

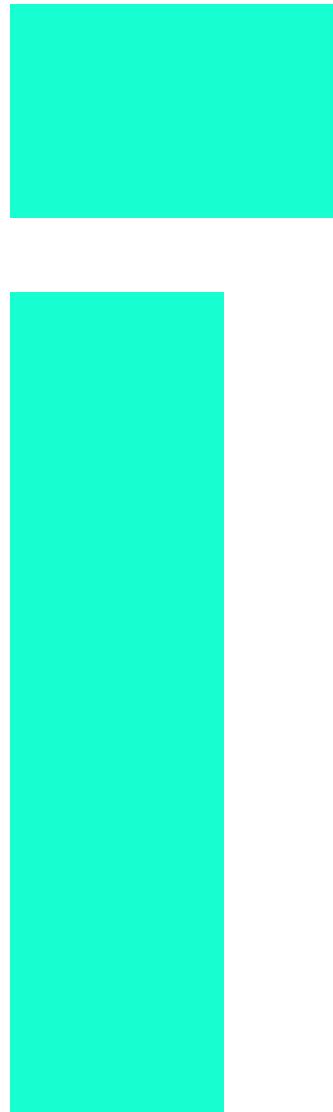
código.py

GALÁXIA EXTRA



Introdução

Olá, seja bem-vindo à Galáxia Extra de GitHub! Neste módulo vamos explorar a maior rede social de códigos no mundo, como trabalhar em equipe e colaborar com outros desenvolvedores, utilizar o versionamento de códigos de um projeto, como criar repositório e trabalhar com branches, fazer commit e pushes. Após aprender os conceitos básicos do GitHub, você estará pronto para explorar recursos mais avançados como issues, pull requests e pipelines.



Mundo 1

1.1. – A importância do Git e GitHub

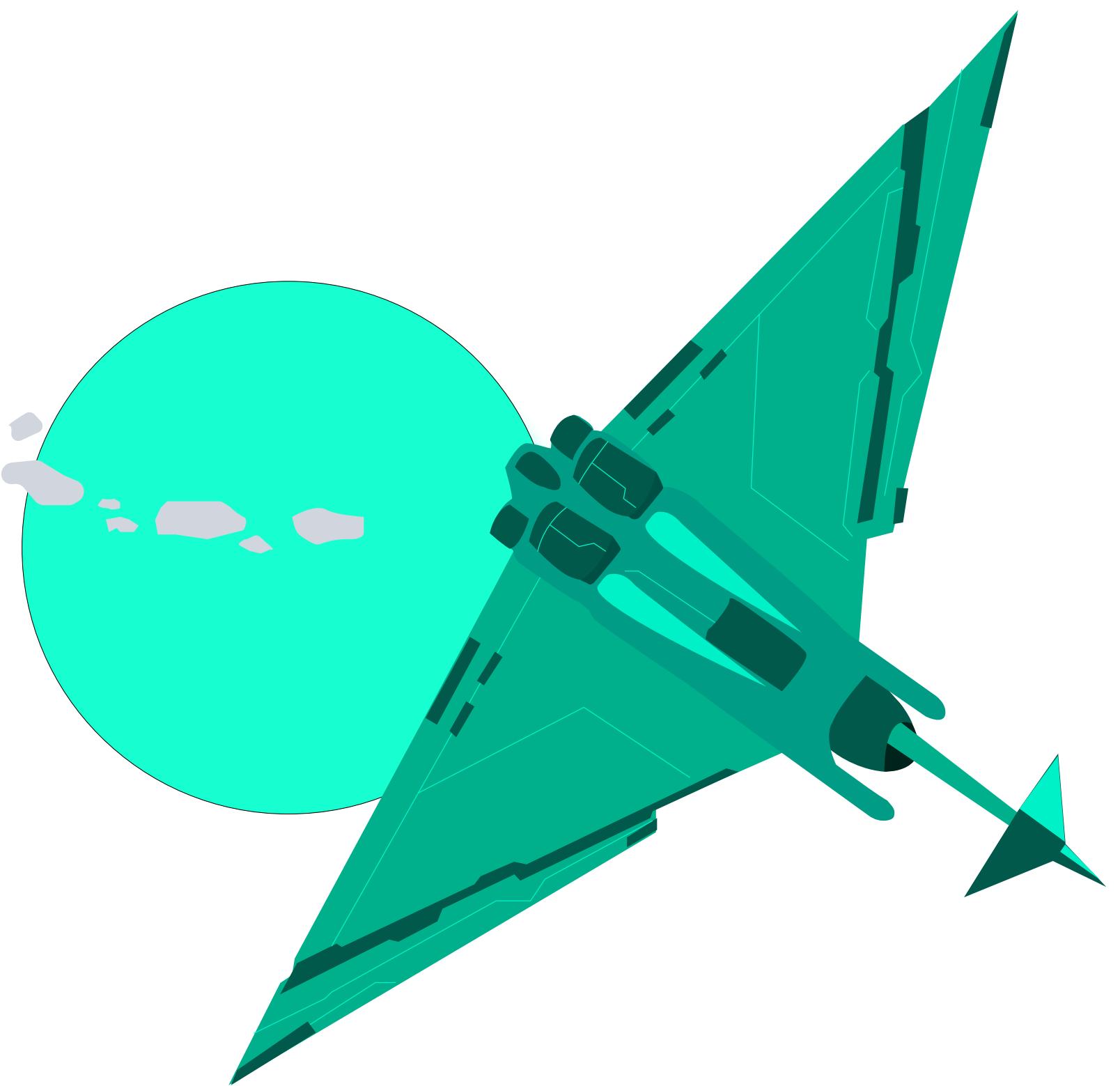
Trabalhar em um projeto de desenvolvimento de software com várias pessoas pode ser desafiador, especialmente quando cada membro da equipe está fazendo modificações diferentes no código.

A ausência de um sistema de controle de versão pode levar a inúmeras complicações. Por exemplo, uma pessoa da equipe pode assumir que uma função específica, chamada de "Função A", retorna um valor particular igual à 2. No entanto, outra pessoa pode ter modificado essa função para criar "Função A.2" que retorne um valor negativo e a função "B.2" depende dela. Isso cria incerteza e risco, pois ninguém pode garantir que a "Função B.2" funcionará corretamente com as alterações na "Função A".

Sem um sistema adequado, o projeto se transformaria em um conjunto caótico de versões conflitantes, sem uma maneira clara de identificar qual é a versão mais atual ou o que cada membro da equipe modificou.

Manter backups diários em pastas separadas é uma solução ineficiente e pouco prática. Essa abordagem rapidamente se tornaria insustentável, ocupando muito espaço em disco e tornando difícil a colaboração simultânea. Além disso, encontrar a versão correta em meio a tantos backups se tornaria uma tarefa árdua.

É por isso que o uso do Git e do GitHub se tornou essencial para o desenvolvimento de software em equipe. Embora muitos possam pensar que são a mesma coisa, eles desempenham papéis distintos, mas complementares.



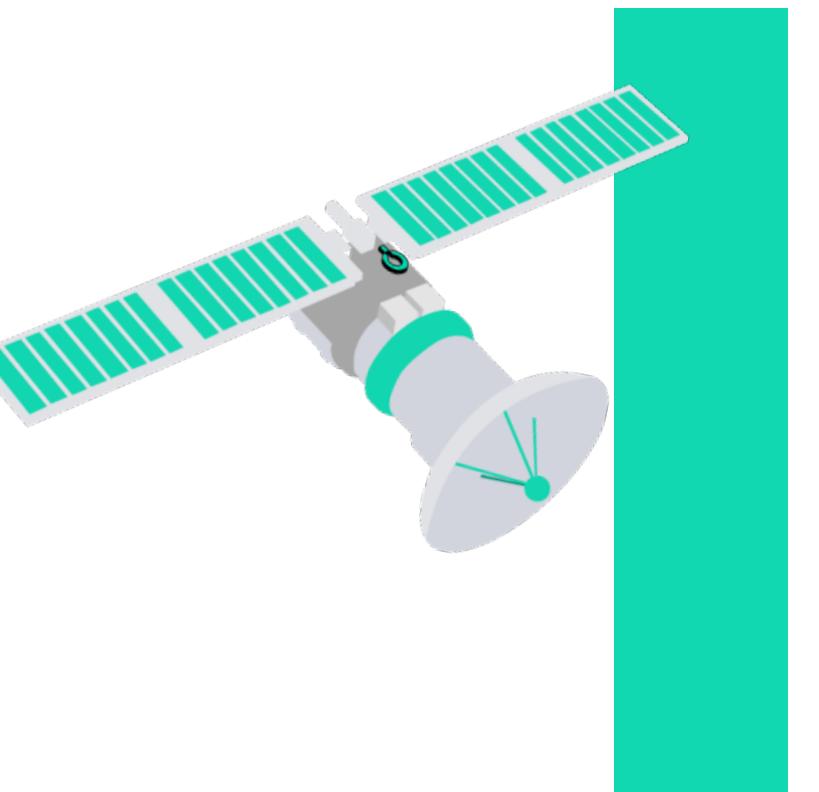
1.2. O que é o Git e quais as suas vantagens?

O Git é um sistema de controle de versão local, o que significa que todas as informações sobre seu projeto são armazenadas em seu próprio computador e, opcionalmente, em um servidor Git ao qual apenas as pessoas autorizadas têm acesso.

Uma das principais vantagens do Git é a capacidade de trabalhar em equipe e compartilhar código. Para isso, muitas pessoas utilizam o GitHub, uma plataforma de hospedagem online que armazena repositórios Git na nuvem. Isso permite que outras pessoas colaborem no projeto e acessem o histórico do código de maneira eficiente.

Vantagens de usar o Git:

1. Controle de Histórico: O Git permite voltar no tempo e visualizar todas as alterações feitas em seu projeto. Isso é útil para entender como o código evoluiu e corrigir erros.
2. Trabalho em Equipe: Com o Git e o GitHub, várias pessoas podem colaborar em um projeto simultaneamente. Elas podem ver as alterações em tempo real e colaborar de maneira eficaz, desde que o projeto seja compartilhado.
3. Ramificação: O Git possibilita que várias pessoas trabalhem em diferentes partes do projeto ao mesmo tempo, o que é chamado de ramificação. Isso facilita o desenvolvimento de recursos ou correções de bugs independentes.
4. Segurança: As alterações feitas no Git são privadas por padrão. Somente você pode visualizá-las e editá-las, a menos que você compartilhe seu trabalho. Isso garante a segurança de suas contribuições.

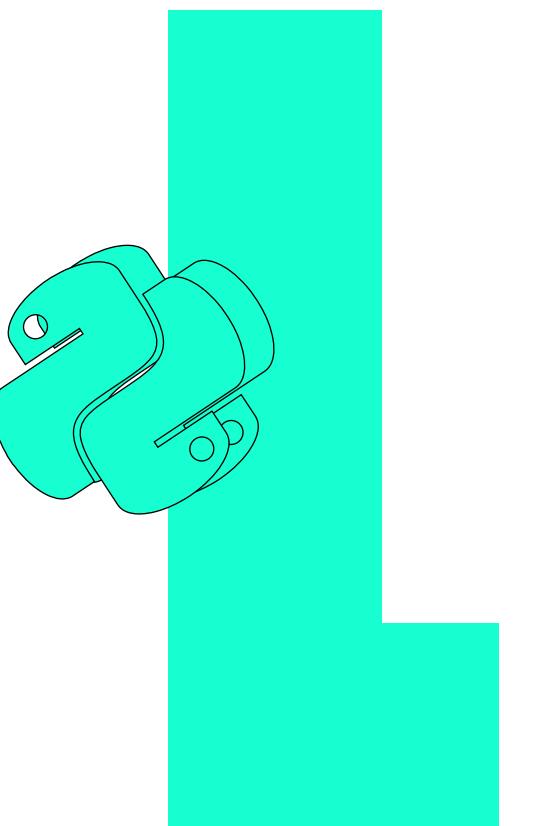


1.3. O que é e para que serve o GitHub?

O GitHub é uma plataforma de hospedagem remota de repositórios Git. Pense no GitHub como uma espécie de rede social para códigos. É importante ressaltar que a maior parte do trabalho é realizada pelo Git, que cuida da administração do seu código localmente. O GitHub entra em cena para fornecer uma interface amigável que organiza e apresenta suas informações de forma clara. só precisa recolher essas informações uma vez.

Para ilustrar, imagine o GitHub como um "Facebook" para códigos. Você pode compartilhar, acessar, salvar, modificar e realizar diversas ações com seus versionamentos, que estão previamente salvos em seu computador. Essa plataforma serve como um ambiente organizado e remoto que complementa o Git.

Embora não seja a mesma coisa, podemos fazer uma analogia com a relação entre SQL e MySQL que você já conheceu anteriormente. Um é a linguagem que realiza os serviços, enquanto o outro é a ferramenta que hospeda e facilita o uso desses serviços.



Mundo 2

O Git e o GitHub possuem alguns conceitos e termos fundamentais

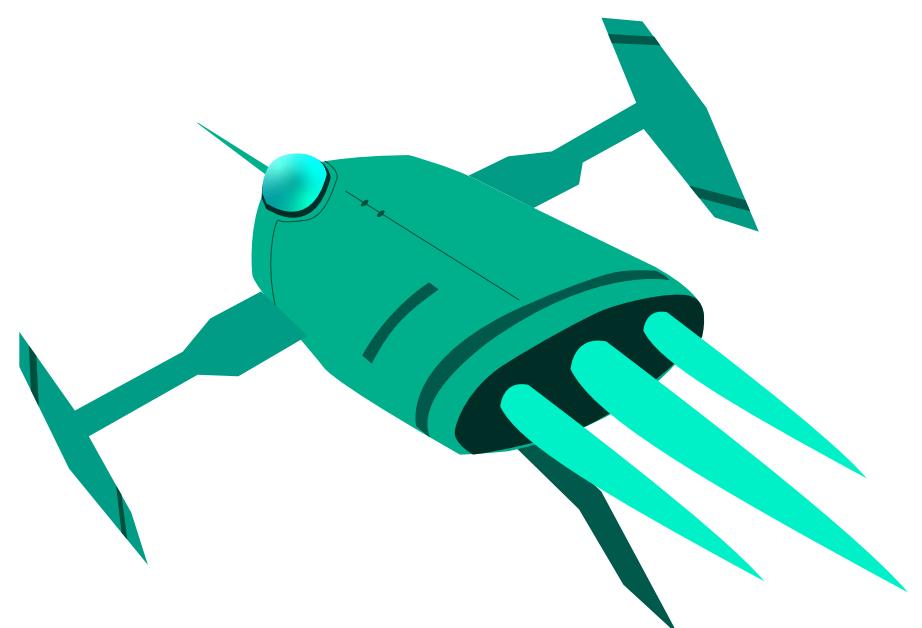
2.1. – Repositório Git.

Um repositório Git é uma pasta comum usada para organizar e gerenciar códigos, imagens, arquivos e etc. O que o torna diferente de uma pasta comum é a presença do arquivo especial ".git" dentro dele.

O arquivo ".git" é como o cérebro do repositório Git. Ele registra todas as alterações feitas nos arquivos dentro da pasta e constrói um histórico detalhado, permitindo que você acesse todas as versões anteriores desse arquivo quando necessário.

Ao criar ou clonar um repositório Git, o arquivo ".git" deve estar presente para transformar a pasta em um repositório Git. Portanto, lembre-se: qualquer alteração nos arquivos dentro desse repositório é registrada e salva, e a remoção de um arquivo é interpretada como exclusão, sem registro de modificações futuras.

Pense em um repositório Git como um ambiente isolado com ações registradas, enquanto qualquer coisa fora dele é ignorada pelo Git e pelo GitHub. O arquivo ".git" geralmente é oculto enquanto navega no diretório, mas é possível visualizá-lo com um comando específico.



2.2. O que é uma commit?

Em termos simples, um commit é uma versão do seu código. Quando você faz uma alteração no código, você pode fazer um commit para salvar essas alterações. Isso permite que você rastreie as alterações que você fez ao seu código e reverta para uma versão anterior se necessário. É o ato de "enviar e salvar", como se fosse um "checkpoint".

2.3. Estágios de um projeto

Modificado: Primeiro estágio do projeto, qualquer alteração feita em um arquivo, o tornará modificado na mesma hora.

Área de preparação: Neste estágio, seu arquivo está pronto para ser registrado. Você já fez todas as modificações necessárias e agora é hora de enviá-lo para a área de preparação com o comando "`git add <nome da pasta>`". Essa etapa é crucial, pois as mudanças precisam ser adicionadas à área de preparação antes de serem incluídas no próximo "commit".

Imagine que você está colocando seu arquivo em uma espécie de área de aprovação fictícia antes de, efetivamente, adicioná-lo à pasta do seu computador.

De forma análoga: Suponha que você tenha 10 arquivos e precise fazer modificações em todos eles. Em vez de criar 10 "commits" separados e acabar com um histórico confuso e poluído, você pode adicionar esses arquivos à área de preparação e, no final, fazer apenas um "commit". Isso ajuda a manter seu histórico Git mais organizado.

Agora, seus arquivos estão prontos, mas ainda não estão salvos no Git.

Repositório Local: Neste estágio, todas as mudanças foram armazenadas no seu repositório local do Git, depois de um "git commit -m "mensagem"". Agora, seus arquivos já estão salvos, mas apenas localmente no seu computador.

Repositório Remoto: Nesse estágio, seus arquivos estão salvos remotamente no GitHub, após o comando "git push origin <nome_da_branch>", caso queira atualizar todas as pastas "git push". Assim, você e sua equipe podem acessar o projeto de diferentes computadores em qualquer lugar, de forma online.

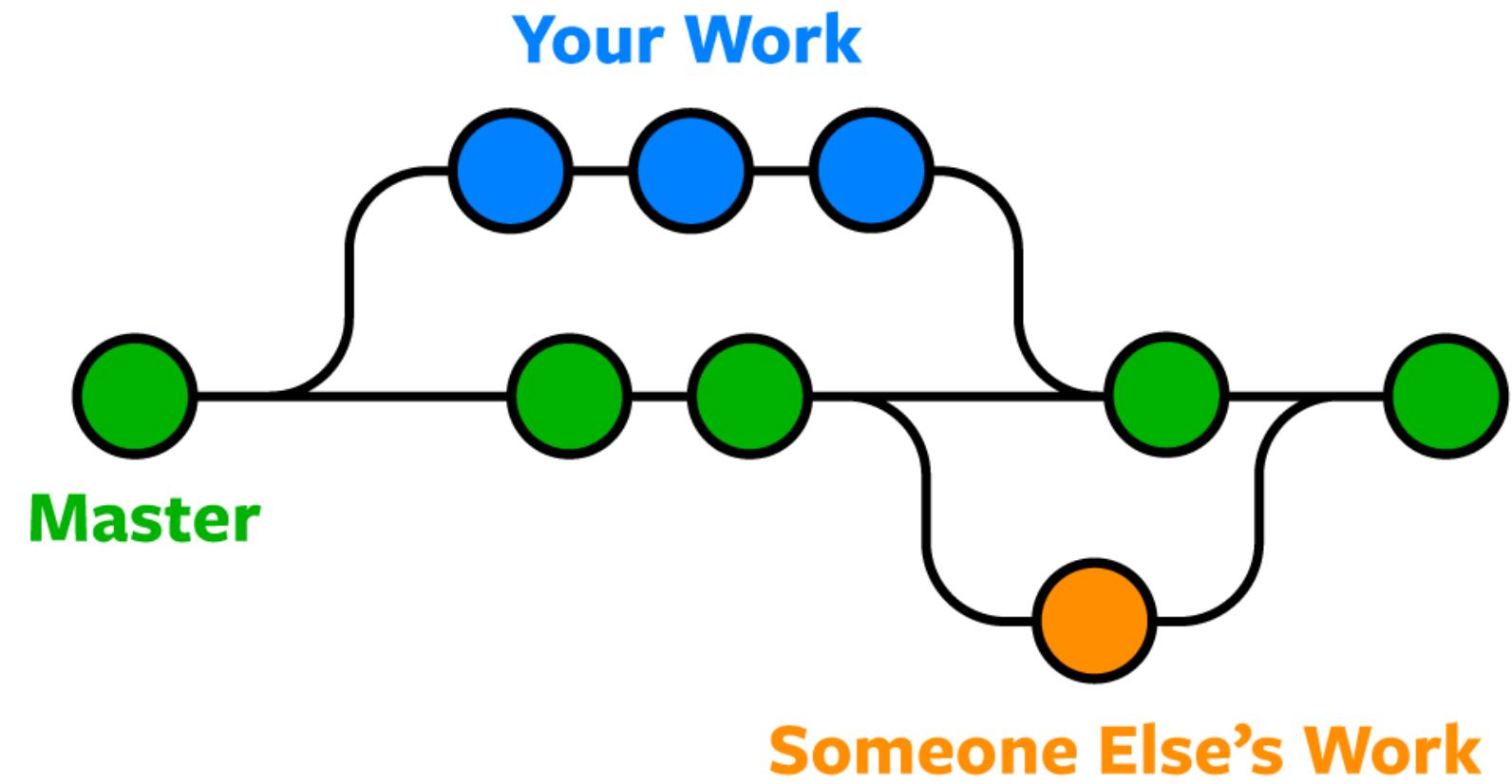


2.4.- Repositório Git.

No Git, "branches" são como ramificações de um projeto. Imagine o tronco principal de uma árvore como a "branch" principal, geralmente chamada de "master" ou "main". A partir dela, você cria outras ramificações para trabalhar em paralelo, sem afetar o código principal. O termo "master" está em desuso, pois é historicamente associado à escravidão e opressão racial.

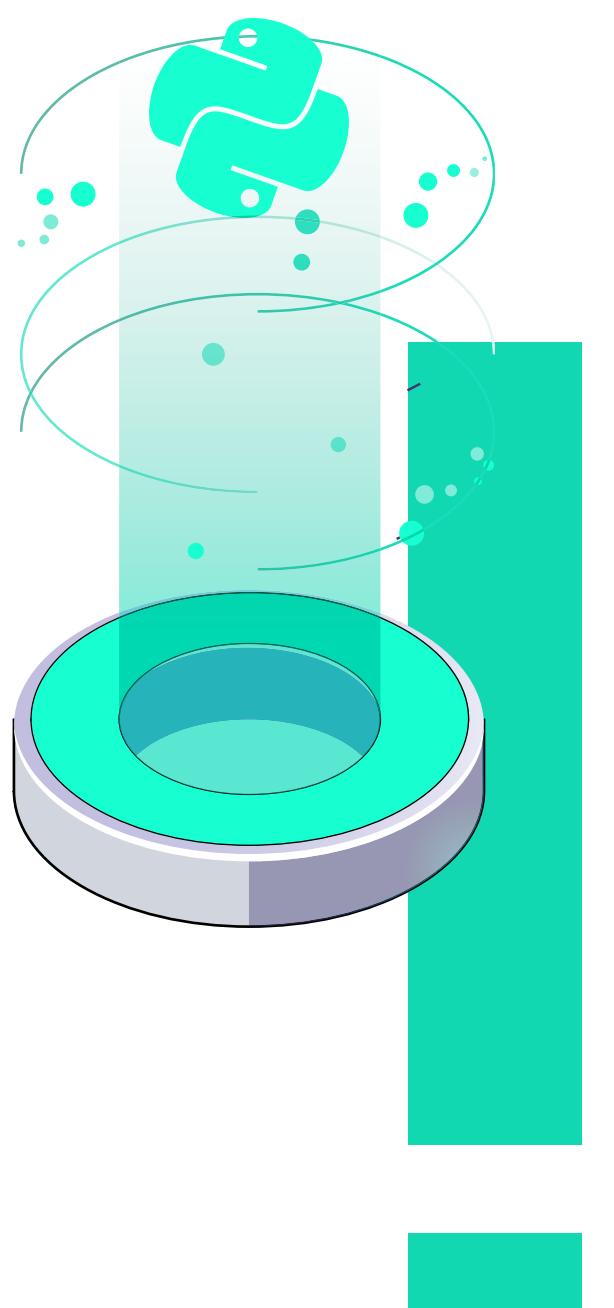
Agora, por que criar essas ramificações? Imagine que você tem um projeto e identifica um problema em uma parte específica do código que precisa ser corrigido. No entanto, você não quer modificar o código principal, pois pode afetar o funcionamento do programa. Nesse caso, você cria uma nova "branch", que serve como um ponto de partida da versão atual do código, e a partir dela, você tenta corrigir o problema. Quando o trabalho na "branch" secundária estiver concluído, você pode mesclar as alterações de volta na "branch" principal, é o que chamamos de "merge".

Quando você cria uma "branch," pode controlar quem pode ou não ver e modificar o código, o que aumenta a segurança e evita conflitos.



2.5.- O que é SSH Keys?

As chaves SSH são uma autenticação usada pelo GitHub para conectar seu computador ao seu repositório remoto. Essa autenticação envolve um par de chaves, uma pública que fica no GitHub e outra privada que fica no seu computador, e a privada não pode ser compartilhada com ninguém. Juntas, elas geram uma chave única que verifica e autoriza sua conexão.



2.6.- O que é uma Fork?

O "Fork" é a ação de copiar um projeto do GitHub. Essa prática é incentivada pelo GitHub e permite modificar projetos que não são seus.

O "Fork" é comumente utilizado quando desejamos modificar um projeto que não é de nossa autoria. Nesse caso, copiamos o projeto original, faremos as modificações desejadas e, em seguida, compartilhamos a nossa versão no GitHub. Isso facilita a colaboração e o desenvolvimento de projetos de forma conjunta, permitindo que várias pessoas

Mundo 3

<https://github.com/>

https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home



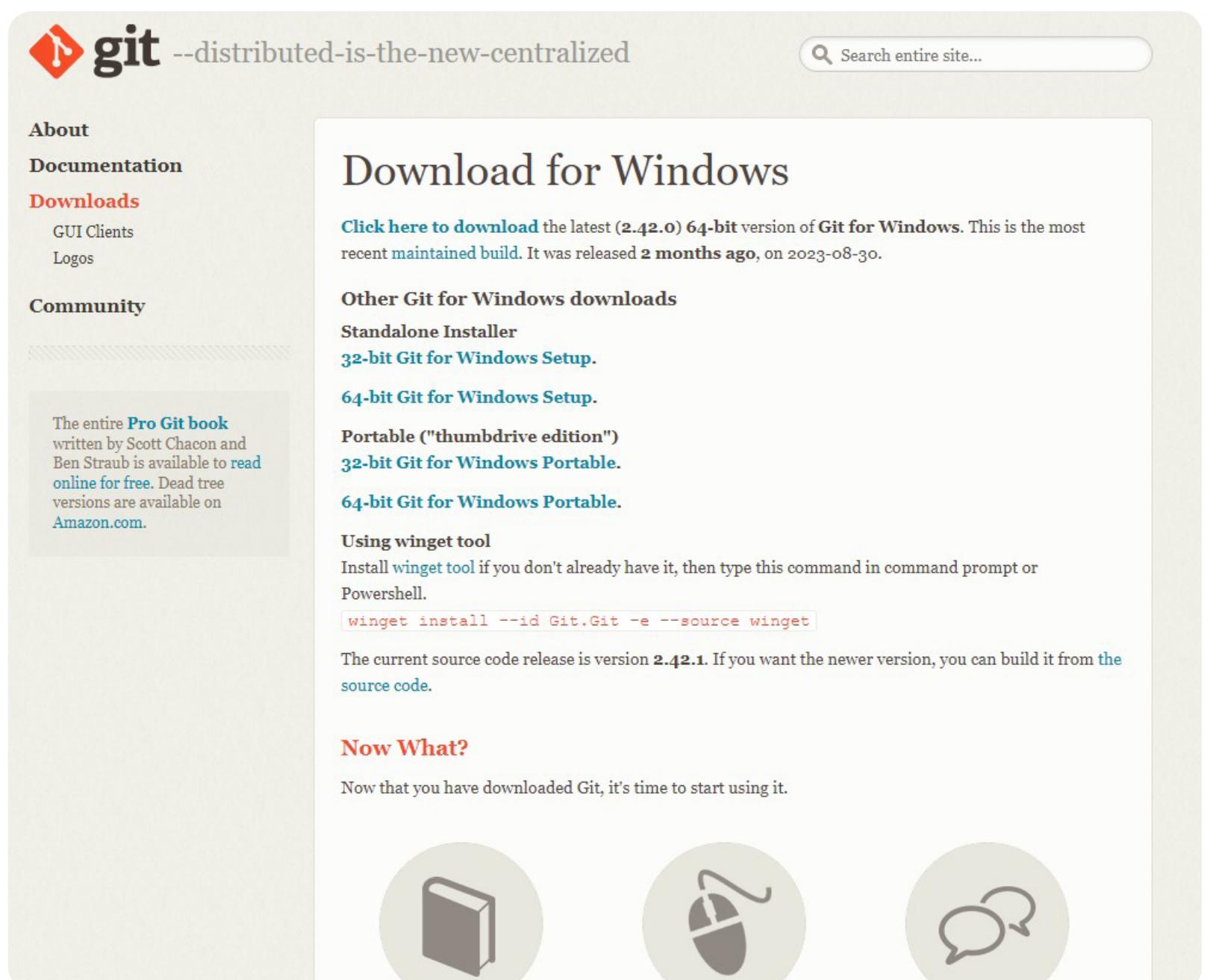
Mundo 4

4.1. - Instalação do Git no Windows

PASSO 1:

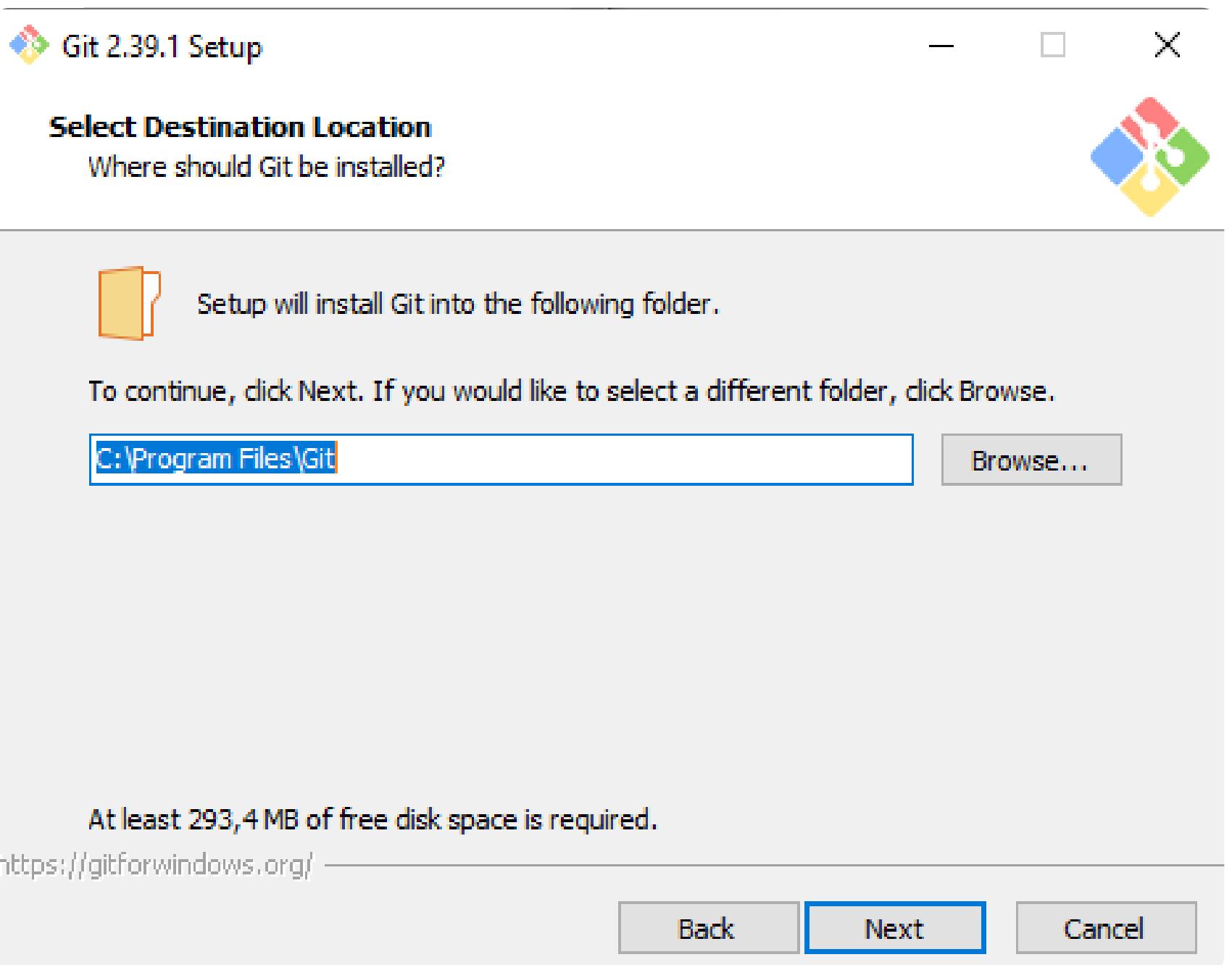
Acesse esse link e clique em download. Escolha onde o instalador ficará dentro do computador:

<https://git-scm.com/download/win>



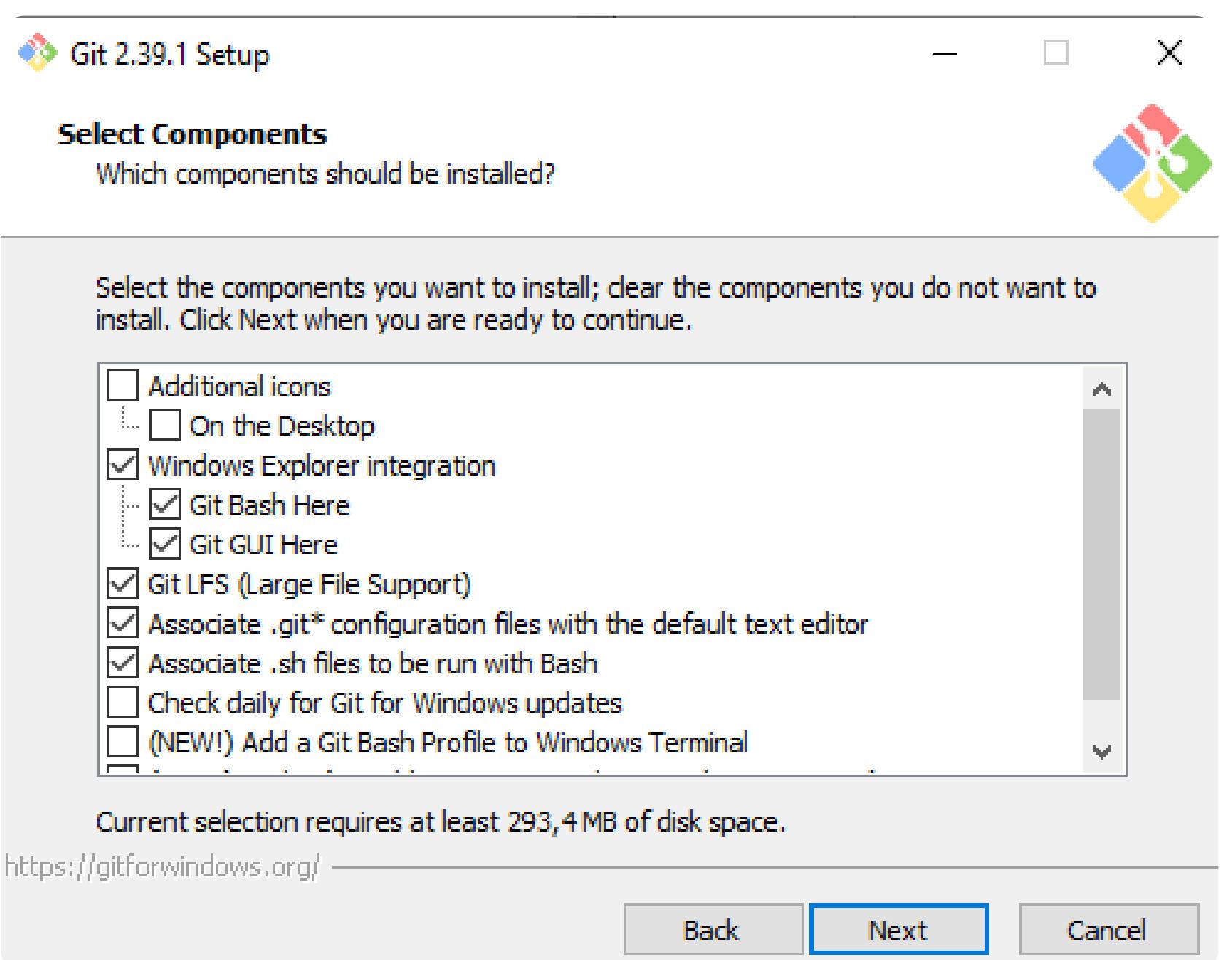
PASSO 2:

Escolha a onde o seu arquivo Git será salvo:

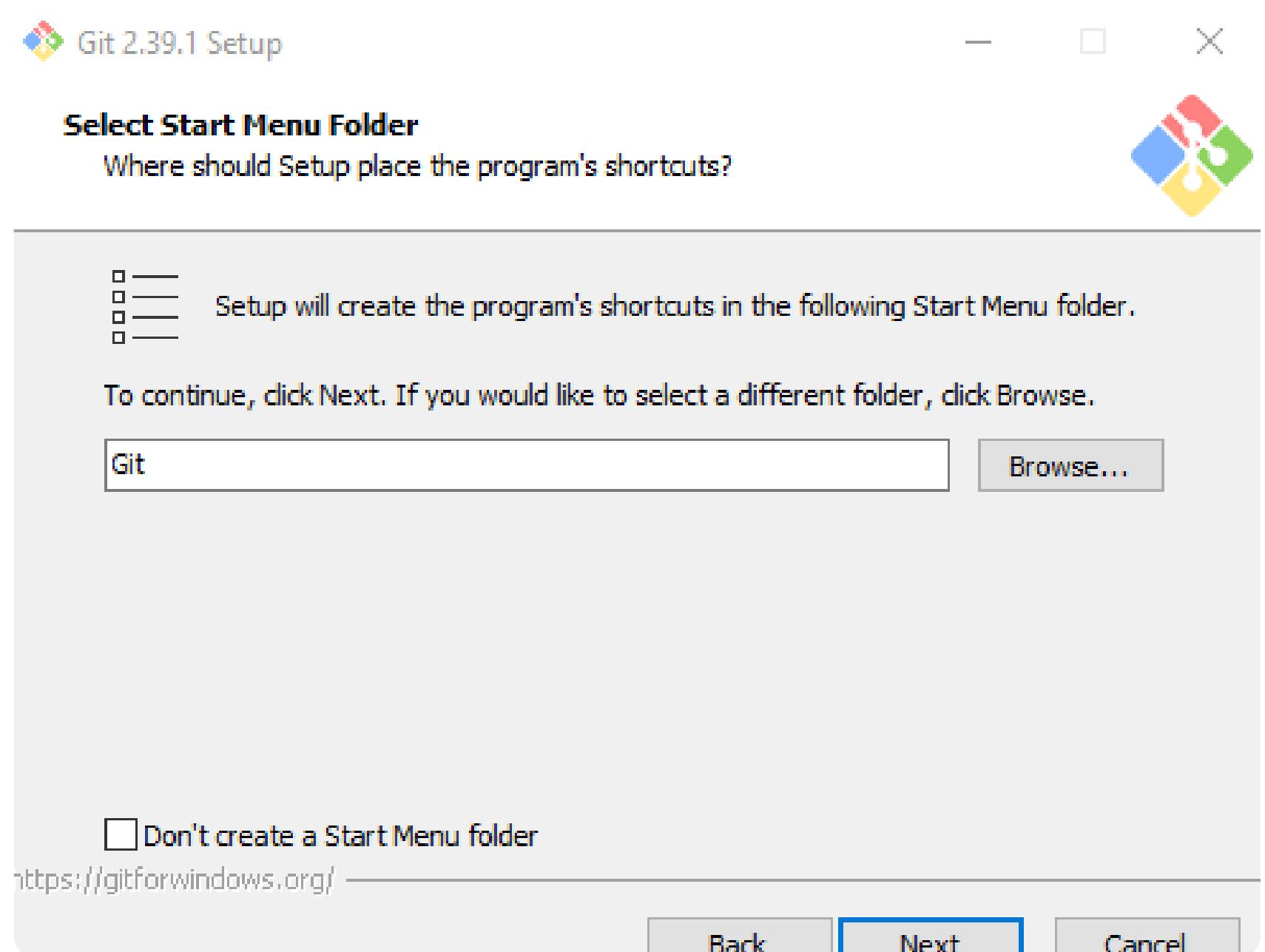


PASSO 3:

Deixe padrão essas opções:

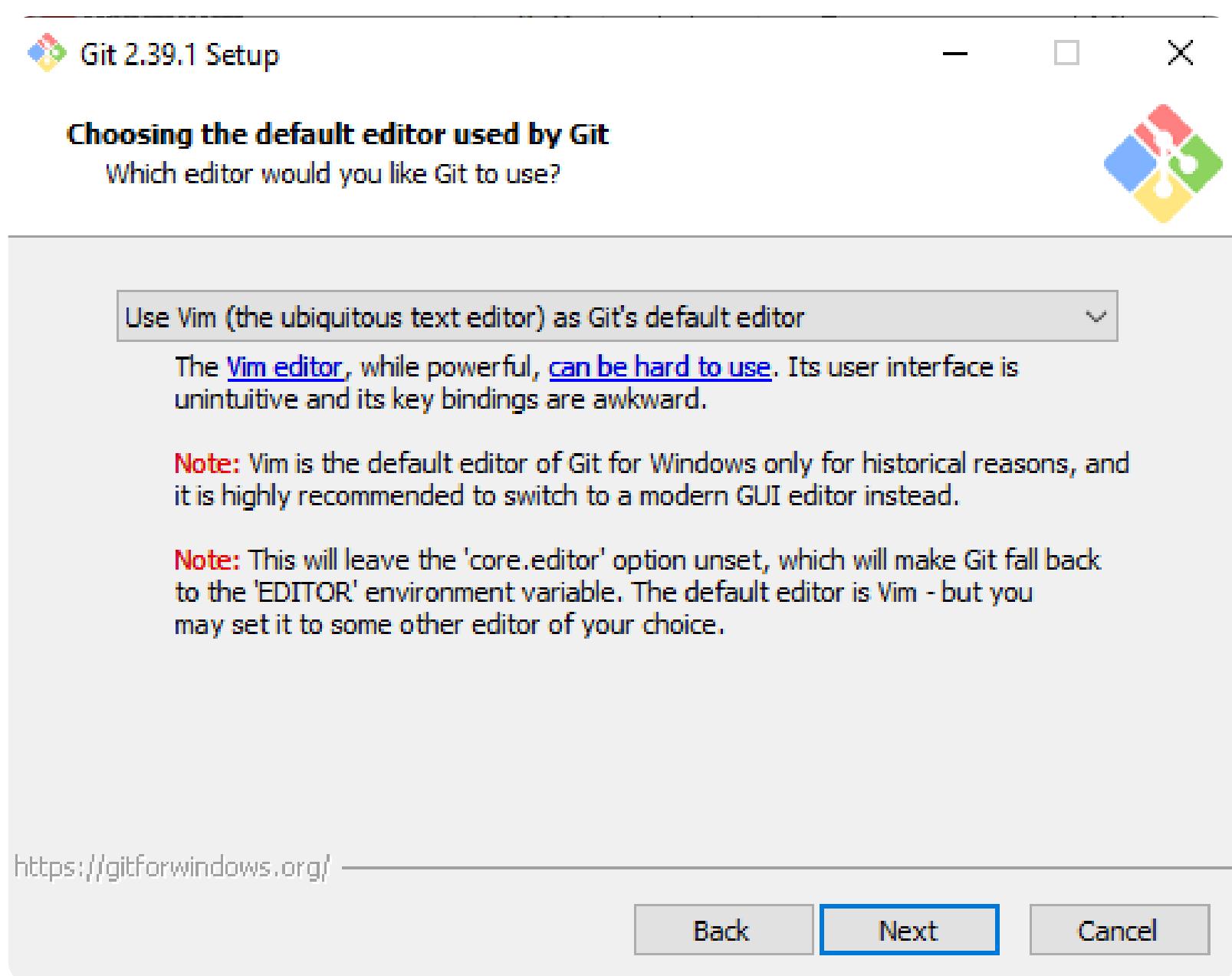
**PASSO 4:**

Nesse passo, vamos definir como o nosso atalho será chamado, e onde ele será salvo. E existe uma opção onde não criamos esse atalho.

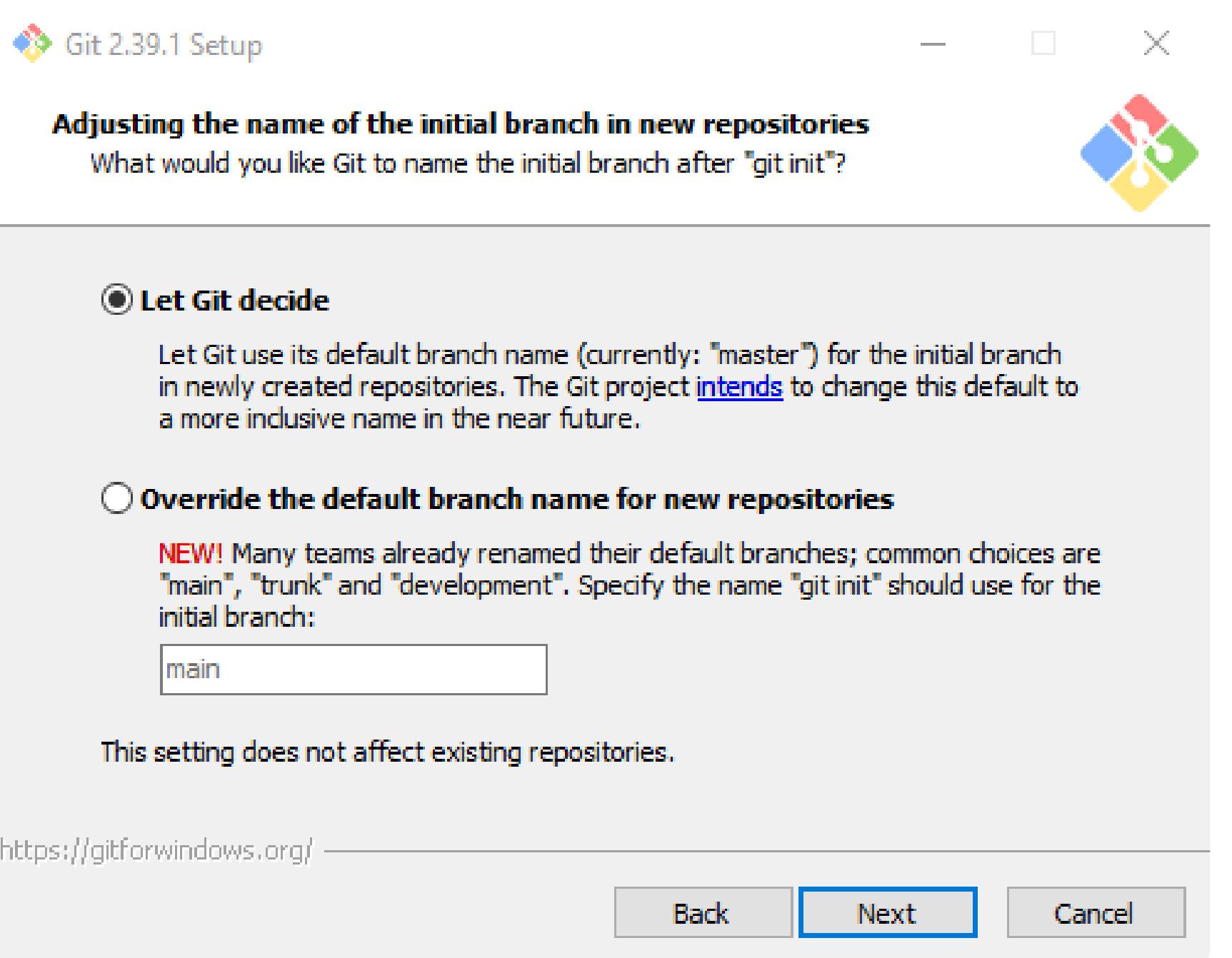


PASSO 5:

Nesse passo, vamos definir o editor de texto a ser utilizado pelo Git, caso deseje escolher algum de sua preferência. Porém aqui vamos deixar o padrão.

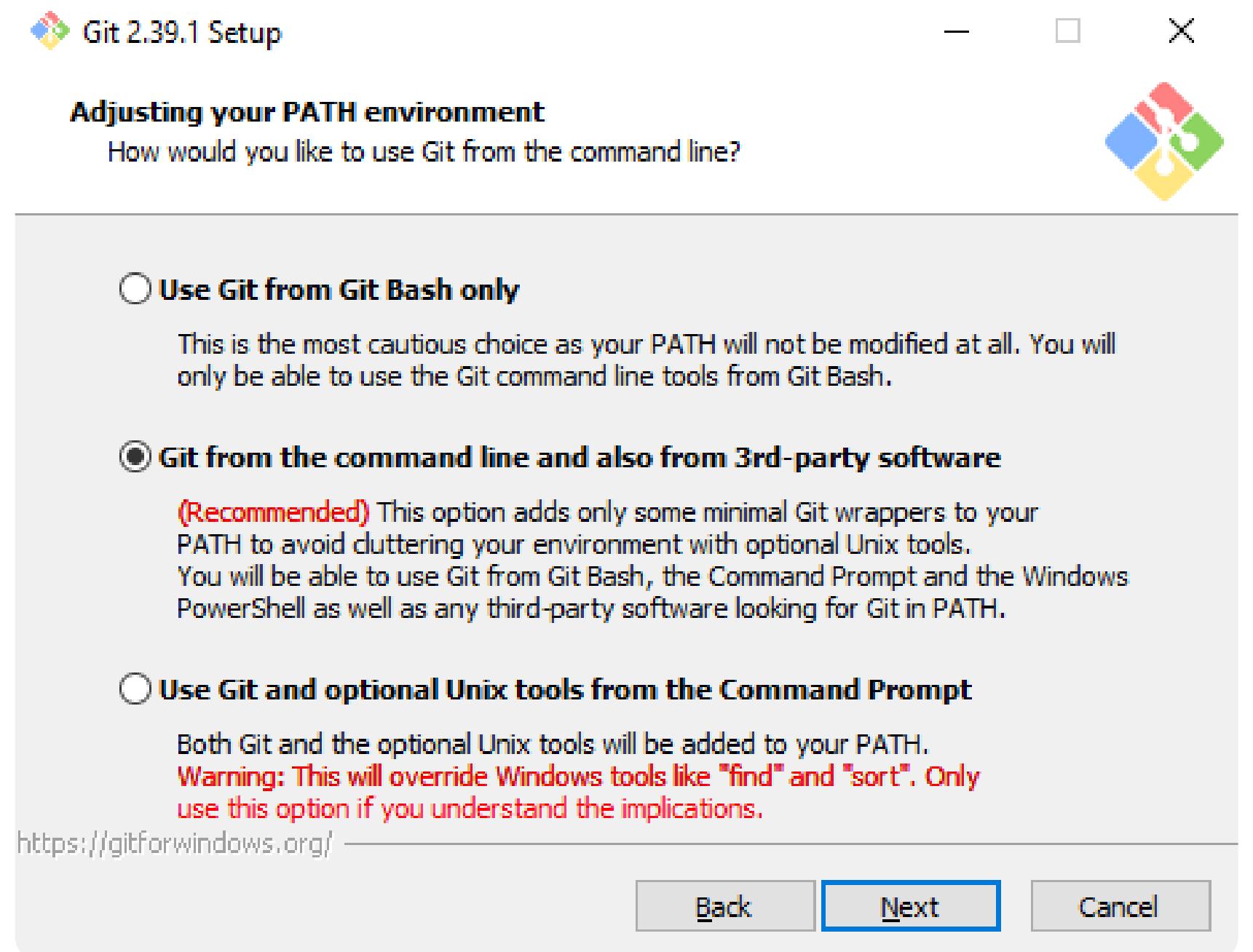
**PASSO 6:**

Nesse passo, vamos definir como será chamada a nossa branch principal. Por padrão, o Git reconhece a branch principal como sendo "master", mas permite a alteração do nome. Vamos manter como "master".

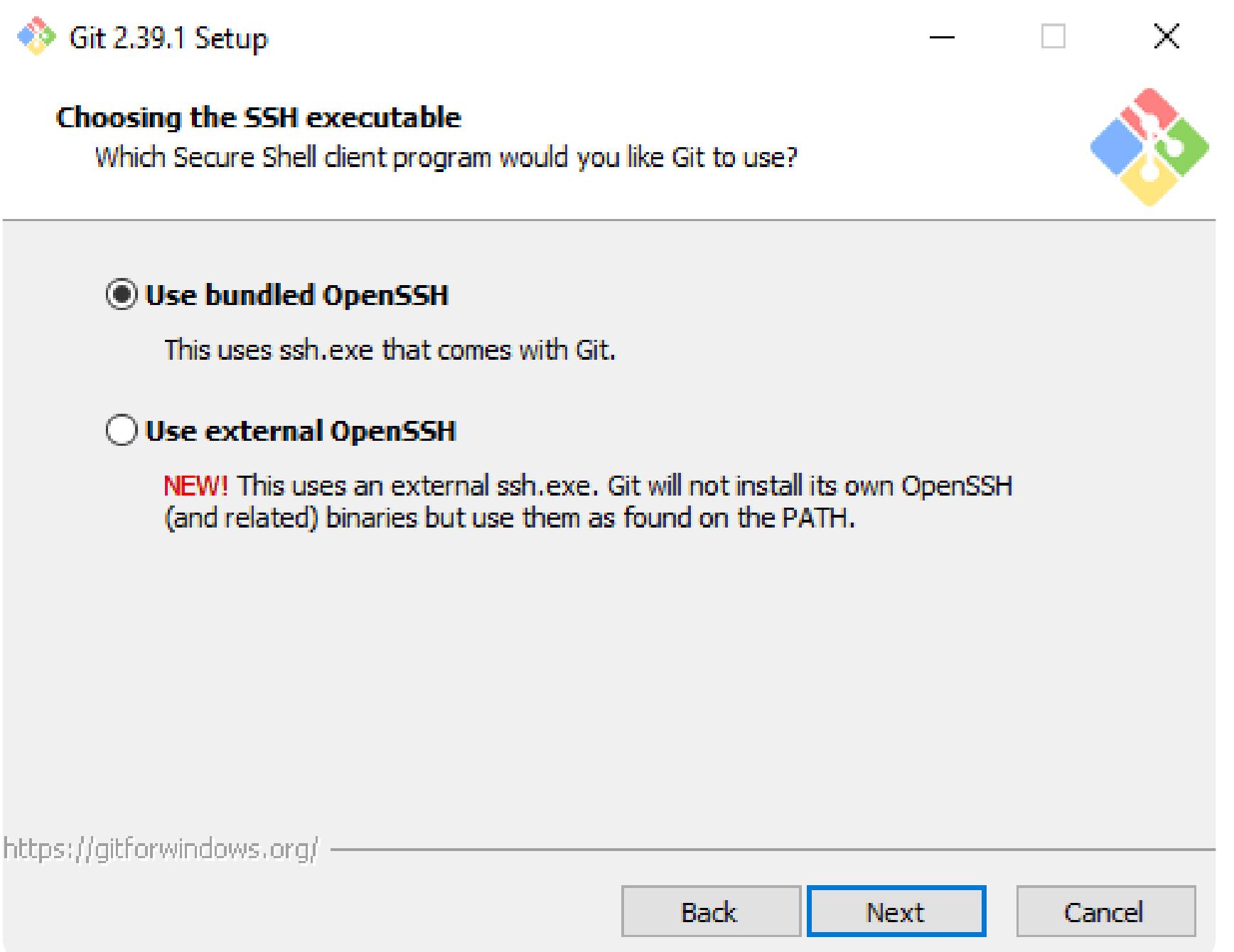


PASSO 7:

Essa opção pergunta como você quer usar o Git. Vamos deixar a opção recomendada que nos diz que podemos utilizar o Git tanto no prompt de comando quanto num terminal do Git, o Git Bash. A primeira opção “Use Git from Git Bash only”, apenas será possível mexer no Git através do terminal e sem integração com outras ferramentas, como o VSCode.

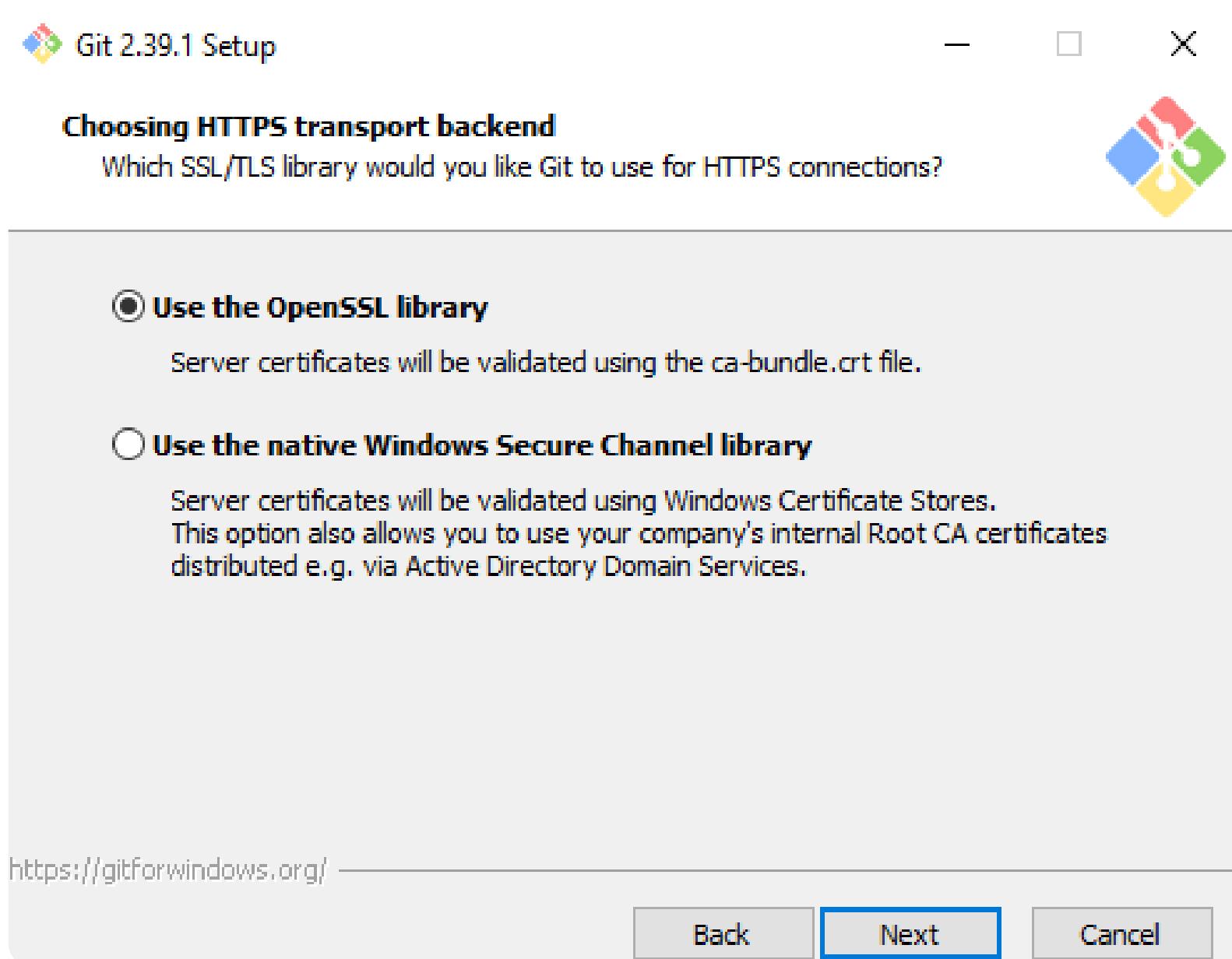
**PASSO 8:**

Essa opção permite que você utilize as mesmas credenciais para logar no Windows, também para conectar num repositório remoto.

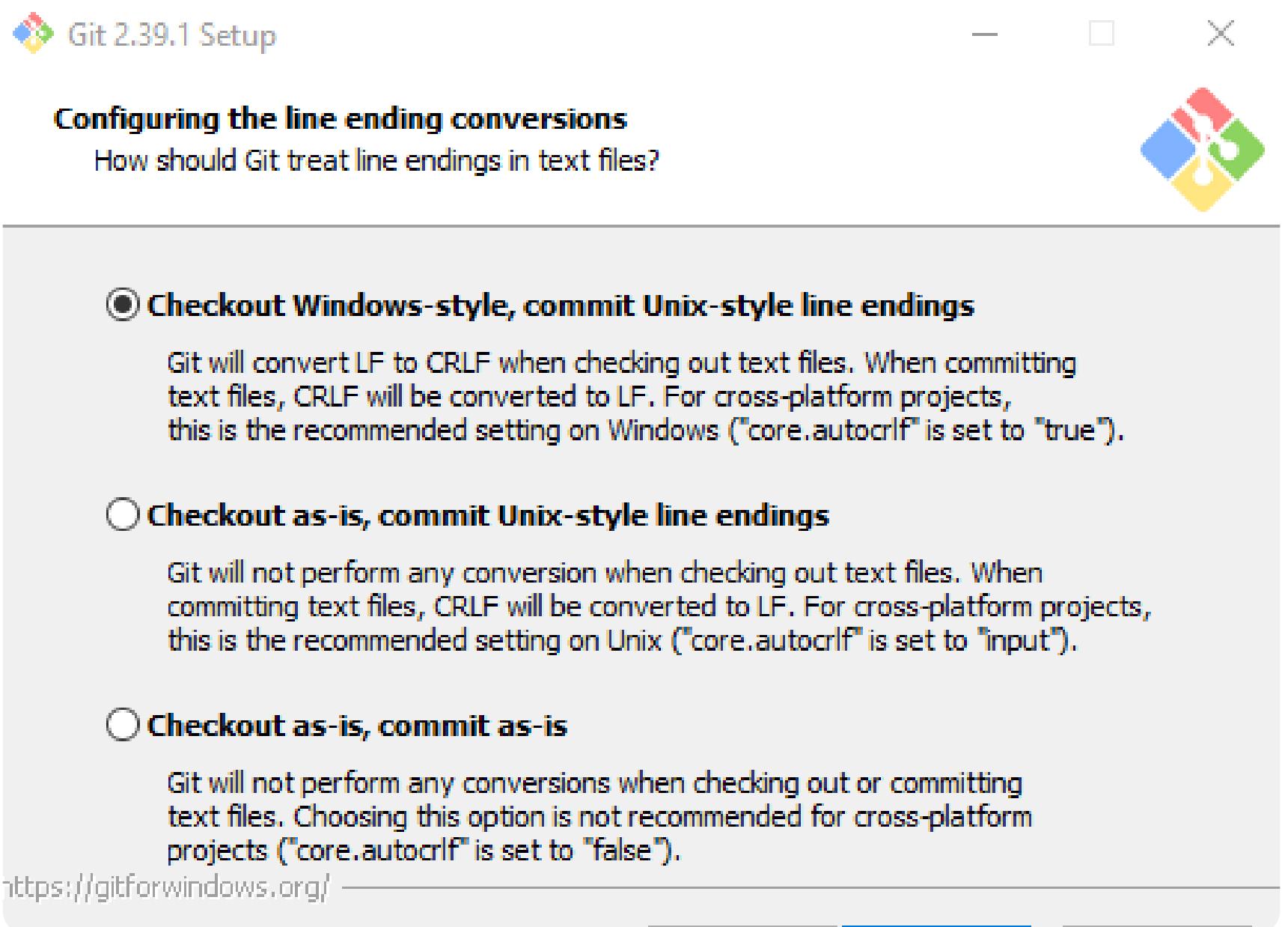


PASSO 9:

OpenSSL Library é um tipo de criptografia utilizada para se conectar ao repositório remoto, vamos escolher essa opção ao invés da segurança do Windows.

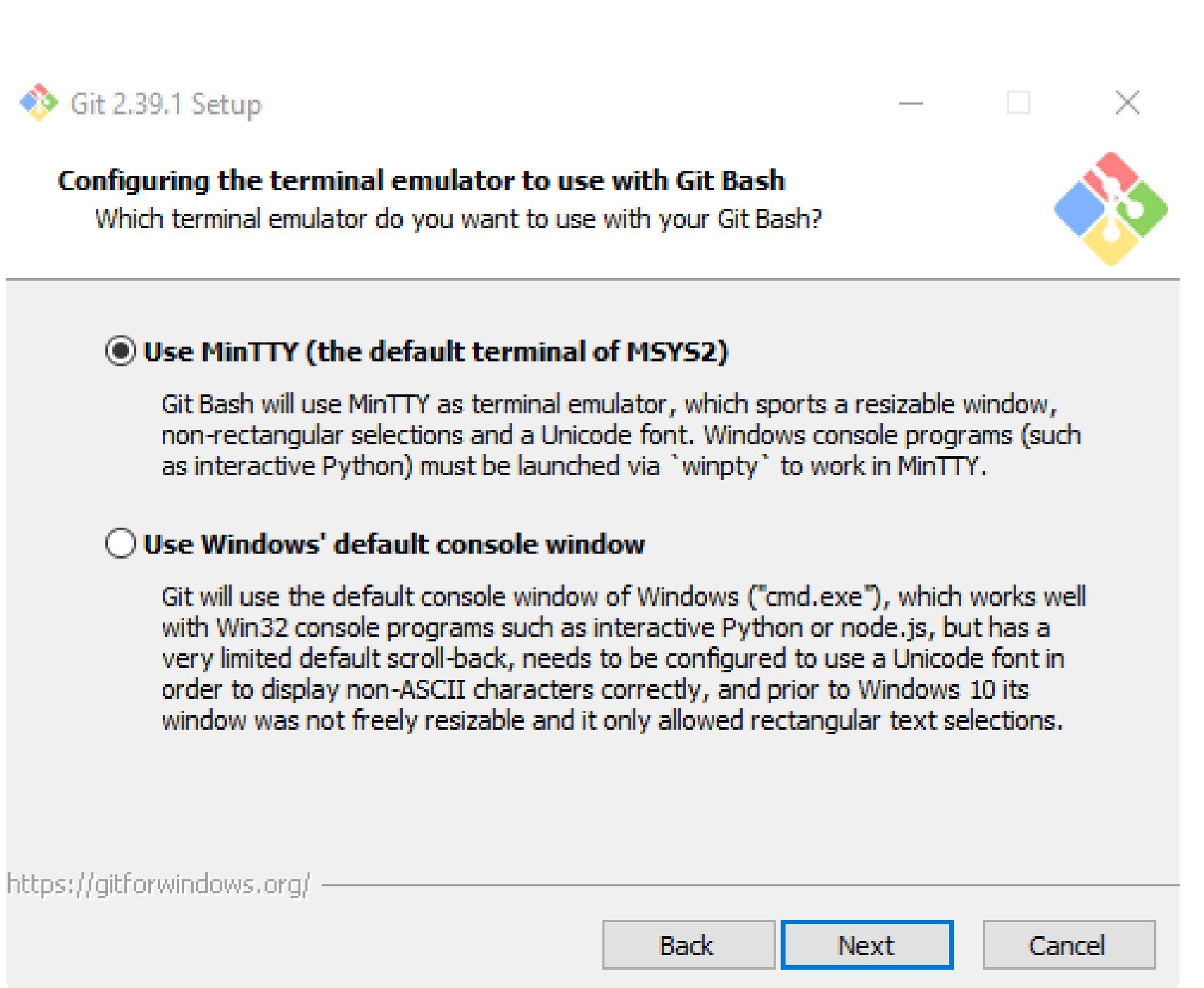
**PASSO 10:**

Isso se resume a estilo, vamos configurar como o padrão também.

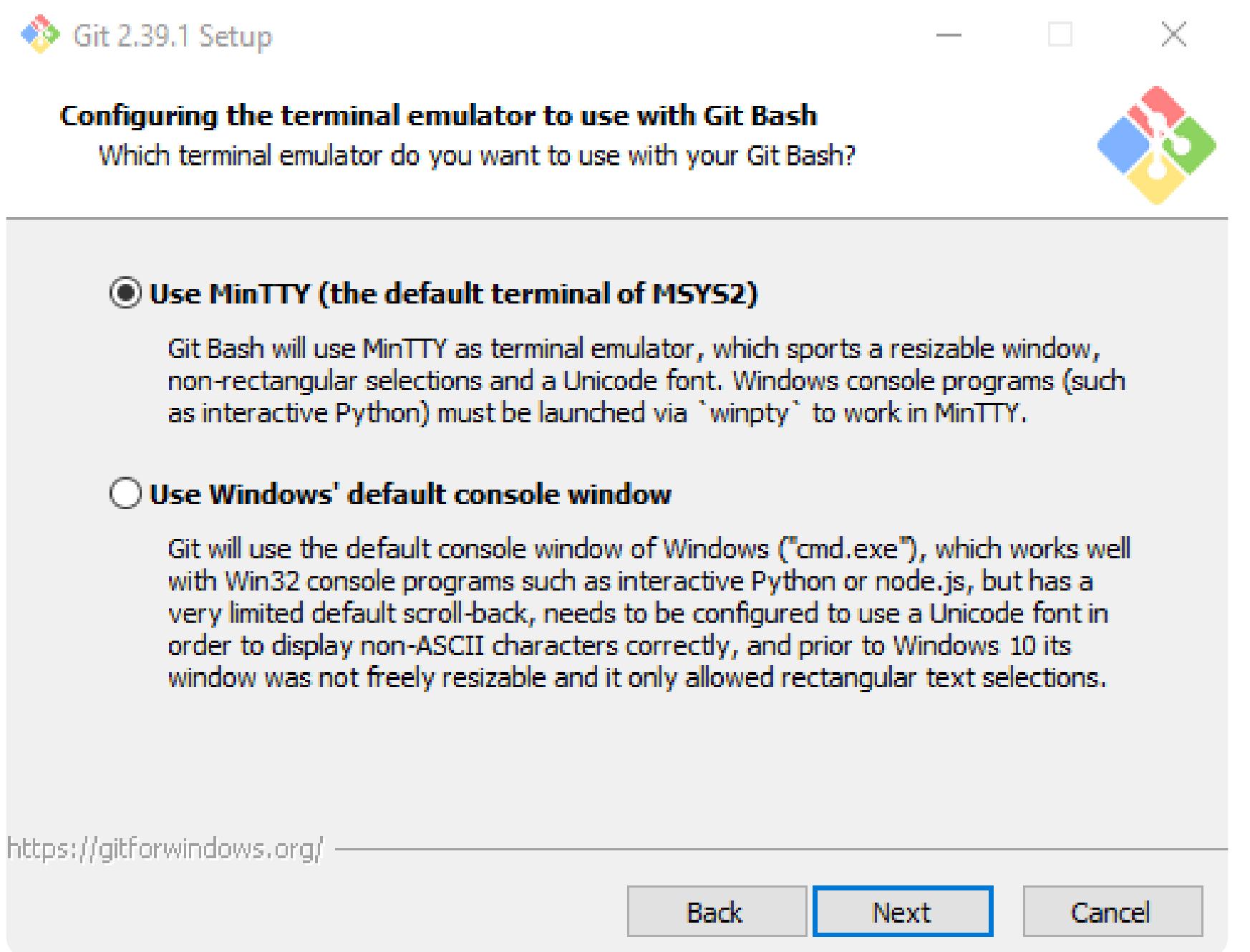


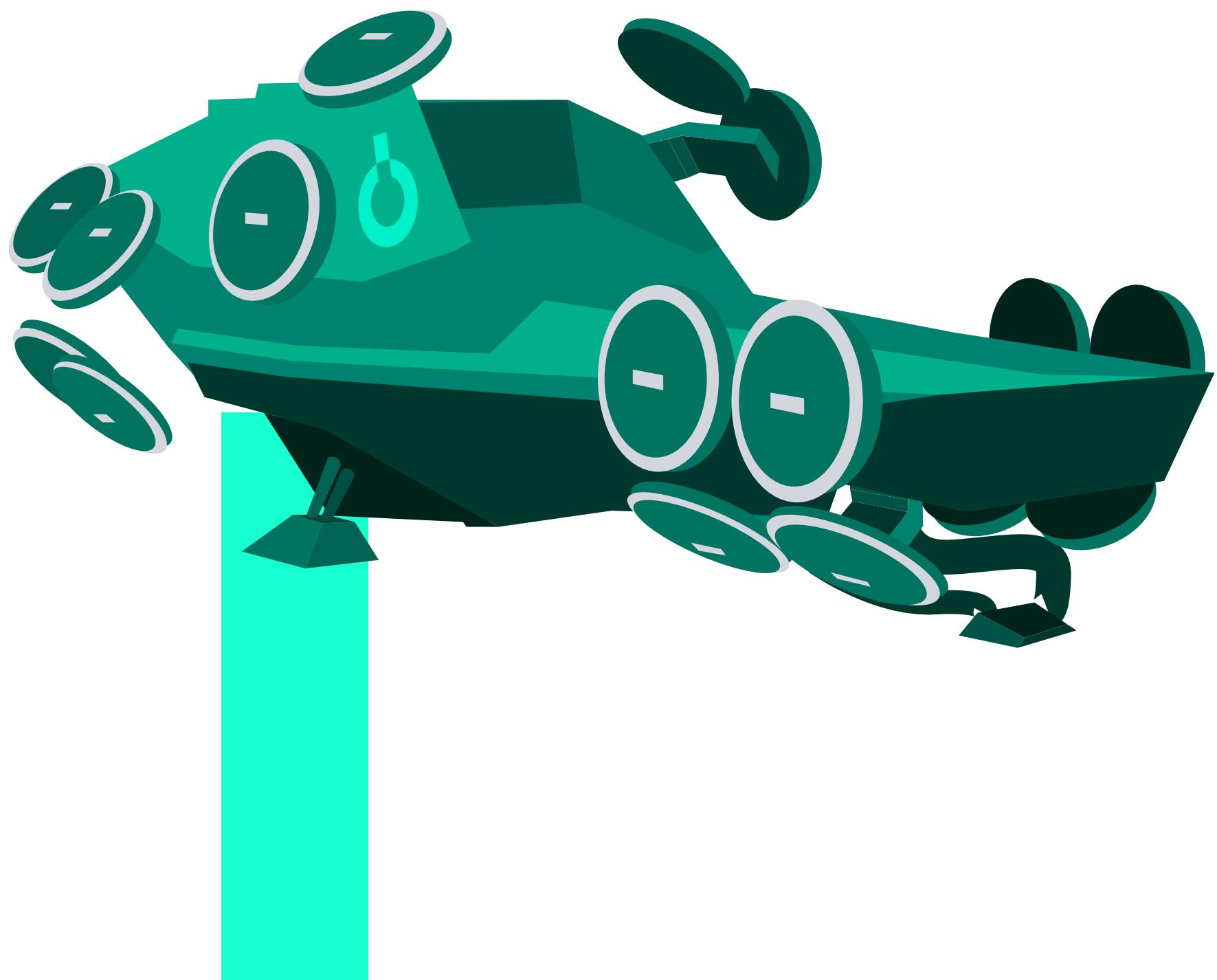
PASSO 11:

Isso é em relação ao terminal, se por padrão o terminal utilizado vai ser o próprio do Git ou se vai ser o do Windows. Nesse caso, utilizaremos o próprio terminal do Git.

**PASSO 12:**

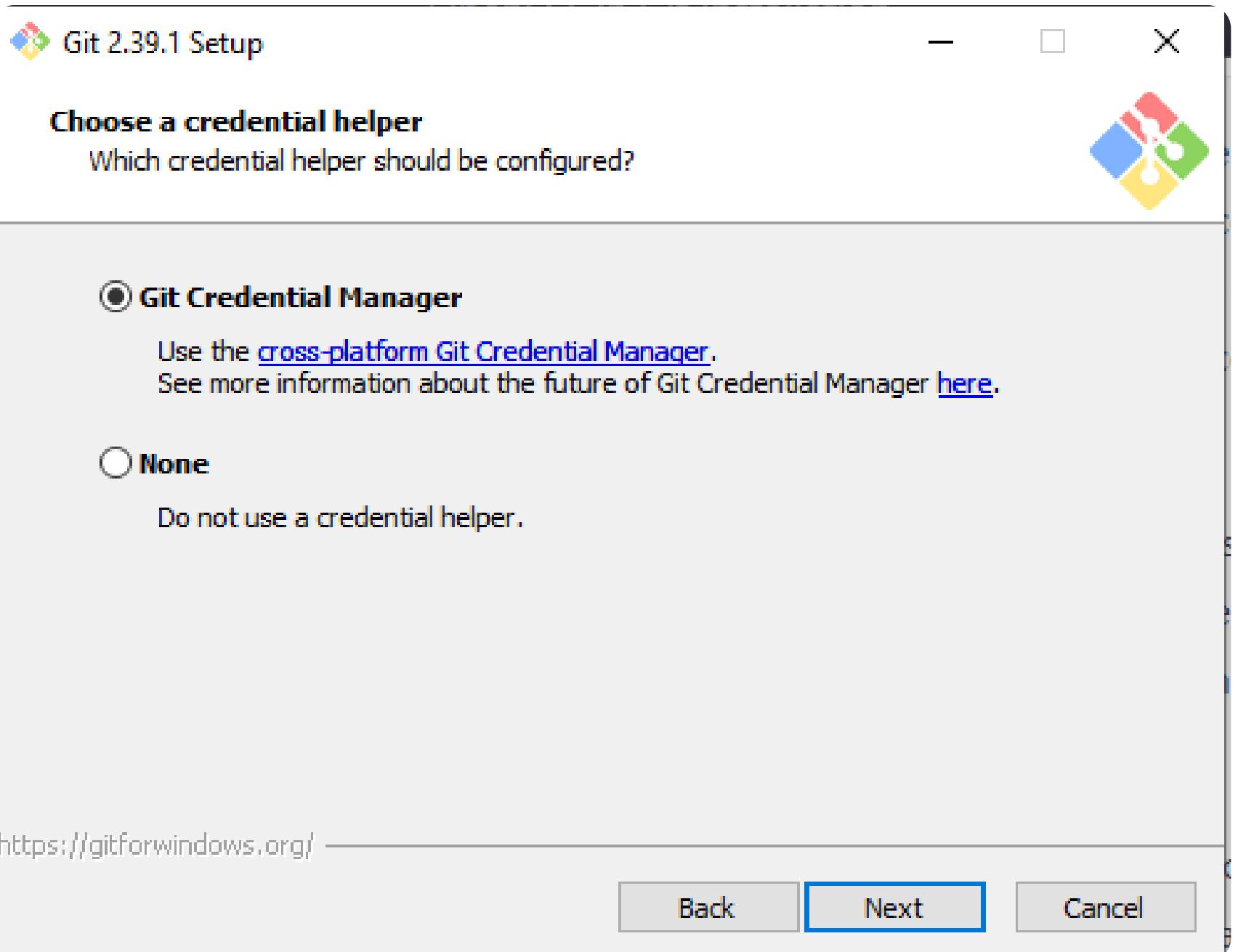
Essa etapa vai especificar a forma que o projeto vai ser enviado para o repositório remoto em caso de conflito, vamos manter como padrão.



**PASSO 13:**

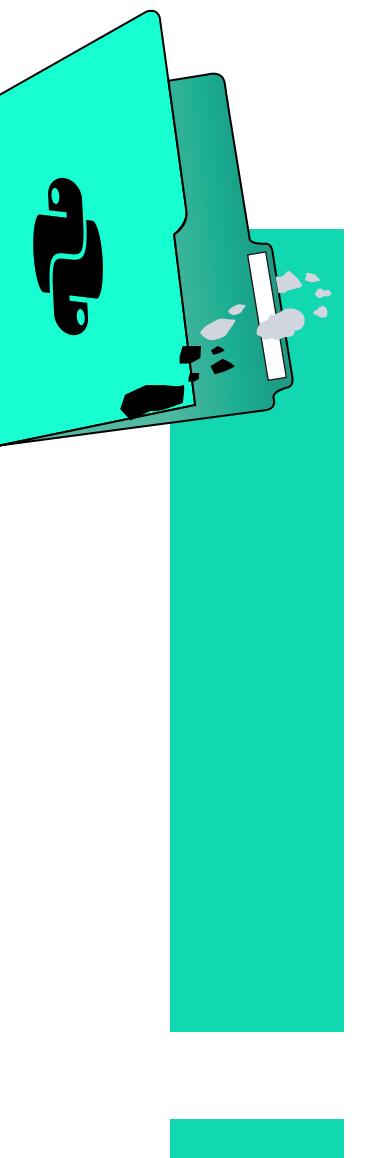
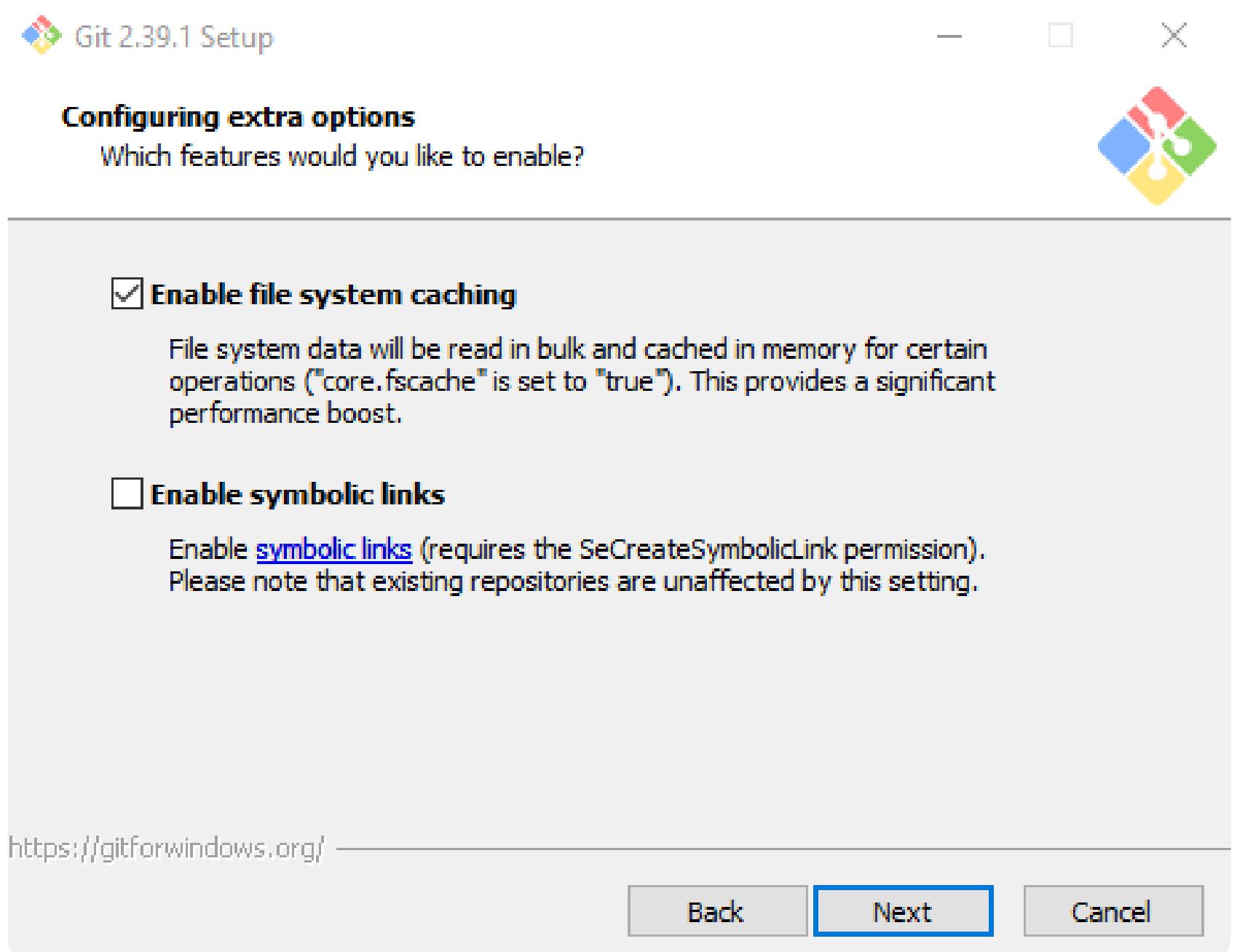
Esse vai especificar se utilizará o Git Credential Manager ou não.

O Git Credential Manager é uma ferramenta útil que facilita o gerenciamento de credenciais de autenticação do Git, tornando as operações mais seguras e mais convenientes.

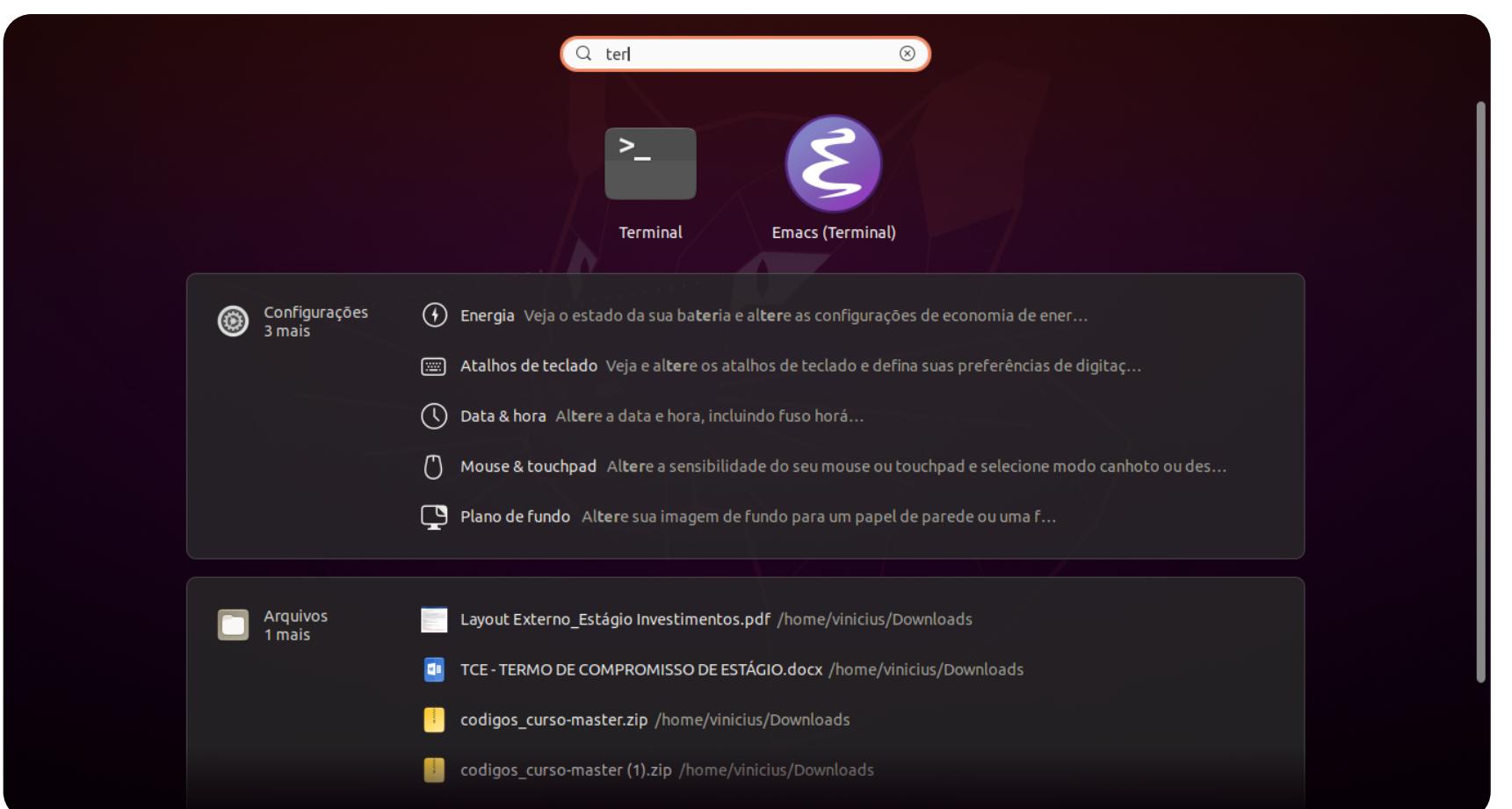


PASSO 14:

O “file system caching” é uma funcionalidade que permite que o sistema operacional mantenha informações recentemente acessadas na memória RAM para que possam ser acessadas rapidamente quando necessária. Isso significa melhora de desempenho.

**4.2. – Instalação do Git no Ubuntu****PASSO 1:**

Abra o terminal através das teclas de atalho Ctrl + Alt + t. Ou, se preferir, vá na tela inicial e digite “terminal”.



PASSO 2:

Digite os seguintes comando no terminal:

Esse primeiro comando é para atualizar a lista de pacotes disponíveis para download e instalação. Sempre que você for instalar alguma coisa no Ubuntu é importante que você utilize esse comando para garantir que sempre terá a última versão sobre os pacotes disponíveis:

```
$ sudo apt-get update
```

Agora, para instalar a biblioteca Git, digite o seguinte comando:

```
$ sudo apt-get install git
```

Agora, verifique a versão instalada do seu git a partir do comando:

```
$ git --version
```



4.3 - Instalação do Git no MAC

Assim como no Linux, é bem fácil a instalação via terminal.

<https://git-scm.com/download/mac>

Acessando o link acima, haverá várias opções para fazer o download do Git.

The screenshot shows the official Git website at <https://git-scm.com/>. The page features the Git logo and the tagline "fast-version-control". A search bar is located in the top right corner. On the left, there's a sidebar with links for "About", "Documentation", "Downloads" (which is currently selected), and "Community". The main content area is titled "Download for macOS" and contains several sections: "Homebrew" (with the command `$ brew install git`), "MacPorts" (with the command `$ sudo port install git`), "Xcode" (noting that Apple ships a binary package of Git with Xcode), "Binary installer" (mentioning an installer provided by Tim Harper, with the version [2.33.0](#) highlighted in red), "Building from Source" (noting tarballs available on [kernel.org](#)), and "Installing git-gui" (mentioning the commit GUI and interactive history browser). A note on the right side of the "Downloads" section states: "There are several options for installing Git on macOS. Note that any non-source distributions are provided by third parties, and may not be up to date with the latest source release." A small note also mentions the availability of the "Pro Git book" online for free.

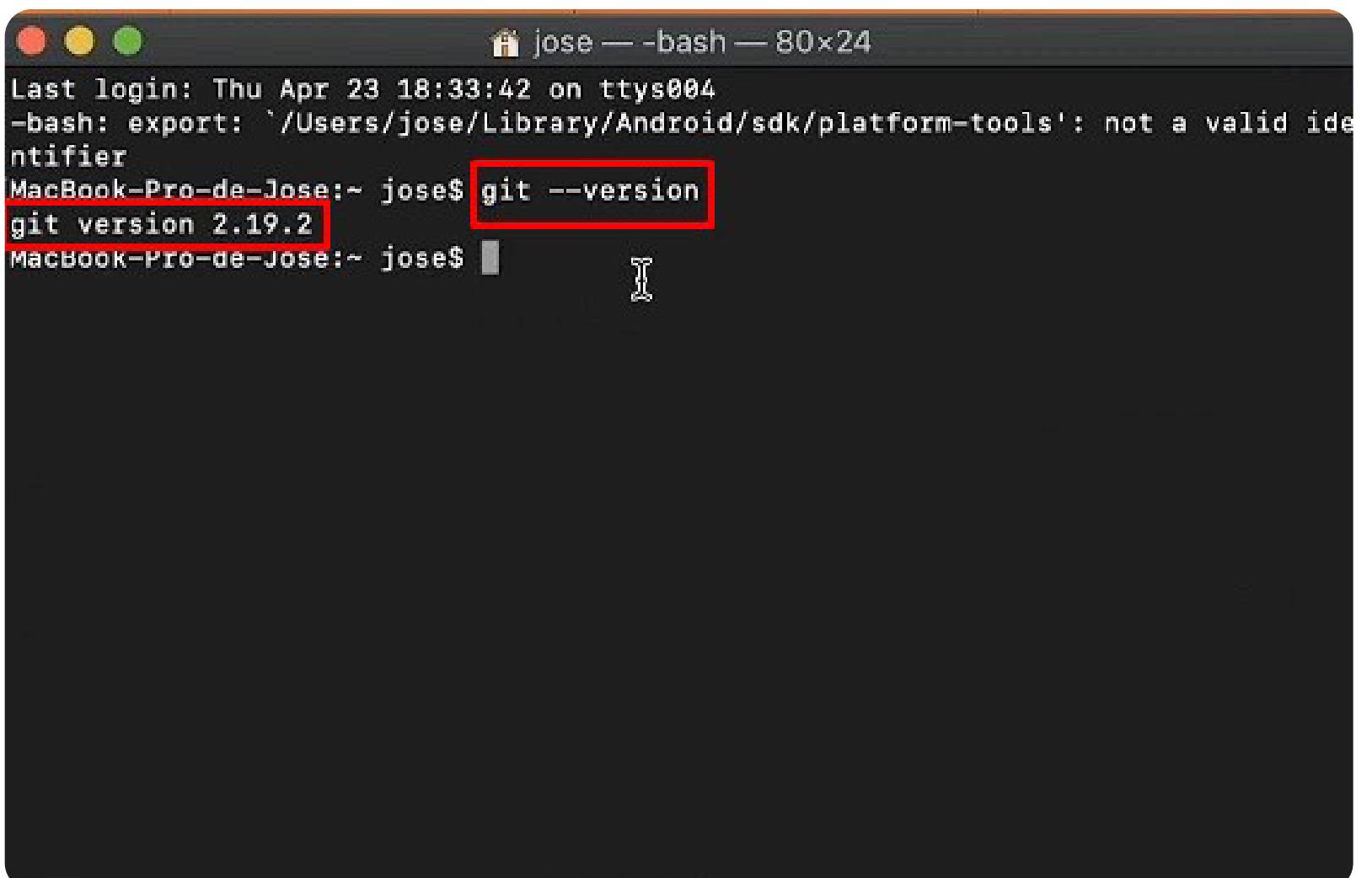


Utilizaremos o “Binary Installer”, clicando no hyperlink da versão.

Faça o download do Git, e depois entre no terminal do Mac e digite o comando:

- git --version

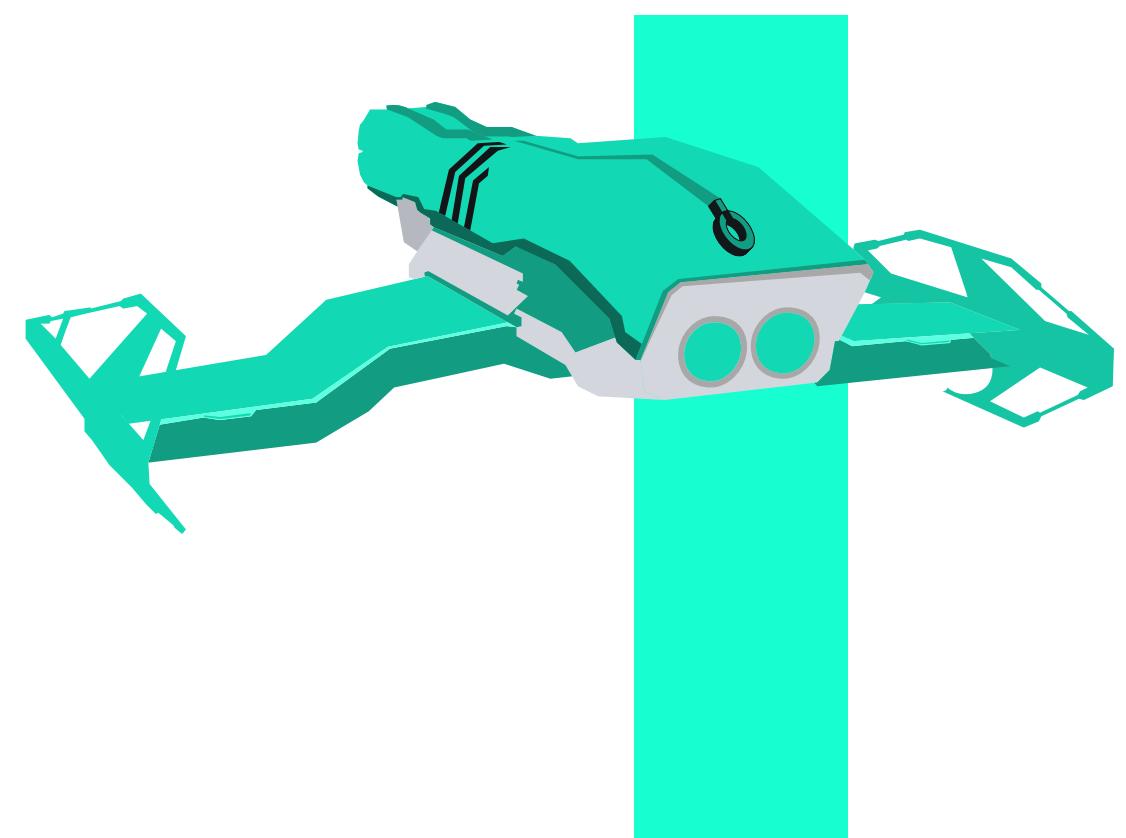
Caso o terminal retorne a versão, o download foi feito com sucesso.



```
jose — -bash — 80x24
Last login: Thu Apr 23 18:33:42 on ttys004
-bash: export: '/Users/jose/Library/Android/sdk/platform-tools': not a valid identifier
MacBook-Pro-de-Jose:~ jose$ git --version
git version 2.19.2
MacBook-Pro-de-Jose:~ jose$
```

4.4. – Configurando o Git e o GitHub

Importante ressaltar que os exemplos abaixo configurando o Git e o GitHub foram feitas via terminal GitBash no Windows, mas que não possui nenhuma diferença se feita no Ubuntu/Mac ([verificar](#)).



4.4.1. – Criando uma chave SSH

Vamos criar uma chave SSH. Com essa chave iremos conectar ao GitHub. E assim conseguiremos fazer a integração entre eles:

PASSO 1:

Digite o seguinte comando no Git Bash:

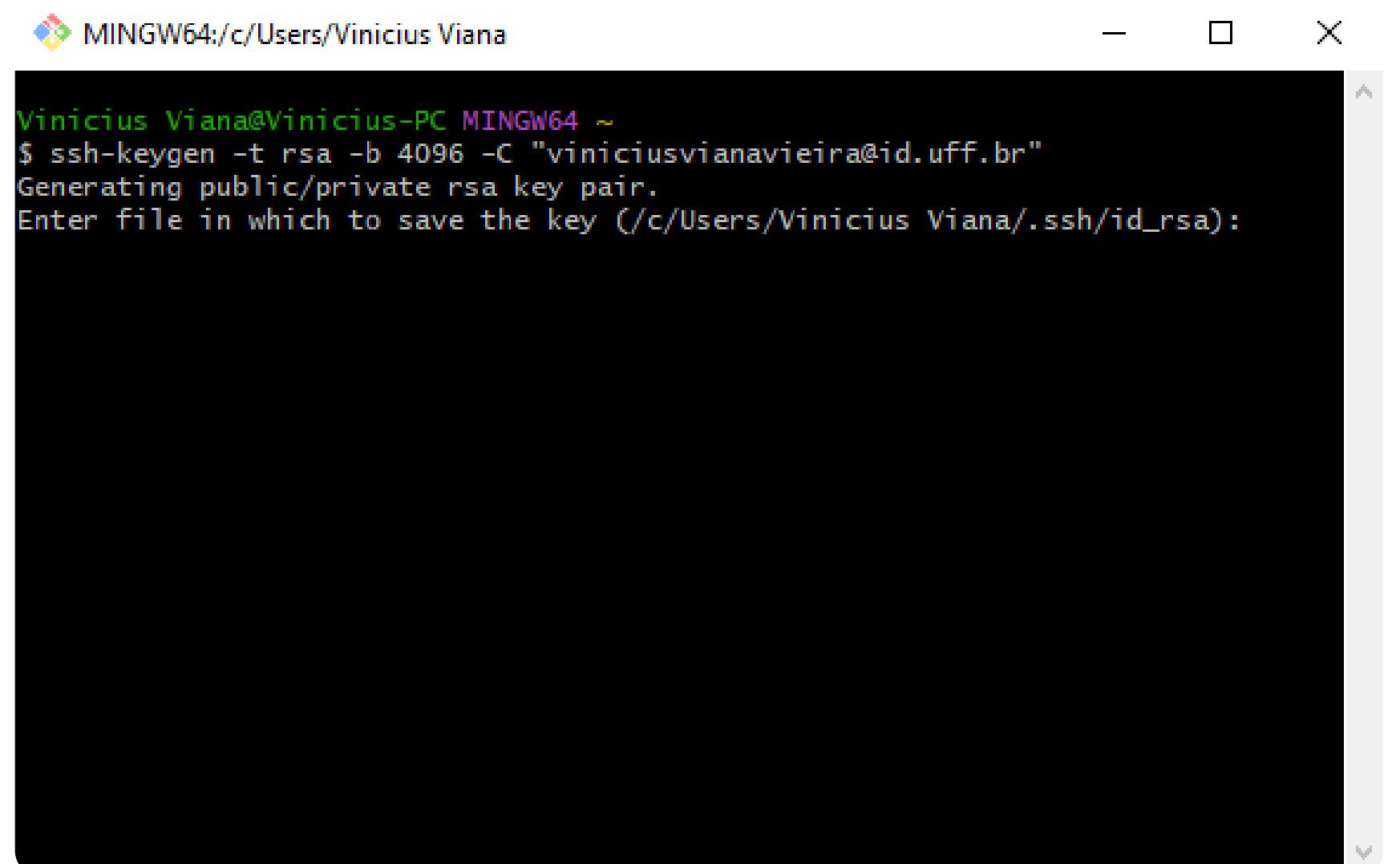
```
ssh-keygen -t rsa -b 4096 -C "seu_email@dominio.com"
```

Não há necessidade de gravar esses comandos, mas é bom ter conhecimento:

- Queremos gerar uma chave: “ssh-keygen”
- Tipo de criptografia: “-t rsa”
- Força da criptografia: “-b 4096”
- Email do GitHub: “-C “seu email”

PASSO 2:

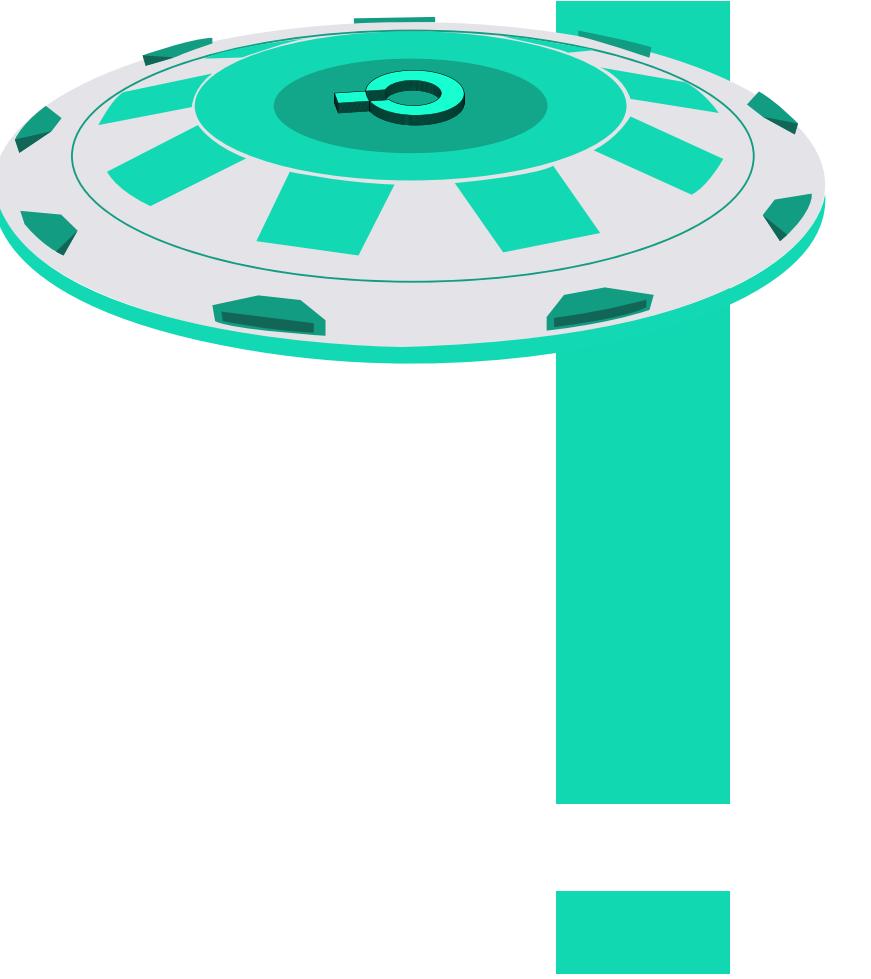
Você quer salvar nesta pasta de arquivo, só apertar enter.



A screenshot of a terminal window titled "MINGW64:/c/Users/Vinicio Viana". The command \$ ssh-keygen -t rsa -b 4096 -C "viniciusvianavieira@id.uff.br" is being typed. The terminal then displays the message "Generating public/private rsa key pair." followed by "Enter file in which to save the key (/c/Users/Vinicio Viana/.ssh/id_rsa):". A large red rectangular box covers the bottom portion of the terminal window, obscuring the key pair generation progress.

PASSO 3:

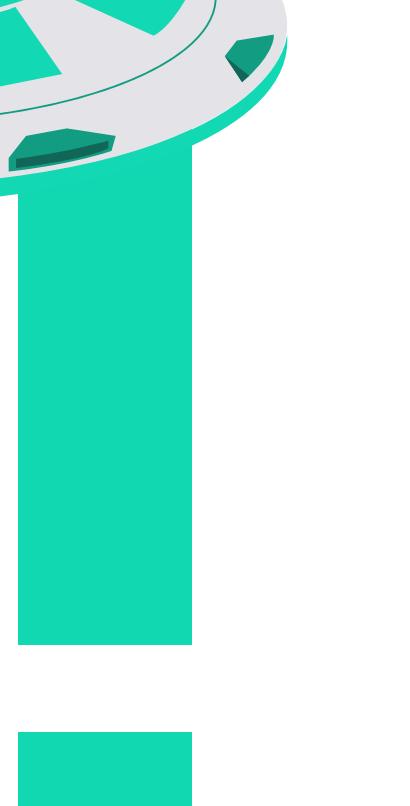
Será solicitada uma senha, e guarde-a em algum lugar seguro. Há opção de deixar sem, a chave SSH já é um ótima segurança, essa senha é mais uma camada de segurança. Essa senha será solicitada sempre que você precisar realizar operações de download ou envio no repositório.



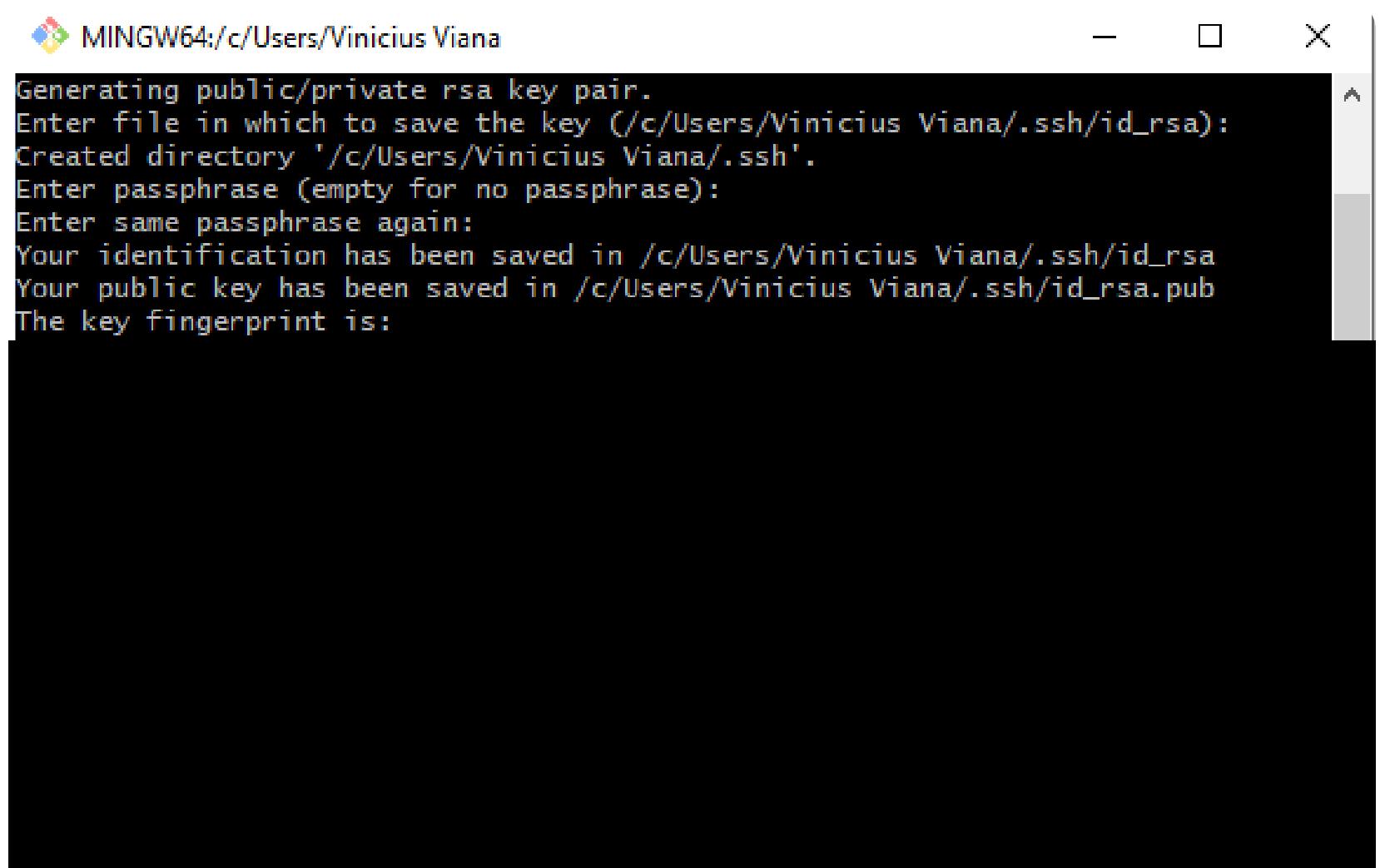
```
MINGW64:/c/Users/Vinicius Viana
-
X
Vinicius Viana@Vinicius-PC MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "viniciusvianavieira@id.uff.br"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Vinicius Viana/.ssh/id_rsa):
Created directory '/c/Users/Vinicius Viana/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again: |
```

PASSO 4:

Confirmação da senha.



```
MINGW64:/c/Users/Vinicius Viana
-
X
Vinicius Viana@Vinicius-PC MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "viniciusvianavieira@id.uff.br"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Vinicius Viana/.ssh/id_rsa):
Created directory '/c/Users/Vinicius Viana/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again: |
```

PASSO 5:

```
MINGW64:/c/Users/Vinicius Viana
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Vinicius Viana/.ssh/id_rsa):
Created directory '/c/Users/Vinicius Viana/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Vinicius Viana/.ssh/id_rsa
Your public key has been saved in /c/Users/Vinicius Viana/.ssh/id_rsa.pub
The key fingerprint is:
```

Após a criação da chave, o Git nos retorna:

1. Chave privada: id_rsa
2. Chave pública: id_rsa.pub

4.4.2. – Configurando a chave SSH**PASSO 1:**

Digite o seguinte comando no Git Bash:

```
cd ~/.ssh/
```



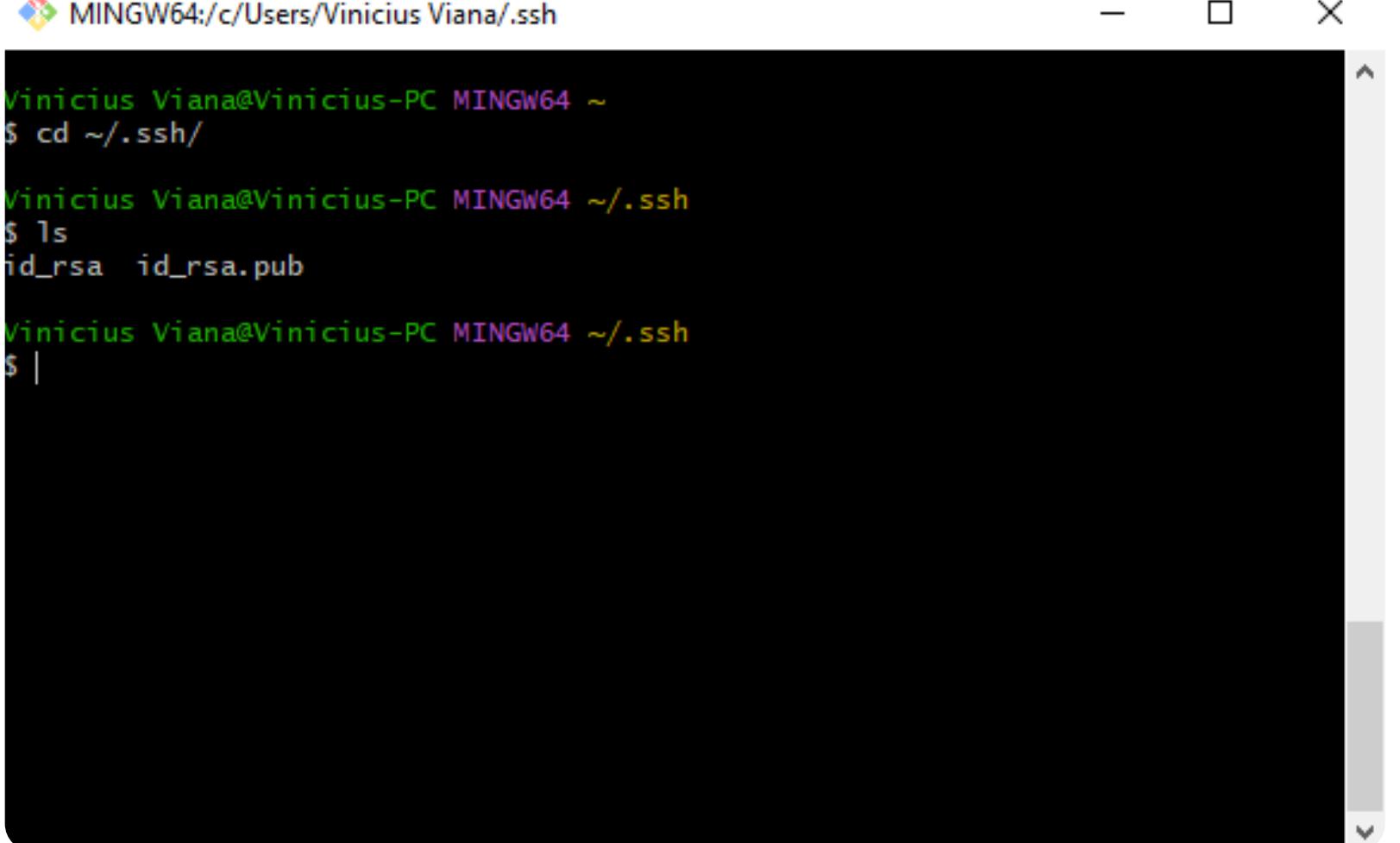
```
Vinicius Viana@Vinicius-PC MINGW64 ~
$ cd ~/.ssh/
Vinicius Viana@Vinicius-PC MINGW64 ~/ssh
$
```

PASSO 2:

Para conferir os itens na pasta, digite o seguinte comando no terminal:

```
ls
```

Logo após, deverá aparecer suas chaves SSH:

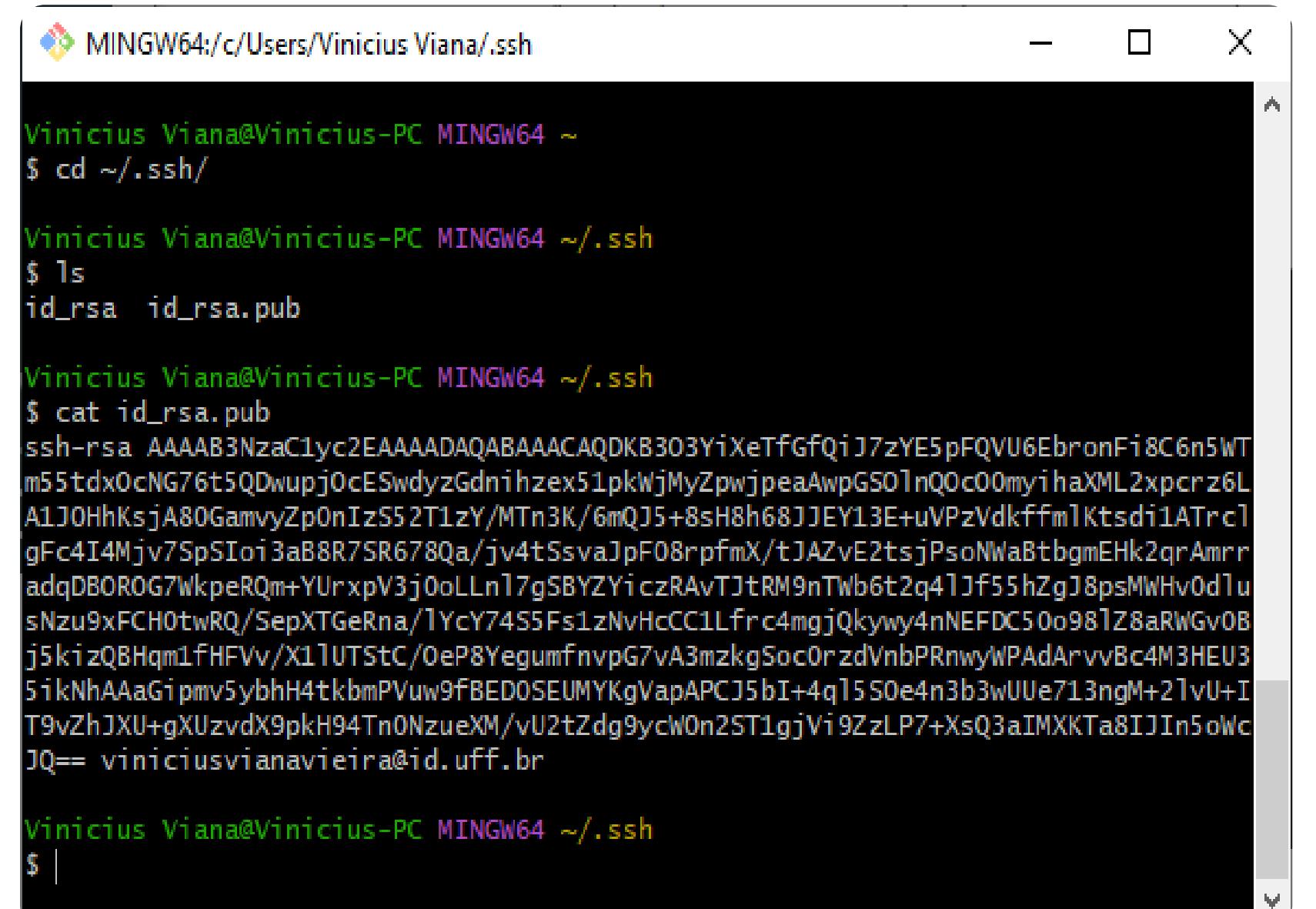


```
Vinicius Viana@Vinicius-PC MINGW64 ~
$ cd ~/.ssh/
Vinicius Viana@Vinicius-PC MINGW64 ~/ssh
$ ls
id_rsa  id_rsa.pub
```

PASSO 3:

Para pegar o valor da sua chave SSH pública, que será integrada ao GitHub, digite o seguinte comando:

- cat id_rsa.pub



```
Vinicius Viana@Vinicius-PC MINGW64 ~
$ cd ~/.ssh/
$ ls
id_rsa  id_rsa.pub
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDKB303YiXeTfGfQiJ7zYE5pFQVU6EbronFi8C6n5WT
m55tdx0cNG76t5QDwupj0cESwdyzGdnihzex51pkWjMyZpwjpeaAwpGS01nQ0c00myihaXML2xpcrz6L
A1J0HhKsjA80GamvyZp0nIz552T1zY/MTn3K/6mQJ5+8sH8h68JJYE13E+uVPzVdkffmlKtsdi1ATrc1
gFc4I4Mjv7SpSIOi3aB8R7SR678Qa/jv4tSsvaJpF08rpfmX/tJAZvE2tsjPsoNWaBtbgmEHk2qrAmrr
adqDBOROG7WkpeRQm+YUrxpV3j0oLLn17gSBYZYiczRAvTJtRM9nTwb6t2q41Jf55hZgJ8psMWhv0dJu
sNzu9xFCH0twRQ/SepXTGeRna/1YcY7455Fs1zNvHcCC1lfrc4mgjQkywy4nNEFDC50o981Z8aRWGv0B
j5kizQ8Hqm1fHFVv/X11UTStC/OeP8YegumfnvpG7vA3mzkg5ac0rzdVnbPRnwypAdArvvBc4M3HEU3
5ikNhAAaGipmv5ybhH4tkbmPVuw9fBED0SEUMYKgVapAPCJ5bI+4q15S0e4n3b3wUUe713ngM+21vU+I
T9vZhJXU+gXUzvdX9pkH94TnONzueXM/vU2tZdg9ycW0n2ST1gjVi9ZzLP7+xsQ3aIMXKTa8IJIn5oWc
JQ== viniciusvianavieira@id.uff.br
```

Caso o comando não funcione, teste um desses códigos:

- more id_rsa.pub
- OU
- ```
vi id_rsa.pub
```

No último comando, estará acessando o editor de texto. Portanto, cole a chave e depois para sair, digite:

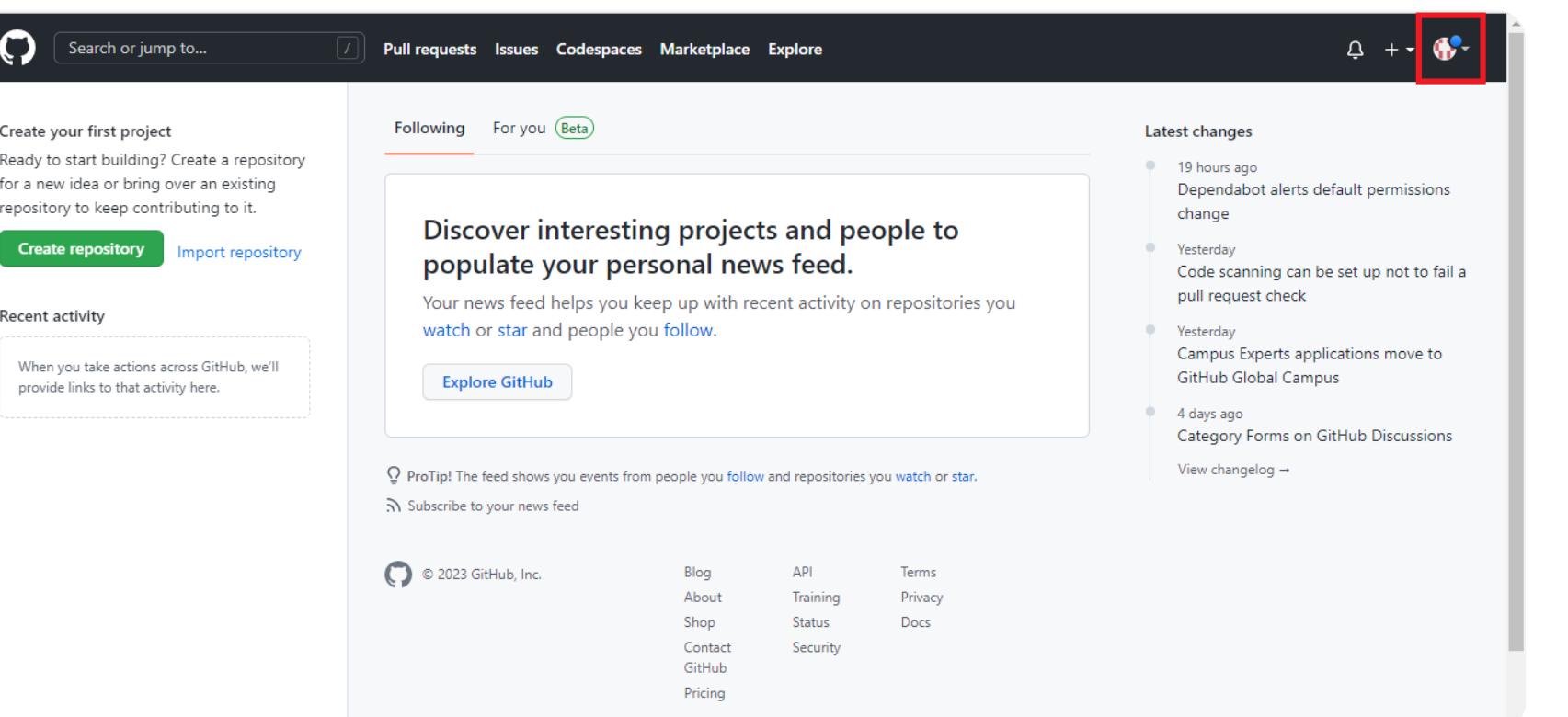
- [esc]:wq

## PASSO 4:

Depois que copiar a chave, entre no GitHub e siga os próximos passos:

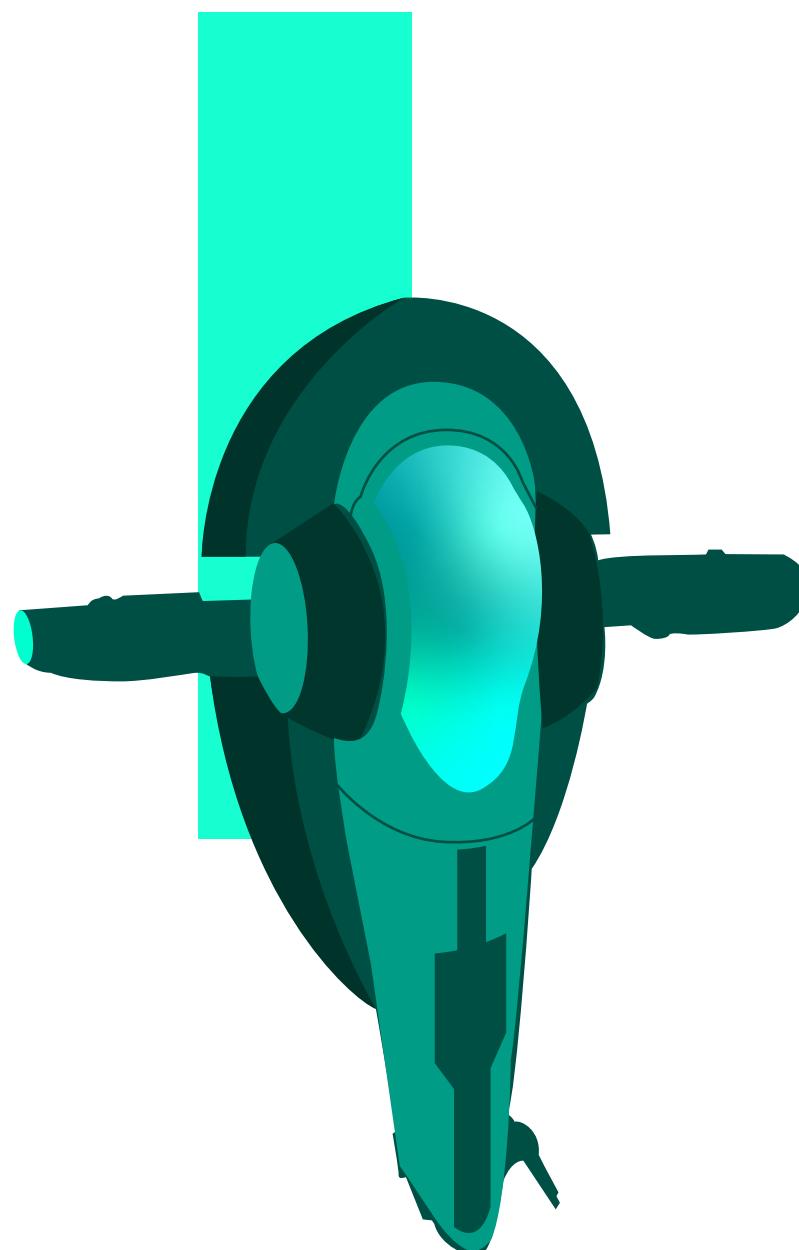
### PASSO 4.1:

Clique no perfil:



**PASSO 4.2:**

Clique em “settings”:



The screenshot shows the GitHub homepage. At the top right, a user dropdown menu is open, showing the signed-in user's name and a list of options. The 'Settings' option is highlighted with a red box. The main content area displays the 'Create your first project' section and a 'Recent activity' summary.

**PASSO 4.3:**

Clique em “SSH and GPG Keys”:

This screenshot shows the GitHub Public profile settings page for the user 'viniciusvianavieirauff'. The left sidebar contains various account management sections like Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys (which is highlighted with a red box), Organizations, and Moderation. The main area displays the Public profile settings, including fields for Name, Profile picture, Public email, Bio, and URL.

**PASSO 4.4:**

Clique em “New v Key”:

This screenshot shows the GitHub SSH keys settings page for the same user. The left sidebar includes Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys (highlighted with a red box), Organizations, and Moderation. The main content area shows the SSH keys section, which currently has no keys associated with the account. It also includes sections for GPG keys and Vigilant mode, with a 'New SSH key' button highlighted with a red box.

**PASSO 4.5:**

Coloque as seguintes informações e depois confirme.

- Title: [nome da chave - sua escolha -]

Key type: Authentication key

Key: Chave id\_rsa.pub obtida a partir do Git Bash terminal

The screenshot shows the GitHub user profile page for 'viniciusvianavieirauff'. The left sidebar has sections like 'Public profile', 'Account', 'Appearance', 'Accessibility', 'Notifications', 'Access', 'Billing and plans', 'Emails', 'Password and authentication', 'Sessions', 'SSH and GPG keys' (which is selected), 'Organizations', and 'Moderation'. The main content area is titled 'SSH keys / Add new'. It has fields for 'Title' (set to 'Chave SSH Windows'), 'Key type' (set to 'Authentication Key'), and a large text area for 'Key' which is redacted. At the bottom is a red button labeled 'Add SSH key'.

**PASSO 4.6:**

Confirme sua identidade



## Confirm access

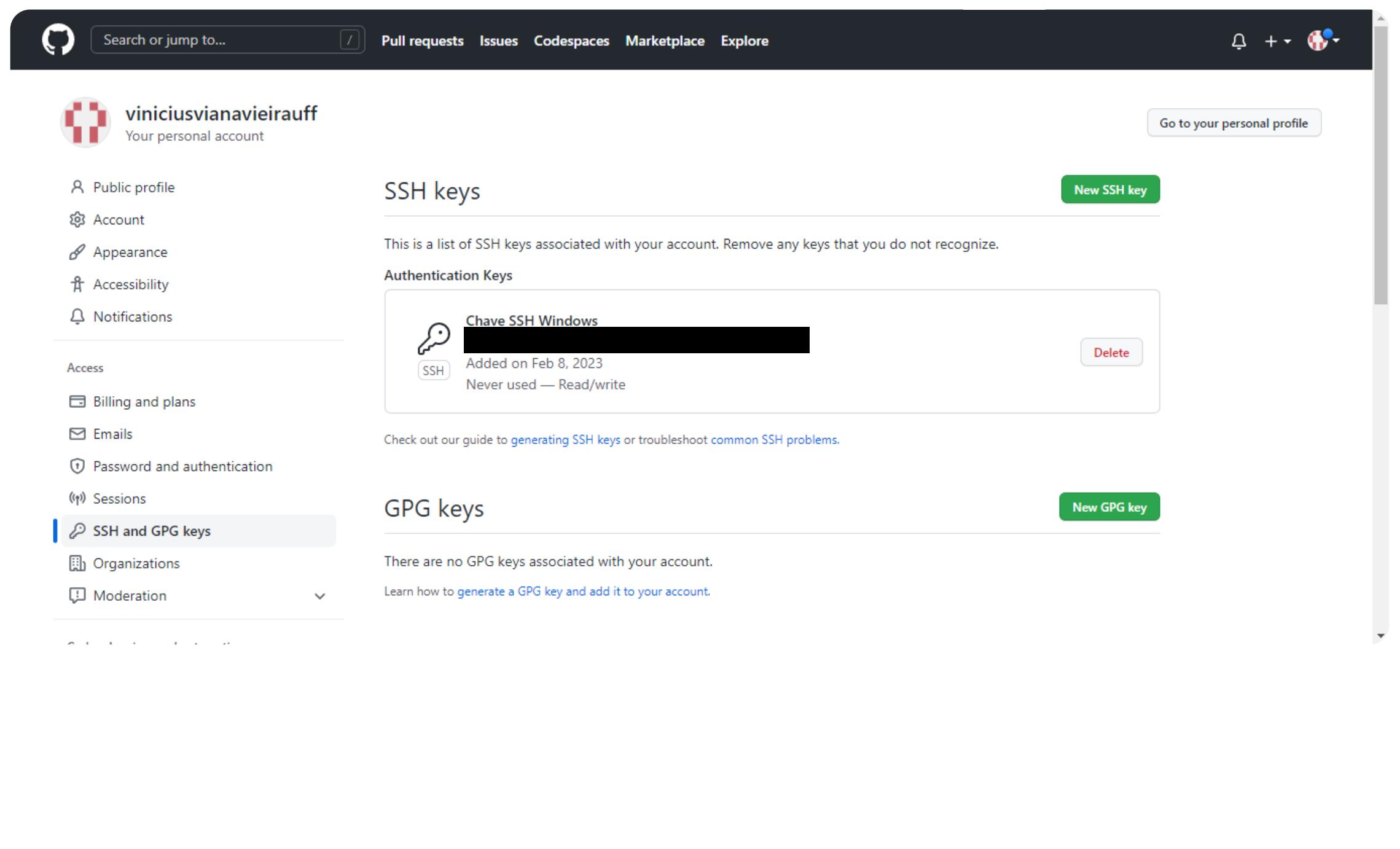
A modal dialog box titled 'Confirm access' contains a 'Password' field with masked input, a 'Forgot password?' link, and a large green 'Confirm' button.

Tip: You are entering **sudo mode**. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.

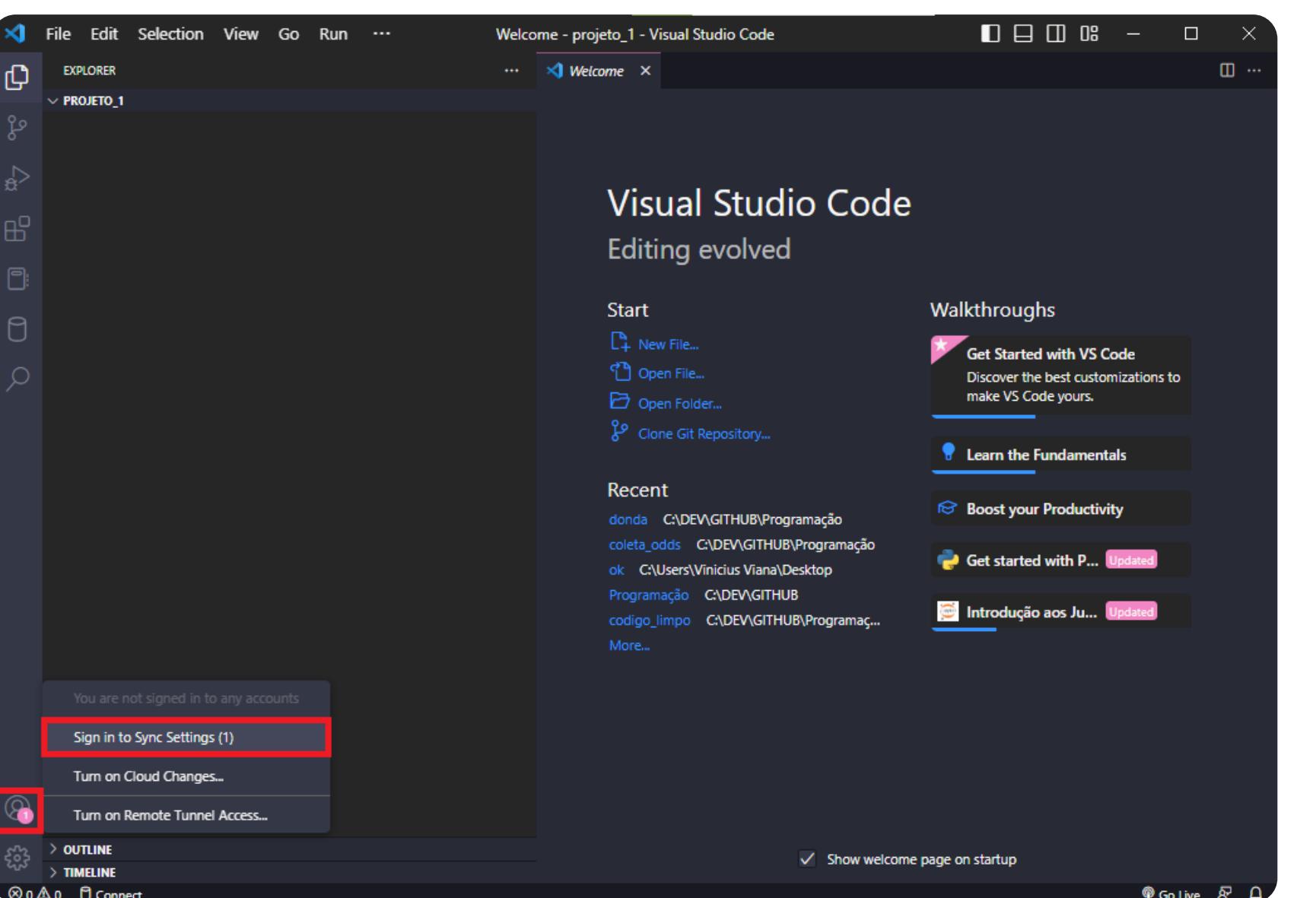
[Terms](#) [Privacy](#) [Security](#) [Contact GitHub](#)

**PASSO 4.7:**

Finalizado. Chave SSH conectada ao GitHub.

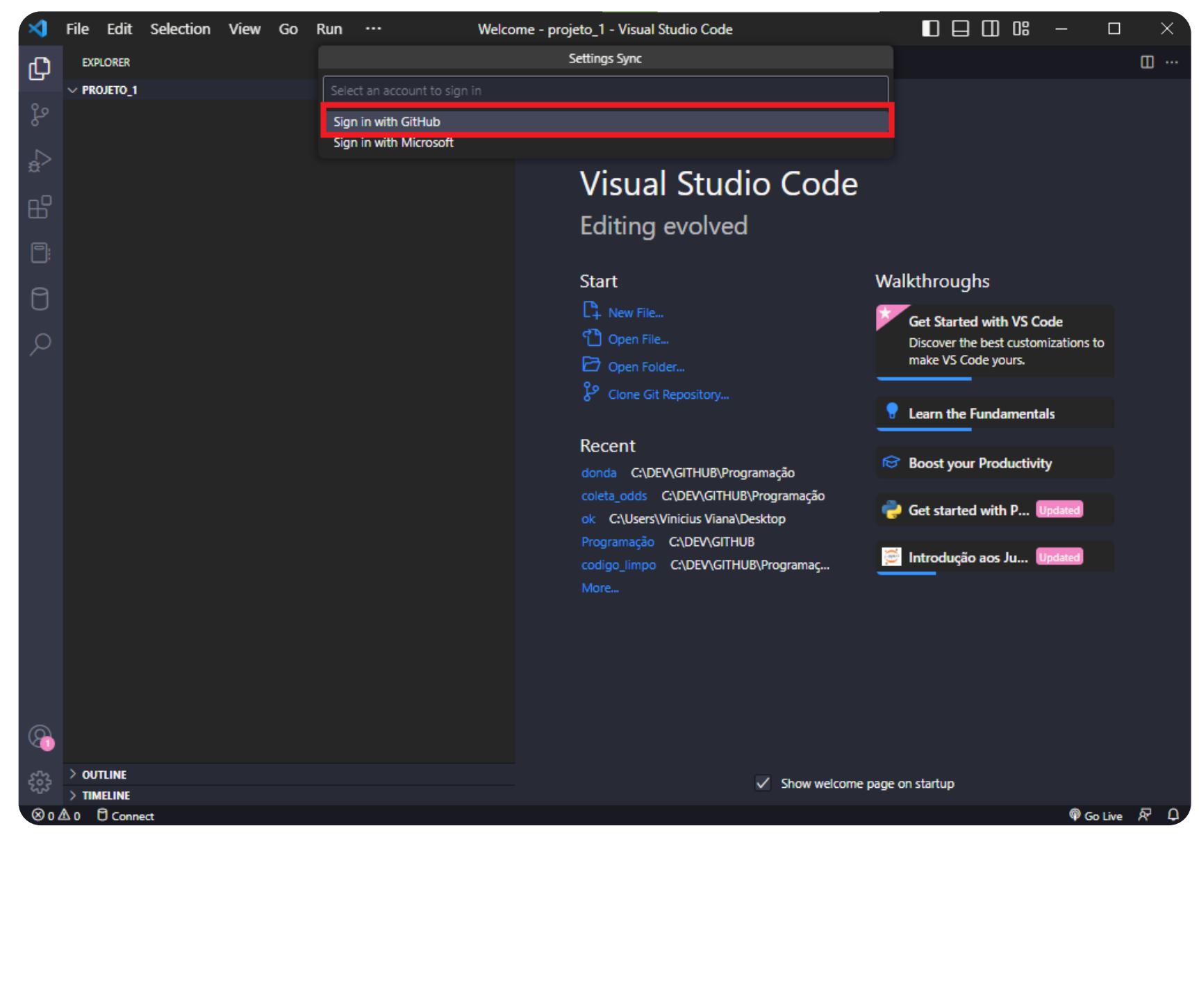
**4.4.3. – Configurando o VSCode e conectando ao GitHub****PASSO 1:**

Na tela principal do VSCode clique no seu perfil e depois clique em "Sign in to sync Settings(1)"

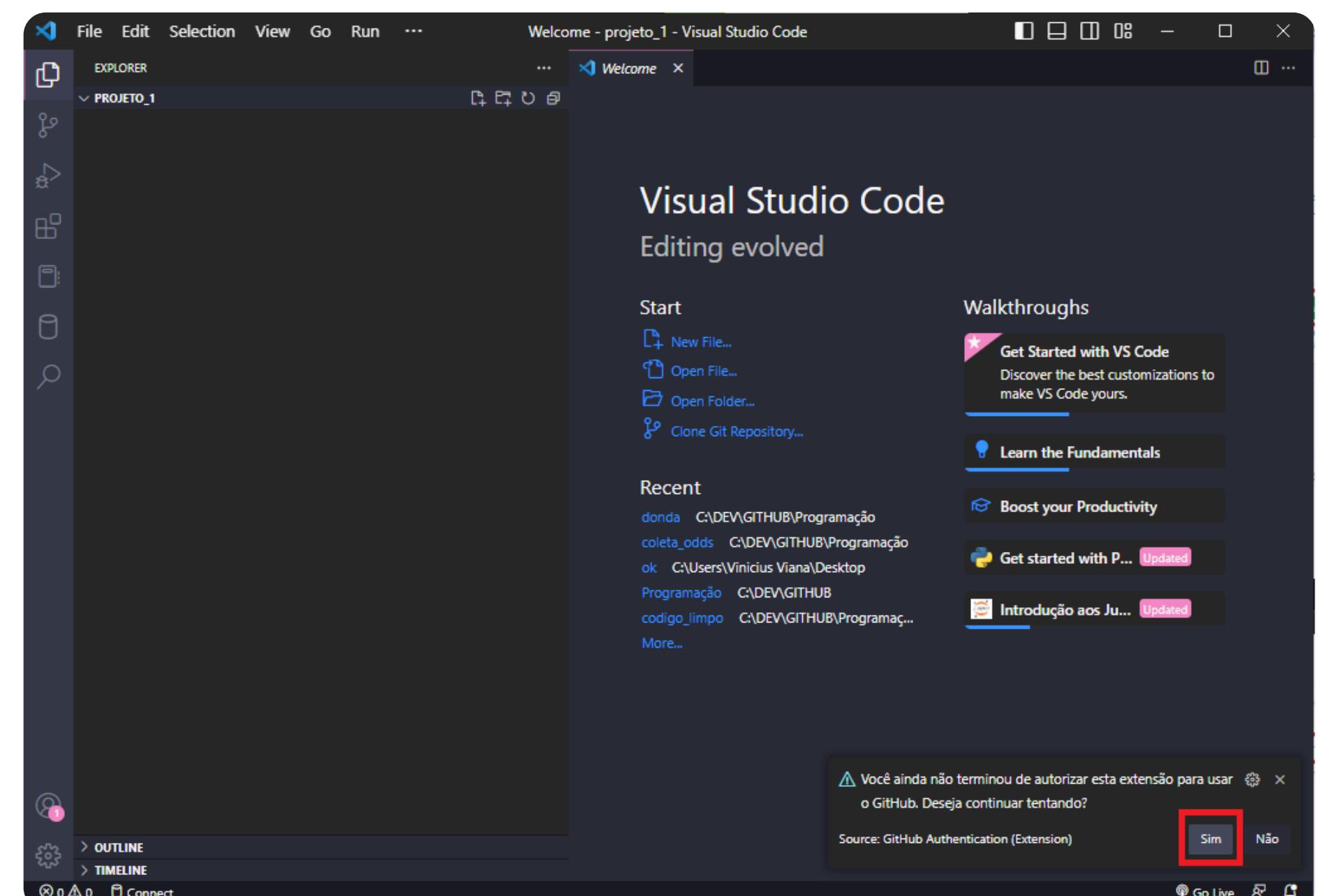


**PASSO 2:**

Aparecerá algumas opções, escolha a “Sign in with GitHub”

**PASSO 3:**

Ele pedirá as credenciais do GitHub. Forneça-o e clique para autorizar a integração com o VSCode e depois retorne. No VSCode, no canto direito inferior, confirme a autorização da integração.



Pronto, agora o seu VSCode está integrado com o GitHub. O que gera muito mais facilidade na hora de importação e exportação de projetos.

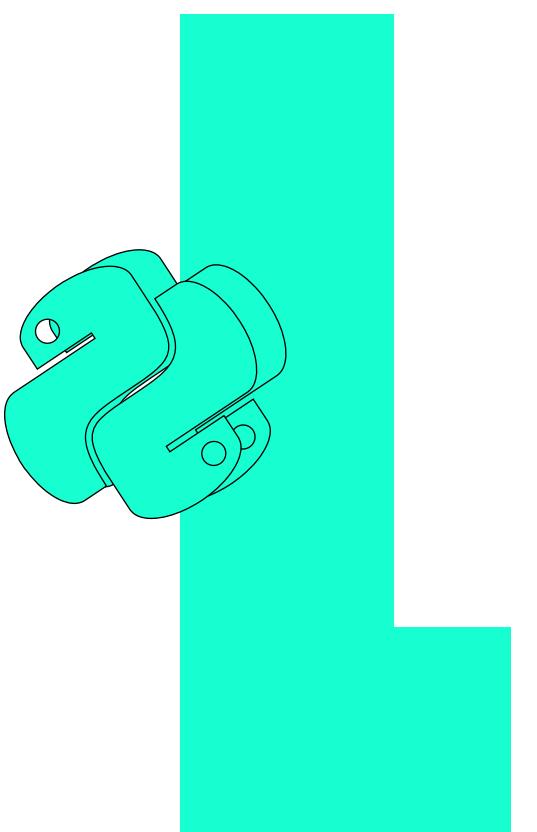
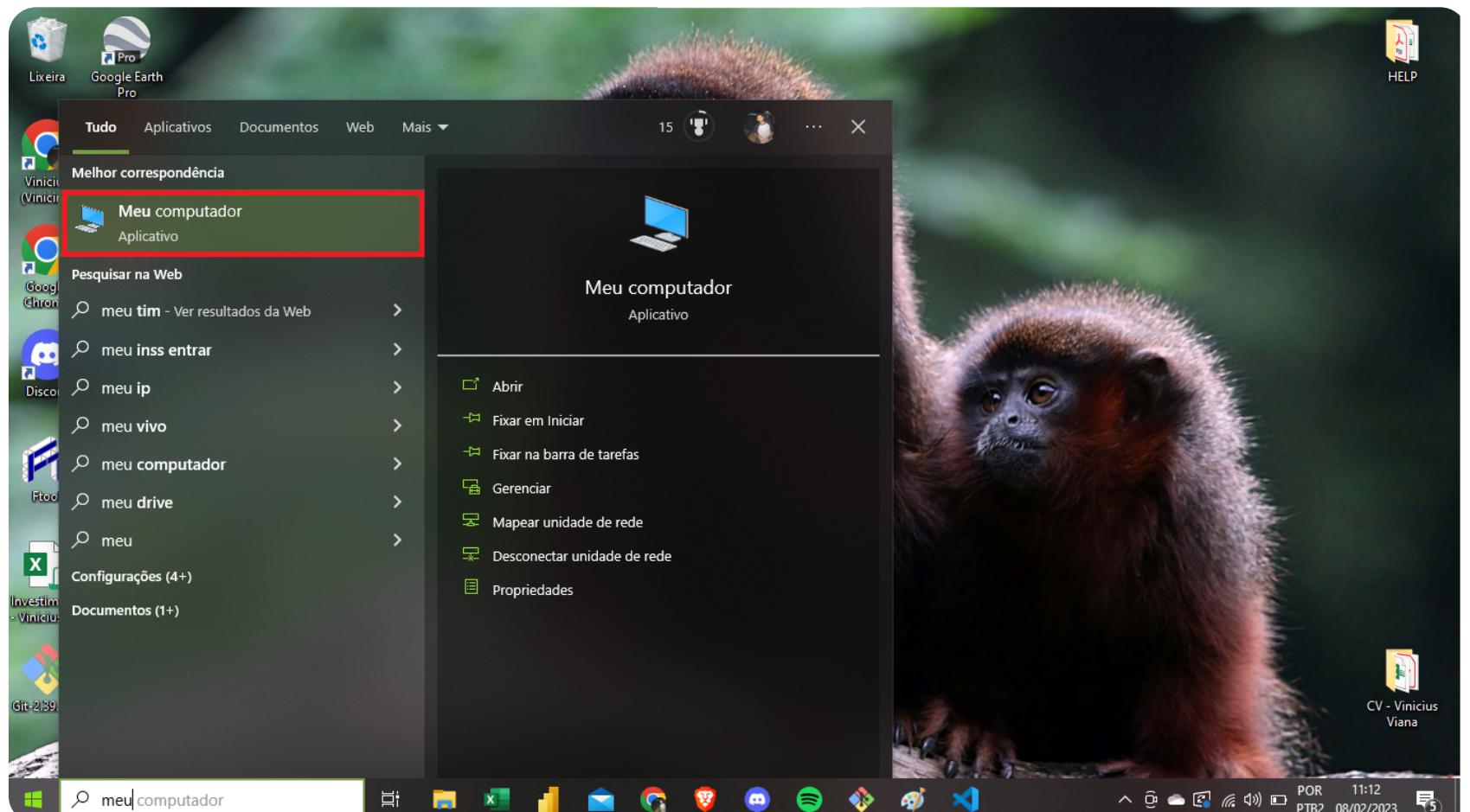
## Mundo 5

### 5.1. – Organizando as pastas no computador

Antes de começar um novo projeto e configurar o VSCode, é importante realizar uma pré-configuração no computador para garantir um ambiente bem organizado.

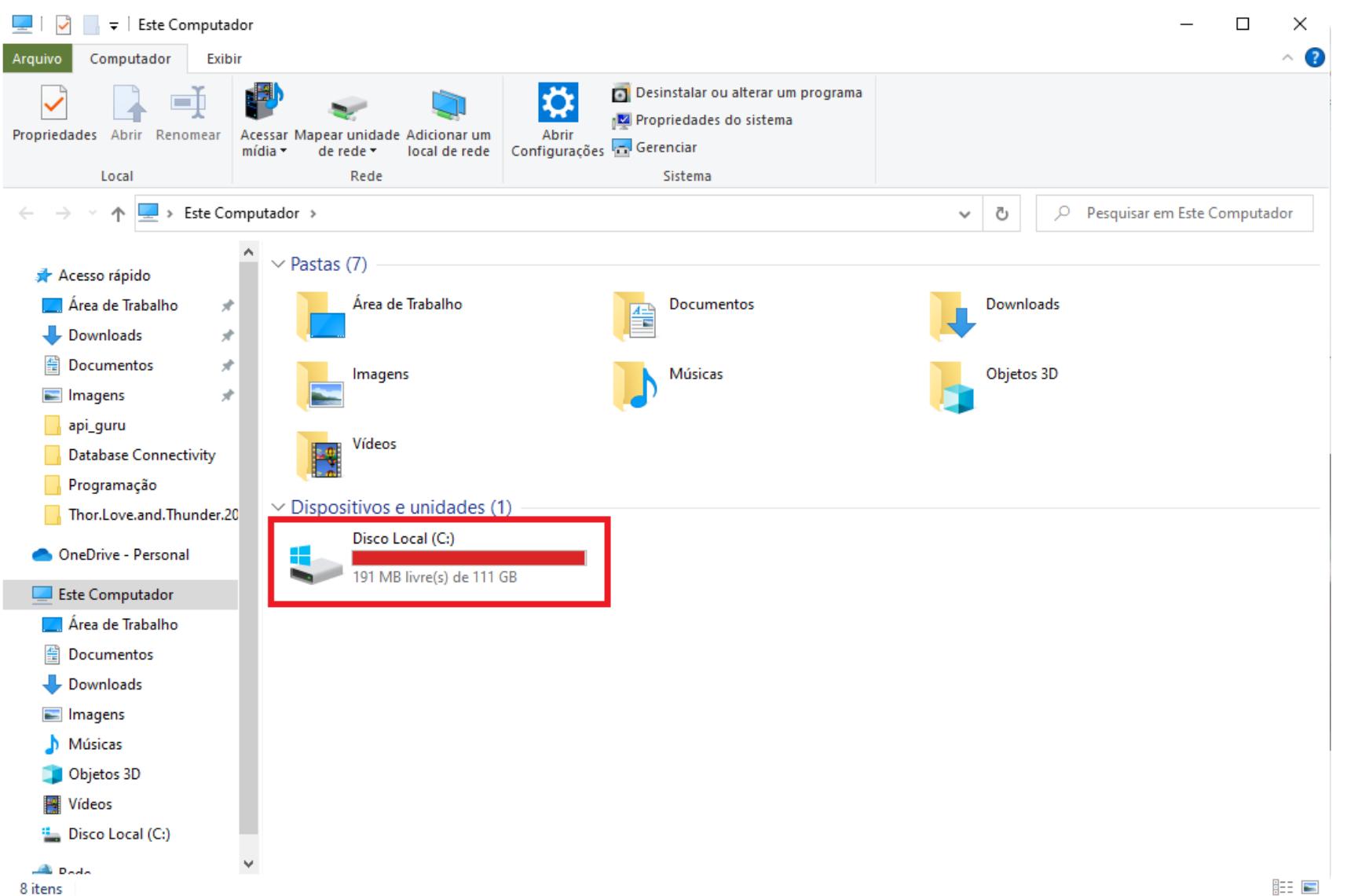
#### PASSO 1:

Vá em "Meu Computador"



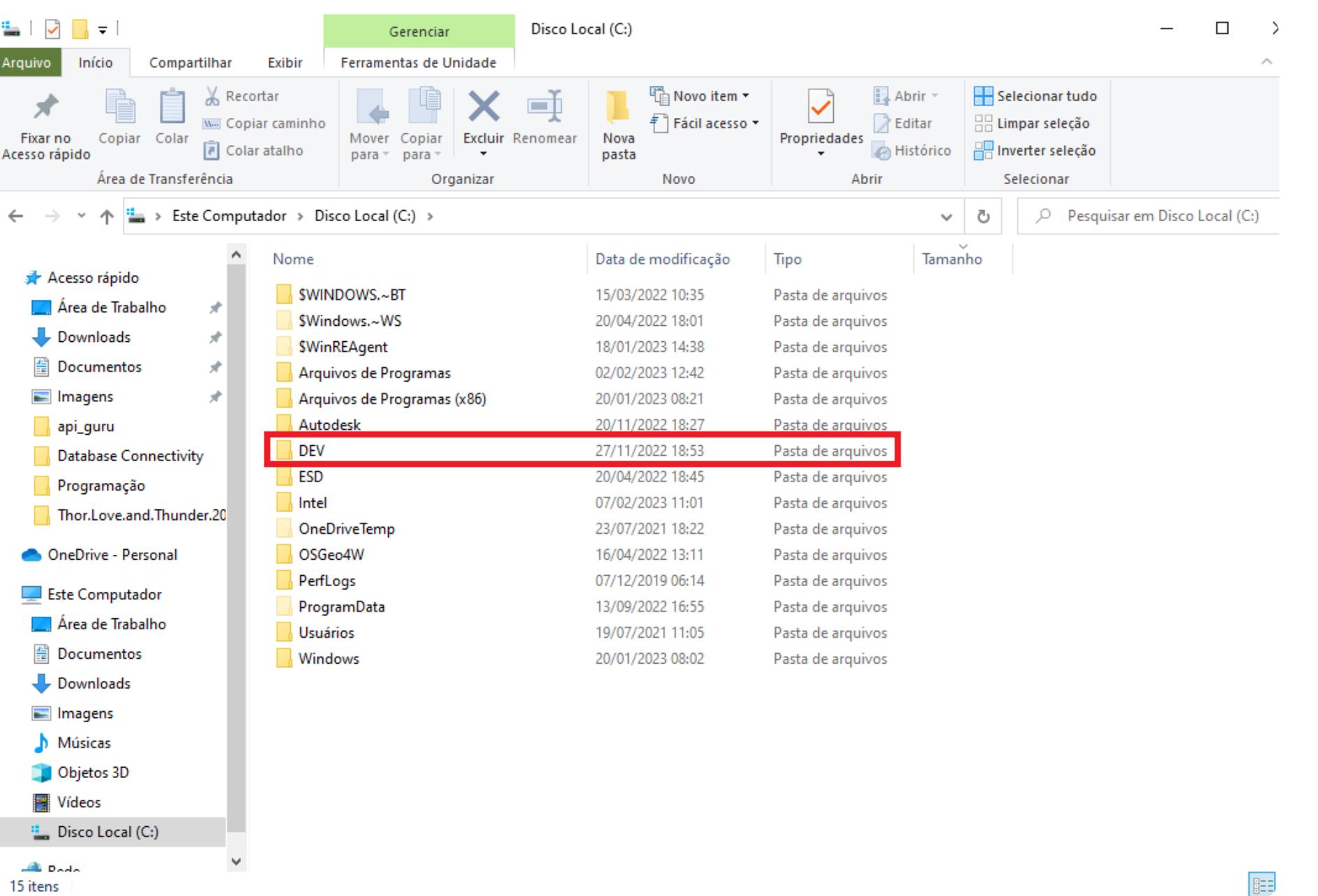
#### PASSO 2:

Escolha seu dispositivo de armazenamento, no nosso caso "Disco Local(C:)".

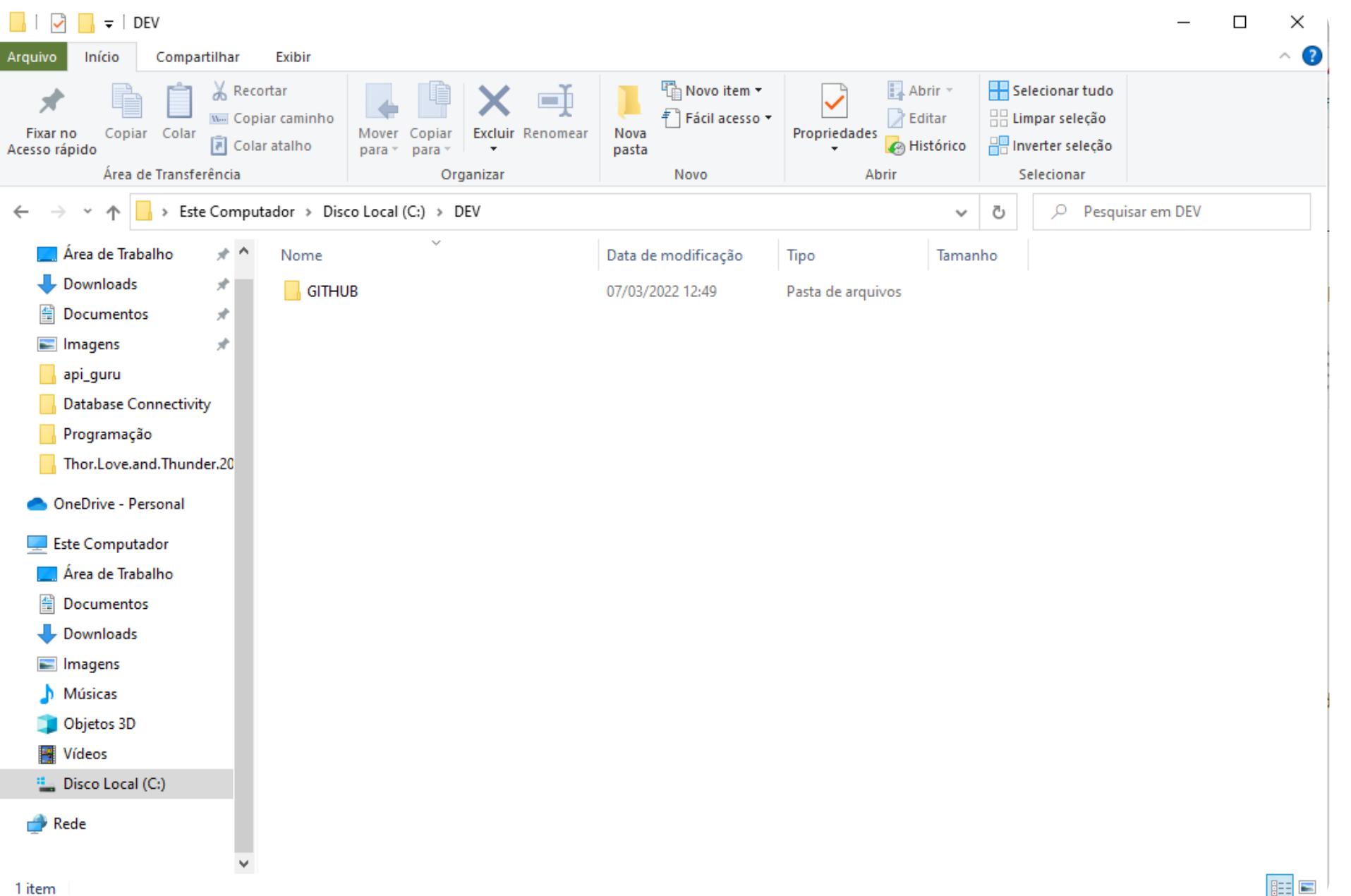


**PASSO 3:**

Crie uma pasta voltada para guardar seus projetos e arquivos relacionados à programação. No nosso caso, nomeamos de "DEV" (Desenvolvedor).

**PASSO 4:**

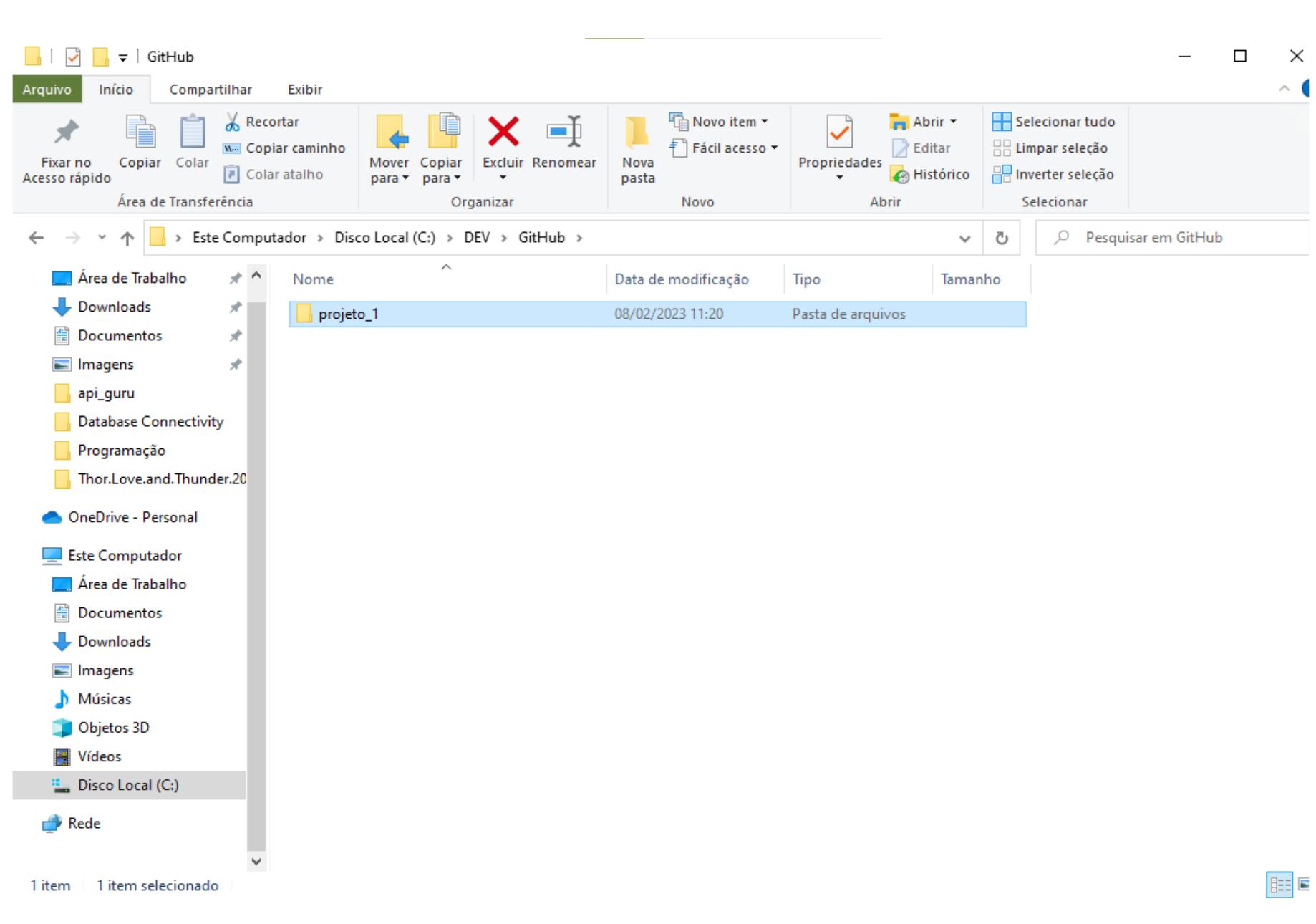
Depois, crie uma nova pasta dentro da pasta "DEV", nomeada de "GitHub", onde ficarão todos os seus projetos, a fim de facilitar a organização.



## 5.2. – Criando um projeto novo no VSCode

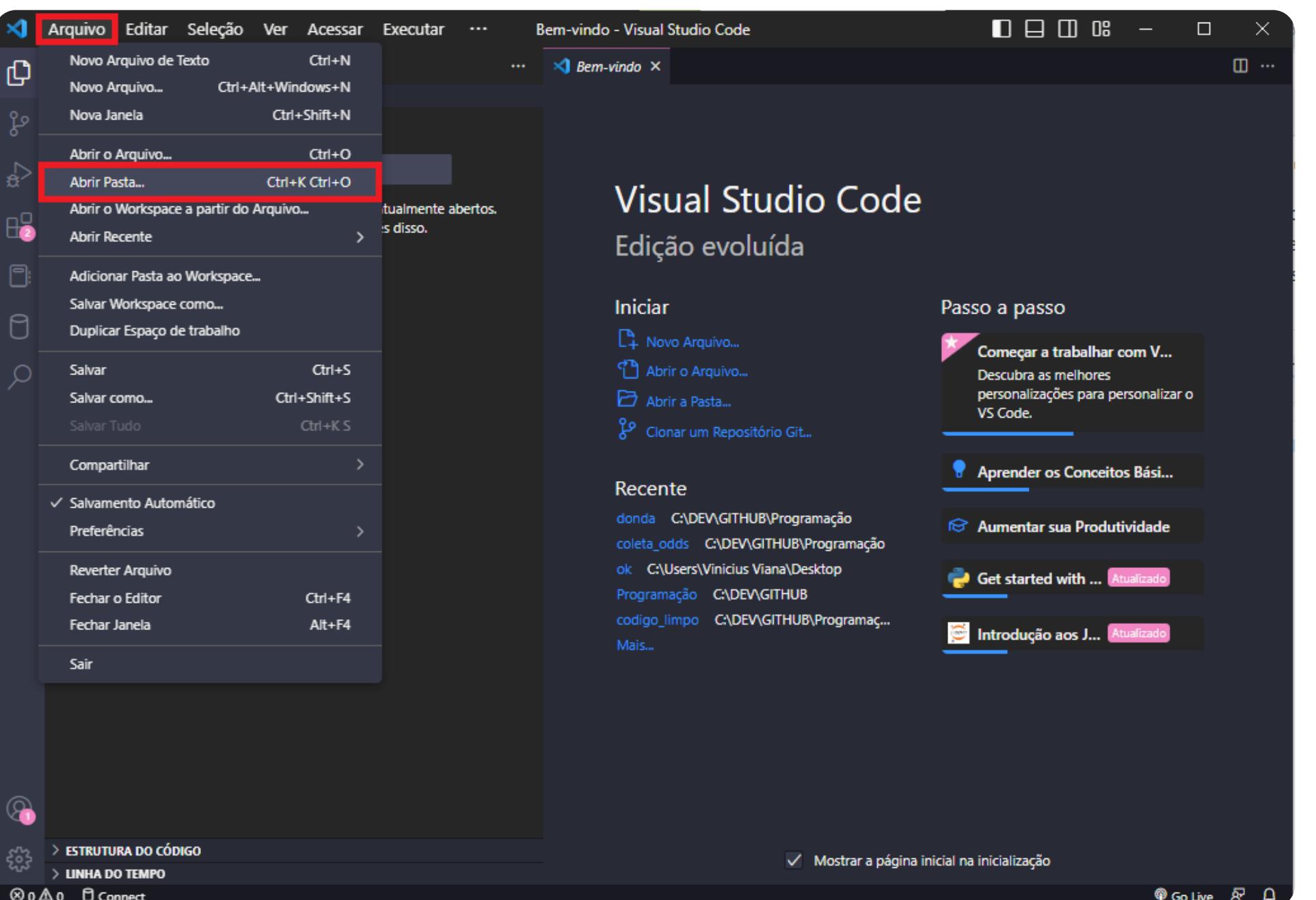
### PASSO 1:

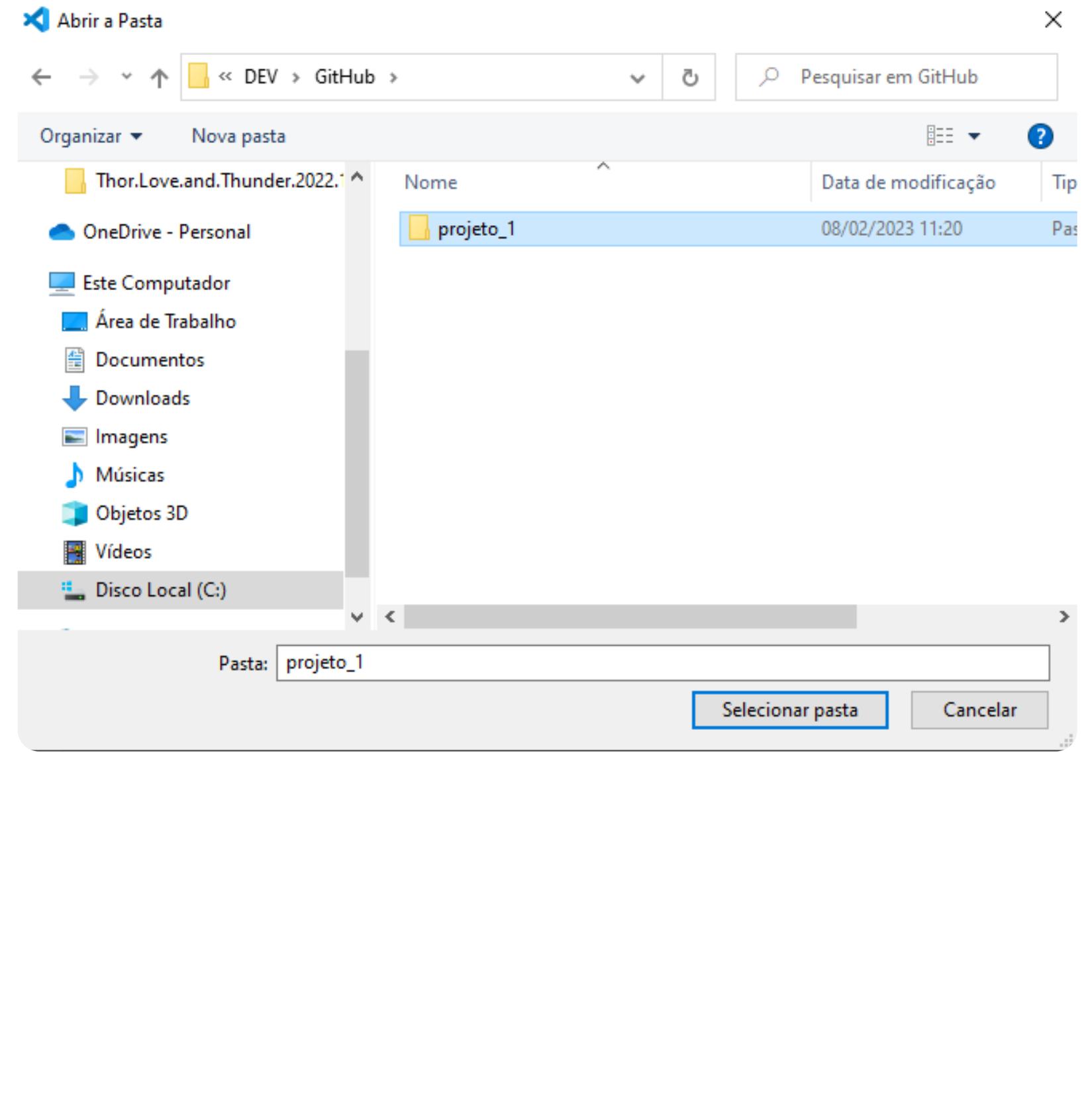
Dentro da pasta recém-criada, chamada "GitHub", criaremos uma nova pasta com o nome do projeto que pretendemos upar para o GitHub.



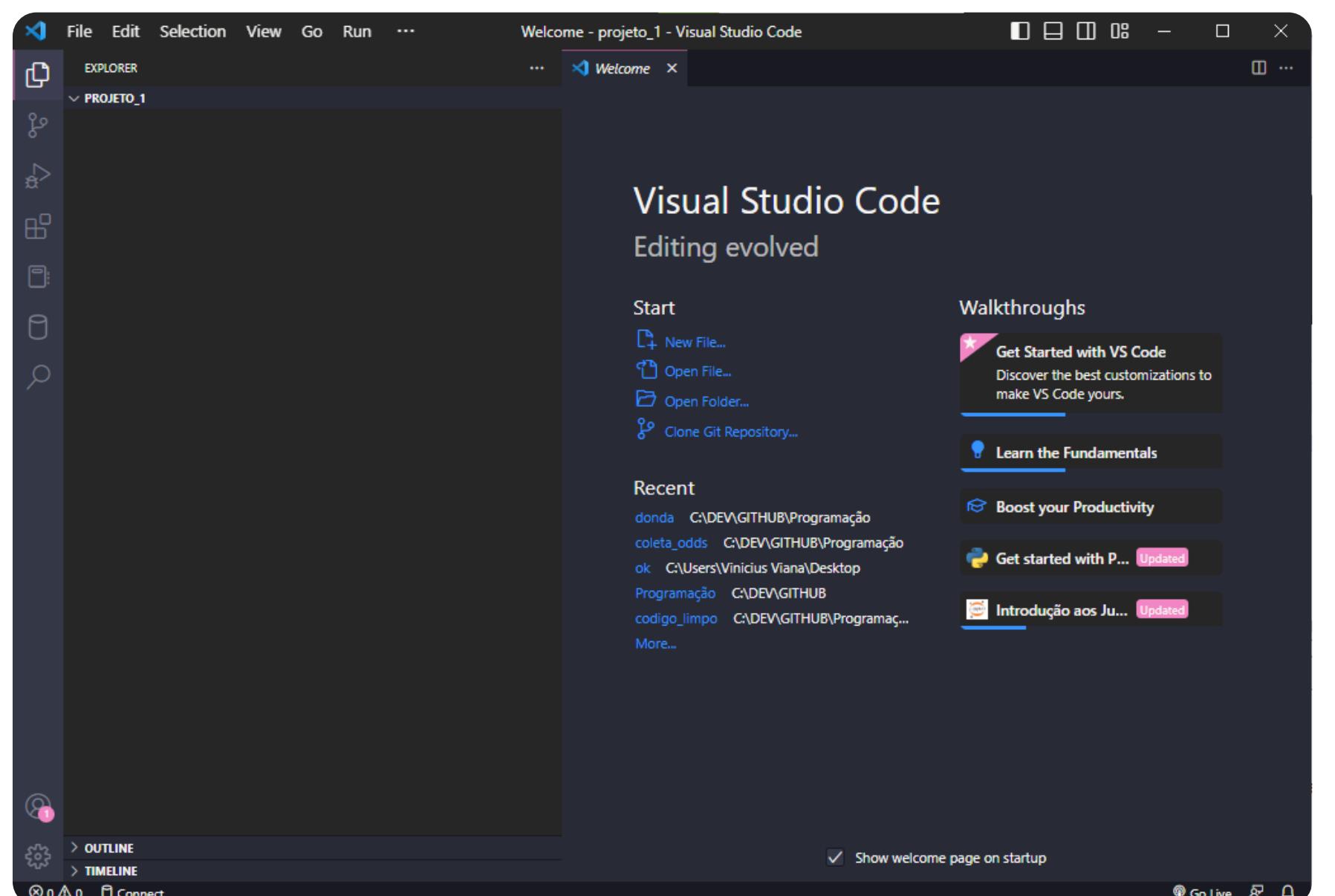
### PASSO 2:

Abra o VSCode e clique em "Arquivos" e depois clique em "Abrir pasta" e vá até a pasta criada por você, clique nela e escolha a opção "Selecionar pasta."



**PASSO 3:**

Pronto, você está dentro da pasta. Tudo que for criado aqui, será direcionado a pasta "projeto\_1".



### 5.3. – Configurando o Git

Antes de publicar nosso primeiro projeto, é importante configurar o Git corretamente. Vamos vincular nosso nome e e-mail ao Git. É importante que essas informações sejam iguais às registradas no GitHub, pois elas são usadas para autenticação, junto com a chave SSH.

Para fazer isso, abra o terminal do "Git Bash" no VSCode, caso você esteja no Windows. No Linux e no macOS, o Git já está integrado no terminal padrão, então você pode continuar nele. O importante é usar um terminal que permita executar comandos Git.

Digite os seguintes comandos no terminal para alterar o nome e o e-mail associados ao Git. Lembre-se de que o e-mail deve ser o mesmo registrado no GitHub, pois isso é uma das maneiras de o GitHub autenticar suas ações.

Utilizaremos o comando "Git config" para definir essas variáveis, existem 3 níveis de configuração:

1. **local**: Esse nível é o padrão, caso você não defina nada no comando "git config". Ele se resume apenas à aquele repositório Git, por isso quando for definir uma variável local tenha certeza que está no repositório correto. Esses valores podem ser encontrados dentro do arquivo ".git" em git/config.

2. **global**: Este nível significa que será definido para um usuário daquele sistema operacional. Os valores podem ser encontrados dentro da pasta ".gitconfig" na home.

3. **system**: Esse é o maior nível, definido para todo o sistema operacional, ultrapassando até usuários.

Esses comandos abaixo, serão aplicados globalmente, ou seja, no seu usuário do sistema operacional.

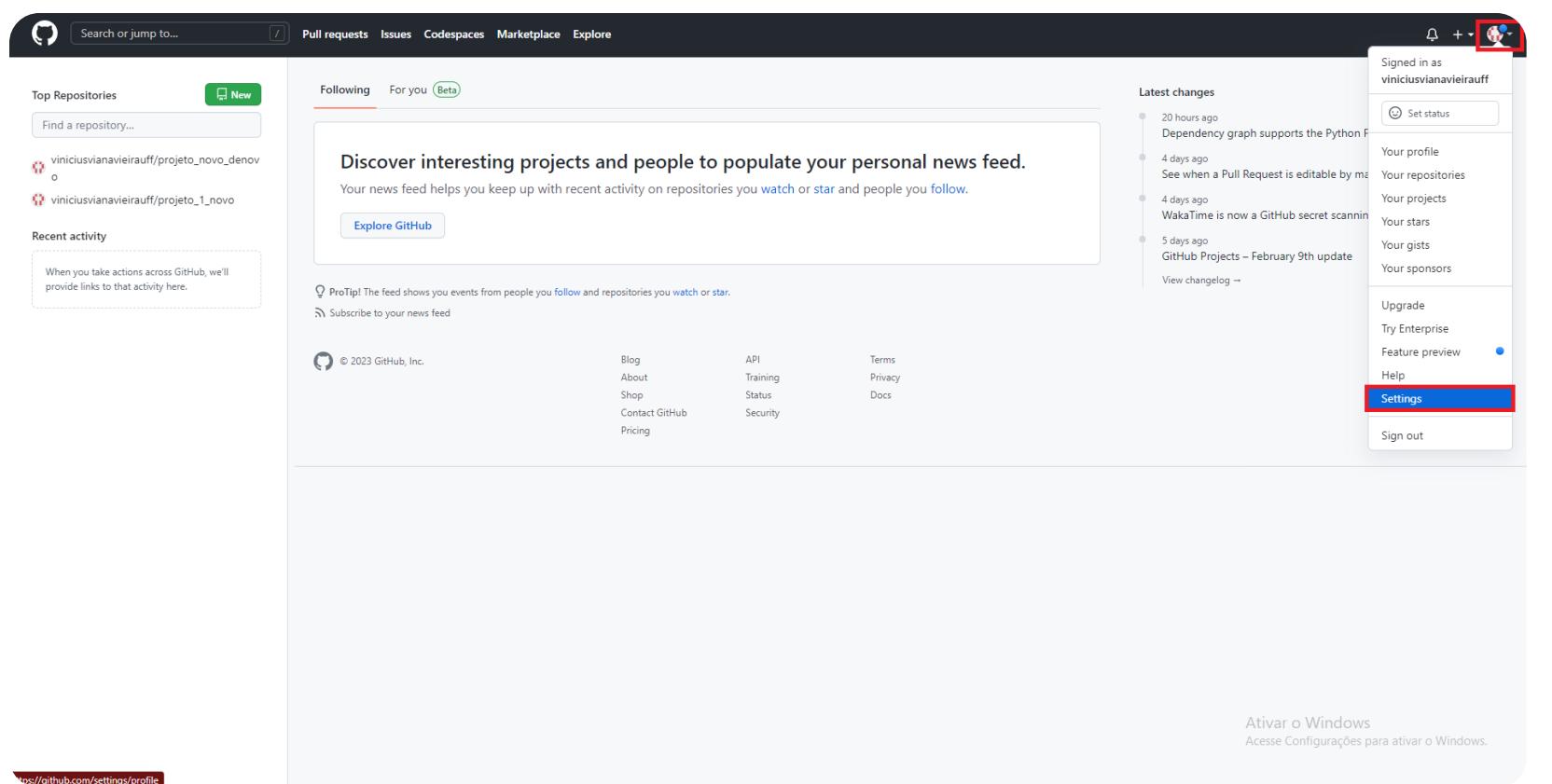
### 5.3.1. – Alterando o nome e e-mail

Vamos começar alterando o nome, e vamos utilizar o mesmo do GitHub como boas práticas de programação, através desse comando:

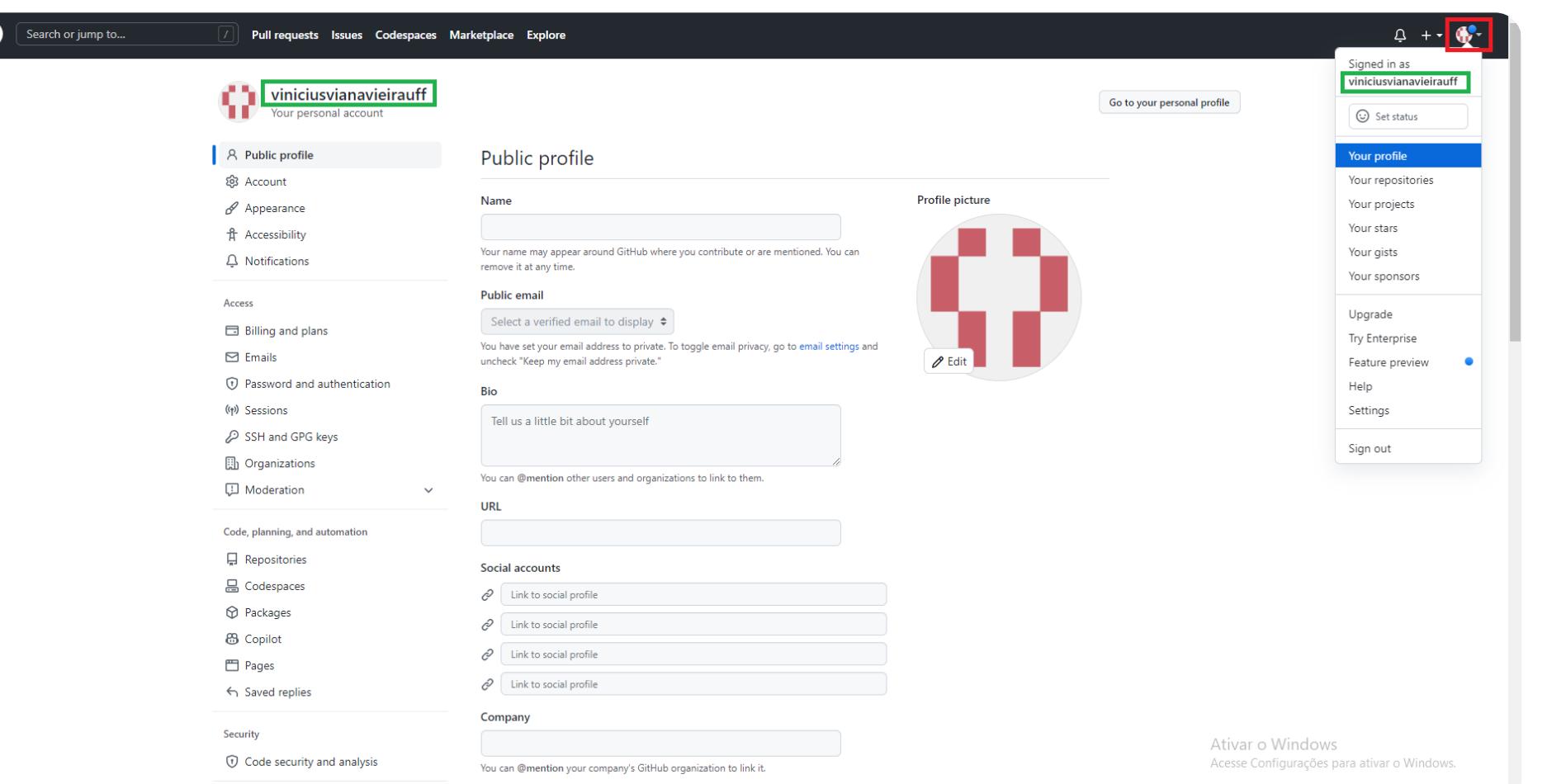
- `git config --global user.name "nome_do_GitHub"`

Caso não lembre o nome no GitHub, siga o passo a passo:

Para saber qual seu nome, entre no seu GitHub e clique na logo no canto superior direito e depois clique em “Settings”:



Esses nomes marcados de verde são os que serão utilizados:



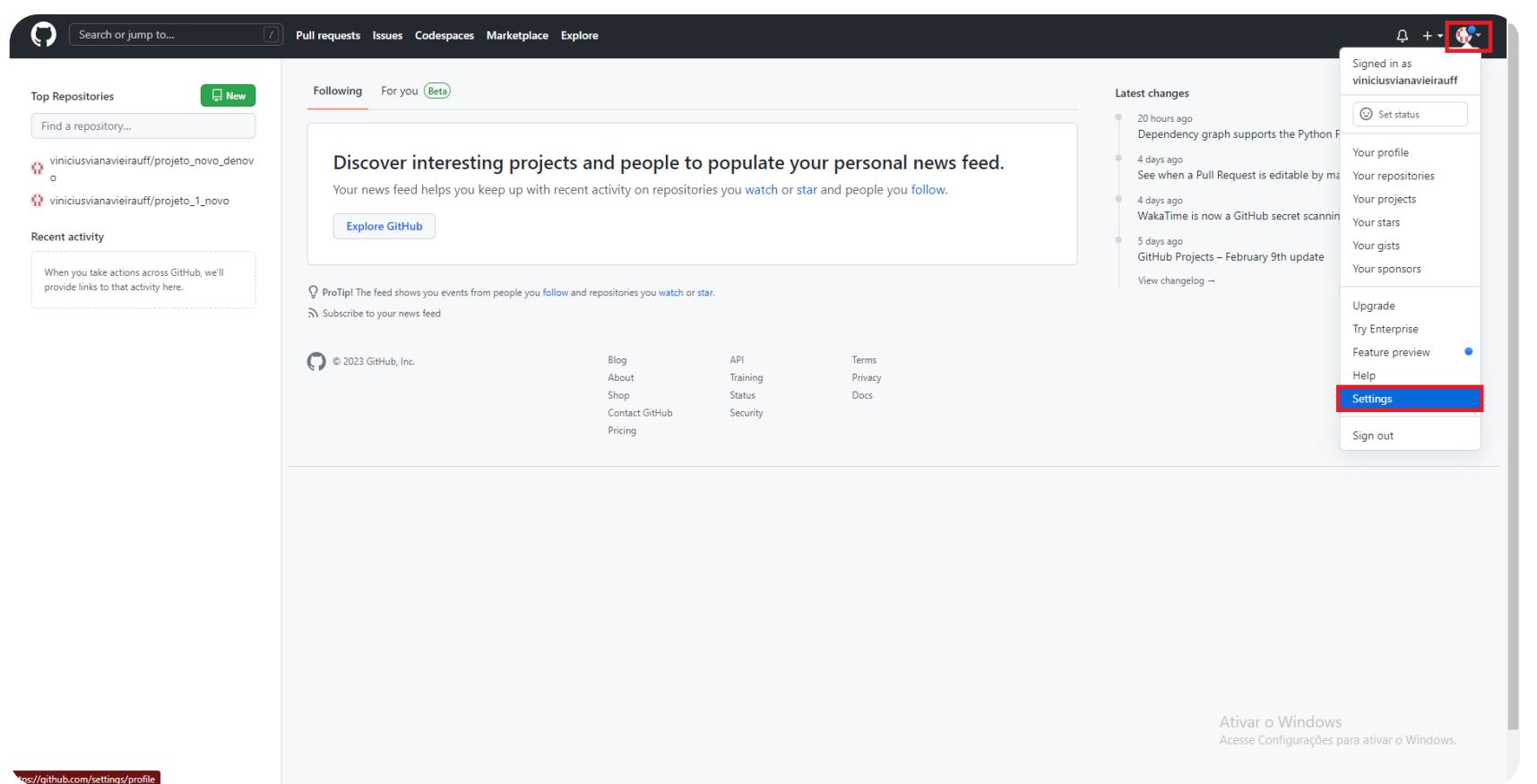
Ativar o Windows  
Acesse Configurações para ativar o Windows.

Após alterar o nome, vamos configurar o e-mail, que também deve ser igual ao GitHub, através desse comando:

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

Caso não lembre o e-mail no GitHub, siga o passo a passo:

Entre no seu GitHub e clique na logo no canto superior direito e depois clique em “Settings”:



Clique na sessão de “emails”:

A screenshot of the GitHub 'Emails' settings page. On the left, a sidebar lists various account settings like Public profile, Account, Appearance, Accessibility, Notifications, Billing and plans, and Emails. The 'Emails' option is highlighted with a red box. The main area shows a list of emails under 'Emails'. The first email, 'viniciusvianaveirauff@id.uff.br – Primary', is selected and highlighted with a green box. Below it, there are sections for 'Add email address' and 'Primary email address'. At the bottom, there are checkboxes for 'Keep my email addresses private' and 'Block command line pushes that expose my email'. The URL at the bottom right is 'github.com/settings/emails'.

Após a configuração do nome e e-mail através dos comando, confirme se teve êxito, através desse outro comando:

- git config user.name "nome\_do\_GitHub"
- git config user.email "seuemail@exemplo.com"

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a folder named 'Bem-vindo' containing a file 'inicio.py'. The code editor shows the following content:

```
1 print("Essa alteração foi a última")
```

Below the code editor is the Terminal panel, which is active and shows the following command history:

```
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git config --global user.name "viniciusvianavieirauff"
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git config --global user.email "viniciusvianavieira@id.uff.br"
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git config user.name
viniciusvianavieirauff
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git config user.email
viniciusvianavieira@id.uff.br
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
```

The status bar at the bottom indicates the file is 'main' and has 0 changes.

## 5.4. – Iniciando um repositório no Git

Após criar seu programa no Python e configurar tudo que é necessário, pode-se iniciar um repositório Git. Mas lembre-se, após a criação do programa, é somente um repositório normal do Windows ainda, caso não tenha o arquivo ".git".

Para conferir se há arquivo oculto, faremos alguns comandos nos terminais. Caso utilize o sistema operacional Linux, esse comando pode ser feito no próprio terminal, diferentemente do Windows que possui alguns como o próprio Git Bash, PowerShell e Command Prompt.

Cada terminal possui sua utilidade. Git Bash é o terminal do Git, apropriado para trabalhar com versionamento de código. PowerShell é apropriado para gerenciamento de sistema operacional Windows, enquanto o Command Prompt adequado para executar sistemas básicos de comando no sistema operacional.

- No Git Bash:

```
ls -la
```

- No PowerShell:

```
Get-ChildItem -Force
```

- No Command Prompt:

```
dir /ah
```

Na ausência do arquivo ".git", é necessário transformar num repositório Git, no canto direito superior do terminal tem um botão "+", depois vá no "git bash" e digite o comando "`git init`". Foi realizada a transformação de um repositório normal, em um repositório Git, com o arquivo ".git".

## 5.5. – Usando o Git Ignora para ocultar dados sensíveis

Como bem sabemos, temos que ter muito cuidado ao expor dados sensíveis nos nossos programas ao upar um repositório de forma pública no GitHub. Para isso, temos uma solução, criaremos uma pasta, que a função dela será ignorar os arquivos dentro do seu repositório. Por exemplo, criou um programa de login automático no seu e-mail, e deixou como público no GitHub, para não expor seus dados sensíveis, eles ficarão em um outro arquivo dentro da pasta `.gitignore`.

Portanto, para criar a parte, digite no terminal o seguinte comando:

```
"touch .gitignore"
```

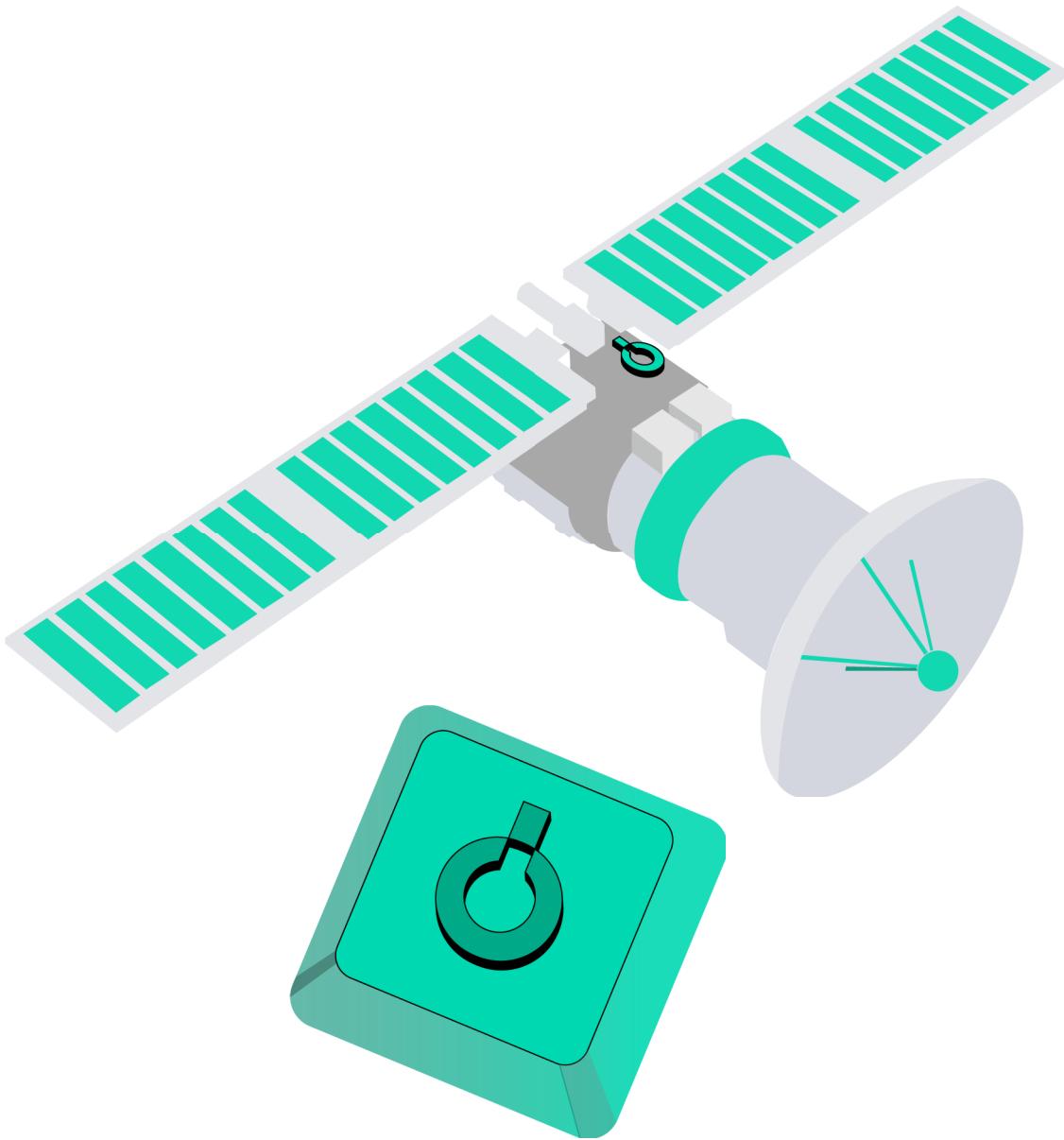
E, para adicionar arquivos dentro dessa pasta:

```
"echo .env >> .gitignore"
```

## 5.6 – O que é o Readme?

O README é como um guia de boas-vindas para o seu projeto no GitHub. É o primeiro lugar que as pessoas vão olhar quando visitam o repositório. Pense nele como um bilhete amigável que explica o que o projeto faz, como começar a usá-lo e como as pessoas podem contribuir. Ele ajuda os usuários a entenderem rapidamente do que se trata o projeto e como podem se envolver. É como um convite para explorar e colaborar!

O arquivo “README” é um documento geralmente em formato de texto simples ou em Markdown. Nele, você fornecerá as informações importantes sobre o seu projeto para os outros usuários ou recrutadores, portanto, é recomendado que seja robusto e bem explicativo. Geralmente, deve conter informações essenciais da descrição do projeto, quais desafios você resolve, como manipular e realizar manutenção no código, qual a forma correta de interpretar o projeto, como os desenvolvedores podem contribuir para a melhoria, exemplos de uso e etc, use a criatividade.

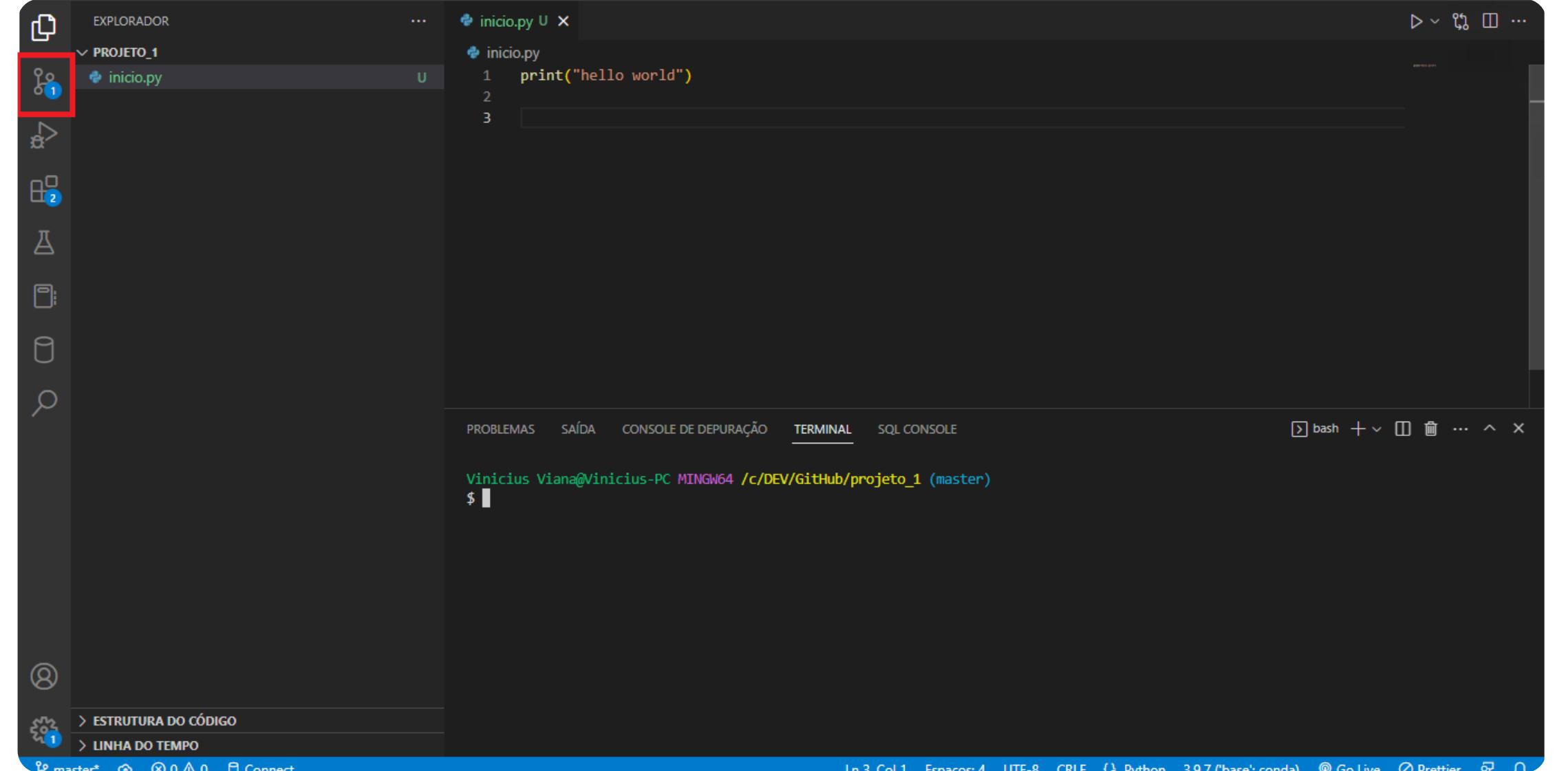


## Mundo 6

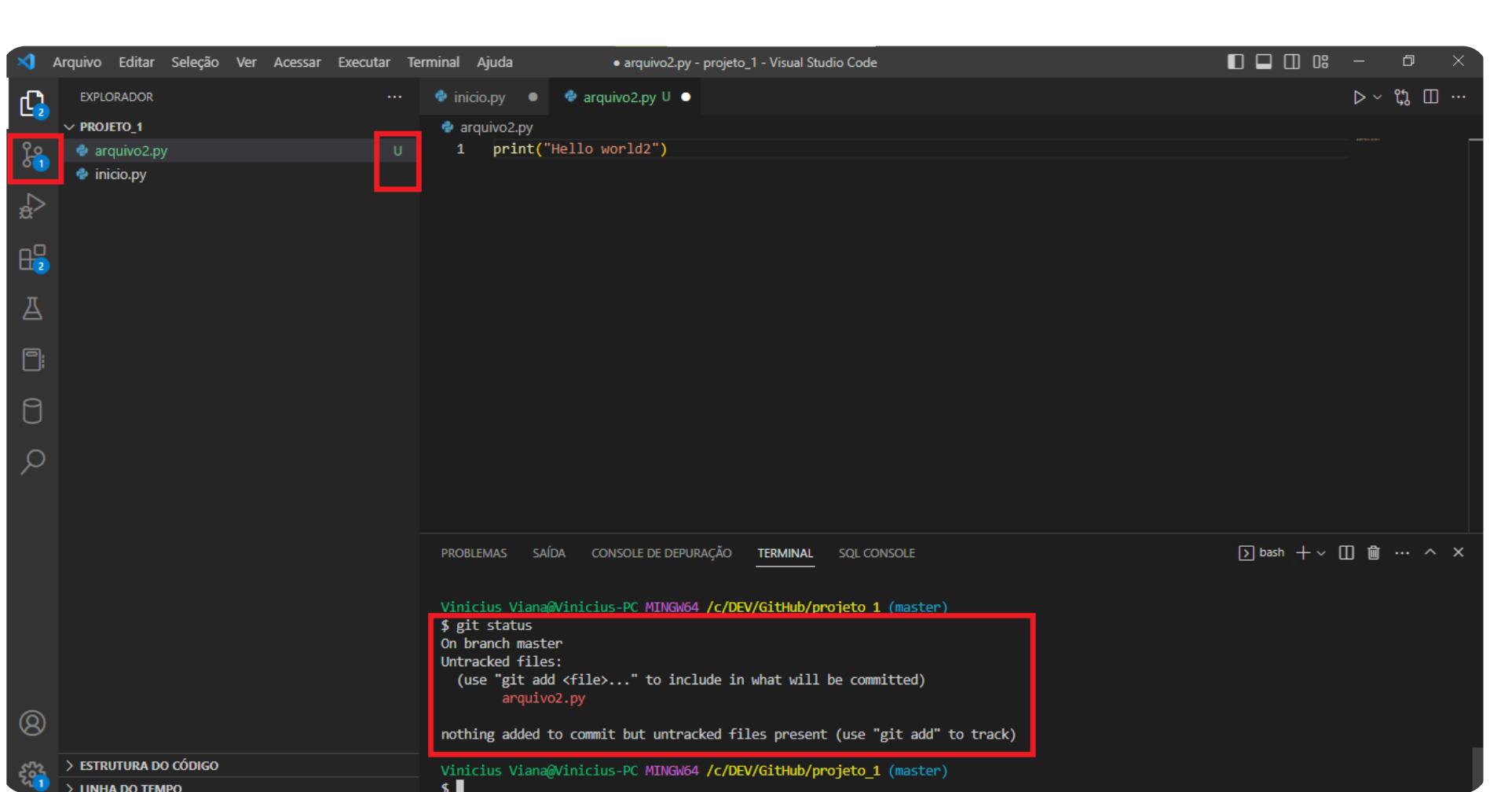
### 6.1. – Como fazer um commit

No último mundo foi ensinado como criar um repositório Git, portanto, o versionamento de código está habilitado e no estágio modificando, como aprendemos anteriormente. A partir de agora, aprenderemos como subir um código novo, dar um “commit”, e entre outras coisas.

Primeiramente, vamos aprender a identificar o estágio do projeto, através da integração do Git e VSCode. Esse pop-up representa o número de arquivos modificados do seu projeto, nesse caso, apenas 1 arquivo. Quando é criado ou modificado outro arquivo, este pop-up modifica e é sinalizado como 2 arquivos não salvos.



Ao clicar no símbolo que representa uma branch, é possível ver quais arquivos dentro do projeto sofreram alterações e ainda não foram salvos.

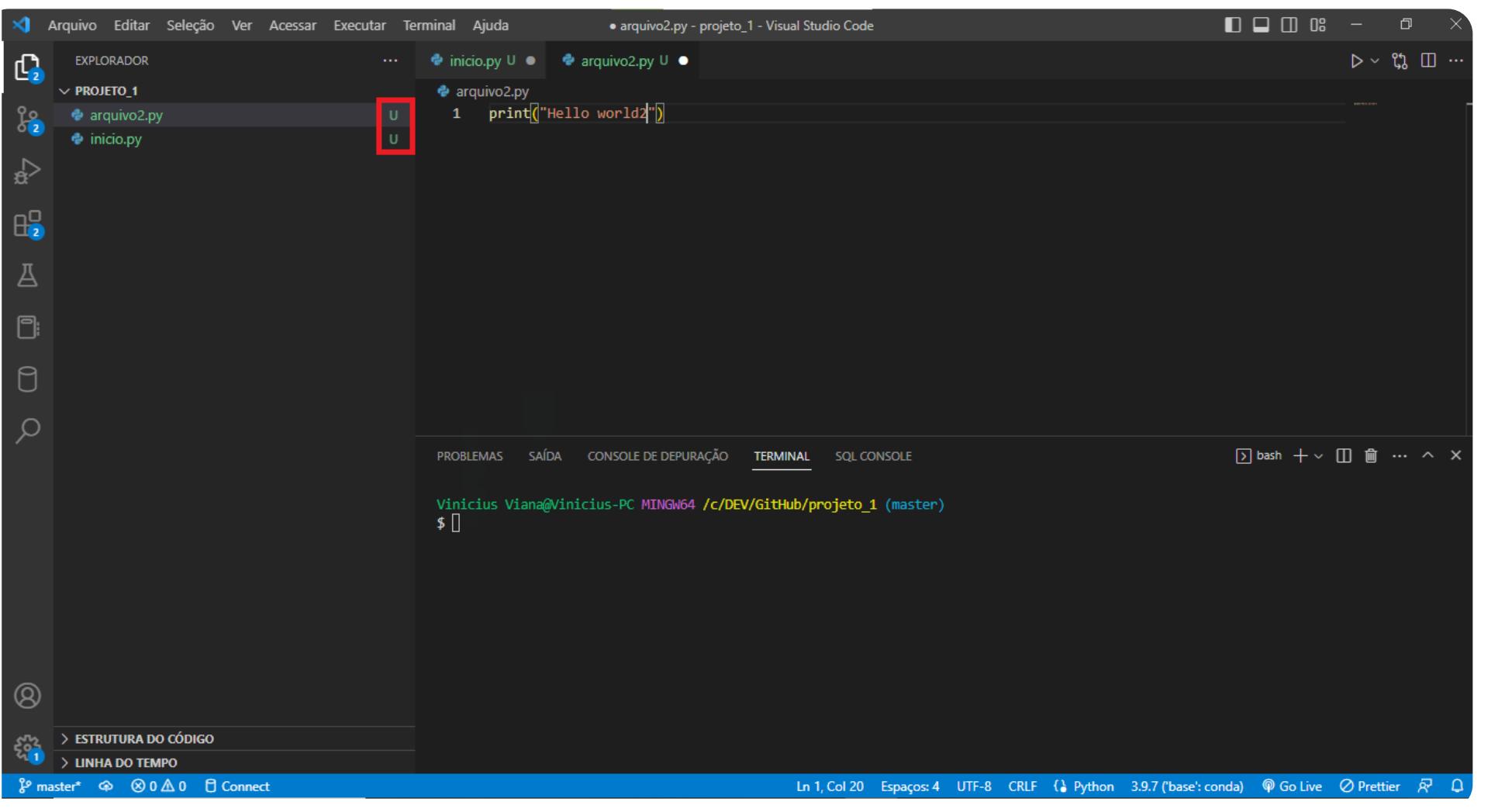


The screenshot shows the Visual Studio Code interface with a dark theme. In the Explorer sidebar, under the 'PROJETO\_1' folder, there are two files: 'arquivo2.py' and 'inicio.py'. Both files have a small blue icon with a white 'U' next to them, indicating they are untracked. A red box highlights this icon for 'arquivo2.py'. The terminal tab at the bottom shows the output of a 'git status' command:

```
Vinicius Viana@Vinicius-PC MINGW64 /c/DEV/GitHub/projeto_1 (master)
$ git status
On branch master
Untracked files:
 (use "git add <file>..." to include in what will be committed)
 arquivo2.py

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

O símbolo “U” verde, significa que está no 1º estágio do projeto, como modificado.



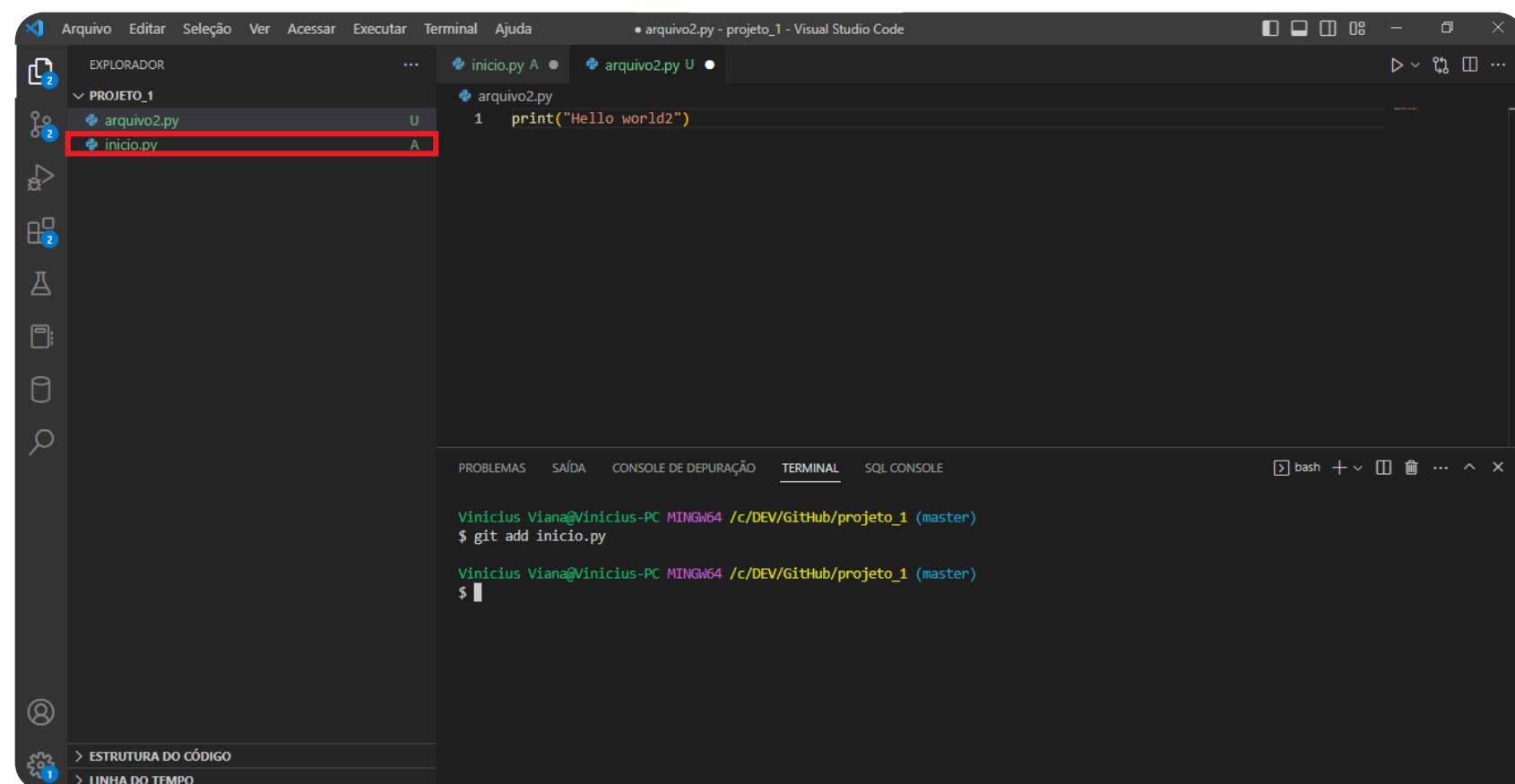
The screenshot shows the Visual Studio Code interface with a dark theme. In the Explorer sidebar, under the 'PROJETO\_1' folder, there are three files: 'arquivo2.py', 'arquivo.py', and 'inicio.py'. The 'arquivo2.py' file has a small blue icon with a white 'U' next to it, while 'arquivo.py' and 'inicio.py' have a small blue icon with a white dot. A red box highlights the 'U' icon for 'arquivo2.py'. The terminal tab at the bottom shows the output of a 'git status' command:

```
Vinicius Viana@Vinicius-PC MINGW64 /c/DEV/GitHub/projeto_1 (master)
$
```

Portanto, realizada todas as modificações necessárias, vamos avançar para o próximo estágio e enviar o arquivo para a área de preparação para que seja possível upar no GitHub, através do comando:

- git add <nome\_do\_arquivo>

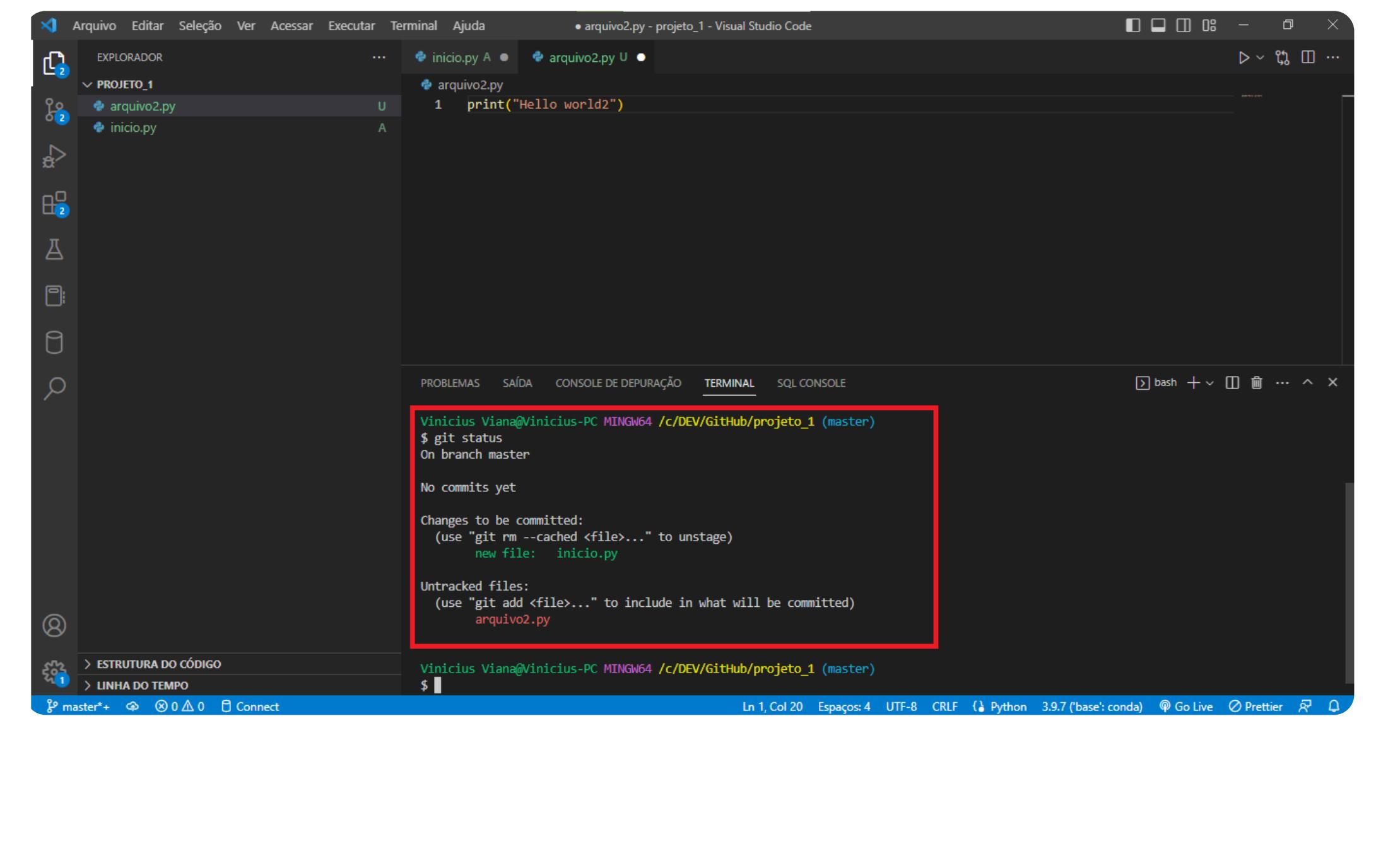
Houve a substituição de “U” para “A”, ou seja, ele foi adicionado à área de preparação, porém ainda não foi confirmado, ou em outras palavras, “commitado”.



A screenshot of the Visual Studio Code interface. The left sidebar shows a project named 'PROJETO\_1' containing two files: 'arquivo2.py' and 'inicio.py'. The 'inicio.py' file has a red border around its status bar indicator 'U', indicating it is untracked. The terminal at the bottom shows the command '\$ git add inicio.py' being run.

### 6.1.1. – Conferência de estágio do seu projeto via terminal

Existem outras formas de ser feita a conferência do estágio do seu projeto, via terminal.



A screenshot of the Visual Studio Code interface. The terminal at the bottom shows the command '\$ git status' being run. The output indicates there are changes to be committed, specifically 'new file: inicio.py'. A red box highlights this section of the terminal output.

**On Branch Master:** Na tradução seria: “Na Branch principal”. Isso significa os arquivos “commitados”, aqueles que já foram salvos no seu computador, ou seja, localmente por meio do comando “git commit”. Como não foi feito “commit” em nenhum arquivo, foi retornado:

“No commits yet” = “Sem “commits” ainda”

**Changes to be committed:** Na tradução seria: “Mudanças a serem comitadas”. Isso significa os arquivos que foram para área de preparação, por meio do comando “git add”. São os arquivos prontos e não salvos. Além disso, são os arquivos que, agora, possuem aquele “A” ao lado do nome.

E de fato, se repararmos, tem apenas um arquivo presente ali. Esse arquivo é o mesmo que adicionamos na área de preparação por meio do comando “git add”.

“Arquivo novo: inicio.py”

O próprio Git fornece a instrução caso deseje retornar o arquivo da área de preparação. Utilizando o comando:

```
git rm --cached <nome_do_arquivo>
```

**Untracked files:** Na tradução seria: “Arquivos não rastreados”. São aqueles que modificamos, mas não utilizamos ainda nenhum comando do Git para mudar o estado deste arquivo. São aqueles que possuem o “U” ao lado do nome.

Após o arquivo ser adicionado para a área de preparação, devemos enviar ao nosso repositório local, através do comando:

```
git commit -m "mensagem_obrigatoria" -m "descricao opcional"
```

Após o commit, houve algumas mudanças. como:

- Desaparecimento das letras de identificação "U" e "A", indicando que o projeto está salvo no Git.
- Na branch, consta somente o número 1, e não 2, como anteriormente, indicado que o arquivo foi “commitado”. Agora consta o número 1, referente apenas ao arquivo: arquivo2.py
- Digitando o comando “git status”, o arquivo não é visível pois foi confirmado e registrado permanentemente no histórico do repositório.

```
$ git status
On branch master
Untracked files:
 (use "git add <file>..." to include in what will be committed)
 arquivo2.py

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

Para ter acesso ao histórico de “commits”, digite o comando no VSCode:

```
git log
```

Caso você precise sair do editor é só seguir os seguintes passos:

Clique em [:] e digite [wq]

## 6.2. – Postando no GitHub | Criando um repositório remoto

Ao contrário das etapas anteriores, não existe um comando direto no Git que crie um repositório diretamente no GitHub. No entanto, há maneiras de conectar um repositório Git a um repositório no GitHub ou até mesmo clonar um repositório do GitHub para o seu computador.

Para simplificar o processo, primeiro precisamos criar um repositório remoto no GitHub separadamente. Em seguida, vamos integrar esse repositório do GitHub ao Git e ao VSCode. Permitindo que faça “commits” de forma quase automática nas etapas subsequentes. Para isso, existem 3 formas.

### 6.2.1. – 1<sup>a</sup> forma

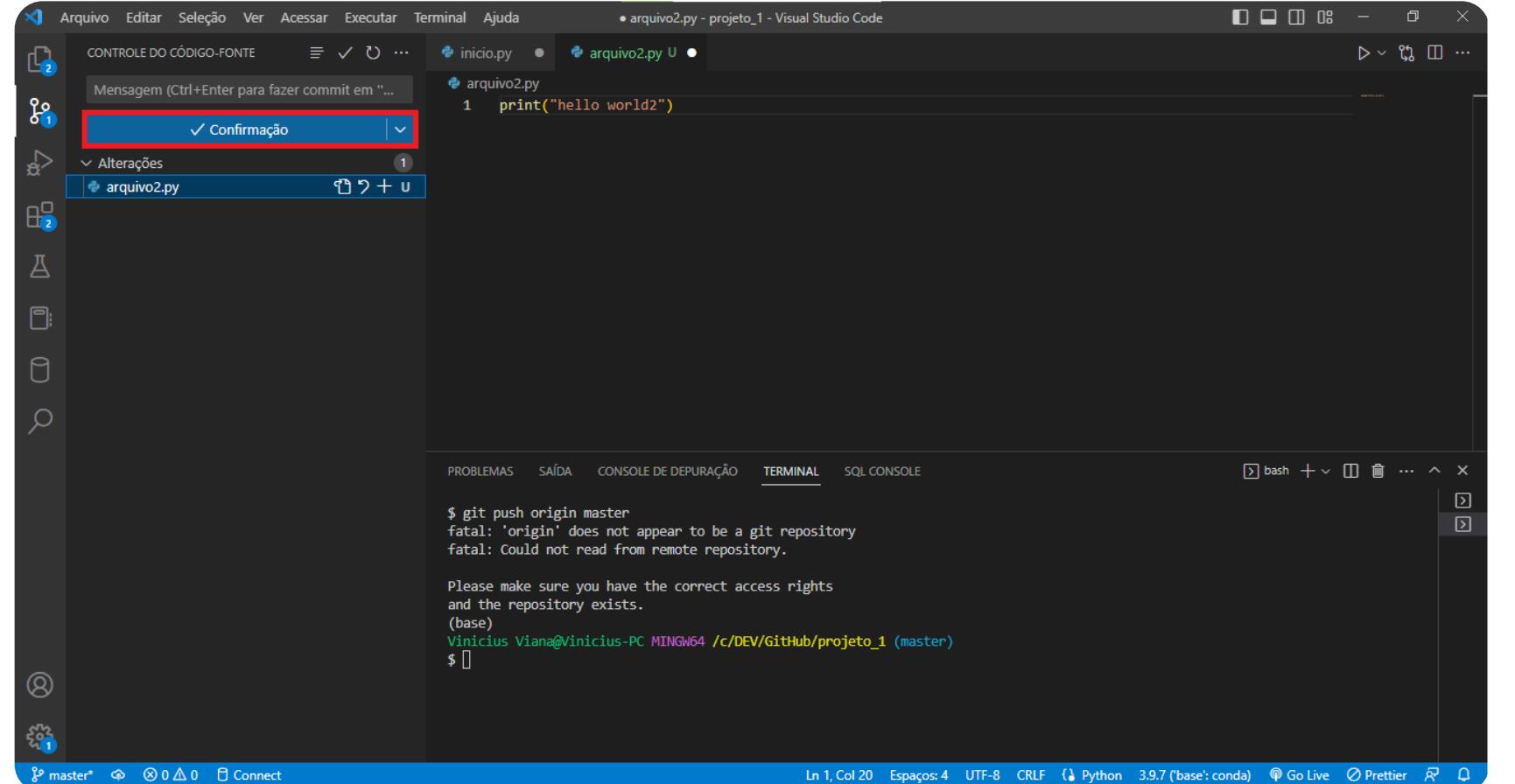
Esta forma é feita através do VSCode, que já possui uma integração com o GitHub.

Antes de iniciarmos, é necessário “commitar” os arquivos. Dessa vez, faremos o “commit” para todos os arquivos, e não de um em um, através do comando:

- `git add *`
- `git commit -m "mensagem"`

Está tudo pronto para publicar dentro do GitHub.

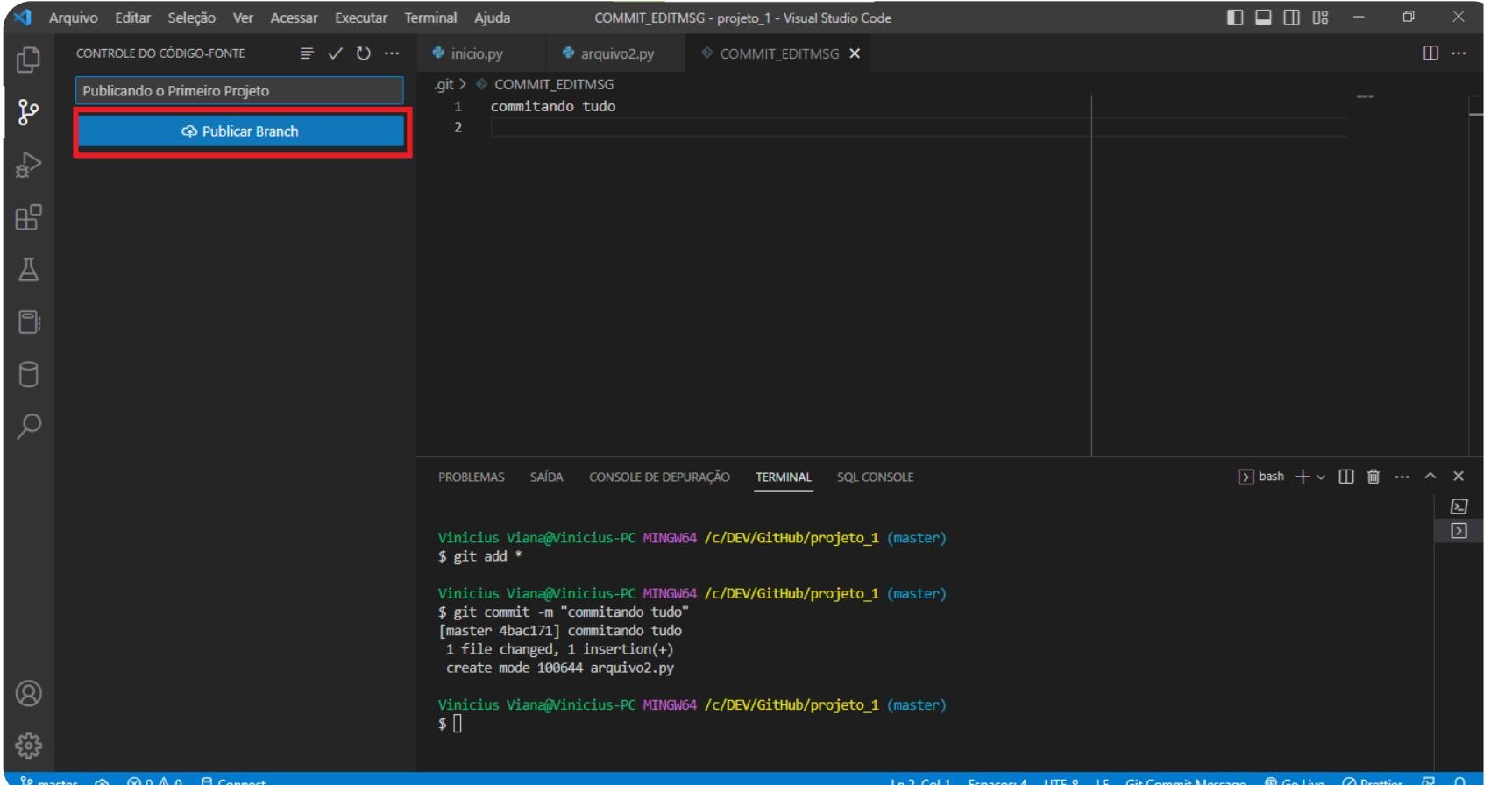
Clique no botão de confirmação. Esse botão é útil quando você deseja realizar um “push”, que é o ato de enviar as informações para o GitHub. No entanto, ao clicar nele, o GitHub identificará que este projeto ainda não está vinculado a nenhum repositório remoto e o ajudará a criar um repositório remoto automaticamente.



A screenshot of the Visual Studio Code interface. The top bar shows the menu: Arquivo, Editar, Seleção, Ver, Acessar, Executar, Terminal, Ajuda. The title bar says "arquivo2.py - projeto\_1 - Visual Studio Code". The left sidebar has sections for "CONTROLE DO CÓDIGO-FONTE", "Alterações", and "arquivo2.py". The main editor area contains the code: "1 print("hello world2")". A confirmation dialog box is open in the center, with the message "Mensagem (Ctrl+Enter para fazer commit em "")" and a dropdown menu showing "✓ Confirmação". The bottom status bar shows "Ln 1, Col 20 Espaços: 4 UTF-8 CRLF Python 3.9.7 (base: conda) Go Live Prettier".

Depois de ter “commitado” todas as alterações necessárias, clique no botão “Publicar Branch”.

- De a permissão para a extensão “GitHub”.
- De permissão ao “VSCode”.
- E digite sua senha para confirmar.



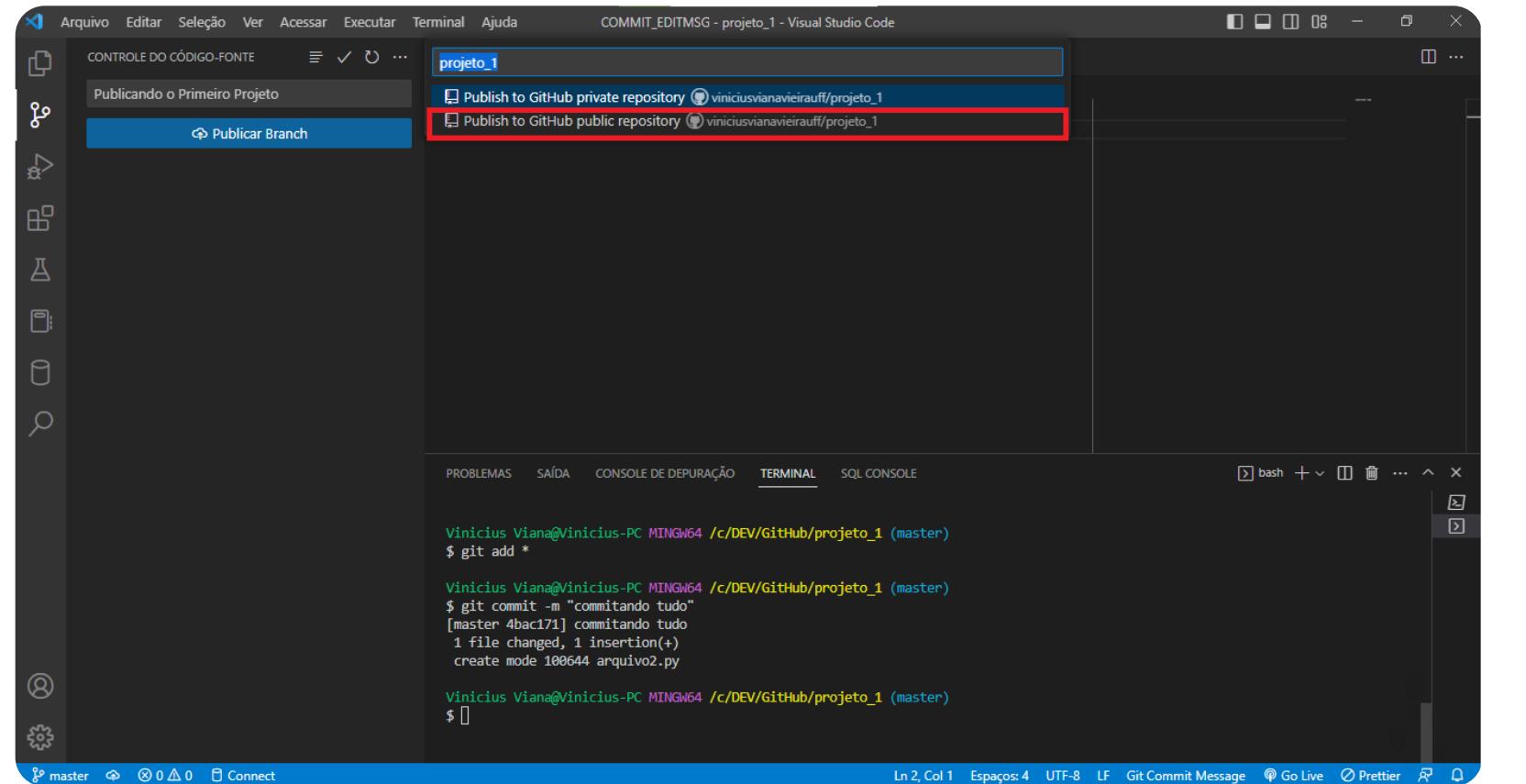
A screenshot of the Visual Studio Code interface. The title bar says "arquivo2.py - projeto\_1 - Visual Studio Code". The left sidebar has sections for "CONTROLE DO CÓDIGO-FONTE" and "arquivo2.py". The main editor area contains the code: "1 print("hello world2")". A dialog box is open with the message "Publicando o primeiro Projeto" and a button "⇨ Publicar Branch". The bottom status bar shows "Ln 2, Col 1 Espaços: 4 UTF-8 LF Git Commit Message Go Live Prettier".

The terminal window shows the following command sequence:

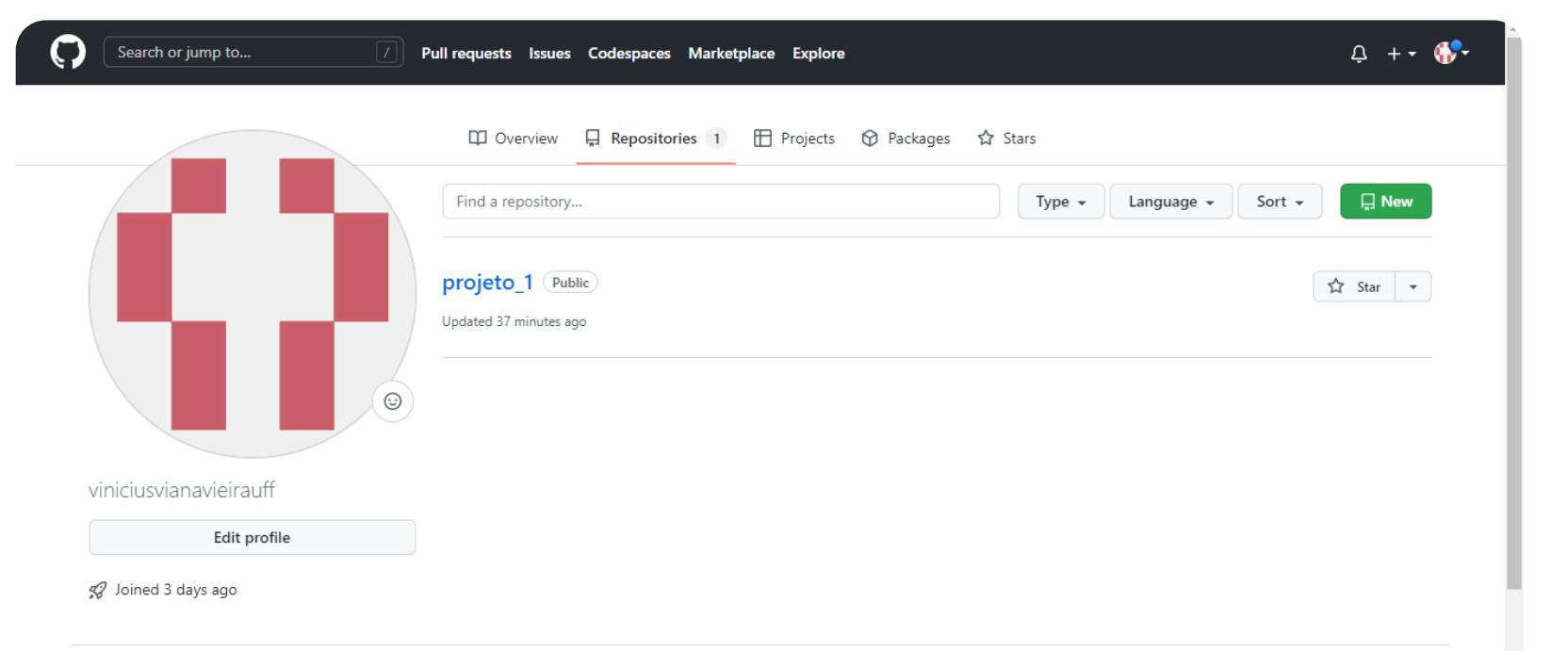
```
Vinicius Viana@Vinicio-PC MINGW64 /c/DEV/GitHub/projeto_1 (master)
$ git add *
Vinicius Viana@Vinicio-PC MINGW64 /c/DEV/GitHub/projeto_1 (master)
$ git commit -m "commitando tudo"
[master 4bac171] commitando tudo
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo2.py
Vinicius Viana@Vinicio-PC MINGW64 /c/DEV/GitHub/projeto_1 (master)
$
```

Após, escolha se o seu repositório será público ou privado. Lembre-se que:

- Repositório público qualquer pessoa terá acesso ao seu projeto, e até mesmo copiar.
- Repositório privado só quem tem permissão poderá acessar.



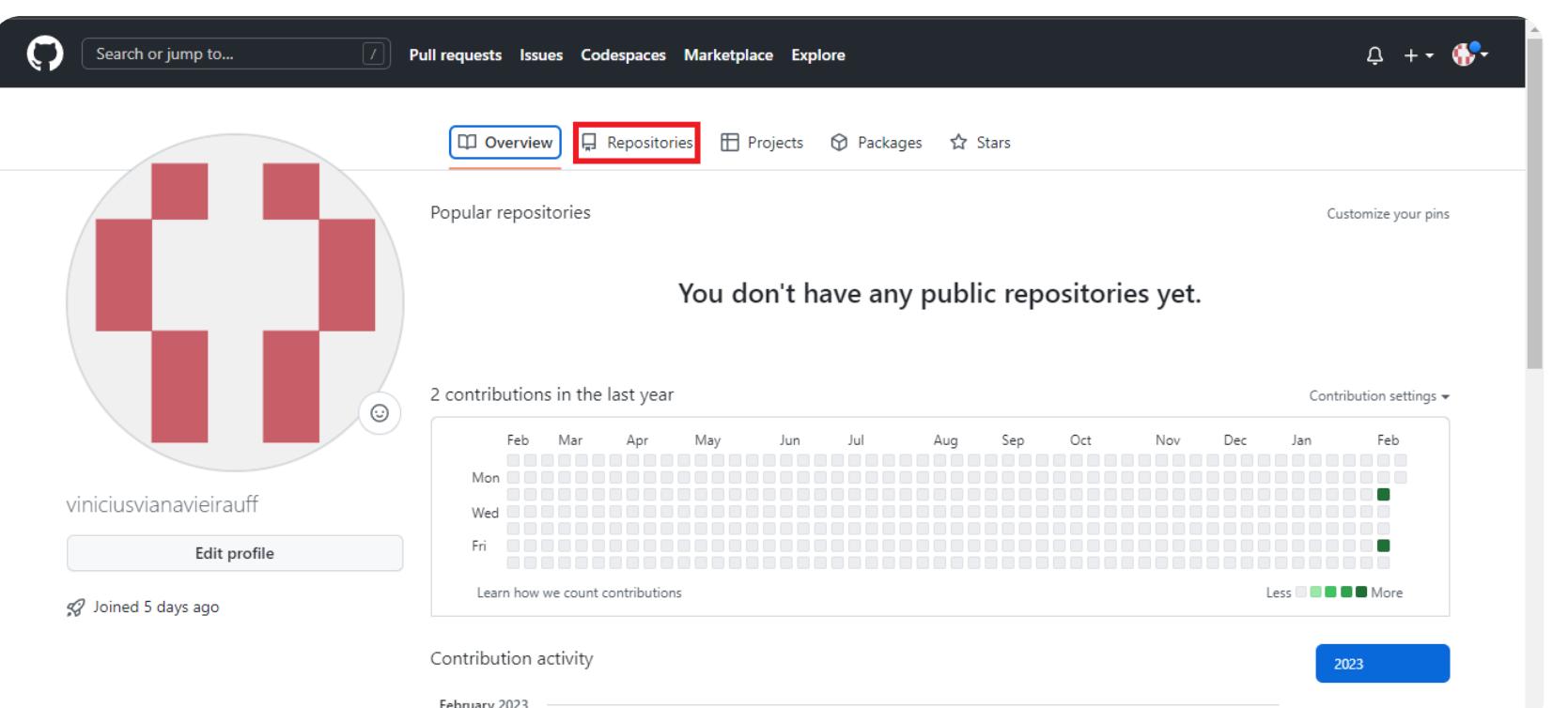
Ao entrar no GitHub, já é possível visualizar seu projeto upado.



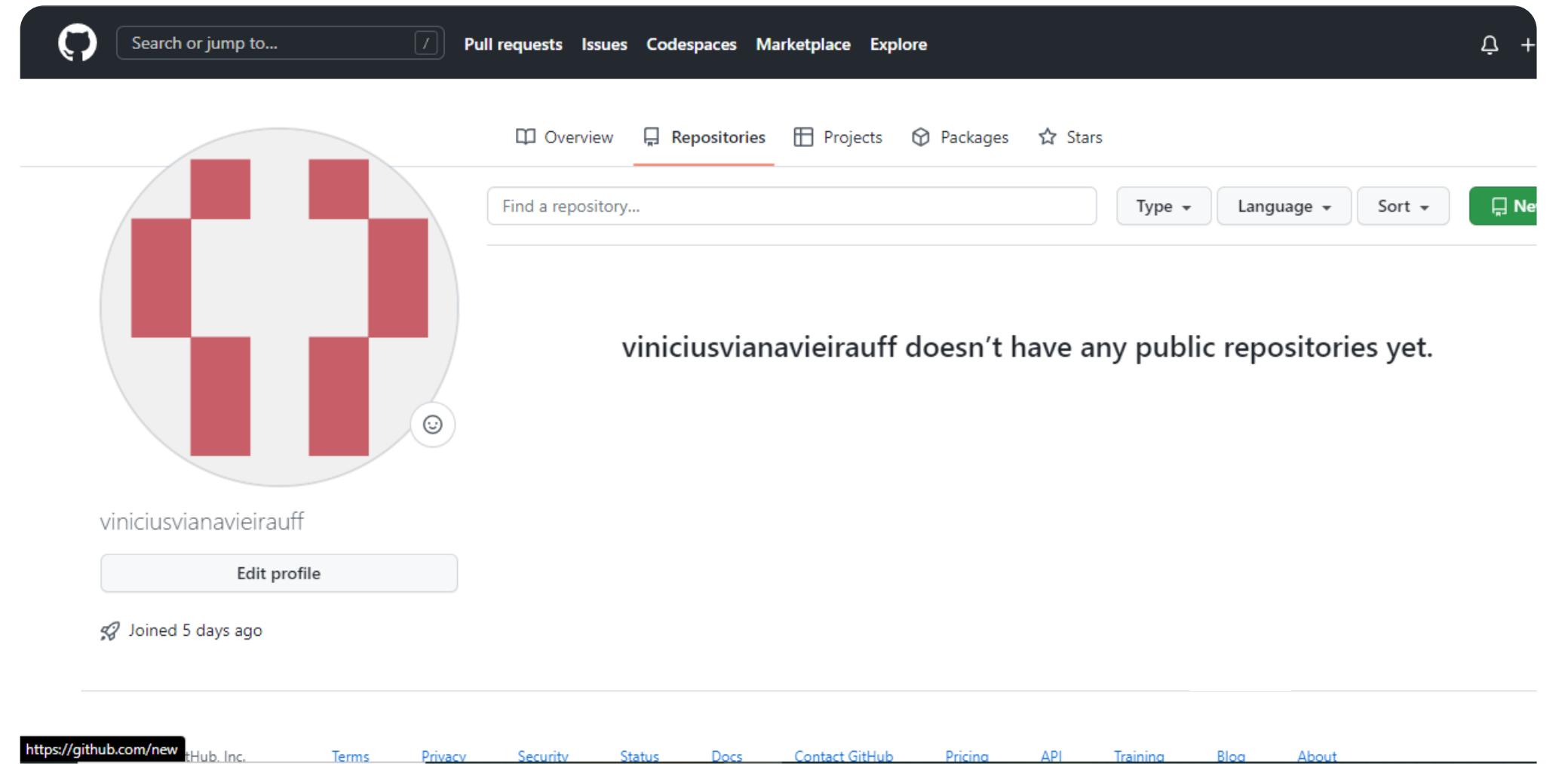
## 6.2.2. - 2<sup>a</sup> forma

Nessa forma, o repositório será criado manualmente, buscando o ID do repositório no GitHub e clonando no seu computador, por meio de comandos Git.

Então entre no seu GitHub e clique:

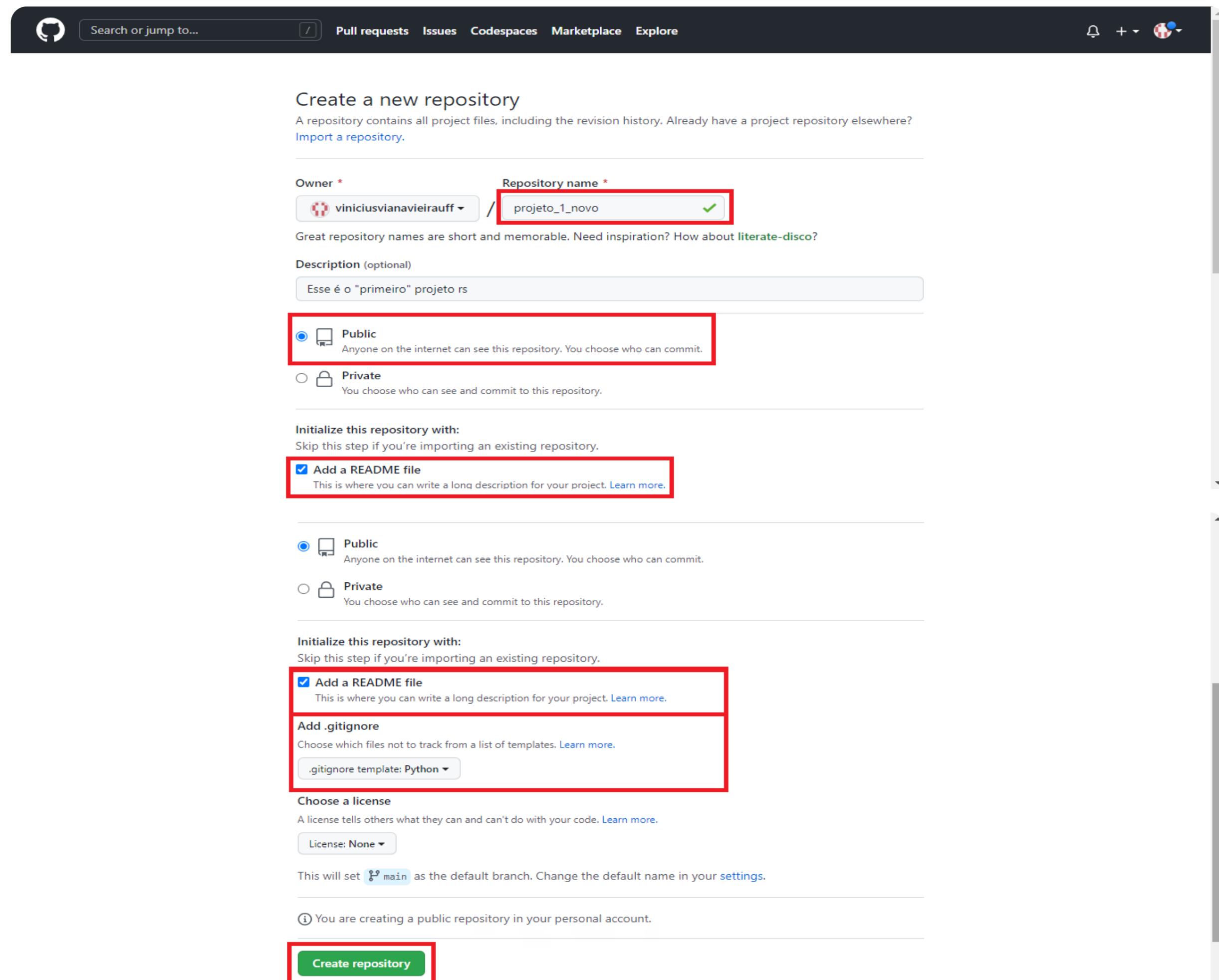


Clique no botão de “New” para criar um repositório.



Preencha os campos:

- **Repository name:** Nome do repositório a ser criado sem espaços
- **Description:** Uma breve descrição do seu repositório
- **README file:** Adicionar um arquivo “readme.file” que é nada mais do que um arquivo que contém uma descrição. É usado para fornecer informações sobre o projeto, propósito, instruções, requisitos, licenças etc. Qualquer coisa que você ache importante colocar.
- **.gitignore:** Um arquivo de configuração que indica ao Git, quais arquivos ou pastas devem ser ignorados, na hora de rastrear alterações no GitHub. Utilizado quando há arquivos que não devem ser incluídos no controle de versão. Um arquivo com senhas ou chaves de acesso, por exemplo. Escolha a linguagem que você vai utilizar, no nosso caso Python.
- **Choose a license:** Escolher licença está atrelado ao objetivo do seu código. Se você deseja que seu projeto seja de código aberto, ou seja, se você quiser que as pessoas reproduzam, distribuam e criem versões sobre seu projeto, você deve escolher uma licença. É um assunto mais complexo, que não vale se aprofundar muito agora.

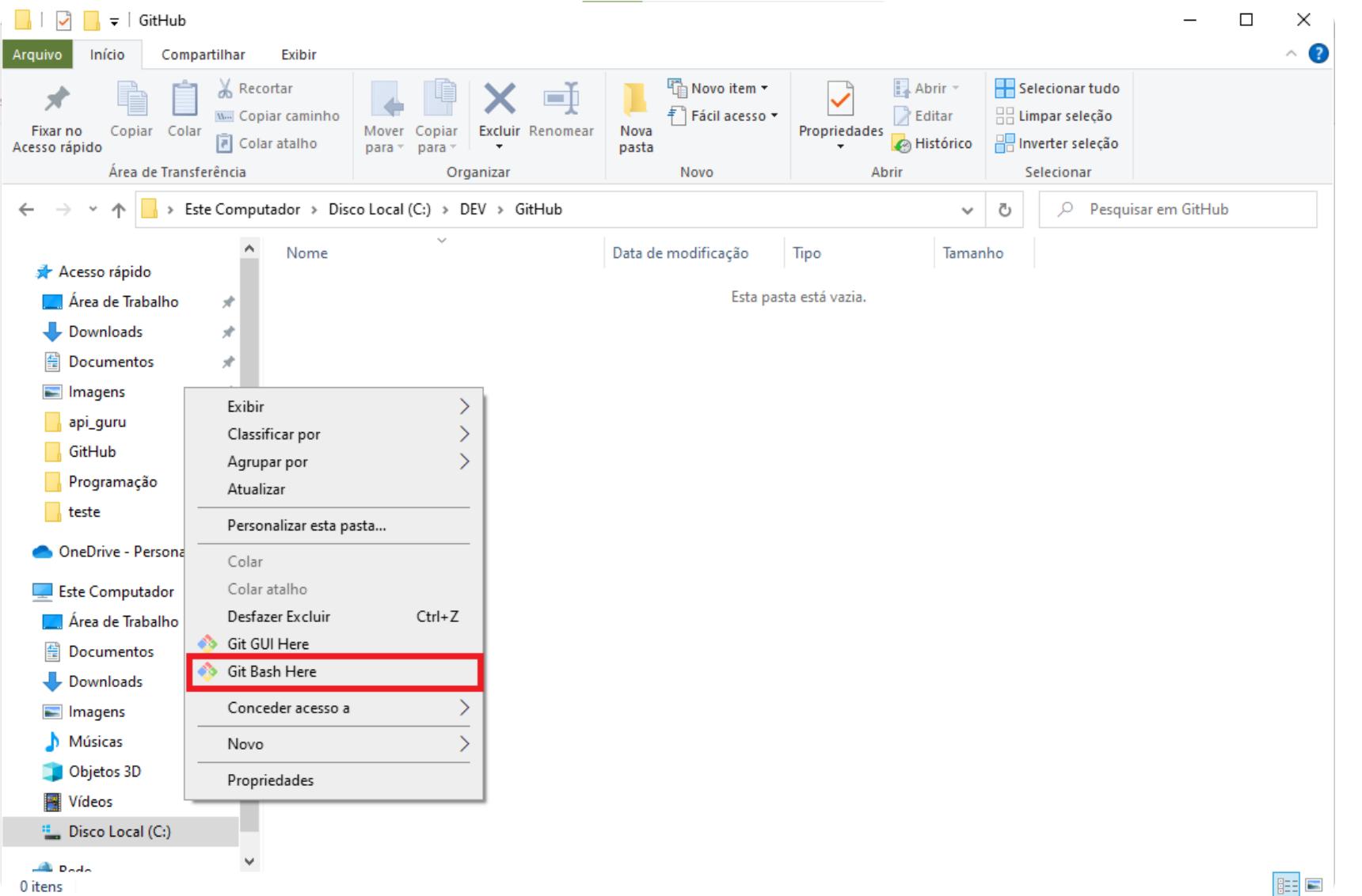


Seu repositório foi criado, mas dentro do GitHub apenas, é necessário clonar em seu computador.

Se direcione até a pasta que você deseja que o arquivo fique. Clique com o botão direito e escolha o “GitBash Here”. Ele abrirá o Terminal GitBash dentro da sua pasta em que você está.

Esse passo a passo, é para Windows mas serve para outros serviços operacionais. O que diferencia é que ao invés de um “Terminal Git Bash”, abrirão um terminal comum onde o Git já está instalado. Os comandos a seguir, também são os mesmos para outros sistemas operacionais.

Clique na opção “Git Bash Here” e o terminal será aberto.



Vá até o GitHub e clique no seu diretório. Clique na opção “CODE” e depois copie o caminho “HTTPS”. O que faremos é clonar esse repositório no seu computador, e esse caminho é para identificar o repositório. Você poderia também utilizar a chave “SSH”, pois já a configuramos dentro do seu Git e do GitHub.

Esse é o “primeiro” projeto

Readme

0 stars

1 watching

0 forks

Releases

No releases published

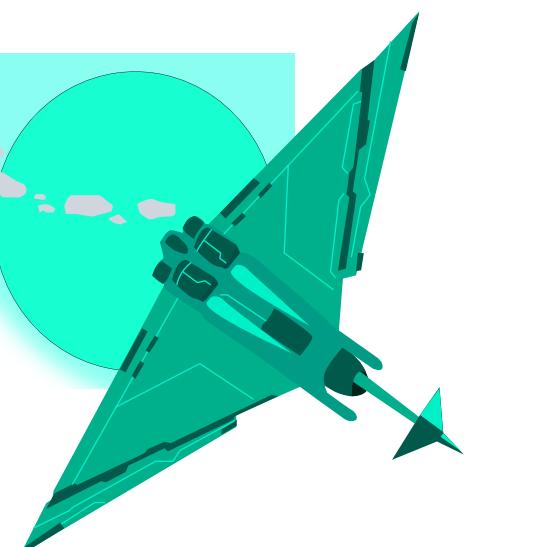
Create a new release

Packages

No packages published

Dentro do Git Bash digite o seguinte comando:

```
git clone <http_link>
```



```
MINGW64:/c/DEV/GitHub
Vinicius Viana@Vinicius-PC MINGW64 /c/DEV/GitHub
$ git clone https://github.com/viniciusvianavieirauff/projeto_1_novo.git
Cloning into 'projeto_1_novo'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

Vinicius Viana@Vinicius-PC MINGW64 /c/DEV/GitHub
$
```

Pronto projeto clonado, agora é só abrir a pasta com VSCode.

Agora, será criado um novo arquivo através do comando “print (“Hello World”)” e realizar as alterações para o GitHub. Após a criação do novo arquivo, utilize os seguintes comandos para alterar no GitHub:

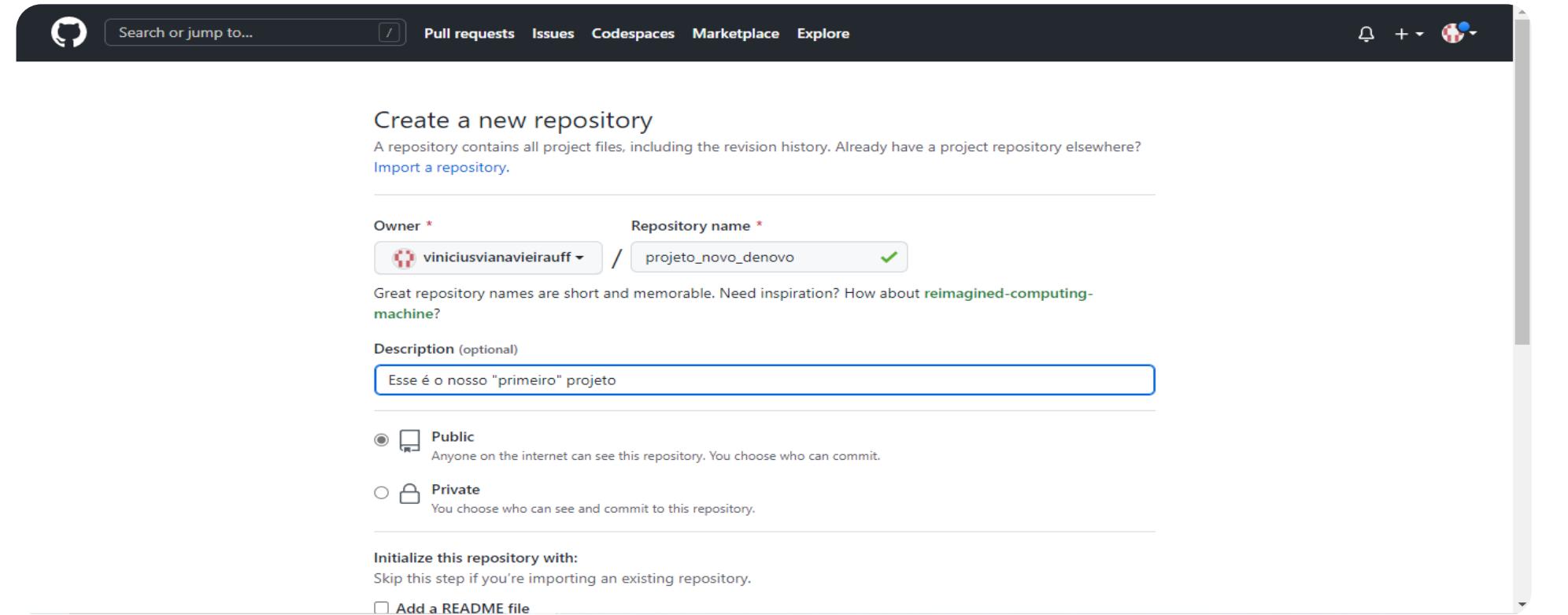
- git add \* ## Para adicionar todos os arquivos modificados na área de preparação
- git commit -m "mensagem" ## Para salvar as alterações dentro do seu arquivo Git Localmente
- git push origin main ## Para jogar as alterações pro GitHub

Depois de executar os comandos, o repositório criado no GitHub estará clonado em seu computador, basta confirmar no GitHub.

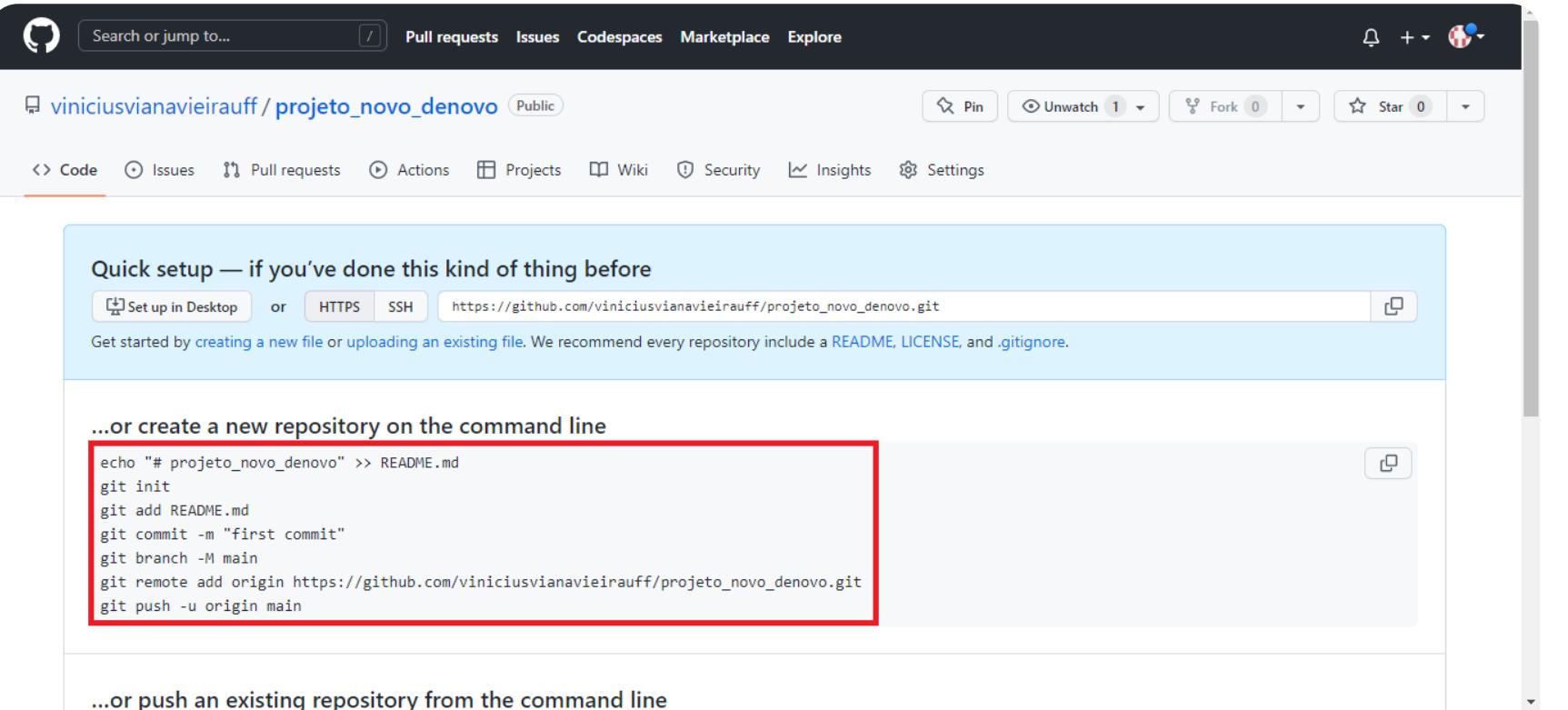
### 6.2.3. - 3<sup>a</sup> forma

A 3<sup>a</sup> forma é similar à forma anterior, mas será criado um repositório tanto no Git quanto no GitHub, e serão conectados. Pelas boas práticas de programação, é sugerido que seja o mesmo nome.

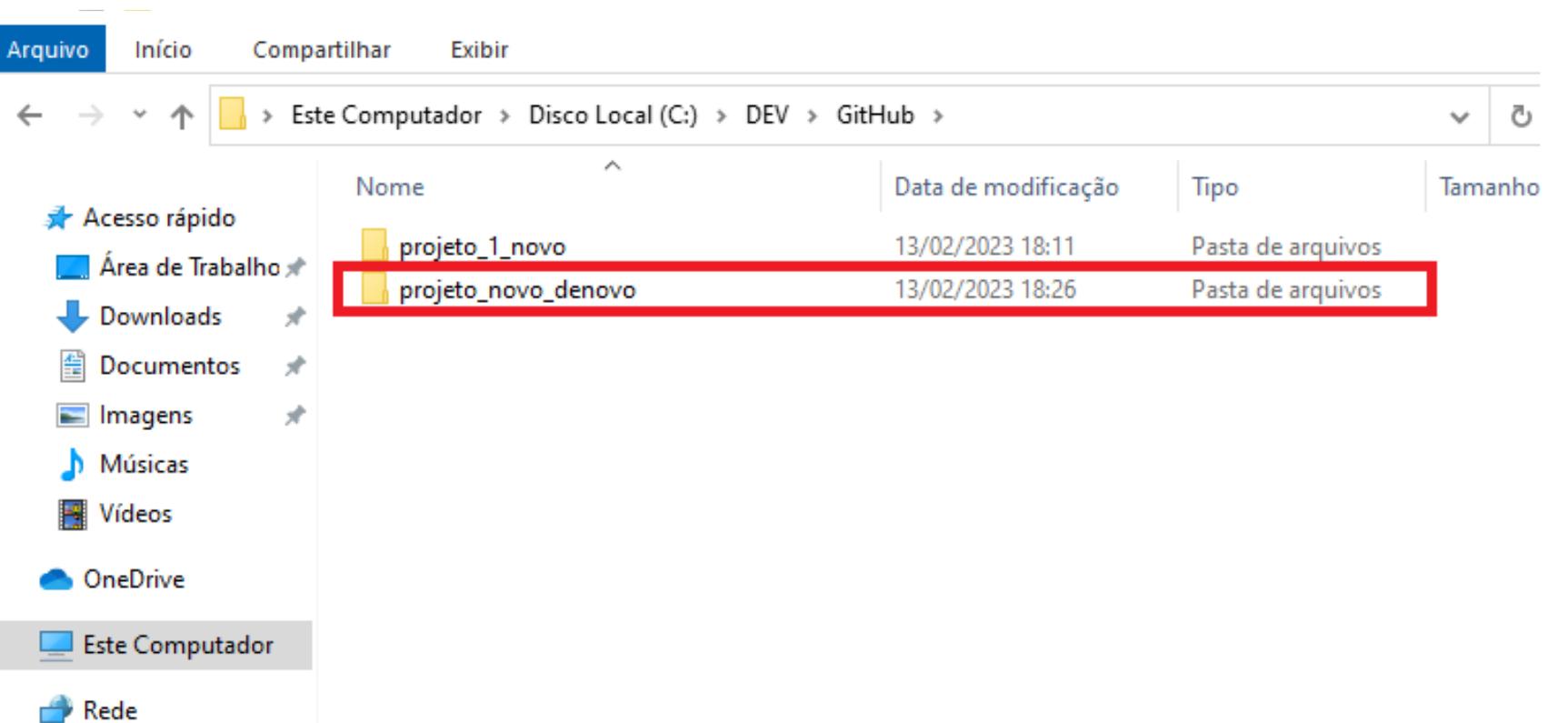
Crie um repositório no Github, mas dessa vez, crie um repositório totalmente vazio, sem os arquivos “README.md” e “.gitignore”.



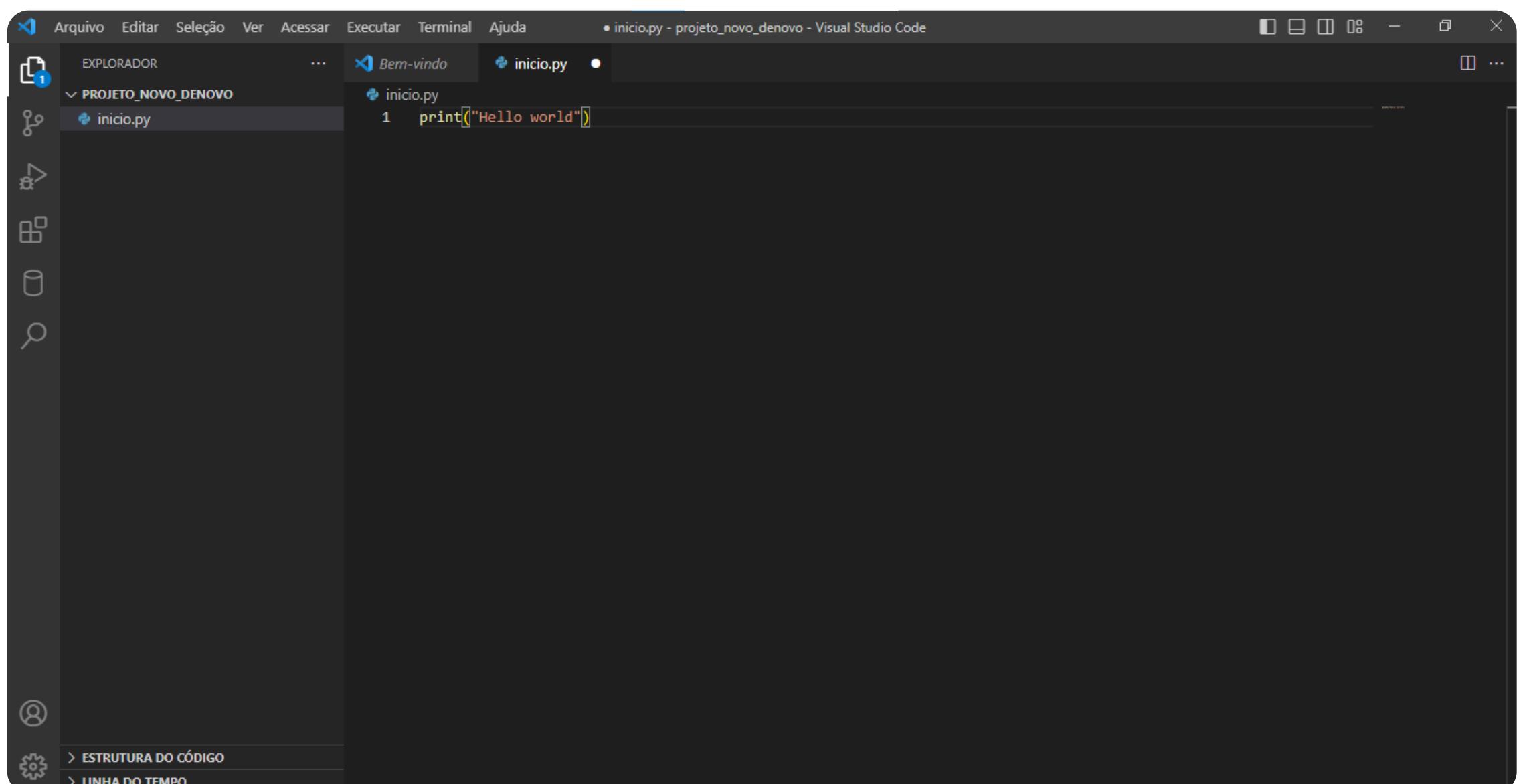
Os comandos e instruções serão dados pelo GitHub após a criação do repositório.



Criaremos uma pasta de um novo projeto na pasta "DEV":



E crie um arquivo para ser jogado dentro do GitHub.



Utilize os seguintes comandos:

```
git init #Iniciando um repositório Git
git add * #Adicionando arquivos na área de preparação
git commit -m "first commit" #Salvando as informações dentro do Git
git remote add origin https://github.com/viniciusvianavieirauff/projeto_novo_denovo.git #Conectando o seu repositório Git ao repositório no GitHub
git push -u origin main #Jogando as modificações para o GitHub
```

Depois de executar os comandos, o repositório criado no GitHub e no Git estarão disponíveis em seu computador, e você pode verificar a sincronia no GitHub para confirmar.

### 6.3. – Extraindo informações do GitHub

Imagine que você está gerenciando um projeto tanto em casa quanto no trabalho, e deseja manter seu código sincronizado entre os dois locais. Claro, clonar o repositório a cada vez que você faz uma alteração seria um processo tedioso. Felizmente, o Git oferece um comando que simplifica essa tarefa. Nesta seção, abordaremos como atualizar seu diretório local para refletir as alterações feitas no repositório do GitHub.

Para praticar essa parte, vamos criar um repositório específico. Crie uma nova pasta chamada "puxando\_alteracoes" dentro da sua pasta "DEV" e inicie um repositório Git nesta pasta. Lembrando que, para iniciar seu repositório git, utilize o comando:

"git init"

Crie uma boa prática de conferir se o arquivo “.git” está na pasta antes de iniciar um projeto.

- No Git Bash:

“ls -la”

- No PowerShell:

“Get-ChildItem -Force”

- No Command Prompt:

“dir /ah”

Crie seu arquivo “inicio.py” e “commite”, para que as alterações sejam salvas no Git. E para enviar as alterações para o GitHub, esse é o comando, mas é necessário entendê-lo antes.

```
git push -m <nome_do_repositório_remoto> <nome_da_branch>
```

Esse comando é utilizado para “exportar” todo o histórico salvo no Git localmente para um repositório remoto, que no nosso caso é o GitHub. Imagine que o “push” faça a mesma função do “commit”, mas ao invés de salvar localmente, ele salva remotamente enviando para um repositório remoto.

<nome\_do\_repositório\_remoto>

Origin é um nome convencionalmente usado para se referir ao repositório remoto no qual você clonou seu projeto local. É comum que o repositório remoto tenha o nome origin, mas você pode usar qualquer outro nome que desejar. O objetivo principal de usar o nome origin é tornar o processo de trabalhar com o repositório remoto mais fácil e intuitivo.

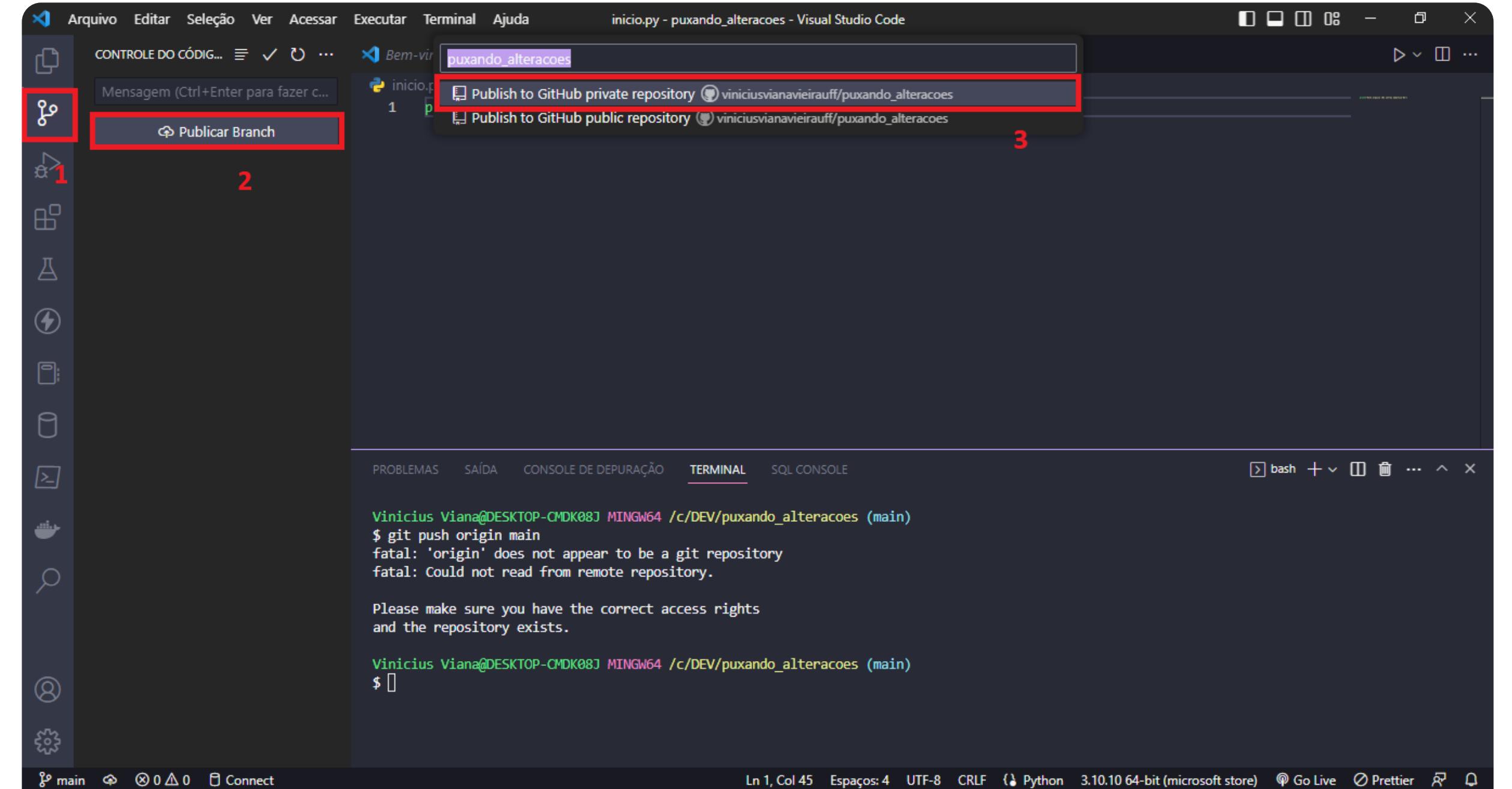
<nome\_da\_branch>

Master é o nome convencional da branch principal em um repositório Git. A branch master é a branch padrão e, geralmente, é a branch mais importante de um projeto, onde todas as atualizações são feitas e mantidas antes de serem publicadas em produção. As outras branches de um projeto são geralmente usadas para desenvolvimento de novas funcionalidades ou correções de bugs.

Devido à problemática do uso da palavra “master”, é possível alterar o nome da branch conforme sua preferência, através do comando:

“git branch -m <nome\_escolhido>”

Vamos criar um repositório público no GitHub:



A screenshot of the Visual Studio Code interface. The top bar shows the title "inicio.py - puxando\_alteracoes - Visual Studio Code". The left sidebar has various icons for file operations. The main area shows a Python file named "inicio.py" with the code: 

```
1 print("Este arquivo não sofreu modificação")
```

. Below it is a "Mensagem" (Message) panel with a "Confirmação" (Confirmation) message: "Este arquivo não sofreu modificação". The bottom part of the screen is a terminal window titled "TERMINAL" with the following content:  
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando\_alteracoes (main)  
\$ git push origin main  
fatal: 'origin' does not appear to be a git repository  
fatal: Could not read from remote repository.  
Please make sure you have the correct access rights  
and the repository exists.  
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando\_alteracoes (main)  
\$ [redacted]  
A GitHub notification message is displayed in a separate window below the terminal:  
O repositório "viniciusvianavieirauff/puxando\_alteracoes" foi  
publicado com êxito no GitHub.  
Origem: GitHub (Extensão) Abrir no GitHub

Enviaremos esse projeto e as alterações para o GitHub através do comando ensinado:

"git push origin main"

No entanto, repetir constantemente o comando "git push origin main" pode se tornar tedioso. Para simplificar o processo, podemos definir "origin main" como o caminho padrão, o que significa que nas próximas vezes, só precisamos digitar "git push". Isso economiza tempo e torna o processo mais eficiente através do comando:

"git branch -u origin/<nome\_da\_branch>"

ou

"git push -u origin <nome\_da\_branch> #esse já envia para o GitHub"

O comando "-u" é o mesmo de "--set-upstream" e é ele que define "nome\_da\_branch" como sendo padrão para o repositório "origin":

Façamos uma alteração qualquer em nosso projeto para conferir se o nosso comando funcionou:

A screenshot of the Visual Studio Code interface. The top menu bar includes Arquivo, Editar, Seleção, Ver, Acessar, Executar, Terminal, Ajuda, and inicio.py - puxando\_alteracoes - Visual Studio Code. The Explorer sidebar shows a folder named 'PUXANDO ALTERACOES' containing 'inicio.py'. The main editor area displays the following code:

```

print("Este arquivo não sofreu modificação!")
print("Fazendo outra alteração para conferir")

```

The terminal below shows the command line session:

```

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git add *

```

After running 'git add \*', the terminal continues:

```

$ git commit -m "branch padrão"
[main 665c478] branch padrão
1 file changed, 2 insertions(+), 1 deletion(-)

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 337 bytes | 337.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/viniciusvianavieirauff/puxando_alteracoes.git
 c6fda56..665c478 main -> main

```

The status bar at the bottom indicates: Ln 2, Col 45 Espaços: 4 UTF-8 CRLF Python 3.10.10 64-bit (microsoft store) Go Live Prettier.

Conferindo no GitHub:

A screenshot of a GitHub repository page. The URL is viniciusvianavieirauff/puxando\_alteracoes. The 'Code' tab is selected. The file 'inicio.py' is shown with the following content:

```

print("Este arquivo não sofreu modificação!")
print("Fazendo outra alteração para conferir")

```

The GitHub interface includes a search bar, navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore. The repository details show 1 contributor and 2 lines (2 sloc) of code with 98 Bytes.

Será feita uma simulação em que outro computador tenha realizado alterações em nosso projeto, editando o arquivo.py:

Faremos uma edição e “commitaremos”:

The screenshot illustrates a workflow for committing changes to a GitHub repository. It consists of three main sections:

- Top Left:** A screenshot of a GitHub repository page for the repository `alteracoes`. The `main` branch is selected. The file `inicio.py` is open, showing two lines of Python code:

```
1 print("Este arquivo não sofreu modificação!")
2 print("Fazendo outra alteração para conferir")
```

A red box highlights the edit icon (`edit`) in the toolbar below the code editor.
- Top Right:** A screenshot of the code editor for `inicio.py` in the `main` branch. The code has been modified to:

```
1 print("Essa alteração foi a última")
```

A red box highlights the first line of code.
- Bottom Right:** A screenshot of the "Commit changes" dialog box. The commit message field contains "Update inicio.py". Below it is a text area for an optional extended description. At the bottom, there are two radio button options:
  - Commit directly to the `main` branch.
  - Create a new branch for this commit and start a pull request. Learn more about pull requests.A red box highlights the "Commit changes" button.

Modificação realizada:

viniciusvianavieirauff / puxando\_alteracoes (Public)

**Code** Issues Pull requests Actions Projects Wiki Security Insights Settings

**main** puxando\_alteracoes / inicio.py / <> Jump to ▾

viniciusvianavieirauff Update inicio.py

1 contributor

1 lines (1 sloc) | 40 Bytes

```
1 print("Essa alteração foi a última")
```

Give feedback

No VSCode, para atualizar digite o comando:

"git pull"

Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda inicio.py - puxando\_alteracoes - Visual Studio Code

EXPLORADOR PUXANDO\_ALTERACOES inicio.py

```
1 print("Essa alteração foi a última")
```

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL SQL CONSOLE

Bem-vindo

**TERMINAL**

```
$ git pull
Updating 665c478..7a4195d
Fast-forward
 inicio.py | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

É dessa forma com que você trabalha dentro do versionamento de código. Quando você deseja fazer um "checkpoint" você faz um "push" enviando alterações para o GitHub e quando você precisa se atualizar, você faz um "pull", colocando seus arquivos em dia.

Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda inicio.py - retornando\_commit - Visual Studio Code

CONTROLE DO CÓDIGO... Mensagem (Ctrl+Enter para fazer c...) Publicar Branch

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL SQL CONSOLE

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando\_commit (nova\_branch)

```
$ git push origin nova_branch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 336 bytes | 336.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'nova branch' on GitHub by visiting:
remote: https://github.com/viniciovianavieirauff/retornando_commit/pull/new/nova_branch
remote:
To https://github.com/viniciovianavieirauff/retornando_commit.git
 * [new branch] nova_branch -> nova_branch
```

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando\_commit (nova\_branch)

```
$ git log
commit 720ec9148dff3e6505dda4c779f58ff145659e90 (HEAD -> nova_branch, origin/nova_branch)
Author: viniciusvianavieirauff <viniciusvianavieira@id.uff.br>
Date: Wed Feb 15 14:40:16 2023 -0300

primeira nova branch

commit d5ba8ea819ddaa4a1f7f653fa3454dd32120288d (origin/master, master)
Author: viniciusvianavieirauff <viniciusvianavieira@id.uff.br>
Date: Wed Feb 15 13:57:40 2023 -0300

segundo commit

commit 249194a27edc8ec5709c7710465024f51a622a5
Author: viniciusvianavieirauff <viniciusvianavieira@id.uff.br>
Date: Wed Feb 15 13:53:43 2023 -0300
```

Ln 2, Col 47 Espaços: 4 UTF-8 CRLF ⓘ Python 3.10.10 64-bit (microsoft store) ⓘ Go Live ⓘ Prettier ⓘ

## Mundo 7

### 7.1. – Gerenciando branch's no Git

Nesta seção, será ensinado como criar, manipular e gerenciar as “branchs” nos seus projetos no Git. Afinal, cada branch trabalha de maneira independente e podem ser unidas através do comando “merge”, criar atualizações e testar os códigos e juntar na branch principal, o que é muito útil quando o trabalho é em equipe.

Digitando este comando, é possível visualizar as branchs presentes em seu projeto:

```
"git branch" #branch presentes localmente
"git branch -r" #branch presentes remotamente
"git branch -a" #para ver todas as branchs
```

Para criar outra branch:

```
"git checkout -b <nome_da_branch>"
```

Automaticamente após a criação da nova branch, o Git troca da “branch main” para a “nova\_branch”

The screenshot shows a Visual Studio Code interface. In the top right corner, there is a green box with the number '61'. The code editor shows a single line of Python code: 'print("Essa alteração foi a última")'. Below the editor is a terminal window with the following content:

```
Vinicius.Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (main)
$ git checkout -b nova_branch
Switched to a new branch 'nova_branch'
Vinicius.Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/puxando_alteracoes (nova_branch)
$
```

The terminal output shows the command '\$ git checkout -b nova\_branch' being run, followed by the message 'Switched to a new branch 'nova\_branch''. The terminal tab is highlighted with a red box. The status bar at the bottom of the terminal window indicates the current branch is '(nova\_branch)'.

Vamos realizar algumas modificações na nova ramificação (branch) e, em seguida, retornaremos à ramificação principal.

```
print ("Essa alteração agora faz parte da branch nova!")
```

Após, é necessário enviar as alterações para o GitHub, através do comando:

```
git push <nome_do_repositório_remoto> <nome_da_branch>
git push origin nova_branch
```

Entendemos que o repositório padrão tem o nome "origin" e a nova ramificação se chama "nova\_branch". Agora que concluímos as alterações necessárias na nova branch, como podemos retornar à branch principal para verificar se as alterações foram incorporadas a ela? Através desse comando:

```
git checkout <nome_da_branch>
```

Caso deseje retornar à outra branch secundária, é feito da mesma forma.

Se você decidir que a nova ramificação está repleta de erros e não pode ser corrigida, você pode excluí-la usando o seguinte comando:

Observação: Lembre-se de não estar atualmente na própria ramificação que deseja deletar.

```
git branch -D <nome_da_branch>
```

Lembrando que, antes de realizar o "merge" entre a main e a branch, sempre faça o "commit"!

Para realizar o "merge", é necessário retornar à branch principal, no caso a master, ou main. Depois, envie o comando com o nome da branch que será unida na master.

```
git merge <nome_da_branch>
```

## Mundo 8

### 8.1. – Restaurando versões antigas

Agora que compreendemos o conceito de ramificações (branches), podemos avançar para o próximo tópico. Existem duas maneiras de desfazer um commit:

1. A primeira maneira permite desfazer o commit, mas ainda manter seu registro no histórico, ou seja, ele ficará armazenado de alguma forma.

2. A segunda maneira de desfazer um commit é mais drástica, pois ela remove não apenas o commit, mas também todas as alterações que ocorreram depois dele, efetivamente retornando ao estado anterior do projeto, como se você estivesse voltando no tempo.

#### 8.1.1. – 1ª Forma

Explicaremos como será feita a restauração através da primeira forma, desfazendo o commit e mantendo o seu registro no histórico. Precisamos visualizar todos os “commits” feito no seu projeto, através do comando:

```
git log --oneline
```

Após o comando, o terminal retornará a hash da commit, que seria como um ID, guarde-a.

A screenshot of Visual Studio Code showing a terminal window with a git log output and a code editor with a Python file named 'inicio.py' containing two print statements.

```
Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
f3616ba (HEAD -> master, origin/master) terceiro commit
d5ba8ea segundo commit
249194a primeiro commit

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git revert 249194a
```

A screenshot of Visual Studio Code showing a terminal window with a git log output and a code editor with a Python file named 'inicio.py' containing two print statements. A red box highlights the terminal output.

```
Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
f3616ba (HEAD -> master, origin/master) terceiro commit
d5ba8ea segundo commit
249194a primeiro commit
```

Começaremos o reset através desse comando, indicando qual commit queremos resetar, através da hash (ID) que identificamos no código anterior, e assim será aberto um editor para resolver conflitos.

**"git revert <hash-do-commit>"**

A screenshot of Visual Studio Code showing a code editor with a Python file named 'inicio.py' containing two print statements. A red box highlights the code editor interface.

```
1 <<<< HEAD (Alteração Atual)
2 print("Vamos fazer nosso segundo commit")
3 print("Em busca do terceiro commit")
4 =====
5 >>>> parent of d5ba8ea (segundo commit) (Alteração da Entrada)
```

Resolver no Editor de Mesclagem

O próprio terminal te retorna alguns possíveis comandos para te auxiliar, é bem intuitivo.

- Caso você desista do comando é só digitar: "git revert --abort"
- Caso você queira pular o commit "git revert --skip"
- Caso você queira continuar o processo "git revert --continue"

Como é possível verificar, nenhuma dessas opções atende ao nosso pedido .Portanto, prossiga em "Resolver no Editor de Mesclagem", e após feita todas as alterações necessárias na suas linhas de código, vá em "Completar Mesclagem".

O Editor de Mesclagem pode assustar de primeira, mas não é um bicho de sete cabeças. Assim que ele funciona:

Entrada: Não entrará nada

Atual: A 2º linha será apagada. ➔ print("Em busca do terceiro commit")

Resultado: Sobrarão apenas a primeira linha, que de fato era exatamente o 2º commit

```

Arquivo Editar Seleção Ver Acessar Executar Terminal Ajuda Mesclagem: inicio.py - retornando_commit - Visual Studio Code
EXPLORADOR ... inicio.py ! Mesclagem: inicio.py !
RETORNANDO_COMMIT inicio.py !
Entrada ...
1 Aceitar Entrada | Aceitar Combinação | Ignorar
1 Aceitar Atual | Aceitar Combinação | Ignorar
1 print("Vamos fazer nosso segundo commit")
2 print("Em busca do terceiro commit")
Atual ...
1 Aceitar Atual | Aceitar Combinação | Ignorar
1 print("Vamos fazer nosso segundo commit")
2 print("Em busca do terceiro commit")
1 Confílio Restante ...
PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL SQL CONSOLE
1 bash + v ... ^ x
Vinicius Viana@DESKTOP-CHDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
f3616ba (HEAD -> master, origin/master) terceiro commit
d5ba8ea segundo commit
249194a primeiro commit
Vinicius Viana@DESKTOP-CHDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git revert d5ba8ea
Auto-merging inicio.py
CONFLICT (content): Merge conflict in inicio.py
error: could not revert d5ba8ea... segundo commit
hint: After resolving the conflicts, mark them with
hint: "git add/r<pathspec>", then run
hint: "git revert --continue".
hint: You can instead skip this commit with "git revert --skip".
hint: To abort and get back to the state before "git revert",
hint: run "git revert --abort".
Vinicius Viana@DESKTOP-CHDK08J MINGW64 /c/DEV/retornando_commit (master|REVERTING)
$ [redacted]
Ln 1, Col 1 Espaços: 4 CRLF ⓘ Python 3.10.10 64-bit (microsoft store) ⓘ Go Live ⓘ Prettier ⓘ

```

Continue o processo até o término, e depois será necessário salvar essas alterações, com os comandos:

```

git add *
git commit -m "mensagem"

```

E, também, manter o Git na mesma versão do GitHub:  
`git push -f origin master #Lembrando que este caso é para branch principal`

Fim, suas alterações foram revertidas mantendo o commit à salvo.

The screenshot shows the Visual Studio Code interface. In the top bar, the menu items are: Arquivo, Editar, Seleção, Ver, Acessar, Executar, Terminal, Ajuda. The title bar says "inicio.py - retornando\_commit - Visual Studio Code". The Explorer sidebar shows a folder named "RETORNANDO\_COMMIT" containing "inicio.py". The terminal tab is active, showing the following command history:

```
Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
db663d8 (HEAD -> master, origin/master) revertendo commit
f3616ba terceiro commit
d5ba8ea segundo commit
249194a primeiro commit

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git reset --soft 249194a

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
249194a (HEAD -> master) primeiro commit

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$
```

The status bar at the bottom shows: Ln 1, Col 42, Espaços: 4, UTF-8, CRLF, Python 3.10.10 64-bit (microsoft store), Go Live, Prettier.

### 8.1.2. – 2<sup>a</sup> Forma

2<sup>a</sup> forma, é como se fosse uma viagem ao passado, pois o commit e todas as alterações feitas depois, irão sumir.

No terminal, digite o comando:

git reset <hash\_do\_commit>

O comando “reset” pode estar acompanhado de dois parâmetros diferentes:

--hard: Com esse parâmetro, todas as alterações posteriores ao commit serão descartadas.

--soft: Com esse parâmetro, as alterações serão mantidas em sua área de preparação

Utilizaremos o soft.

git reset --soft <hash\_do\_commit>

Repare que esse comando ele exclui todos os commits posteriores e mantém as alterações ainda no arquivo. A primeira versão do nosso arquivo era um arquivo vazio.

The screenshot shows a Visual Studio Code interface. In the top right, there's a terminal window titled 'Terminal' with the following command history:

```
Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
db663d8 (HEAD -> master, origin/master) revertendo commit
f3616ba terceiro commit
d5ba8ea segundo commit
249194a primeiro commit

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git reset --soft 249194a

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
249194a (HEAD -> master) primeiro commit

Vinicio Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$
```

The bottom status bar indicates the current branch is 'master+'.

Não vamos esquecer de colocar o GitHub na mesma versão do Git localmente, utilizando o comando:

- git push -f origin master #Lembrando que este caso é para branch principal

Utilizaremos a opção mais forte agora, que descarta todas as alterações. Mas antes disso, precisamos enviar "commit" qualquer, para poder ter histórico para voltar:

- git add \*
- git commit -m "segundo/terceiro commit"
- git push

Retornando à primeira versão, que de fato era um arquivo vazio. Repare que todas as alterações que tinham sido feitas após o primeiro commit, foram descartadas.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a folder named 'RETORNANDO\_COMMIT' containing a file 'inicio.py'. The main editor area has one line of code: '1'. Below the editor is a terminal window with the following history:

```
$ git log --oneline
1855984 (HEAD -> master, origin/master) segundo/terceiro commit
249194a primeiro commit

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git reset --hard 249194a
HEAD is now at 249194a primeiro commit

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$ git log --oneline
249194a (HEAD -> master) primeiro commit

Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando_commit (master)
$
```

The status bar at the bottom indicates the file is on 'master', has 0 changes, and is connected.

Não vamos esquecer de colocar o GitHub na mesma versão do Git localmente, utilizando o comando:

E atualizando o GitHub novamente antes de finalizar.

git push -f origin master #Lembrando que este caso é para branch principal

## Mundo 9

### 9.1. – Como criar atalhos no GitHub

Se você é alguém que aprecia a automação de tarefas, esta lição será particularmente interessante. Nela, aprenderemos a criar atalhos dentro do Git.

A ferramenta que nos permite economizar tempo é o "[alias]", e ela é utilizada em conjunto com o comando "[config]".

Eis como funciona:

- git config --global alias.<atalho> <comando\_git>

Com base no projeto do Mundo 8, acrescentamos uma linha e adicionamos à área de preparação.

- git add inicio.py

Vamos dar início, através de um comando simples, para que seja de fácil compreensão.

- git status

Supondo que fosse um comando grande, como poderia ser criado um atalho para esse comando?

- git config --global alias.st status

Ou seja, o que antes digitávamos "git status", agora será apenas "git st". Sim, olhando por esse código, talvez não faça sentido, mas pensando em códigos maiores que você utilizará mais à frente, será muito eficiente.

Lembrando que, você pode definir os atalhos para alguns níveis dentro do seu computador.

1) --local: Este nível é o padrão quando nenhum valor é definido no comando "git config". Ele se aplica apenas ao repositório Git em questão. Portanto, ao definir uma variável local, certifique-se de estar no repositório correto. Esses valores estão armazenados no arquivo ".git" em git/config.

2) global: Este nível implica que a configuração será definida para um usuário específico no sistema operacional. Os valores podem ser encontrados no arquivo ".gitconfig" na pasta home do usuário.

3) system: Este é o nível mais abrangente, aplicado a todo o sistema operacional e superando as configurações de usuários.

## 9.2. – Atalho para 2 ou mais comandos

Vimos que é bem fácil criar um atalho no GitHub, mas também é possível criar apenas um atalho para vários comandos em sequência, mas lembre-se, os comandos devem ser interligados com "&&" e iniciar com "!", por exemplo:

- git config --global alias.<atalho> '!<comando1> && <comando2> && <comando3>'

Para elucidar, vamos criar um comando em que: adicione o arquivo na área de preparação, envie para o Git e depois para o GitHub.

- git config --global alias.acp '!git add \* && git commit && git push'

Como pode notar, não incluímos a mensagem de "commit", e é obrigatório fornecer uma mensagem para cada "commit". Portanto, o Git abrirá automaticamente um terminal solicitando uma mensagem.

Quando o terminal abrir, solicitará que insira as mensagens para o "commit", informa que linhas iniciadas com "#" serão ignoradas e alerta que o "commit" será cancelado se não houver mensagens. Além disso, indica para qual "branch" estamos enviando e para qual repositório:

- "branch" git: "on branch master"
- "branch" GitHub: "origin/master" no repositório "origin" e "branch" "master".

O terminal também lista os arquivos que estão sendo modificados e "committados".

Para digitar, clique três vezes em qualquer tecla até que o cursor apareça na parte superior.

Para sair, pressione [esc], digite [:] e insira [wq] (write and quit).

Finalizado, arquivos commitados e salvos no GitHub.

The screenshot shows a Visual Studio Code interface. On the left is the Explorer sidebar with a folder named 'RETORNANDO\_COMMIT' containing 'inicio.py'. The main editor area shows the code: 

```
1 print("Vamos criar atalhos nessa aula!")
```

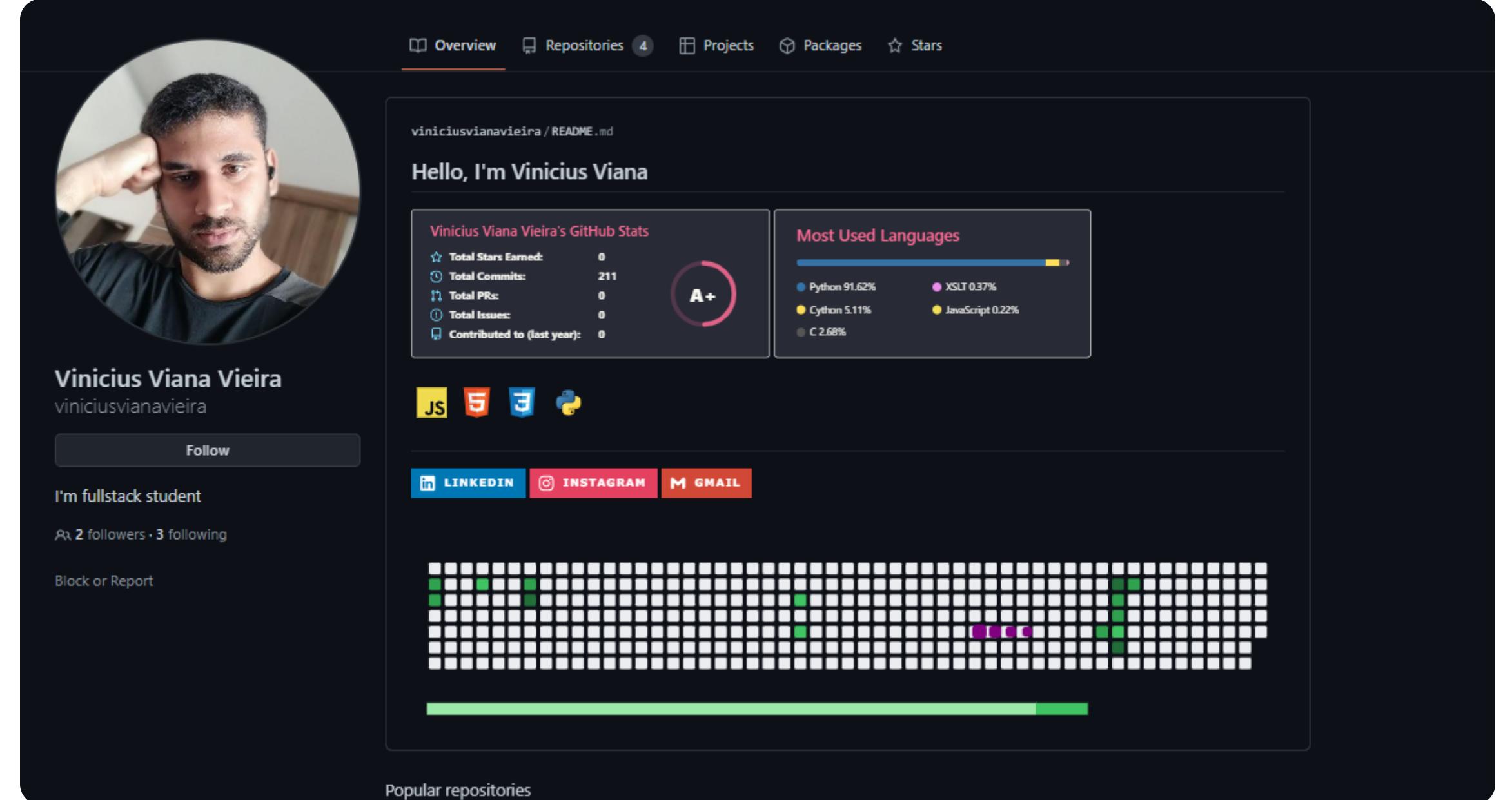
. Below the editor is the Terminal tab, which displays a terminal session:  
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando\_commit (master)  
\$ git config --global alias.acp '!git add \* && git commit && git push'  
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando\_commit (master)  
\$ git acp  
[master ac97c2c] commitando com atalho  
 1 file changed, 1 insertion(+), 1 deletion(-)  
Enumerating objects: 5, done.  
Counting objects: 100% (5/5), done.  
Writing objects: 100% (3/3), 290 bytes | 290.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/viniciusvianavieirauff/retornando\_commit.git  
 e0b784b..ac97c2c master -> master  
Vinicius Viana@DESKTOP-CMDK08J MINGW64 /c/DEV/retornando\_commit (master)  
\$

## Mundo 10

### 10.1. - Estilizando seu GitHub

O GitHub é como se fosse um LinkedIn, onde você pode estar guardando e compartilhando seus projetos, e é aconselhável que esteja sempre organizado, afinal, nunca se sabe quando um recrutador pode entrar no seu perfil.

Nesta seção, você será instruído a ter um GitHub parecido com esse:



O intuito é mostrar como funciona a dinâmica, para que você possa estar fazendo seu próprio estilo de personalização.

Começaremos, criando o bloco de “git status”, que é este aqui:

Ele nos diz algumas estatísticas do perfil, como a quantidade de commits.



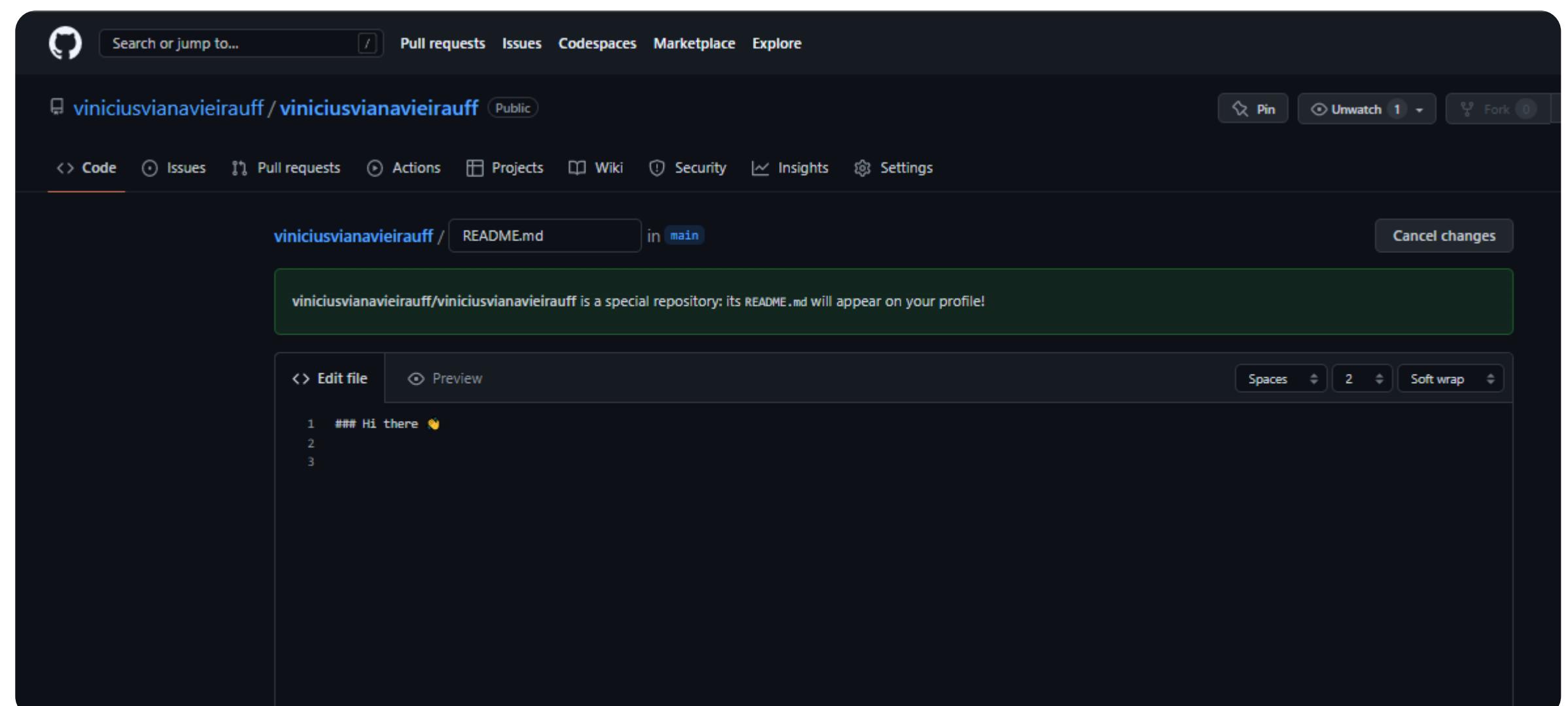
Primeiro passo é criar um repositório, como ensinado no Mundo 5.

Observe que assim que você define o nome do seu repositório, o GitHub envia uma mensagem indicando que este repositório é especial e sugere adicionar um arquivo README.md. Esse arquivo é o local onde seria feita todas as alterações e documentações dos arquivos.

O README.md é um arquivo de texto salvo com a extensão Markdown. Que é uma linguagem de marcação, assim como o HTML, que permite formatar o texto.

Além de aceitar o Markdown, este arquivo aceita HTML e CSS que é exatamente o que utilizaremos para formatação do nosso portfólio.

Iniciaremos nossa edição:

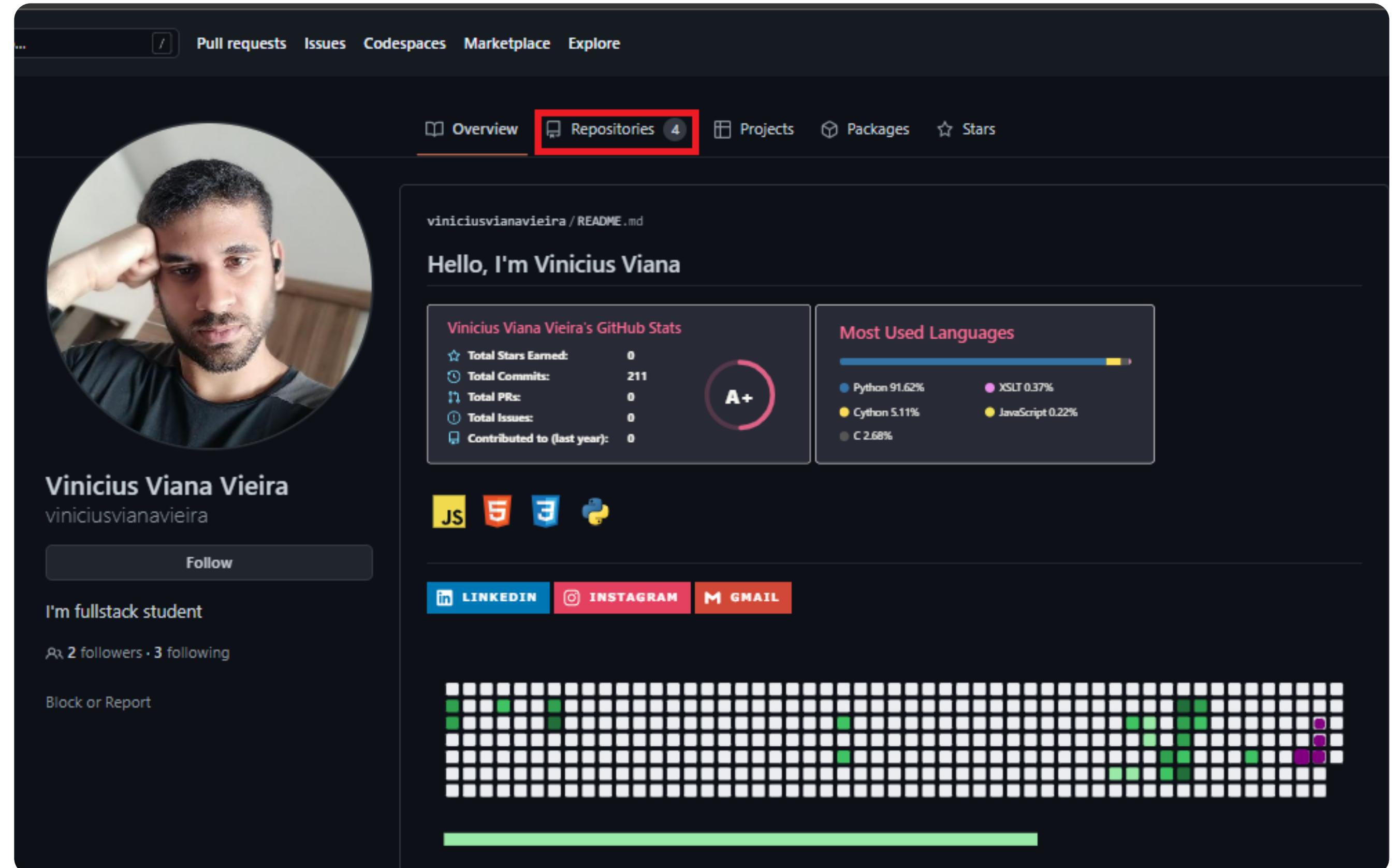


Substitua o "Hi there" por alguma mensagem que te deixe mais confortável, é como se fosse uma breve introdução, algumas pessoas colocam em inglês, por conta dos recrutadores. Se notarmos, há 3 hashtags, elas indicam o tamanho do texto, quanto mais hashtags, menor seu texto, 1# é como se fosse h1, 2# seria h2, e assim sucessivamente.

E assim eu coloquei:

```
Hello, my name is Vinicius, and welcome to my GitHub 😊## Hello,
my name is Vinicius, and welcome to my GitHub 😊
```

O próximo passo é inserir o GitStatus. No entanto, não se preocupe em decorar ou entender o código, pois é um código HTML e CSS criado por outras pessoas. Acesse meu perfil e clique em editar no meu README.md. Ao fazer isso, ele criará automaticamente um "fork", que é uma configuração prévia do GitHub (pode ser apagada posteriormente).



A screenshot of a GitHub user profile for `viniciusvianavieira`. The profile picture is a circular photo of a man with dark hair and a beard, resting his head on his hand. Below the profile picture, the user's name is displayed as **Vinicius Viana Vieira** and their GitHub handle as `viniciusvianavieira`. A 'Follow' button is present. The user has four repositories listed:

- viniciusvianavieira** (Public): Description: "Config files for my GitHub profile.", tags: config, github-config, updated 3 hours ago.
- coleta\_odds** (Public): Description: "I'm fullstack student", tags: Python, updated on Jan 13.
- Curso-FullStack** (Public): Description: "JavaScript", tags: JavaScript, updated on Mar 31, 2022.
- Curso-Python** (Public)

A screenshot of a GitHub repository page for `viniciusvianavieira / viniciusvianavieira`. The repository is public and contains 2 branches and 0 tags. The main file listed is `README.md`. The content of the `README.md` file is:

```
Hello, I'm Vinicius Viana
```

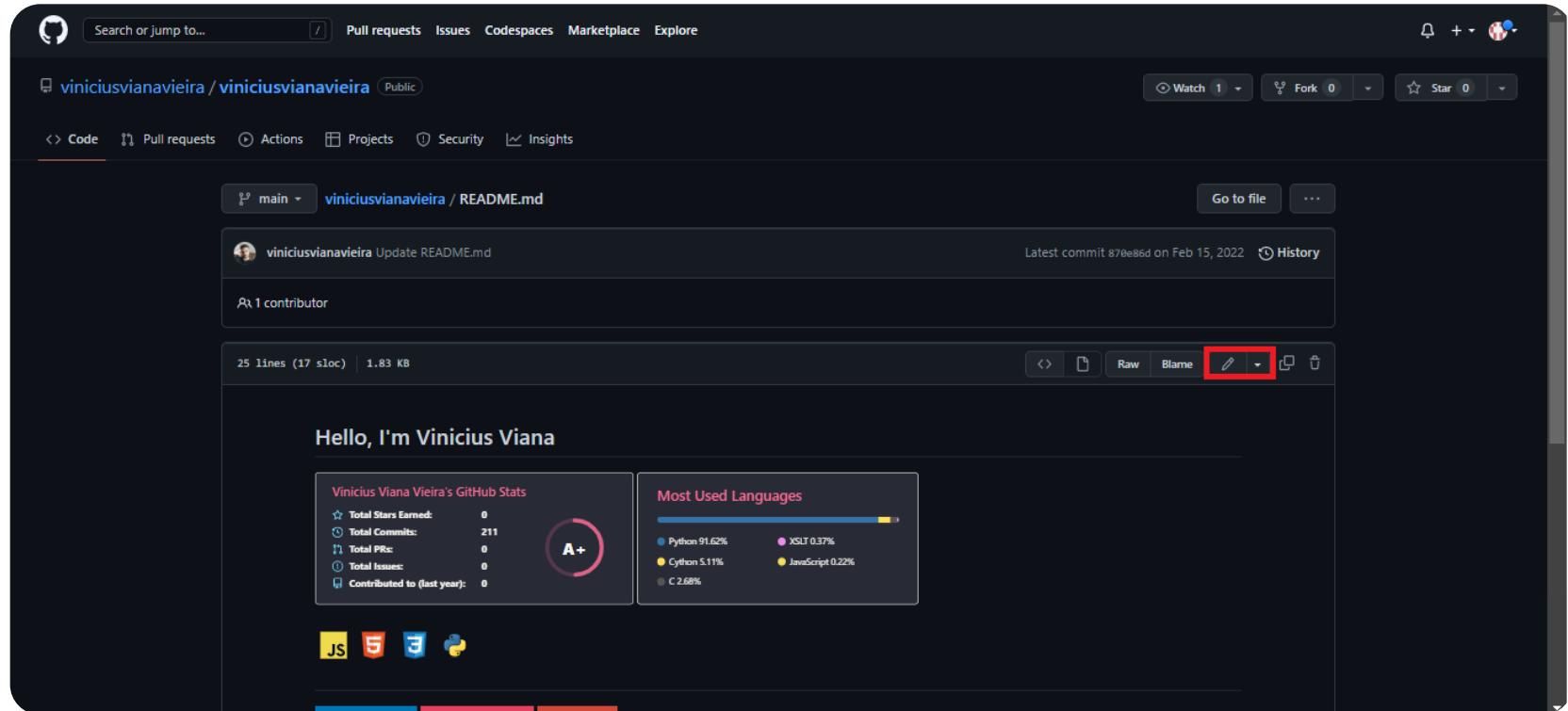
The page also displays Vinicius Viana Vieira's GitHub Stats and Most Used Languages.

**Vinicius Viana Vieira's GitHub Stats**

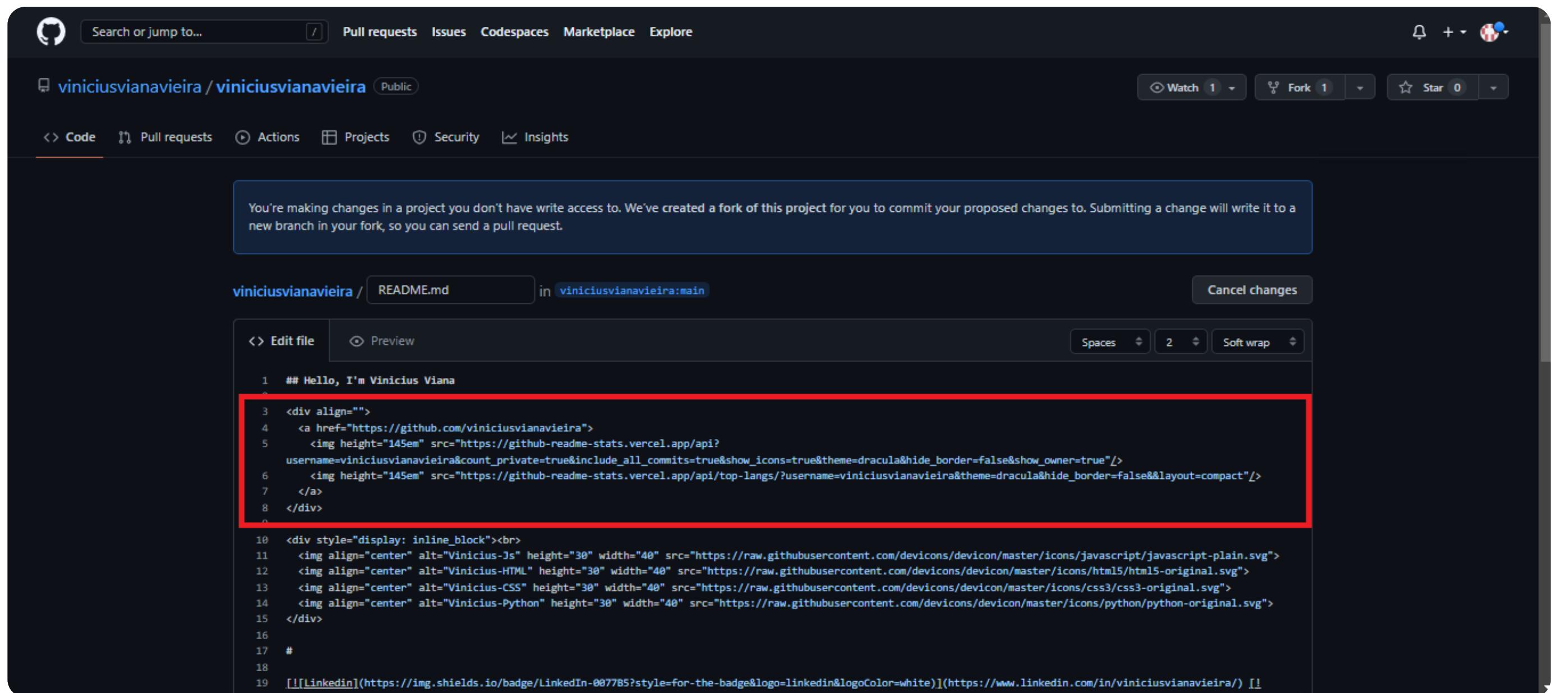
- Total Stars Earned: 0
- Total Commits: 211
- Total PRs: 0
- Total Issues: 0
- Contributed to (last year): 0

**Most Used Languages**

| Language   | Percentage |
|------------|------------|
| Python     | 91.62%     |
| Cython     | 5.11%      |
| XSLT       | 0.37%      |
| JavaScript | 0.22%      |
| C          | 2.68%      |



Na aba de códigos deste projeto, ele foi codificado em HTML, que funciona através de parâmetros, e o parâmetro que precisamos, chama "div".



```
<div align="center">

</div>
```

O link que aparece após o "src" é de um site responsável por exibir os modelos no "GitStatus".

Esse é o link do GitHub para que você possa explorar e entender cada detalhe. É bastante interessante, com muitas opções que podemos implementar em nosso repositório.

<https://github.com/anuraghazra/github-readme-stats>

Nesse código em HTML acima, ele pode ser modificado conforme seu gosto, seja para modificar o tamanho ou tema, é subjetivo. Dentro do parâmetro < a >, indicará para onde será direcionado após o clique, modifique e insira um link de sua escolha. Os links restantes referem-se aos card's.

O próximo código, refere-se às imagens que foram utilizadas, representam as linguagens de programação, e já está configurado com alinhamento, altura e largura. Lembrando que é personalizável.

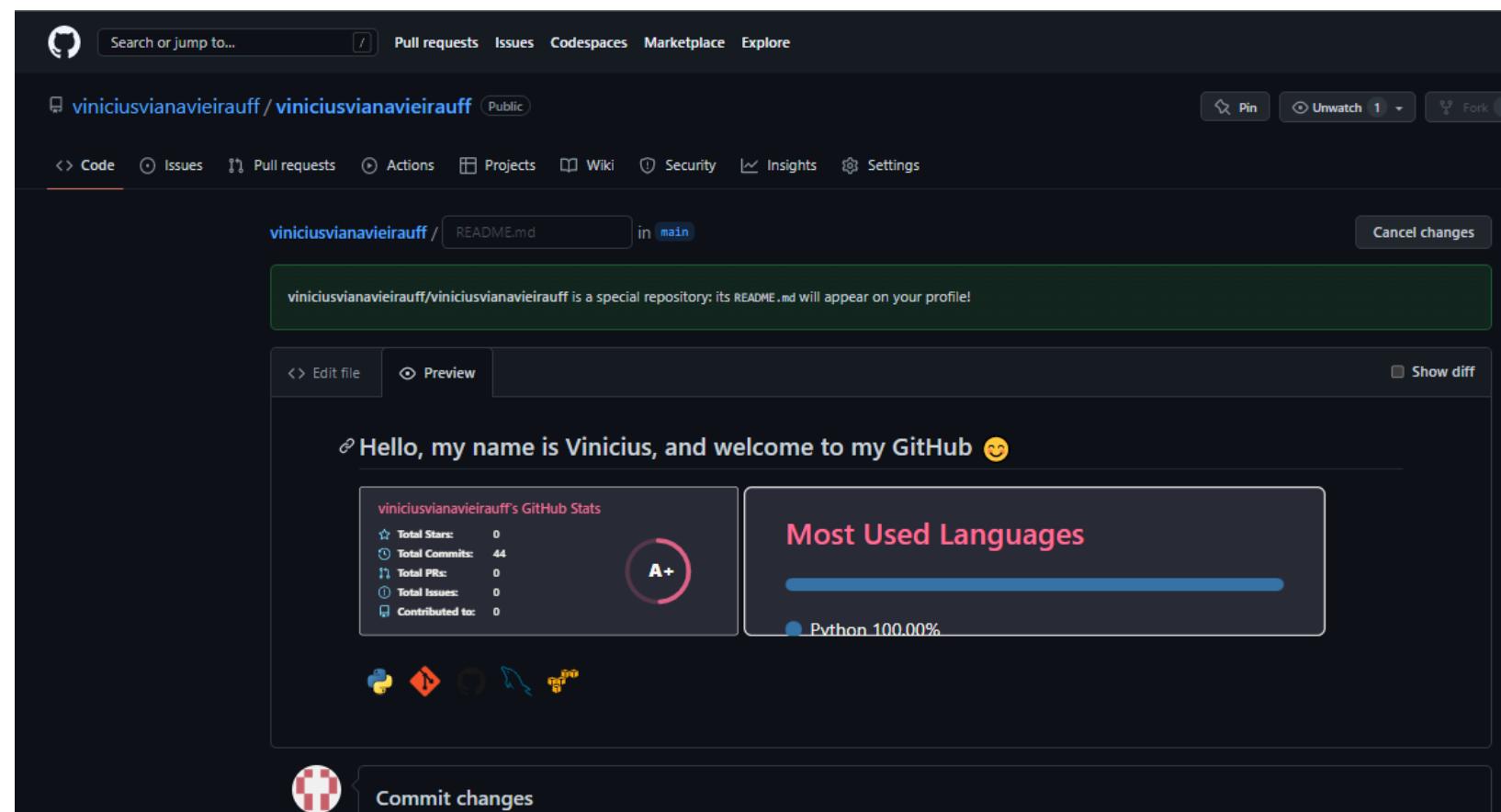
```
<div style="display: inline_block">

</div>
```

O "src" é a parte onde inserimos o ícone, mas é importante notar que não podemos simplesmente pegá-lo de qualquer lugar, deixarei um site apropriado. Neste site, utilize imagens "SVG version", copie o código e insira no arquivo README.md, e personalize.

<https://devicon.dev/>

Seu GitStatus está começando a ganhar forma:



Chegou a hora de inserir o ícone das redes sociais, para facilitar o contato do recrutador. Utilizaremos o site:

[https://dev.to/envoy\\_/150-badges-for-github-pnk](https://dev.to/envoy_/150-badges-for-github-pnk)

Após copiar o código, construiremos assim em seu README.md:

```
#
[![LinkedIn](https://img.shields.io/badge/LinkedIn-0077B5?style=for-the-badge&logo=linkedin&logoColor=white)](https://www.linkedin.com/in/viniciusvianavieira/)
[![Instagram](https://img.shields.io/badge/Instagram-E4405F?style=for-the-badge&logo=instagram&logoColor=white)](https://www.instagram.com/viniciusvianavieira/)
[![Gmail](https://img.shields.io/badge/Gmail-D14836?style=for-the-badge&logo=gmail&logoColor=white)](https://mail.google.com/mail/u/0/#inbox?compose=CllgCKCCSPQpzJvtTrjrwqGPXHDzxxMshlpTXdwNrBQBXLhVJSTtDXNjsfnrcFcpRMmzsxZDq)
```

Agora, para a última etapa, vamos colocar a cobrinha do commit. Mas precisamos habilitar as permissões necessárias.

Vá no seu repositório, procure por "Settings" e por fim em "general". Habilite as permissões:

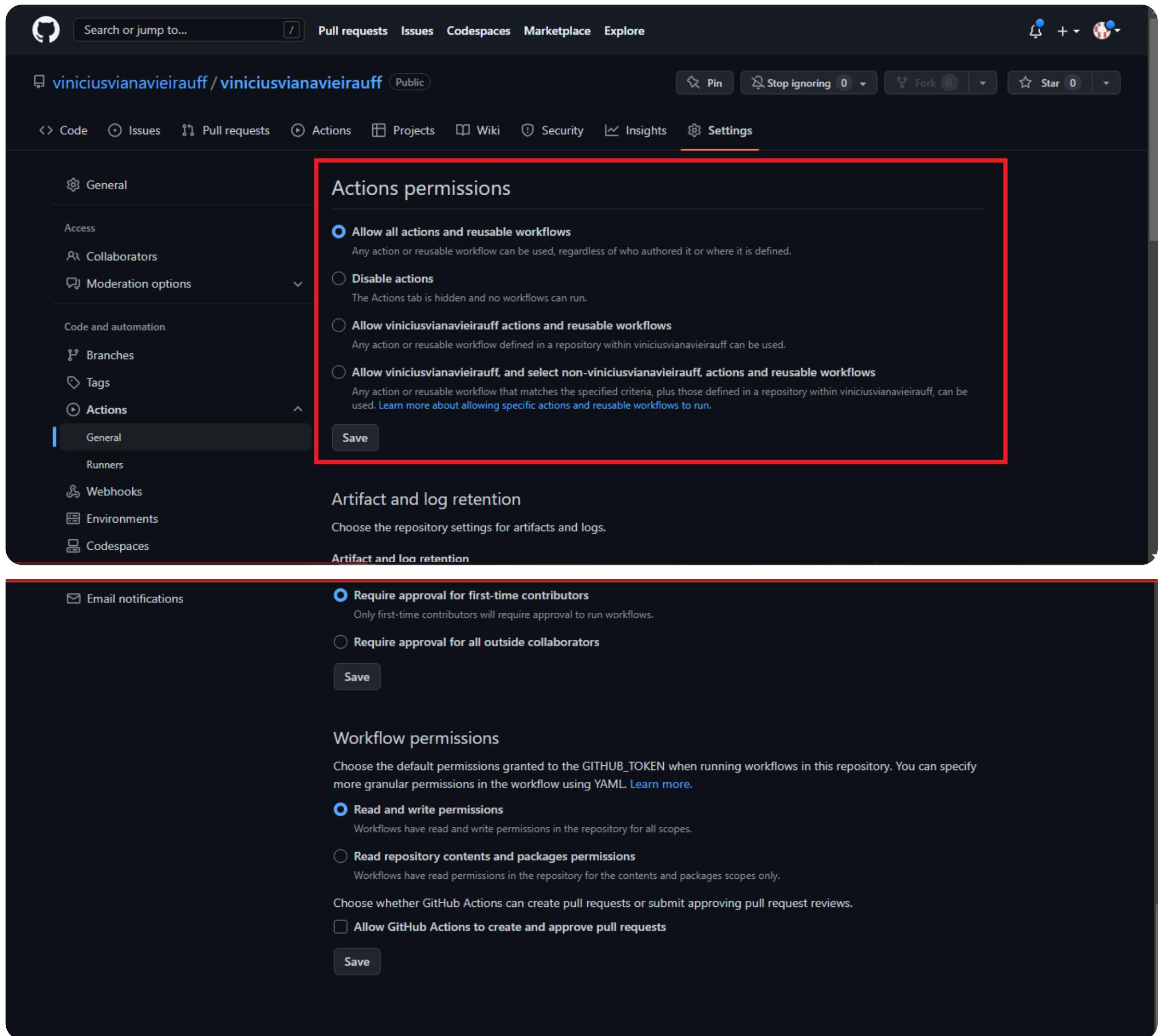
1

General

Actions

Social Preview

Habilite essas permissões:

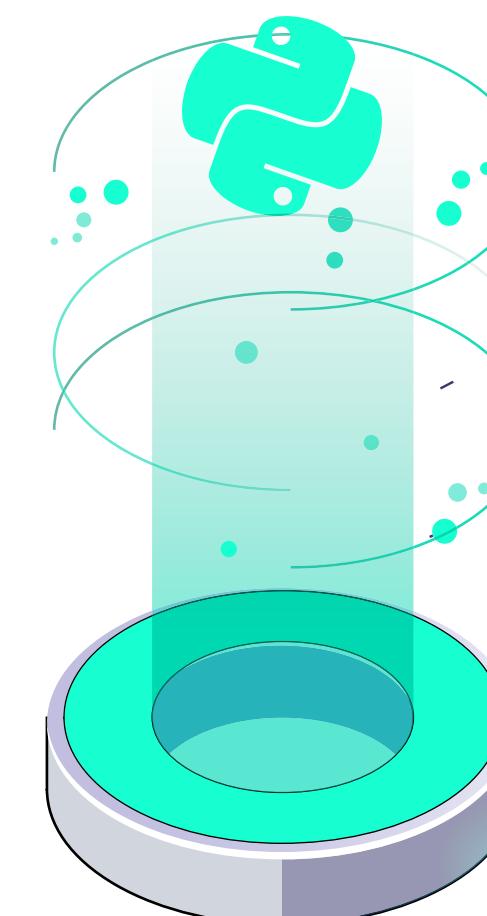


The screenshot shows the GitHub repository settings page for the repository "viniciusvianavieirauff/viniciusvianavieirauff". The "Actions" tab is selected under the "General" settings. A red box highlights the "Actions permissions" section. It contains four options:

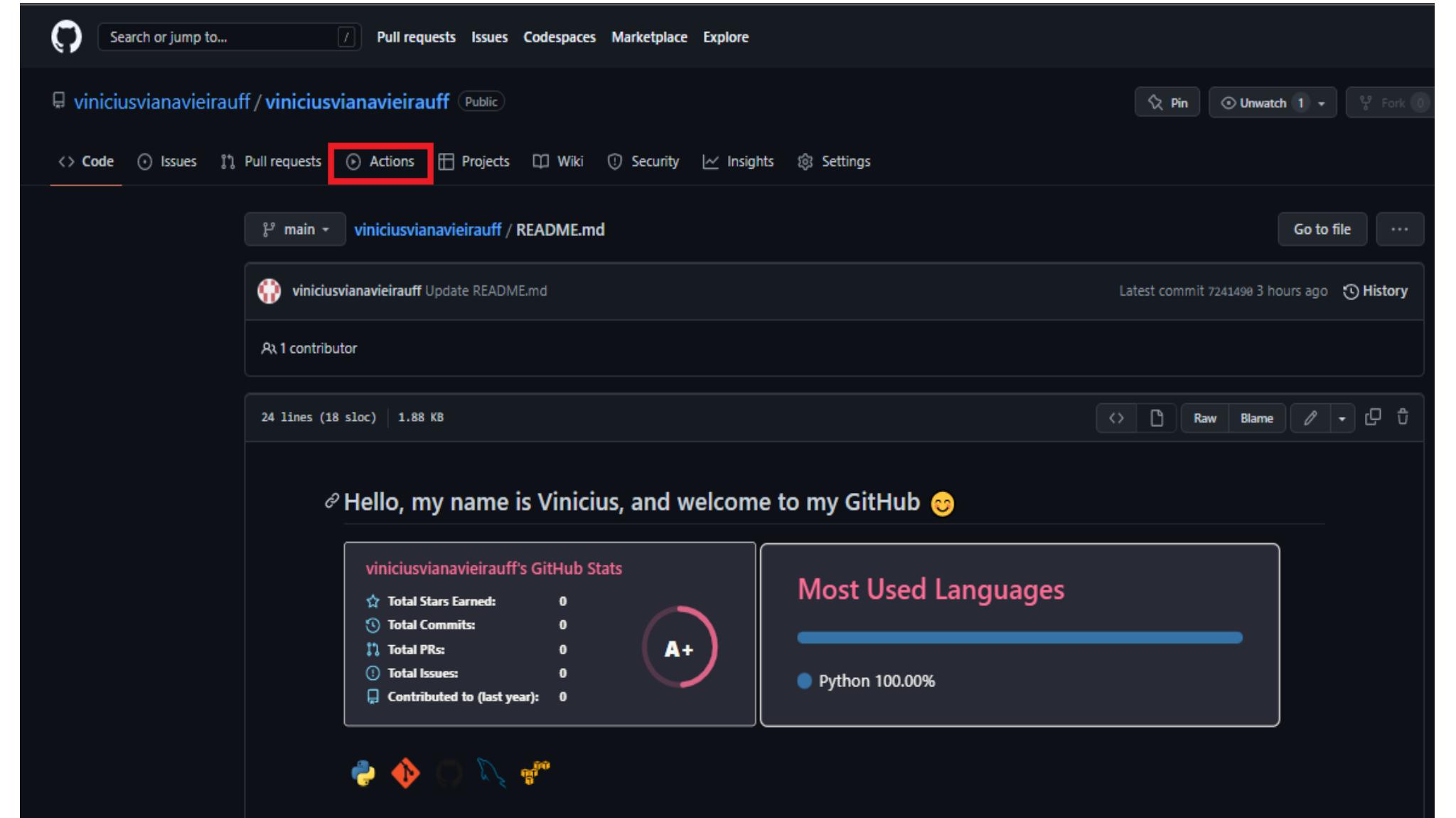
- Allow all actions and reusable workflows: Any action or reusable workflow can be used, regardless of who authored it or where it is defined.
- Disable actions: The Actions tab is hidden and no workflows can run.
- Allow viniciusvianavieirauff actions and reusable workflows: Any action or reusable workflow defined in a repository within viniciusvianavieirauff can be used.
- Allow viniciusvianavieirauff, and select non-viniciusvianavieirauff, actions and reusable workflows: Any action or reusable workflow that matches the specified criteria, plus those defined in a repository within viniciusvianavieirauff, can be used. [Learn more about allowing specific actions and reusable workflows to run.](#)

Below this section is the "Artifact and log retention" section, which is currently collapsed. At the bottom of the "Actions" tab, there is a "Save" button.

Retorne ao README.md e insire este comando, mudando para seu nome e salve as alterações com o commit.



Depois vá em “Actions”, depois clique em “configure” simple workflow, e estará visualizando uma página com códigos, apague por completo e insira um novo, alterando também o nome do usuário, e salve as alterações com um “start commit”. Retorne ao README.md e insire este comando, mudando para seu nome e salve as alterações com o commit.



The screenshot shows the GitHub Actions setup interface for a repository named 'viniciusvianavieirauff/viniciovianavieirauff'. The top navigation bar includes 'Pull requests', 'Issues', 'Codespaces', 'Marketplace', and 'Explore'. Below the repository name, there are links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. A prominent section titled 'Choose a workflow' contains the text: 'Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow or skip this step.' A red box highlights the blue link 'Skip this and set up a workflow yourself →'. Below this is a search bar labeled 'Search workflows'. A section titled 'Suggested for this repository' features a card for a 'Simple workflow' by GitHub, which starts with a file named 'blank.yml'. A 'Configure' button is present. At the bottom, there's a 'Deployment' section with three cards: 'Deploy Node.js to Azure Web App' (By Microsoft Azure), 'Deploy to Amazon ECS' (By Amazon Web Services), and 'Build and Deploy to Google Cloud'.

The screenshot shows the GitHub Actions configuration interface for the 'blank.yml' file in the '.github/workflows' directory of the repository 'viniciusvianavieirauff/viniciovianavieirauff'. The top navigation bar is identical to the first screenshot. The main area displays the YAML code for the workflow:

```
1 # This is a basic workflow to help you get started with Actions
2
3 name: CI
4
5 # Controls when the workflow will run
6 on:
7 # Triggers the workflow on push or pull request events but only for the "main" branch
8 push:
9 branches: ["main"]
10 pull_request:
11 branches: ["main"]
12
13 # Allows you to run this workflow manually from the Actions tab
14 workflow_dispatch:
15
16 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
17 jobs:
18 # This workflow contains a single job called "build"
19 build:
20 # The type of runner that the job will run on
21 runs-on: ubuntu-latest
22
23 # Steps represent a sequence of tasks that will be executed as part of the job
24 steps:
```

Below the code, a note says 'Use **Control** + **Space** to trigger autocomplete in most situations.'

```

Nome da Actions:
name: Snake Game

Controlador do tempo que sera feito a atualização dos arquivos.
on:
 schedule:
 # Será atualizado a cada 5 horas.
 - cron: "0 */5 * * *"

Permite executar na na lista de Actions (utilizado para testes de build).
workflow_dispatch:

Regras
jobs:
 build:
 runs-on: ubuntu-latest
 steps:
 # Checks repo under $GITHUB_WORKSHOP, so your job can access it
 - uses: actions/checkout@v2

Repositorio que será utilizado para gerar os arquivos.
- uses: Platane/snk@master
 id: snake-gif
 with:
 github_user_name: brennosullivan#Seu usuario
 gif_out_path: dist/github-contribution-grid-snake.gif
 svg_out_path: dist/github-contribution-grid-snake.svg

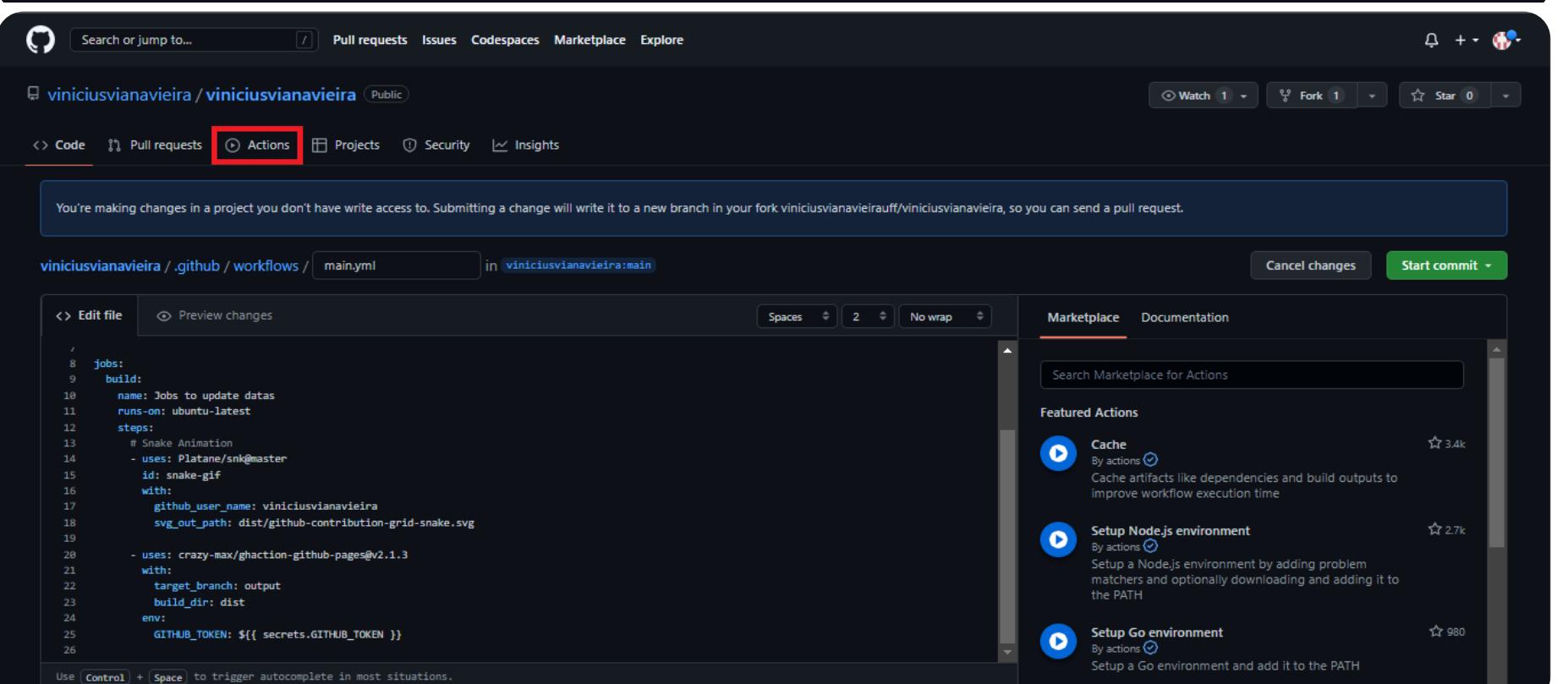
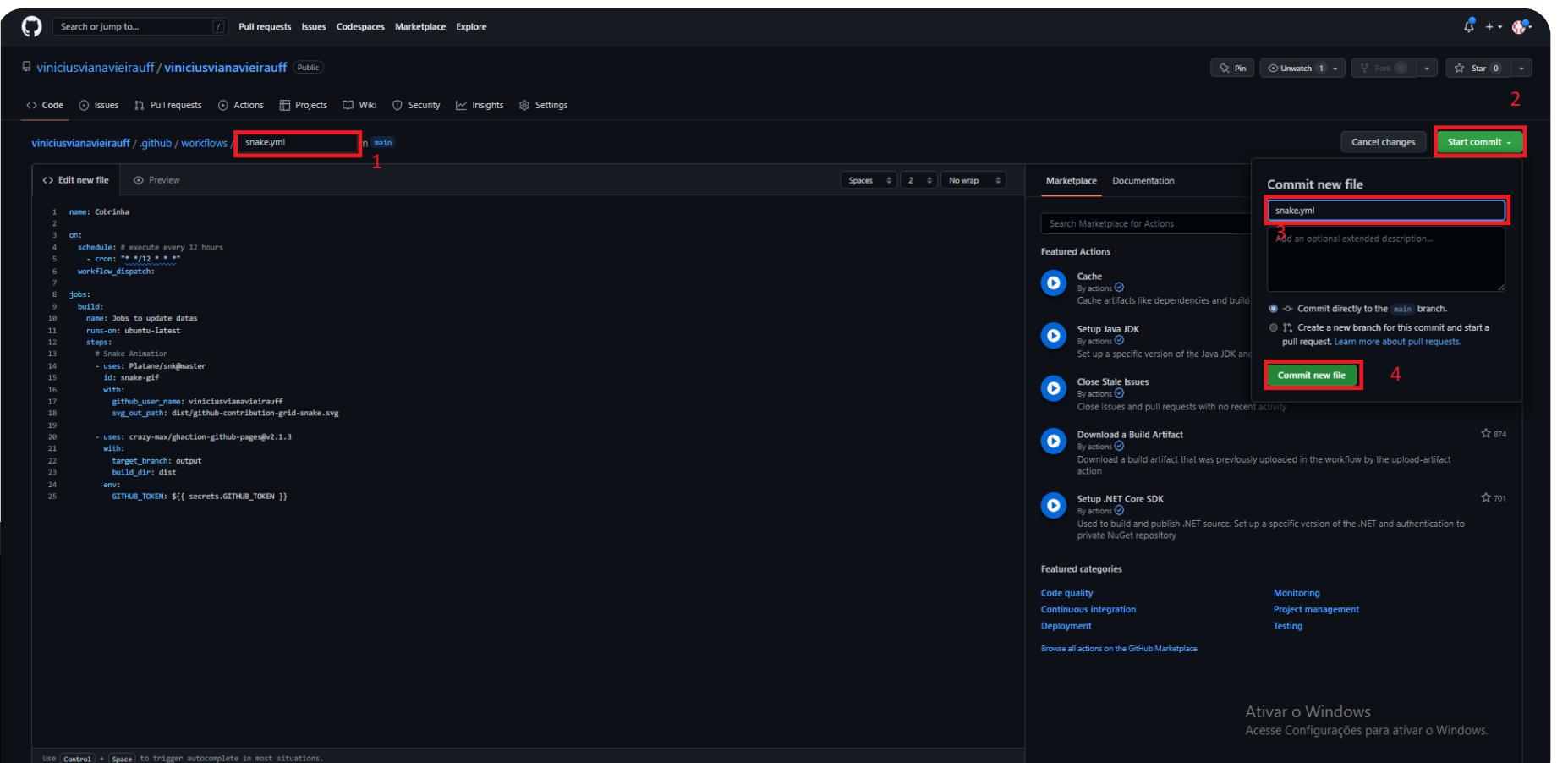
 - run: git status

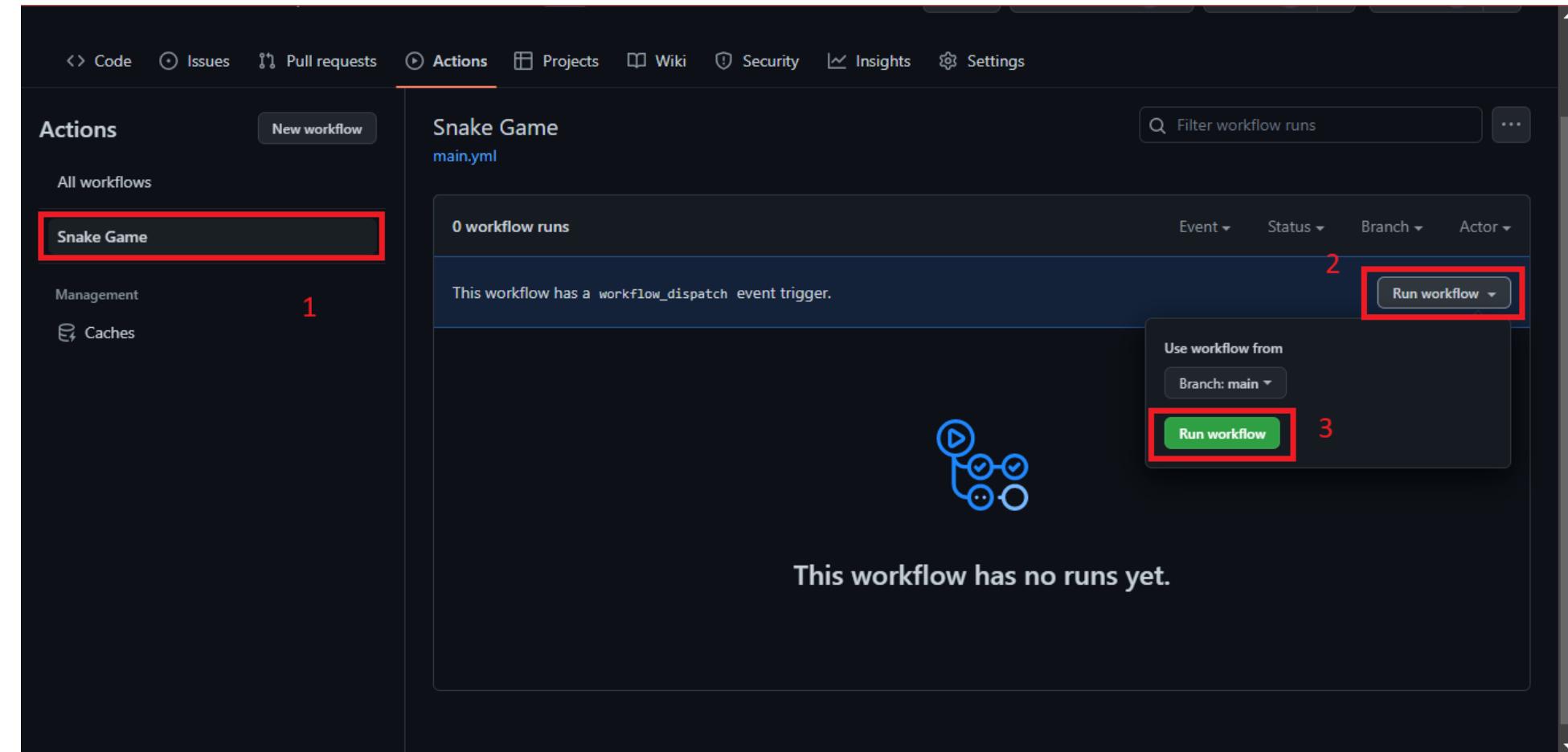
Para as atualizações.
- name: Push changes
 uses: ad-m/github-push-action@master
 with:
 github_token: ${{ secrets.GITHUB_TOKEN }}
 branch: master
 force: true

- uses: crazy-max/ghaction-github-pages@v2.1.3
 with:
 # the output branch we mentioned above
 target_branch: output
 build_dir: dist
 env:
 GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

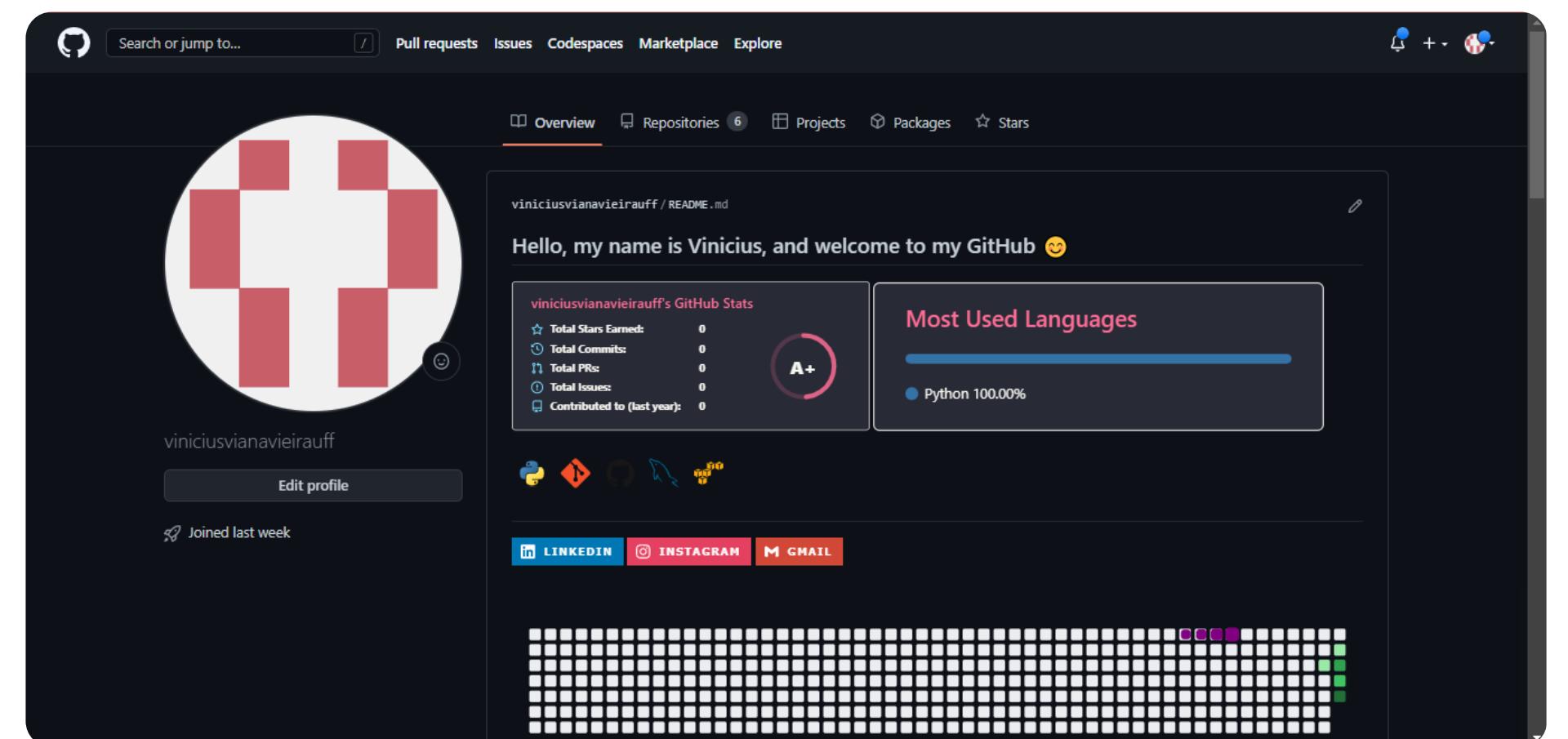
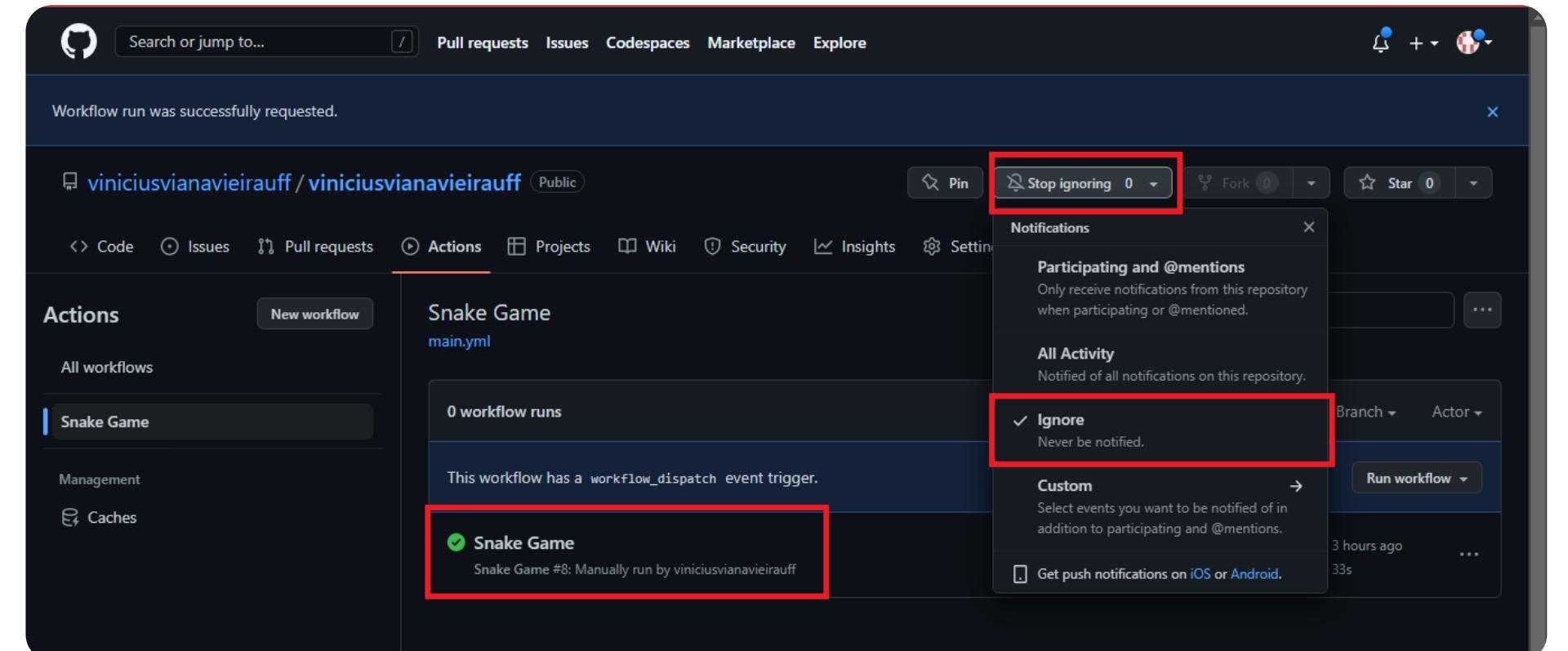
```

Para finalizar, siga estes comandos





E pronto, só conferir se está tudo certo (é se estiver verde), e não esqueça de configurar para que as mensagens não cheguem no seu email, se não, vai chegar mensagem de 12 em 12 minutos.



E esse será seu portfólio!