



Git e Github: Do Básico ao Pull Request

Agenda.

Introdução ao Git

- Controle de versões
- O que é o Git
- Configuração e comandos
- Trabalhando com repositórios locais

GitHub - Colaborando com o Mundo

- Git x GitHub
- Repositórios remotos
- Trabalhando com projetos existentes
- Fluxo de colaboração
- Colaborando no Mural





Controle de Versões.

Controle de Versões.

O que é.

É um sistema que registra as mudanças feitas em arquivos ao longo do tempo, permitindo o acompanhamento e gerenciamento de diferentes versões.

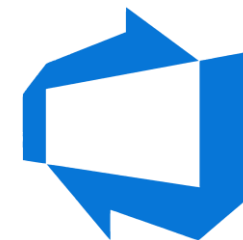
Como funciona.

Cada alteração no código é registrada como um 'snapshot' no tempo. Cada 'snapshot' é um commit, um ponto seguro ao qual você pode retornar.

Ex: Imagine que você está escrevendo um livro. Um sistema de versionamento permite que você salve várias versões: 'Capítulo 1 completo', 'Adicionado novo personagem', 'Revisado parágrafo 3'. Se você não gostar de uma mudança, pode voltar à versão anterior facilmente.



Rational ClearCase

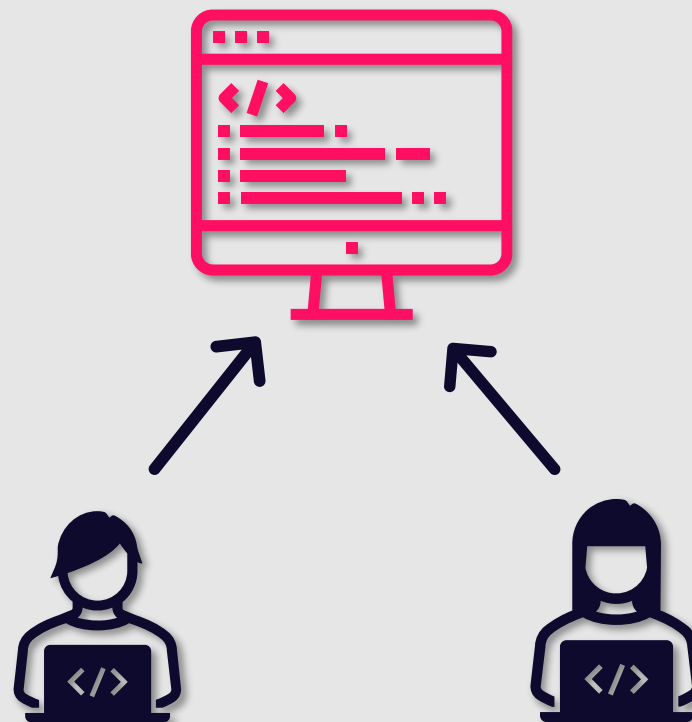


Azure DevOps



Porquê usar Controle de Versão.

- Rastreamento de Alterações
- Colaboração Eficiente
- Documentação de Código
- Backup de Código
- Desenvolvimento Paralelo
- Gestão de Versões
- Análise de Erros e Depuração
- Facilidade de Distribuição



Entendendo alguns termos.

Repositório (Repo): É o diretório onde o versionamento acontece.

Staging Area: Área temporária onde você prepara as alterações antes de um commit.

Commit: Uma "foto" salva das suas alterações no histórico do repositório.

Branches (Ramificações): Linhas de desenvolvimento independentes.

Issues (Problemas): Usadas para rastrear tarefas, aprimoramentos ou bugs.



Git.

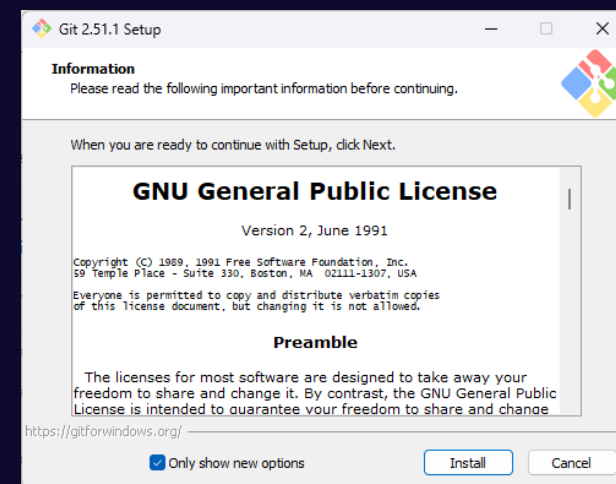
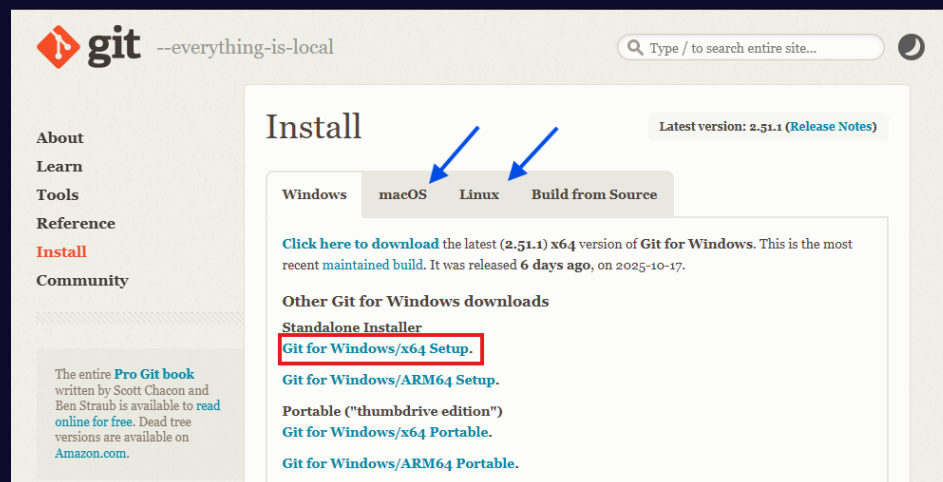
O que é.

O Git é uma das ferramentas mais importantes no mundo do desenvolvimento. É um sistema de controle de versões distribuído que revolucionou a forma como os desenvolvedores colaboram, acompanham e gerenciam o código-fonte de seus projetos.



Instalação.

[Download do Git](#)



Principais comandos.



Definições e configurações

`git config`

Iniciar um repositório

`git init`

Adicionar arquivos à Staging Area

`git add`

Verificar estado do repositório

`git status`

Salvar alterações no histórico

`git commit`

Guardar alterações temporariamente

`git stash`

Visualizar o histórico de alterações

`git log`

Ramificações

`git branch`

Mudar de ramificação

`git checkout`

Unir históricos de ramificações

`git merge`

Configuração.



Após a instalação, devemos realizar algumas configurações iniciais.

Para garantir controle, todo commit é necessariamente identificado com seu autor. Definimos nossas informações de autoria através do comando

```
git config
```

```
# Define o nome que aparecerá nos commits
```

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

```
# Define o e-mail que aparecerá nos commits
```

```
git config --global user.email "seu-email@example.com"
```

Criando o repositório.



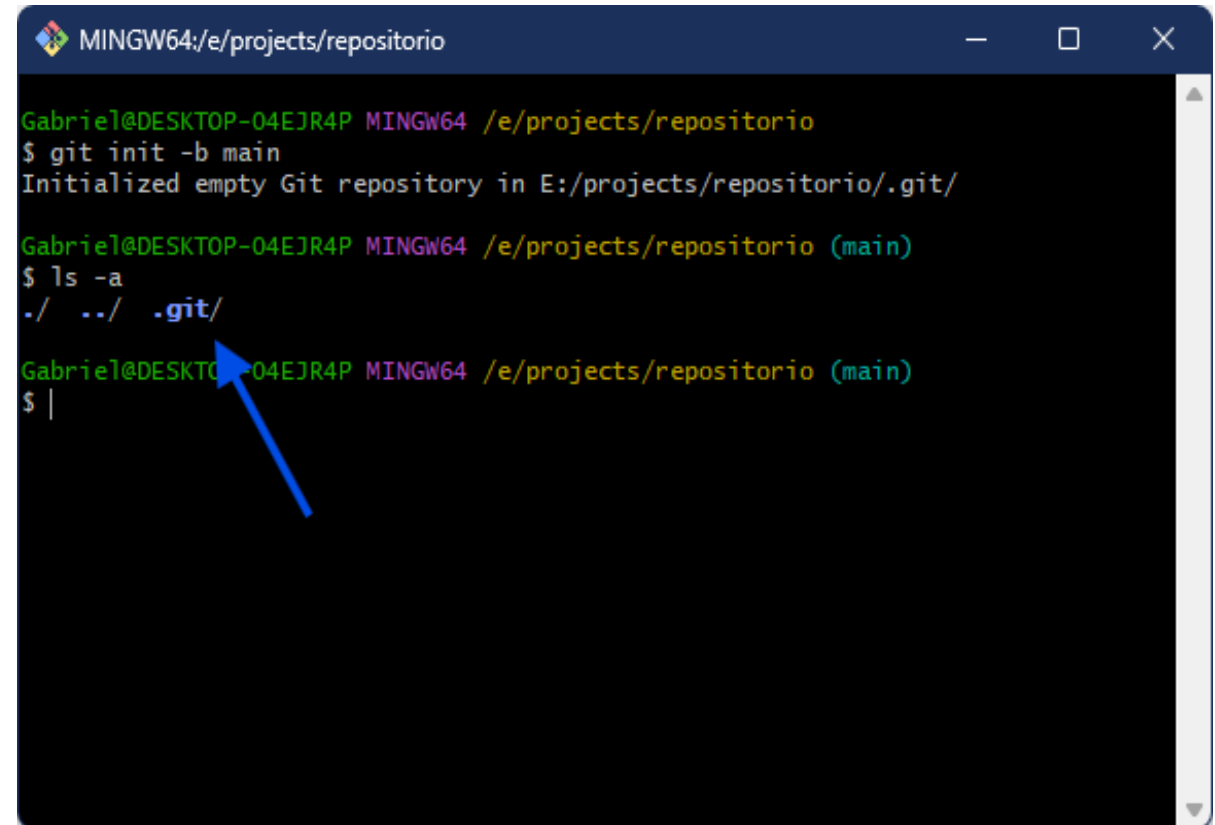
Iniciamos um repositório Git no diretório do projeto onde faremos o controle de versão

Inicia um repositório no diretório em que você está

```
git init
```

Dica de 2025: Inicia o repositório com o nome de branch "main"

```
git init -b main
```

A screenshot of a Windows terminal window titled 'MINGW64:/e/projects/repositorio'. The terminal shows the following commands and output: 1. 'git init -b main' is executed, resulting in 'Initialized empty Git repository in E:/projects/repositorio/.git/'. 2. 'ls -a' is executed, resulting in ' ./ ../ .git/'. A blue arrow points to the '.git/' directory listing. 3. The prompt returns to '\$ |' after the second command.

```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio
$ git init -b main
Initialized empty Git repository in E:/projects/repositorio/.git/

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ ls -a
./ ../ .git/

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ |
```

Preparar alterações.

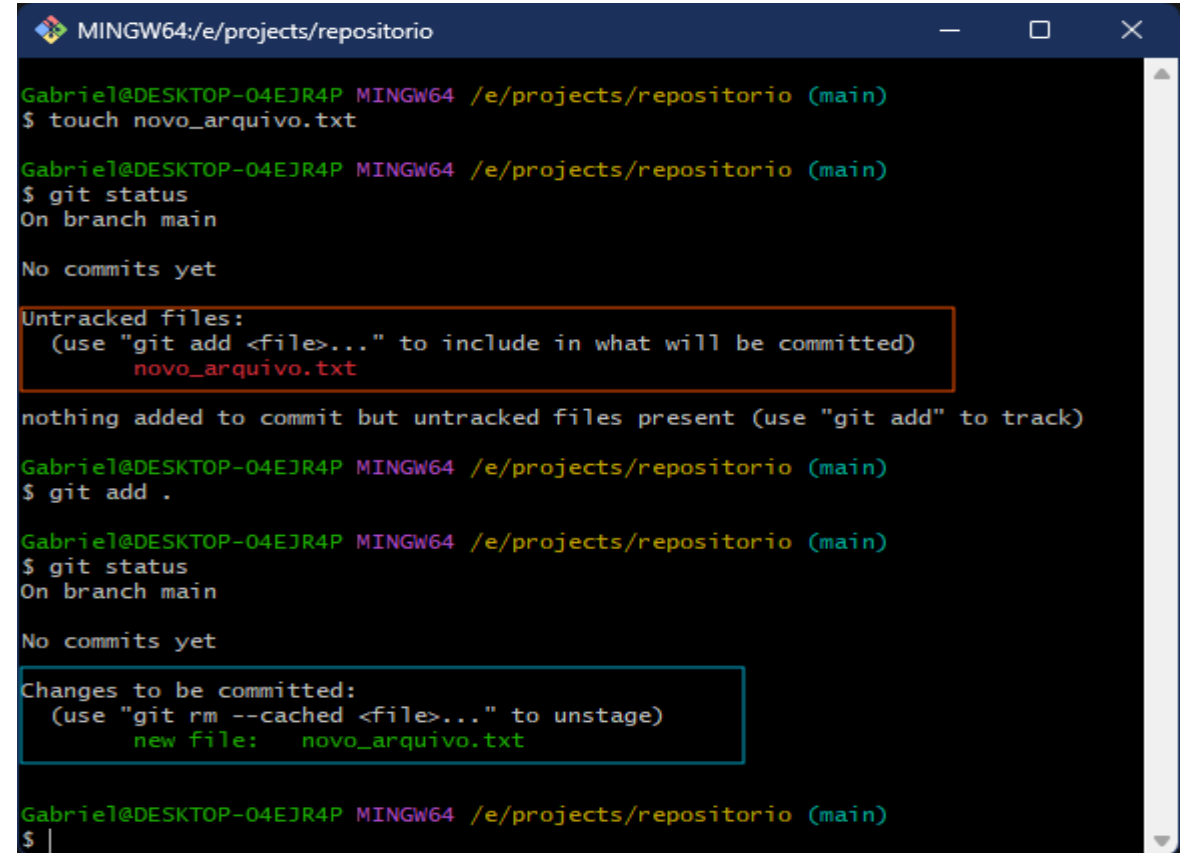


Com o repositório iniciado, todas alterações feitas passam a ser monitoradas pelo Git

Com o comando `git add` adicionamos nossas alterações à "Staging Area"

```
# Adiciona um arquivo específico
git add nome-do-arquivo.txt

# Adiciona todos os arquivos modificados
git add .
```

A screenshot of a Windows terminal window titled 'MINGW64:/e/projects/repositorio'. The terminal shows the following sequence of commands and outputs: 1. Command: 'touch novo_arquivo.txt'. Output: 'Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main) \$ touch novo_arquivo.txt'. 2. Command: 'git status'. Output: 'Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main) \$ git status', 'On branch main', 'No commits yet', and a box highlighting 'Untracked files: (use "git add <file>..." to include in what will be committed) novo_arquivo.txt'. 3. Command: 'git add .'. Output: 'nothing added to commit but untracked files present (use "git add" to track)'. 4. Command: 'git status'. Output: 'Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main) \$ git status', 'On branch main', 'No commits yet', and a box highlighting 'Changes to be committed: (use "git rm --cached <file>..." to unstage) new file: novo_arquivo.txt'. 5. The prompt '\$ |' is visible at the bottom.

```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ touch novo_arquivo.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git add .

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
No commits yet

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

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ |
```

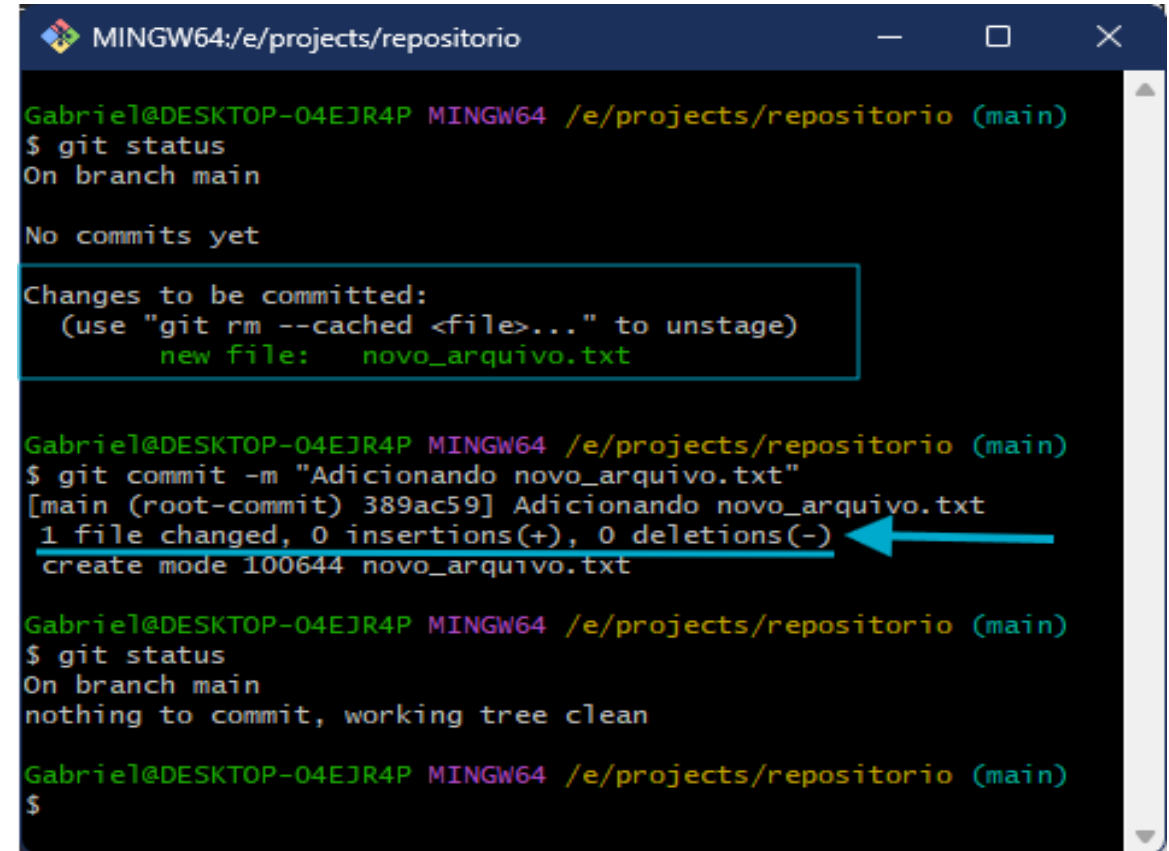
Salvar alterações.



O `git commit` pega tudo que está na "Staging Area" e cria um ponto permanente no histórico do projeto.

Cada *commit* deve ter uma mensagem que descreve *o que* foi mudado. Isso é fundamental para entender o histórico do projeto no futuro.

```
# Salva as alterações da Staging Area no
histórico
# O -m significa "message"
git commit -m "Uma mensagem que
descreve o que foi feito"
```

A screenshot of a Windows terminal window titled 'MINGW64:/e/projects/repositorio'. It shows the execution of 'git status' and 'git commit' commands. The 'git status' output shows 'On branch main' and 'No commits yet', with a box highlighting the 'Changes to be committed' section: 'new file: novo_arquivo.txt'. The 'git commit' command is run with the message 'Adicionando novo_arquivo.txt', and the output shows the commit hash '389ac59' and a summary of changes: '1 file changed, 0 insertions(+), 0 deletions(-)'. A blue arrow points to this summary line. The final 'git status' output shows 'nothing to commit, working tree clean'.

```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main

No commits yet

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

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git commit -m "Adicionando novo_arquivo.txt"
[main (root-commit) 389ac59] Adicionando novo_arquivo.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 novo_arquivo.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
nothing to commit, working tree clean

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$
```

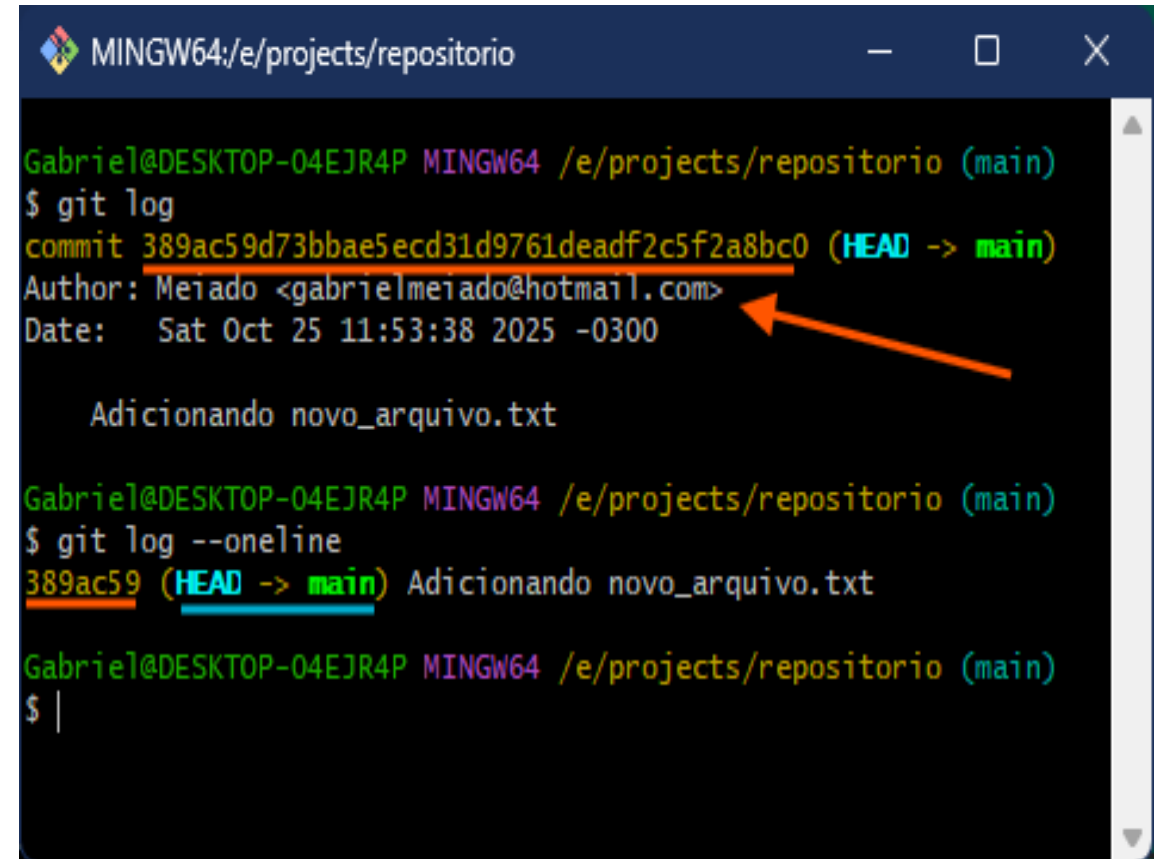
Visualizando Histórico.



O `git log` é usado para listar o histórico de commits da ramificação em que estamos, permitindo a visualização de toda evolução do projeto, autores das alterações e quando aconteceram.

```
# Exibe o histórico completo  
git log
```

```
# Exibe um log simplificado  
git log --oneline
```

A screenshot of a Windows terminal window titled 'MINGW64:/e/projects/repositorio'. The terminal shows the output of 'git log' and 'git log --oneline'. The first command shows a commit with a long hash, author 'Meiado', and date 'Sat Oct 25 11:53:38 2025 -0300'. The second command shows the same commit in a simplified format. An orange arrow points to the commit hash in the first output.

```
MINGW64:/e/projects/repositorio  
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)  
$ git log  
commit 389ac59d73bbae5ecd31d9761deadf2c5f2a8bc0 (HEAD -> main)  
Author: Meiado <gabrielmeiado@hotmail.com>  
Date: Sat Oct 25 11:53:38 2025 -0300  
  
    Adicionando novo_arquivo.txt  
  
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)  
$ git log --oneline  
389ac59 (HEAD -> main) Adicionando novo_arquivo.txt  
  
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)  
$ |
```

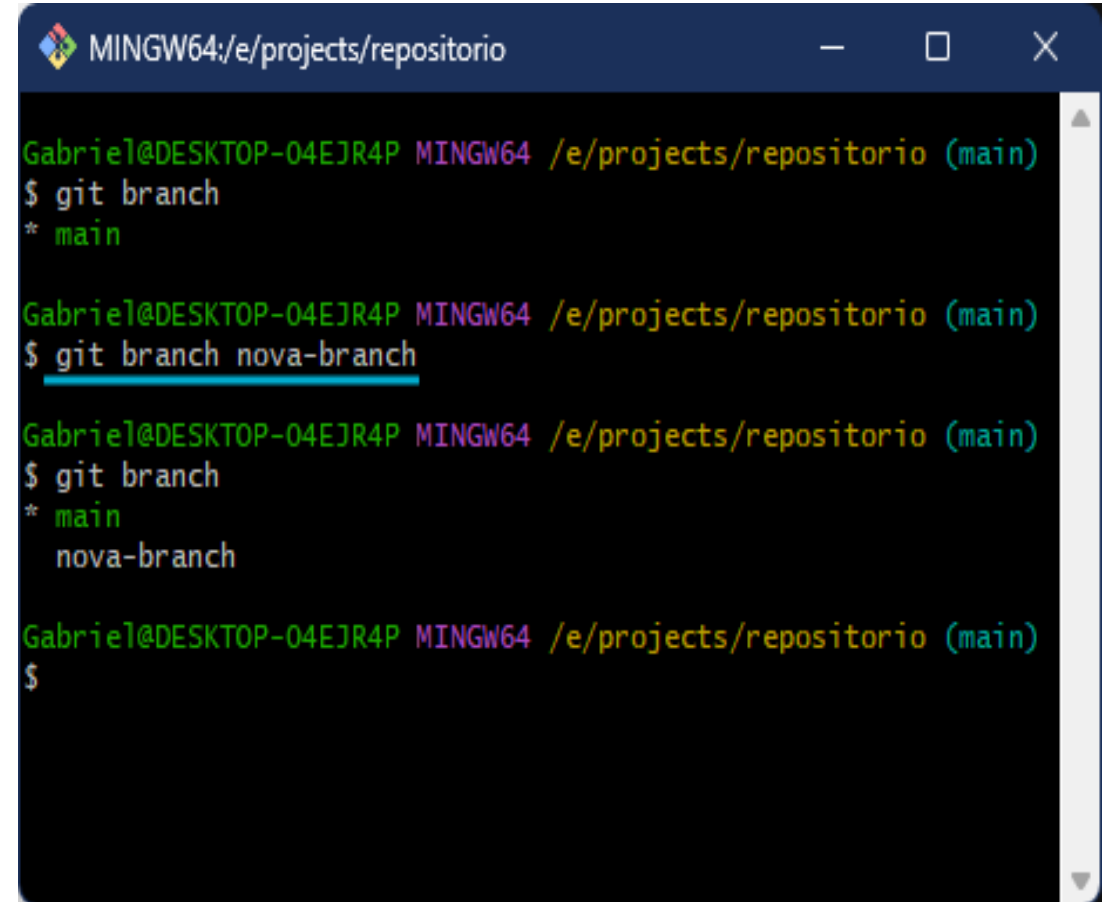
Ramificações.



Ramificações são "linhas do tempo" independentes no seu projeto. Elas permitem que você trabalhe em diferentes funcionalidades ou correções ao mesmo tempo, sem que uma afete a outra.

```
# Lista todas as ramificações locais
# A ramificação marcada com * é a que você
está
git branch

# Cria uma nova ramificação com o nome que
você informar
git branch nova-branch
```

A screenshot of a Windows terminal window titled 'MINGW64:/e/projects/repositorio'. The terminal shows the execution of 'git branch' commands. The first command shows '* main'. The second command creates a new branch 'nova-branch'. The third command shows both '* main' and 'nova-branch'. The terminal text is as follows:

```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git branch
* main

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git branch nova-branch

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git branch
* main
  nova-branch

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$
```

Ramificações.



Agora que temos nossa nova ramificação, precisamos de um jeito de "entrar" nela para começar a trabalhar. O comando `git checkout` é usado para navegar entre as ramificações (branches) e também entre commits antigos.

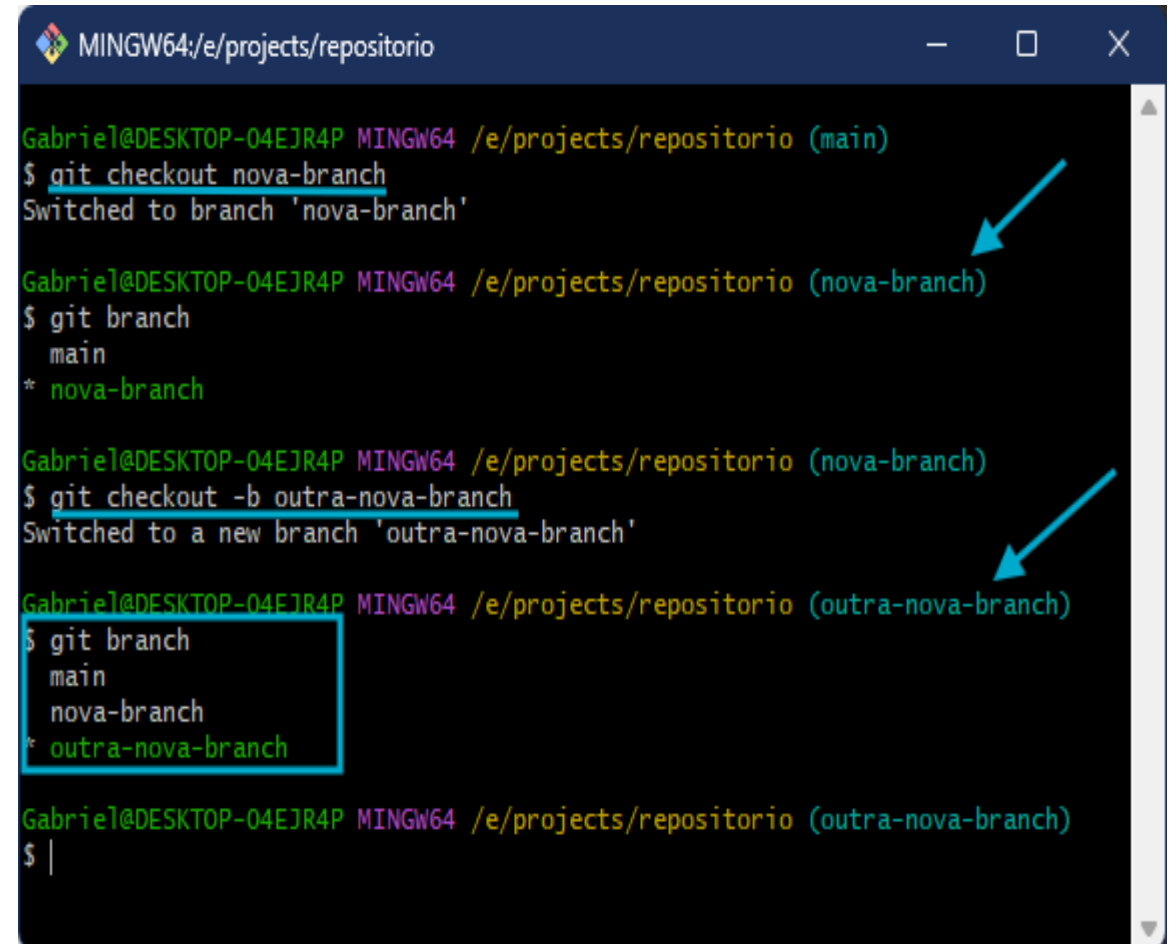
```
# Muda da ramificação atual para a "nova-branch"
```

```
git checkout nova-branch
```

```
# Atalho para CRIAR e MUDAR ao mesmo tempo
```

```
# (Isso faz o 'branch' e o 'checkout' em um passo só)
```

```
git checkout -b outra-nova-branch
```

A screenshot of a Windows terminal window titled 'MINGW64:/e/projects/repositorio'. The terminal shows a series of Git commands and their outputs. The user starts on the 'main' branch and runs 'git checkout nova-branch', which switches them to 'nova-branch'. Then, they run 'git branch' and see 'main' and '* nova-branch'. Next, they run 'git checkout -b outra-nova-branch', which switches them to 'outra-nova-branch'. Finally, they run 'git branch' again, showing 'main', 'nova-branch', and '* outra-nova-branch'. Two blue arrows point to the 'nova-branch' and 'outra-nova-branch' entries in the branch lists. A red box highlights the output of the second 'git branch' command. The prompt character is '\$'.

Unir Ramificações.



Depois de terminar o trabalho na nova branch, você vai querer trazer as alterações de volta para a branch de onde saiu a ramificação

O comando `git merge` faz a fusão dos históricos de uma branch na outra

```
# Voltar para a branch original
```

```
git checkout main
```

```
# Trazer as alterações da "nova-feature" para a "main"
```

```
git merge nova-branch
```

Unir Ramificações.



```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-branch)
$ touch arquivo_nova_branch.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-branch)
$ ls
arquivo_nova_branch.txt  novo_arquivo.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-branch)
$ git add .

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-branch)
$ git commit -m "Adicionando arquivo à nova branch"
[nova-branch d358605] Adicionando arquivo à nova branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 arquivo_nova_branch.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-branch)
$ |
```

Unir Ramificações.



```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-branch)
$ git checkout main
Switched to branch 'main'

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git log
commit 389ac59d73bbae5ecd31d9761deadf2c5f2a8bc0 (HEAD -> main)
Author: Meiado <gabrielmeiado@hotmail.com>
Date: Sat Oct 25 11:53:38 2025 -0300

    Adicionando novo_arquivo.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ |
```



```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git merge nova-branch
Updating 389ac59..d358605
Fast-forward
 arquivo_nova_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 arquivo_nova_branch.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git log
commit d358605626cf3aade3c611e06436939acd8f8d34 (HEAD -> main, nova-branch)
Author: Meiado <gabrielmeiado@hotmail.com>
Date: Sat Oct 25 13:06:36 2025 -0300

    Adicionando arquivo à nova branch

commit 389ac59d73bbae5ecd31d9761deadf2c5f2a8bc0
Author: Meiado <gabrielmeiado@hotmail.com>
Date: Sat Oct 25 11:53:38 2025 -0300

    Adicionando novo_arquivo.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ |
```

Guardar alterações temporariamente.



O `git stash` é o comando que você usa quando está no meio de uma alteração, mas ainda não está pronto para *commitar*. De repente, você precisa trocar de branch, mas o Git não deixa, pois você tem trabalho não salvo.

O *stash* guarda essas alterações de lado, limpando a branch e permite a troca

```
# Salva as alterações atuais com uma mensagem
git stash push -m "Minha alteração
incompleta"

# Lista todos os "stashes" guardados
git stash list

# Aplica o último stash guardado E o remove da
lista
git stash pop
```

Guardar alterações temporariamente.



```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ ls
main.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git branch nova-feature

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ echo "Atualizando main..." >> main.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
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:   main.txt

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

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git add .
warning: in the working copy of 'main.txt', LF will be replaced by CRLF the next time Git touches it

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git commit -m "Atualizando main"
[main 6f55caa] Atualizando main
1 file changed, 1 insertion(+), 2 deletions(-)
```

Guardar alterações temporariamente.



```
MINGW64:/e/projects/repositorio
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git checkout nova-feature
Switched to branch 'nova-feature'

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-feature)
$ echo "Modificando a main" >> main.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-feature)
$ git status
On branch nova-feature
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:   main.txt

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

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-feature)
$ git checkout main
error: Your local changes to the following files would be overwritten by checkout:
    main.txt
Please commit your changes or stash them before you switch branches.
Aborting

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-feature)
$ |
```

```
MINGW64:/e/projects/repositorio
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-feature)
$ git stash push -m "Salvando alterações pra depois"
warning: in the working copy of 'main.txt', LF will be replaced by CRLF the next time Git touches it
Saved working directory and index state On nova-feature: Salvando alterações pra depois

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (nova-feature)
$ git checkout main
Switched to branch 'main'

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ cat main.txt
Versão da Main
Atualizando main...

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$
```

Ignorar arquivos e pastas.

O Git possui um mecanismo para ignorar certos arquivos e pastas no controle de versões, o arquivo `.gitignore`

Essa função é fundamental para evitar que arquivos desnecessários e arquivos sensíveis sejam adicionados ao histórico de alterações:

- Pastas de dependências (ex: `node_modules/`)
- Arquivos de segredos (ex: `.env`)
- Arquivos de Sistema (ex: `.DS_Store` no Mac)
- Pastas de Código compilado (ex: `build/`
`dist/` `target/`)



Ignorar arquivos e pastas.



```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ touch .gitignore

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ touch secret.txt

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        secret.txt ←

nothing added to commit but untracked files present (use "git add" to track)

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$
```

```
MINGW64:/e/projects/repositorio

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ echo secret.txt >> .gitignore

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/repositorio (main)
$ |
```




GitHub.

O que é.

O GitHub é uma plataforma ou serviço online. Pense nele como um “serviço de hospedagem na nuvem” feito especificamente para repositórios Git.

Ele fornece um site com uma interface gráfica que facilita a colaboração entre desenvolvedores, permitindo que equipes trabalhem juntas no mesmo código, acompanhem *Issues* (tarefas/bugs) e façam *Pull Requests*.



Git x GitHub.



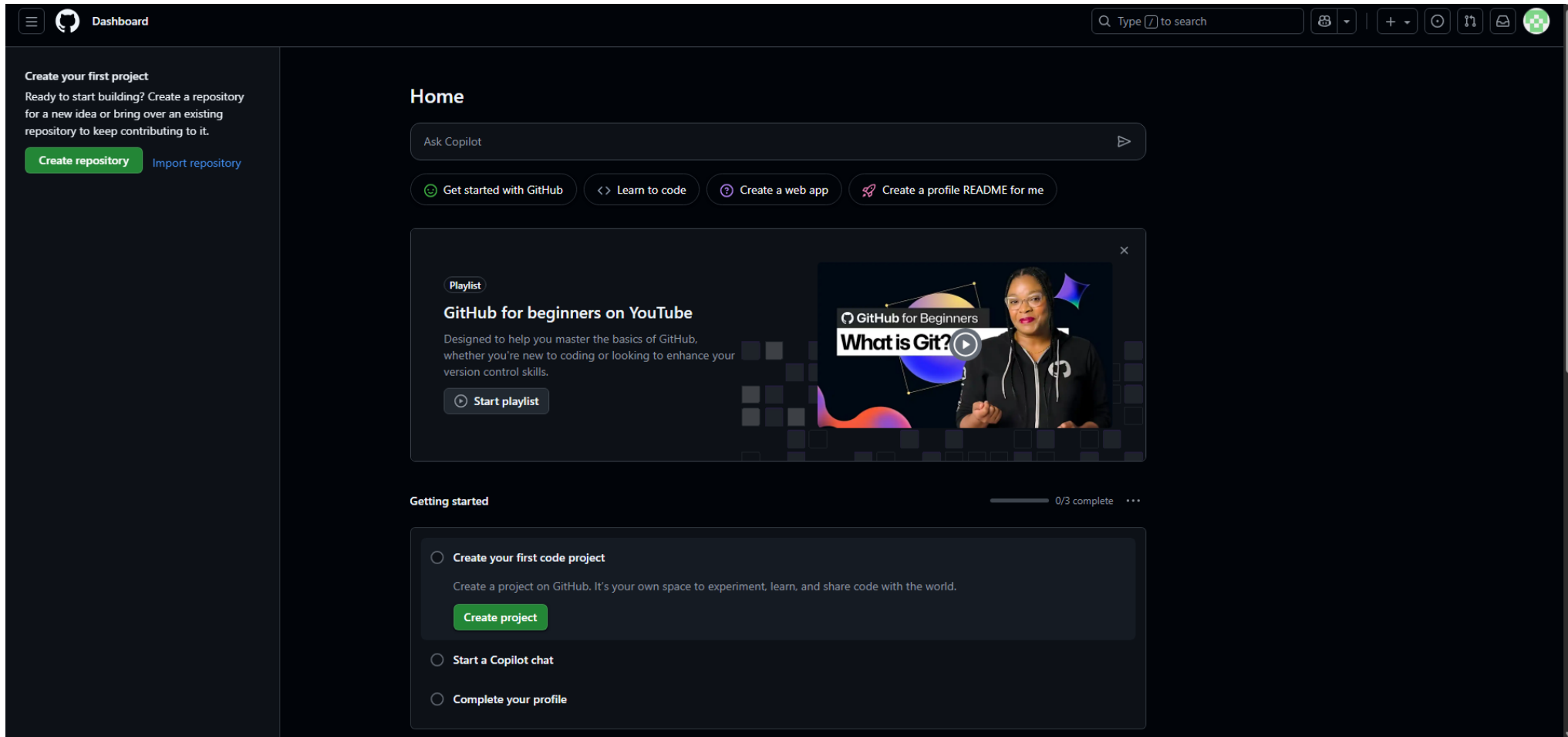
- É a ferramenta
- Software de linha de comando
- Instalado no computador
- Controle de histórico de versão



- É o serviço/plataforma
- Site (interface gráfica)
- Hospedado na nuvem
- Facilita a colaboração e hospedagem

GitHub.

Tour pela plataforma.



Repositórios Remotos.



Para que possamos comunicar nossas mudanças locais ao repositório criado na plataforma é necessário sinalizar essa relação e criar a ponte entre os dois

Usamos o comando *git remote add* para isso, sinalizando ao Git qual a URL do repositório remoto onde as alterações também serão salvas

```
# Adiciona um "apelido" (remote) chamado "origin"
# para a URL do seu repositório no GitHub
git remote add origin https://github.com/seu-usuario/seu-repo.git
```

```
# Para verificar se o "remote" foi adicionado com
sucesso
git remote -v
```

Repositórios Remotos.




Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).
Required fields are marked with an asterisk (*).

1

General


Owner *

 gabrielmeiado20

 /

Repository name *

demo-project

 demo-project is available.

Great repository names are short and memorable. How about **probable-waffle**?

Description


0 / 350 characters

2

Configuration

Choose visibility *

Choose who can see and commit to this repository

 Public

Add README

READMEs can be used as longer descriptions. [About READMEs](#)

Off ☐

Add .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)

No .gitignore

Add license

Licenses explain how others can use your code. [About licenses](#)

No license

Create repository

```
MINGW64:/e/projects/demo-project

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$ git remote -v

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$ git remote add origin https://github.com/gabrielmeiado20/demo-project.git

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$ git remote -v
origin https://github.com/gabrielmeiado20/demo-project.git (fetch)
origin https://github.com/gabrielmeiado20/demo-project.git (push)

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$ |
```

Repositórios Remotos.



Com o repositório conectado e as alterações já no histórico após os commit, o comando *git push* é que finalmente envia as alterações locais para o GitHub

A primeira vez que fazemos usamos a flag *-u* para salvar nossa escolha indicando ao Git que a branch *main* local deve seguir a branch *main* remota em *origin*

```
# Envia os commits da "main" local para a "origin"
# O -u (ou --set-upstream) conecta as duas branches
git push -u origin main

# Depois da primeira vez, podemos usar apenas:
git push
```

Repositórios Remotos.

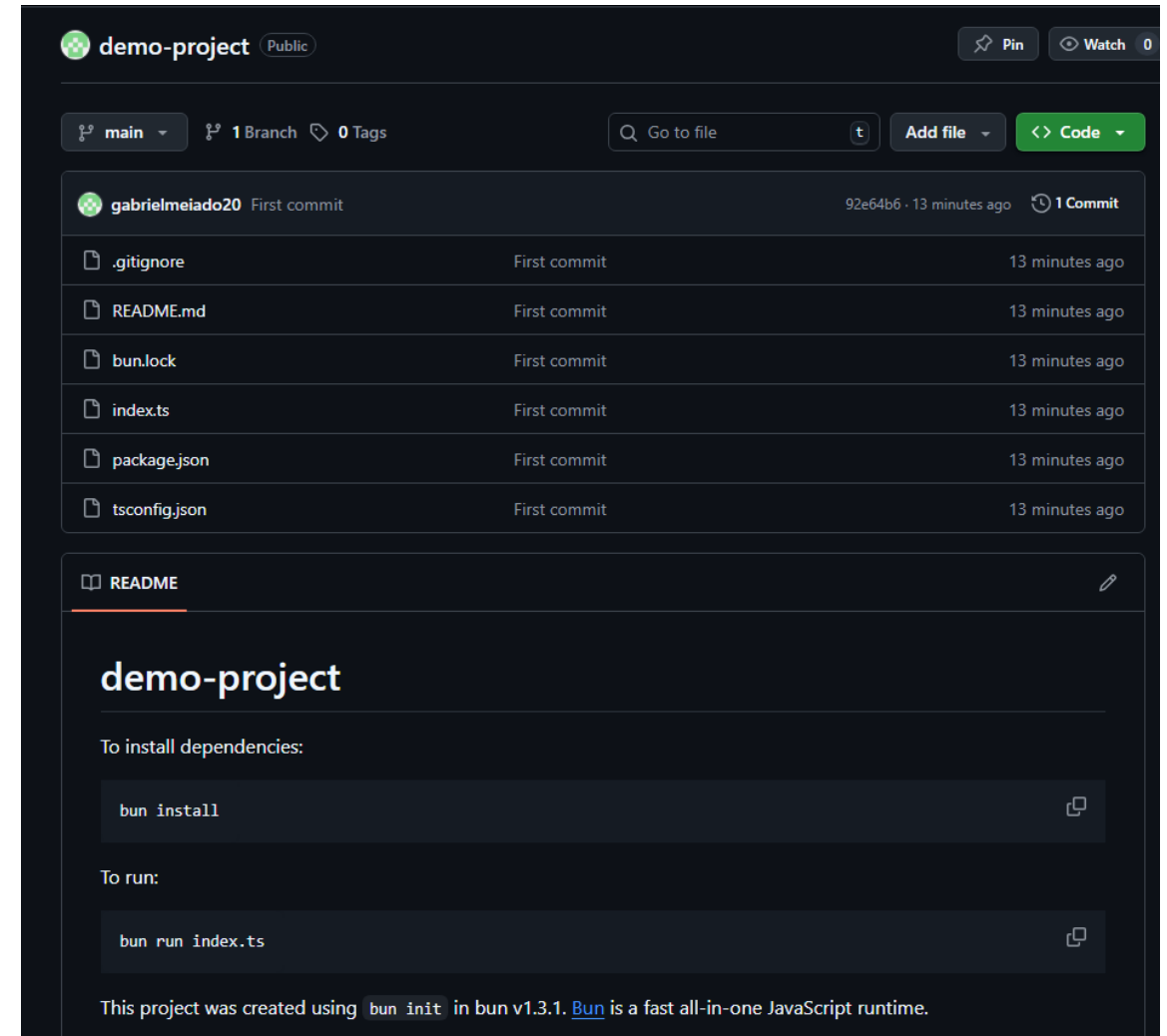


```
MINGW64:/e/projects/demo-project

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$ git log --oneline
92e64b6 (HEAD -> main) First commit

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$ git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 12 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 2.13 KiB | 728.00 KiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/gabrielmeiado20/demo-project.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/demo-project (main)
$
```



The image shows the GitHub web interface for a repository named "demo-project". The repository is public and has one branch, "main", with no tags. It shows a single commit by user "gabrielmeiado20" with the message "First commit". The commit details show a list of files: .gitignore, README.md, bun.lock, index.ts, package.json, and tsconfig.json, all added in the first commit. Below the file list, the README is displayed, showing the project name "demo-project" and instructions on how to install dependencies using "bun install" and how to run the project using "bun run index.ts". The README also mentions that the project was created using "bun init" and that Bun is a fast all-in-one JavaScript runtime.

demo-project Public

main 1 Branch 0 Tags

gabrielmeiado20 First commit 92e64b6 · 13 minutes ago 1 Commit

File	Commit	Time
.gitignore	First commit	13 minutes ago
README.md	First commit	13 minutes ago
bun.lock	First commit	13 minutes ago
index.ts	First commit	13 minutes ago
package.json	First commit	13 minutes ago
tsconfig.json	First commit	13 minutes ago

README

demo-project

To install dependencies:

```
bun install
```

To run:

```
bun run index.ts
```

This project was created using `bun init` in bun v1.3.1. [Bun](#) is a fast all-in-one JavaScript runtime.

Repositórios Remotos.



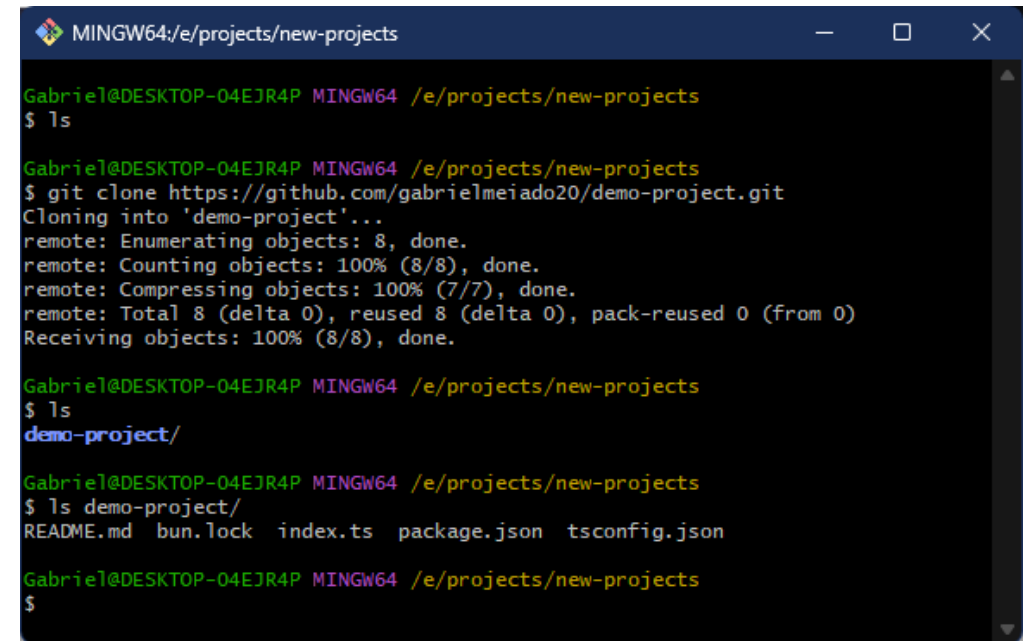
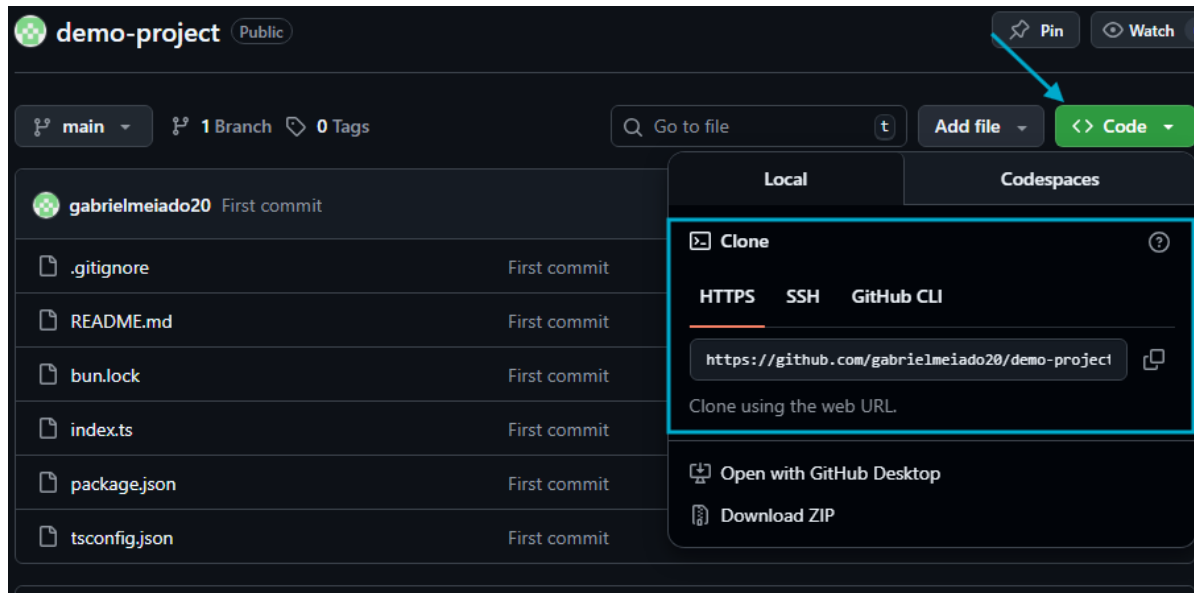
Até agora, nós "subimos" um projeto que criamos localmente. Mas, e se quisermos trabalhar num projeto que já existe?

O comando *git clone* faz exatamente isso. Ele "clona" (cria uma cópia local) de um repositório remoto.

```
# Copia o projeto para uma nova pasta na sua máquina
git clone https://github.com/usuario/repositorio-do-projeto.git
```

```
# Ele faz três coisas num só passo:
# 1. Cria uma nova pasta com o nome do projeto.
# 2. Baixa todo o histórico de commits e ficheiros.
# 3. Configura automaticamente o remote "origin".
```

Repositórios Remotos.



Repositórios Remotos.



O *git clone* é feito apenas *uma vez*, no início.

Depois disso, para trazer novas atualizações do repositório remoto (commits feitos por outros colegas, por exemplo), nós usamos o *git pull*.


Este comando faz duas coisas: ele "busca" (*fetch*) as novidades do GitHub e depois "mescla" (*merge*) essas novidades na sua *branch* local. É a forma de garantir que você está sempre a trabalhar na versão mais recente do código.

```
# Busca e mescla as atualizações da branch  
# "main" do remoto "origin" para a sua Branch  
# local  
git pull origin main
```


Repositórios Remotos.



Commit 573f4ce

 gabrielmeiado20 committed 11 minutes ago

Add new line to index.ts

 main

Filter files...

index.ts

1 file changed +1 -1 lines changed

	@@ -1 +1 @@
1	- console.log("Hello via Bun!");
1	+ console.log("Hello via Bun!");console.log('Nova linha no codigo');

Comments 0

```
MINGW64:/e/projects/new-projects/demo-project

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/new-projects/demo-project (main)
$ cat index.ts
console.log("Hello via Bun!");
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/new-projects/demo-project (main)
$ git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 304 bytes | 30.00 KiB/s, done.
From https://github.com/gabrielmeiado20/demo-project
* branch          main          -> FETCH_HEAD
   92e64b6..573f4ce main        -> origin/main
Updating 92e64b6..573f4ce
Fast-forward
 index.ts | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/new-projects/demo-project (main)
$ cat index.ts
console.log("Hello via Bun!");console.log('Nova linha no codigo');

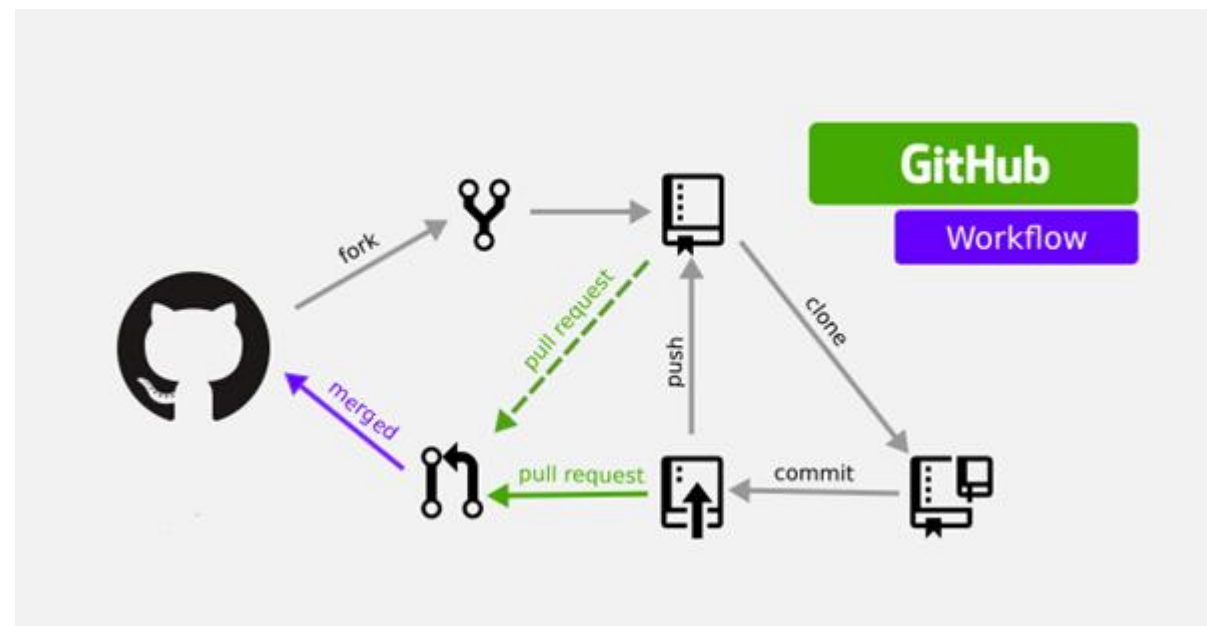
Gabriel@DESKTOP-04EJR4P MINGW64 /e/projects/new-projects/demo-project (main)
$
```

Fluxo de Colaboração.



Quando queremos contribuir para um projeto que não é nosso (um projeto open-source ou do seu local de trabalho), normalmente não temos as permissões para fazer um *push* direto

Nesse cenário nós seguimos um fluxo de trabalho diferente, envolvendo duas novas ideias: Fork e Pull Request



Fork e Pull Request.



- Fork é uma cópia pessoal sua de um repositório de outra pessoa
- Fica na sua conta do GitHub
- Onde você tem permissão para salvar seu trabalho



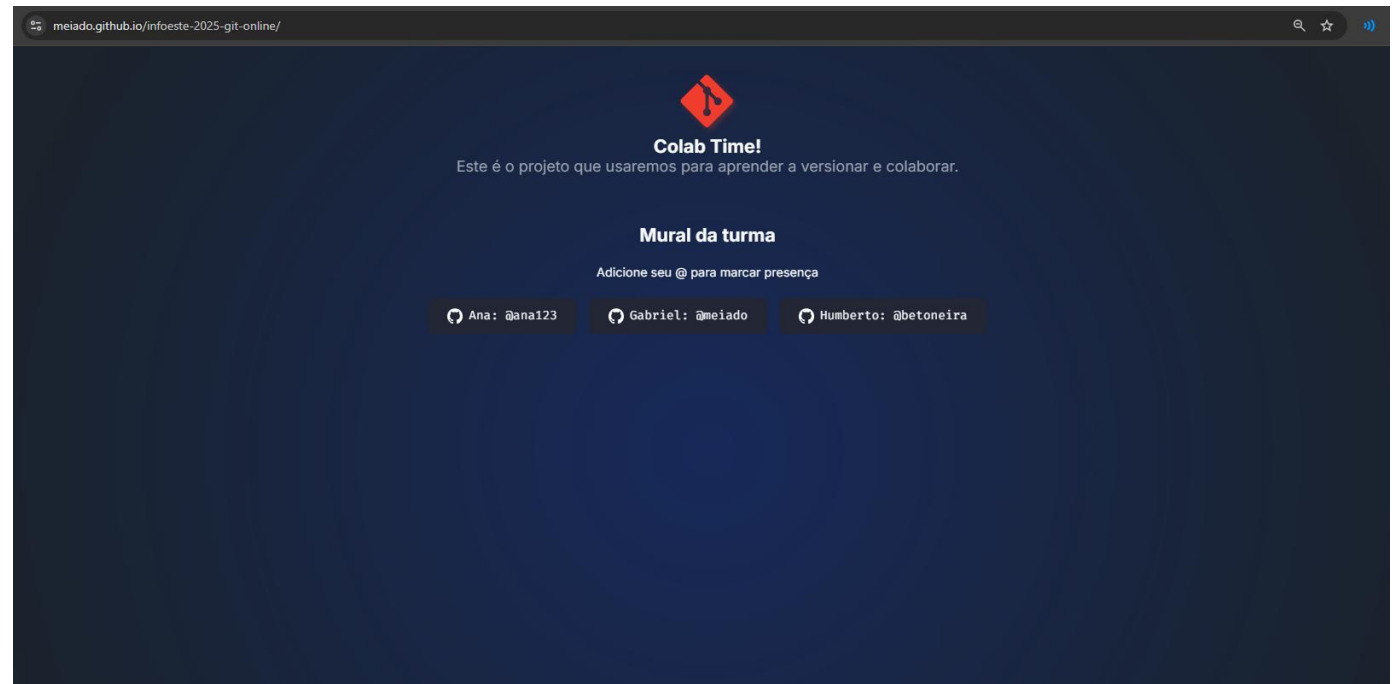
- Pull Request é um pedido formal para o dono do projeto "puxar" suas alterações
- Você abre o pedido do seu fork para o repositório original
- Onde acontece o Code Review e a interação com outros colaboradores

Colaborando.



Fizemos um projeto para ser um mural da turma onde vocês podem colocar em prática tudo o que vimos até aqui.

O desafio agora é usar de tudo que vimos no decorrer do curso e fazer a contribuição de vocês com um projeto existente.



Colaborando.



The screenshot shows the GitHub repository page for 'infoeste-2025-git-online'. The repository is private and has 0 forks and 0 stars. A red box highlights the 'Fork' button, with a red arrow pointing to it. The repository has 1 branch (main) and 0 tags. The file list includes: Meiado (Delete Material Completo.pdf), assets (feat: adicionando projeto para colaboração (#1)), .gitignore (Initial commit), LICENSE (Initial commit), README.md (feat: adicionando projeto para colaboração (#1)), and index.html (feat: adicionando projeto para colaboração (#1)). The 'About' tab is selected, showing no description, website, or topics provided. The 'Releases' section shows no releases published.

The screenshot shows the 'Create a new fork' form in GitHub. The form includes a description field, a 'Copy the main branch only' checkbox (checked), and a 'Create fork' button. The repository name is 'infoeste-2025-git-online'.

Create a new fork

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Required fields are marked with an asterisk (*).

Owner * Repository name *

Choose an owner / infoeste-2025-git-online

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description

0 / 350 characters

☒ Copy the main branch only

Contribute back to Meiado/infoeste-2025-git-online by adding your own branch. [Learn more.](#)

Create fork

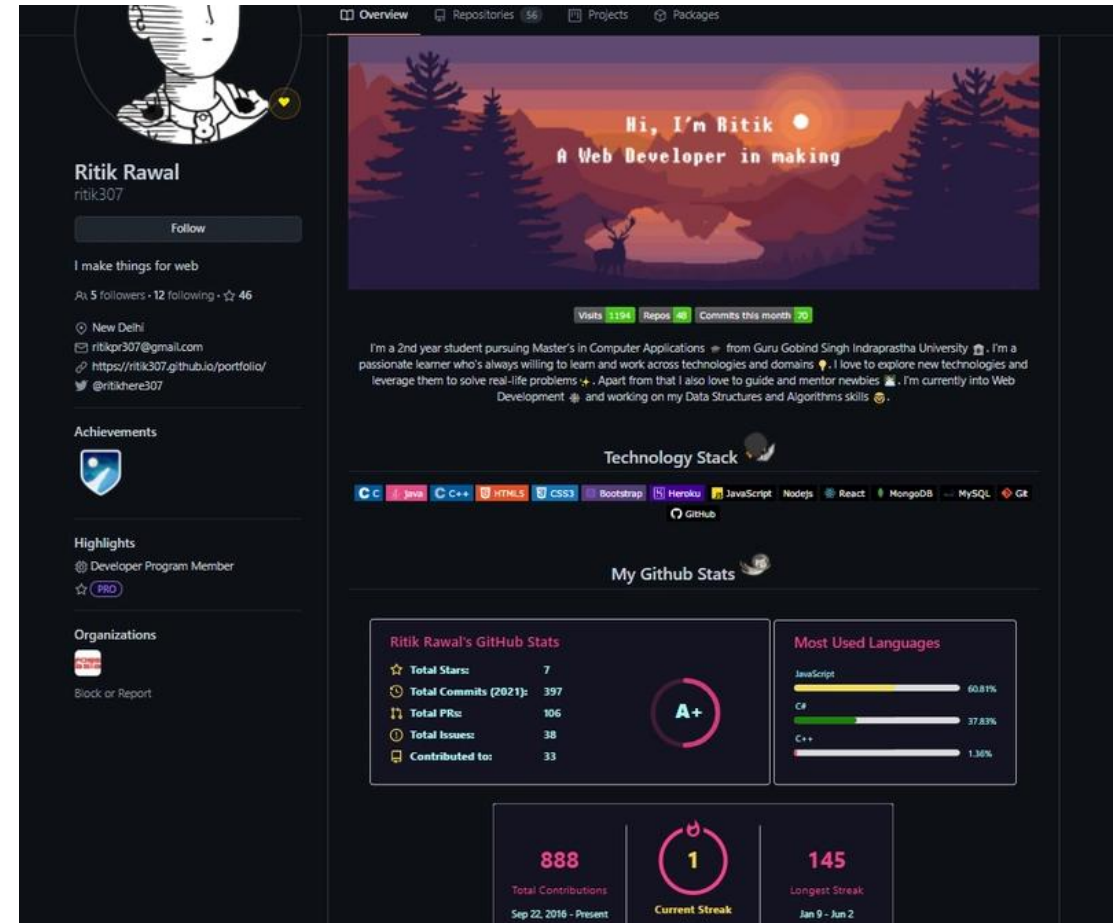
Construindo seu Portfólio.



Seu perfil no GitHub é seu Portfólio.

Recrutadores e empresas hoje olham o seu GitHub para ver o que você sabe fazer. Tudo o que aprendemos é a base para você mostrar seu trabalho ao mundo.

- Contribua para projetos open-source
- Crie seus próprios projetos
- Use o README.md
- Mantenha a “grama verde”



Obrigado pela presença!



Ana de Paula

 [@acp-athena](#)

 [Ana C. Almeida Paula](#)




Gabriel Meiado

 [@Meiado](#)

 [Gabriel Meiado](#)



Humberto Stuari

 [@HumbertoStuari](#)

 [Humberto Stuari](#)