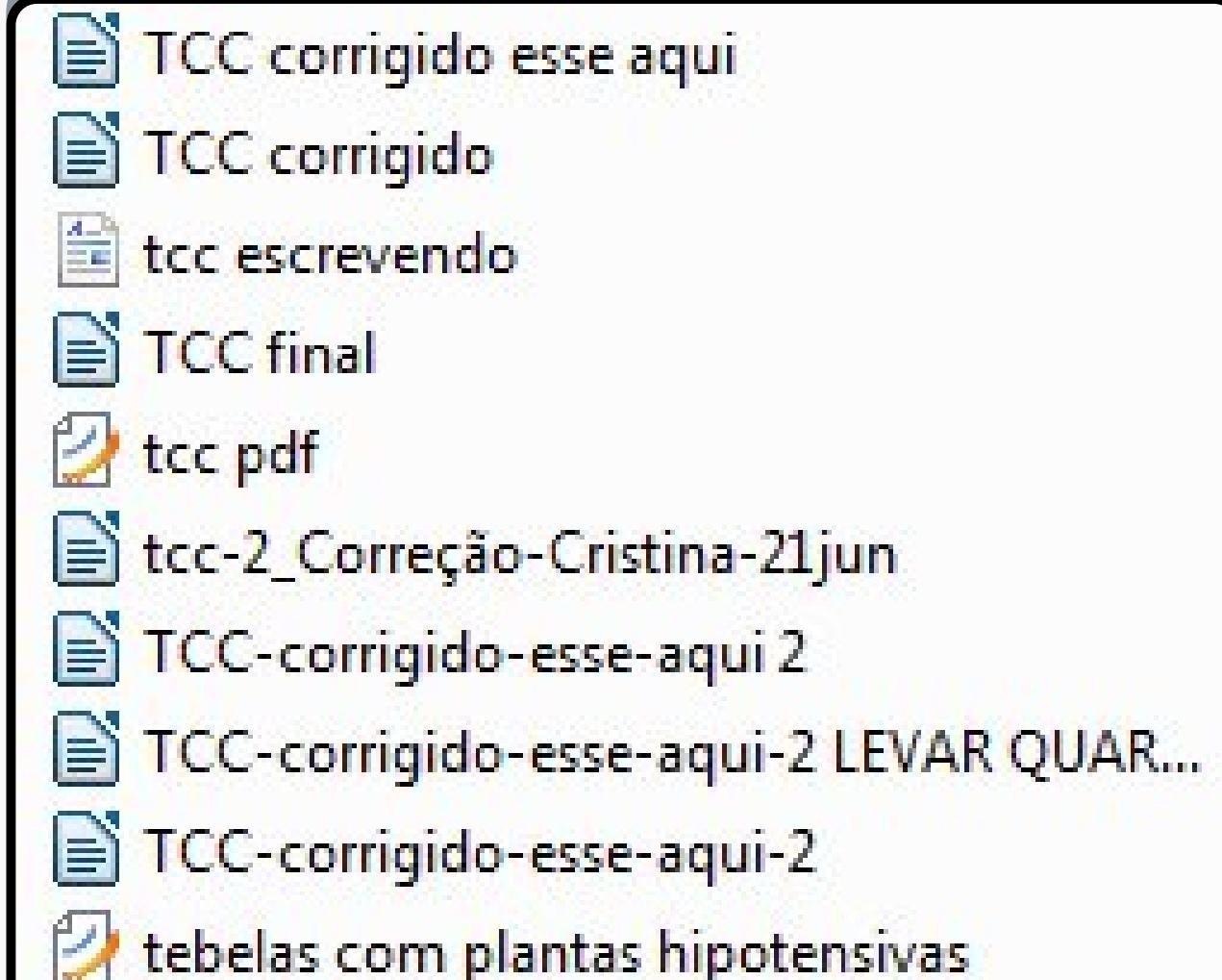


v3

Mas, aos poucos isso vira um problema...

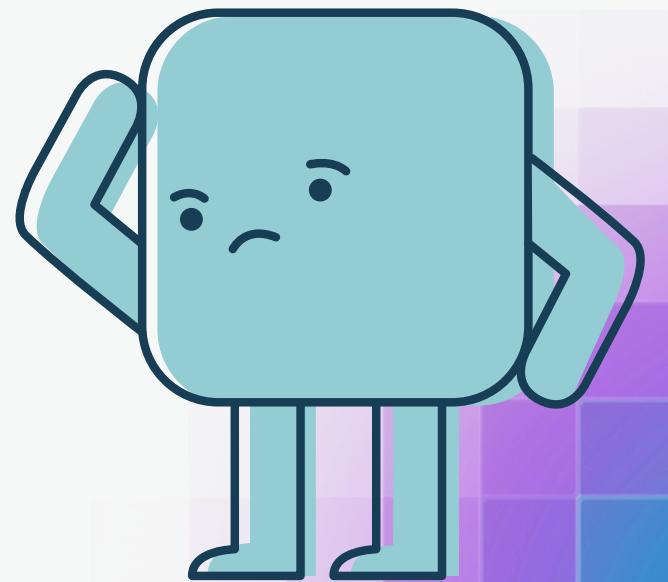
# VERSÃO NAMENTO DE CÓDIGO

- Como gerenciar todos esses códigos?
- Como observar a linearidade das versões?



# VERSIONAMENTO DE CÓDIGO

- Como trabalhar **colaborativamente** em um projeto grande onde várias pessoas podem mexer constantemente e ao mesmo tempo nos mesmos arquivos?



# HISTÓRICO

1973

SCSS

1982

RCS

1986

CVS

2000

SUBVERSION

2005

BITKEEPER

2005

GIT



## SURGIMENTO DO GIT

- Linus Torvalds
- A Kernel Linux era mantida usando “patches” e “tarballs”
- A ferramenta de versionamento de código que utilizavam na época, o BitKeeper, não atendia mais às necessidades do projeto

# SURGIMENTO DO GIT

- Linus então projetou o **Git** com três objetivos principais em mente:
  - **Velocidade**
  - **Modelo Distribuído**
  - **Integridade de Dados**

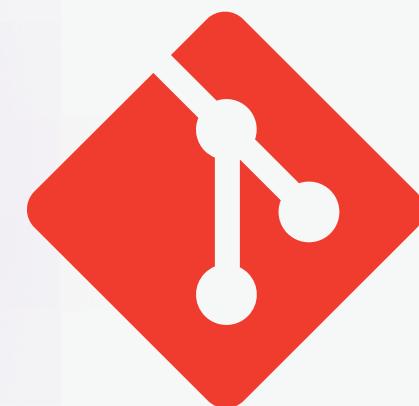


## MAS, AFINAL, O QUE É O GIT?

- É uma ferramenta que gerencia e armazena o histórico de modificações de um projeto;
- A cada alteração significativa, o Git salva um "snapshot" (um registro instantâneo) do estado de todos os arquivos, criando um histórico detalhado e permitindo a navegação entre as diferentes versões.

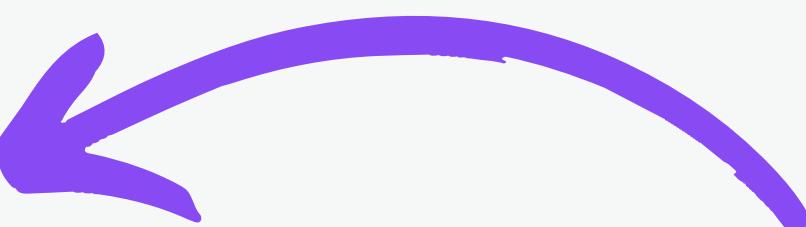


## GIT X GITHUB



git

Usa



GITHUB

Gerenciamento local de código

Colaboração e compartilhamento de  
código remoto

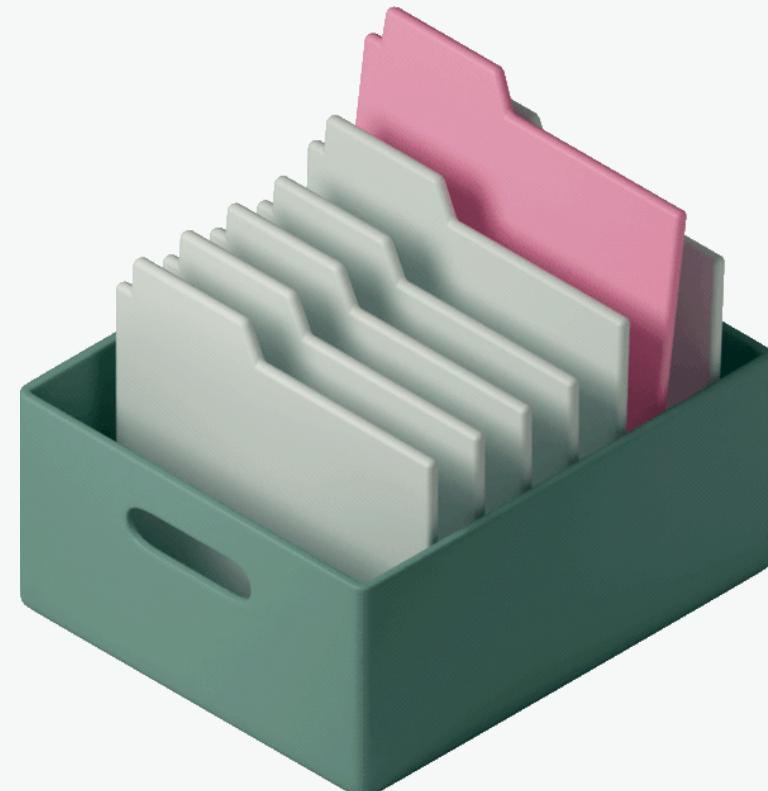
# GIT X GITHUB

- Não existe só o `github`;
- Existe também o `gitlab`, `bitbucket`, `Gitea`, `Gogs` e várias outras.



# ESTRUTURA do GIT

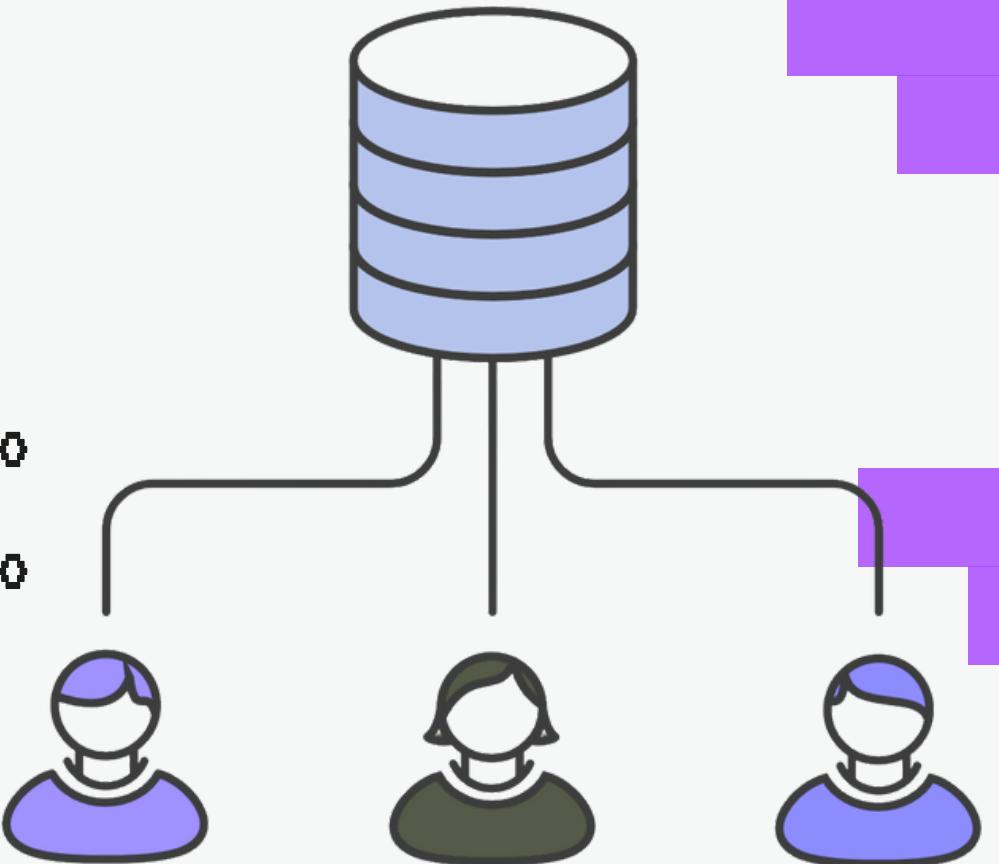
- Para rastrear as alterações de forma eficiente, o Git divide seu projeto em três "árvores" ou ambientes principais que existem localmente na sua máquina:
  - Working Directory (Área de Trabalho)
  - Staging Area (Área de Preparação ou "Index")
  - Local Repository (Repositório Local - a pasta .git)



# ESTRUTURA

## do GIT

- Directory → Staging Area → Local Repository
- Comandos que realizam essa transição:
  - `git add`
  - `git commit`
- Depois que as alterações estão seguras no seu Repositório Local, você pode então sincronizá-las com um Repositório Remoto usando os comandos:
  - `git push` (para enviar)
  - `git pull` (para receber)



## PRINCIPAIS COMANDOS

- Agora que entendemos a estrutura do Git, vamos conhecer os comandos que nos permitem interagir com ele

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

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



# PRINCIPAIS COMANDOS



## INICIAR O PROJETO

`git init`: Transforma uma pasta existente em um repositório

`git clone [URL]`: Cria uma cópia local de um repositório remoto que já existe



# PRINCIPAIS COMANDOS



## CICLO BÁSICO DE TRABALHO

`git status`: Este comando mostra o estado atual do seu Working Directory e da Staging Area

`git add [arquivo]`: Move as alterações de um arquivo específico do Working Directory para a Staging Area, indicando que elas serão incluídas no próximo salvamento.

`git commit -m "Sua mensagem de commit"`: salva definitivamente todas as alterações que estão na área de preparação, criando um ponto no histórico do projeto



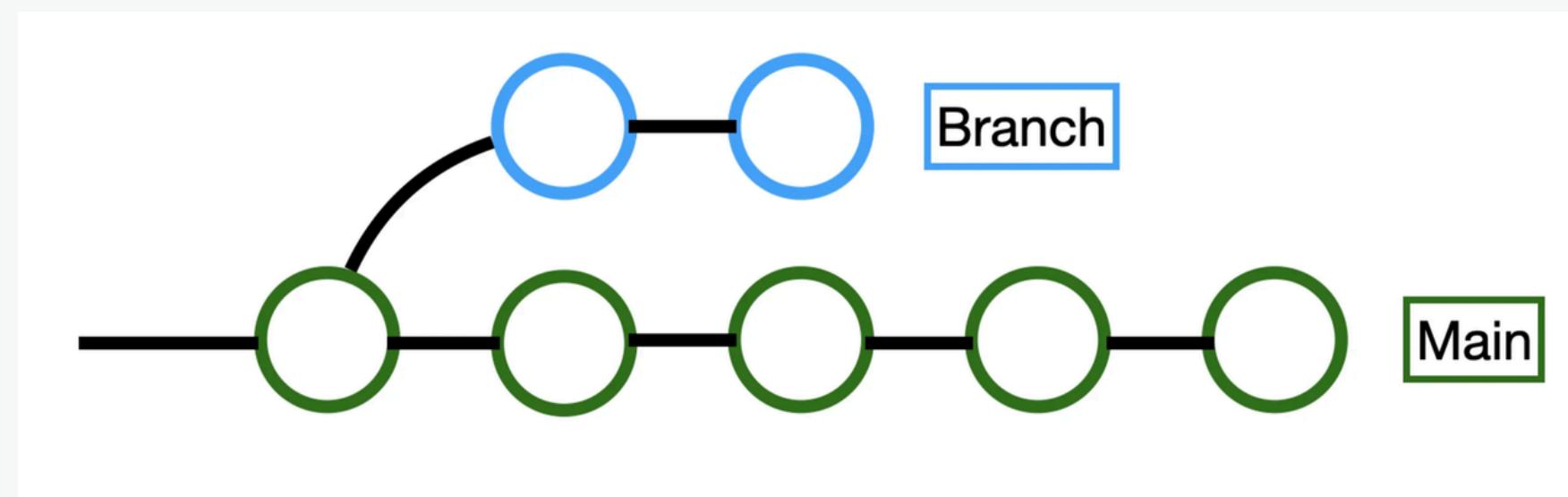
## RAMIFICAÇÕES

No universo do controle de versão com Git, a ramificação (ou branching) é um mecanismo poderoso que permite divergir da linha principal de desenvolvimento e continuar a trabalhar de forma isolada, sem afetar o código base principal. Pense em uma ramificação como a criação de uma linha do tempo paralela do seu projeto.



# RAMIFICAÇÕES

A ramificação principal em um repositório Git é comumente chamada de **main** (anteriormente **master**). Quando você cria uma nova ramificação, o Git cria um novo ponteiro para o mesmo commit em que a **main** está, permitindo que você desenvolva um novo recurso, corrija um bug ou experimente novas ideias em um ambiente contido.



# RAMIFICAÇÕES



## principais comandos

`git branch`: Lista, cria ou deleta branches. `git branch novo-nome` cria uma nova branch

`git checkout [branch]` ou `git switch [branch]`: Muda para a branch especificada

`git merge [branch]`: Une o histórico da branch especificada à sua branch atual, combinando as alterações.

# REPOSITÓRIOS REMOTOS

Para interagir com os repositórios remotos, usamos alguns comandos essenciais do Git

**git remote add <nome> <URL>**: Conecta seu repositório local a um remoto. O nome padrão é origin

**git fetch**: Baixa as novidades do remoto, mas não as integra ao seu trabalho local

**git pull**: Baixa as novidades do remoto e tenta mesclá-las (merge) automaticamente

**git push**: Envia seus commits locais para o repositório remoto

## PULL REQUESTS

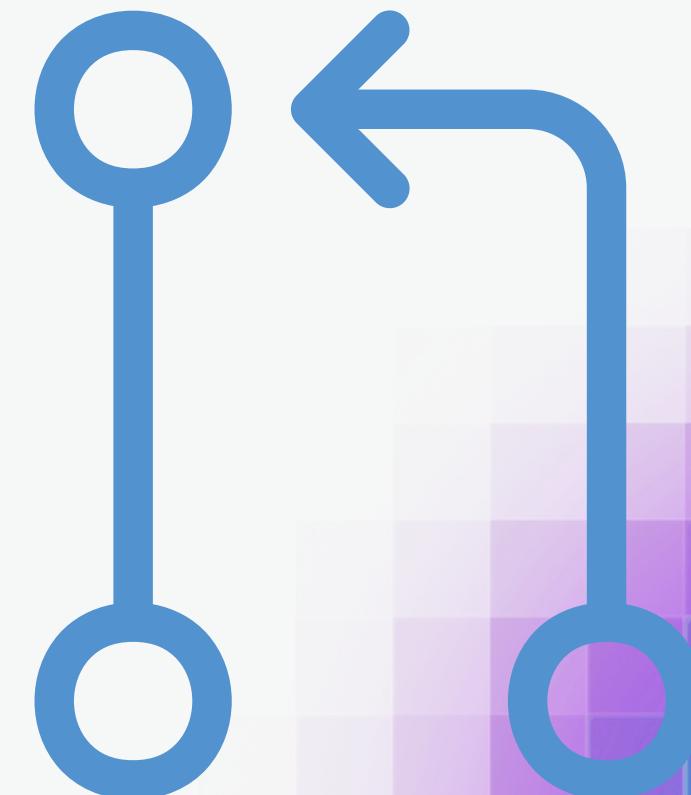
Pull Request não é um comando do Git, mas uma funcionalidade de plataformas como GitHub e GitLab. O Pull Request é um pedido formal para que suas alterações sejam revisadas e integradas à branch principal do projeto.



# PULL REQUESTS

Como funciona:

1. Você trabalha em uma branch **separada**
2. Faz commits das suas alterações
3. Envia a branch para o repositório remoto com **git push**
4. Abre um **Pull Request** na plataforma (GitHub, por exemplo)
5. Outros desenvolvedores revisam seu código
6. Após aprovação, as alterações são **integradas (merged)** à branch principal



# PULL REQUESTS

## Vantagens:

- Permite revisão de código
- Facilita discussões sobre as alterações
- Mantém a qualidade do código
- Documenta decisões técnicas





YAMOS PRATICAR?

YES!

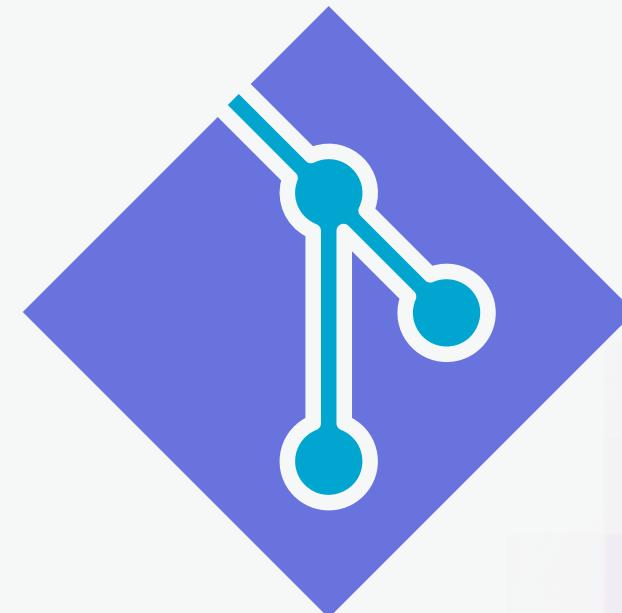


Acesse o link e siga as intruções do Readme para fazer a atividade:

<https://github.com/evellyn489/minicurso-git-aracomp-eniac>

# BRANCHES E COMMITS PADRÕES

- Usar um padrão para nomear branches é fundamental em projetos com mais de uma pessoa. Ajuda a entender rapidamente o que está sendo desenvolvido em cada ramificação e a automatizar processos.



# BRANCHES E COMMITS PADRÕES



## branches - git flow simplificado

- Branchs principais:
  - main (ou master)
  - develop

# BRANCHES E COMMITS PADRÕES



## branches - git flow simplificado

- Branchs de suporte:
  - feature
  - bugfix
  - hotfix

# BRANCHES E COMMITS PADRÕES

## commits

- <tipo>(<escopo>): <descrição>
- [corpo opcional]
- [rodapé opcional]

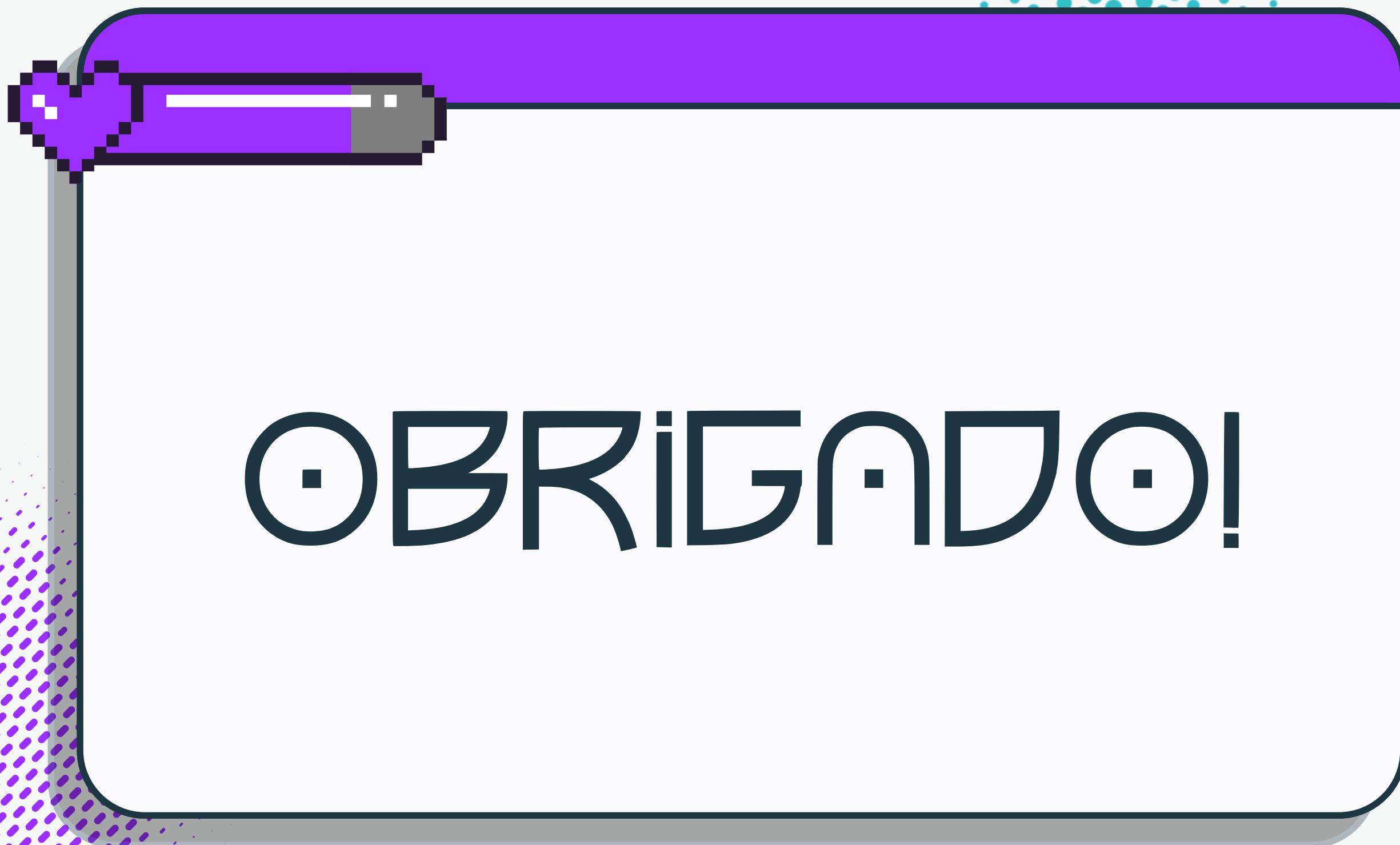
# BRANCHES E COMMITS

## PADRÕES

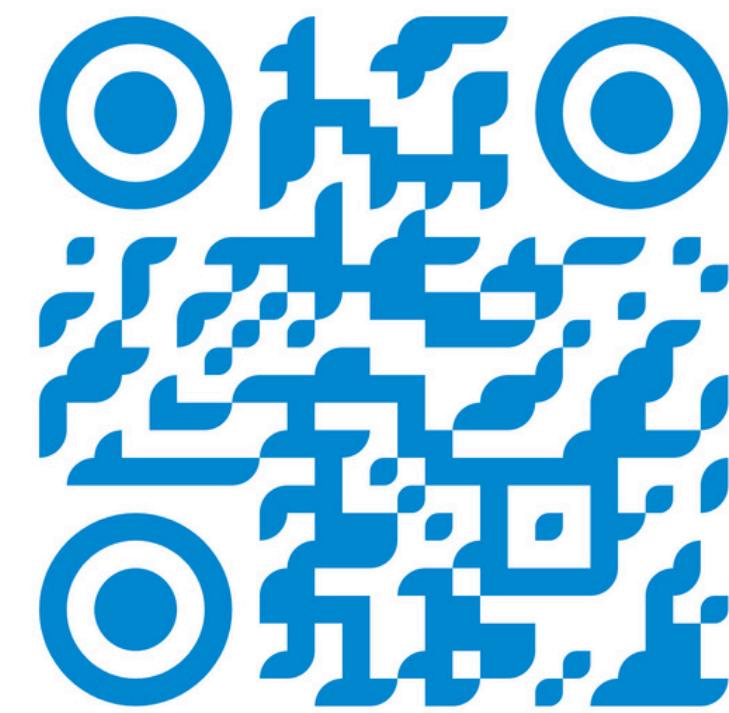


### TIPOS DE COMMITS

- **feat**: Nova funcionalidade visual que adiciona algo novo
- **fix**: Correção de bug
- **docs**: Mudanças na documentação
- **style**: Mudanças de formatação de código (indentação, espaços, ponto e vírgula) ou de aparência visual (CSS, cores, espaçamentos)
- **refactor**: Reestruturação de código que muda a lógica interna
- **test**: Adição ou correção de testes
- **chore**: Tarefas de manutenção, configurações, dependências



## FORMS - FEEDBACK



<https://gqr.sh/taDM>