

MINI E-BOOK SOBRE GIT E GITHUB

Um pequeno guia sobre
versionamento de
arquivos feito para
iniciantes.

FERNANDA CARLA DE FREITAS NASCIMENTO

<https://github.com/nascimentofernanda>



MINI E-BOOK SOBRE GIT E GITHUB

Um pequeno guia sobre versionamento de arquivos feito para iniciantes.

Se você está iniciando na área da tecnologia provavelmente ouviu falar sobre o Git e o Github, e se, assim como eu você ficou meio perdido nos primeiros contatos com esses dois, esse mini e-book pode te ajudar.

“Cuidado com gente que não tem dúvida. Gente que não tem dúvida não é capaz de inovar, de reinventar, não é capaz de fazer de outro modo. Gente que não tem dúvida só é capaz de repetir.”

Mario Sergio Cortella

O desenvolvimento de software é uma jornada complexa, que requer colaboração, precisão e organização. Conforme as equipes de desenvolvimento crescem e os projetos se tornam mais ambiciosos, a necessidade de um sistema eficiente de controle de versão se torna imprescindível. Neste cenário, surge o Git, uma poderosa ferramenta de controle de versão distribuído que revolucionou a forma como os desenvolvedores gerenciam e colaboram em seus projetos.

Este mini e-book tem como objetivo proporcionar aos leitores uma pequena jornada através do mundo do Git. Para você que é um iniciante que busca entender os princípios básicos do controle de versão e quer mergulhar no universo do Git, espero que este e-book possa te auxiliar.

1. O que é o Git?

O Git é um sistema de controle de versão distribuído, desenvolvido em 2005 por Linus Torvalds, o mesmo criador do kernel Linux. Sua principal finalidade é gerenciar o versionamento de arquivos e códigos fonte, permitindo que várias pessoas possam trabalhar em um mesmo projeto de forma colaborativa e coordenada, mantendo um histórico detalhado de todas as alterações realizadas.

Internamente o Git cria conjuntos de dados e metadados para armazenar o histórico de um projeto monitorado pela ferramenta. A esses conjuntos de dados damos o nome de objetos **Git**, e eles podem ser de 3 tipos: **blobs, trees e commits**.

No contexto do Git, "blobs", "trees" e "commits" são os três principais tipos de objetos usados para armazenar e representar diferentes partes de um repositório e seu histórico. Eles desempenham papéis fundamentais no funcionamento interno do Git.

- **Blobs**

Um blob representa a parte básica e indivisível de um arquivo. Ele armazena o conteúdo real de um arquivo em formato binário. Quando você cria ou modifica um arquivo no Git, ele é convertido em um blob e, em seguida, é armazenado no banco de dados do Git, associado a um hash SHA-1 exclusivo, que é utilizado para referenciá-lo posteriormente. Os blobs são imutáveis, o que significa que, sempre que um arquivo é alterado, um novo blob é criado e armazenado no banco de dados, mantendo o histórico de alterações.

- **Trees**

Um tree é um objeto que corresponde a um diretório no sistema de arquivos do Git. Ele contém referências para blobs (objetos binários) e outras árvores, organizando assim a estrutura de diretórios do repositório. Quando você cria um diretório ou faz modificações nos arquivos dentro dele, o Git cria uma tree que registra essas mudanças e aponta para os blobs correspondentes aos arquivos modificados. As trees também são imutáveis, e sempre que ocorrem mudanças em um diretório, uma nova tree é criada.

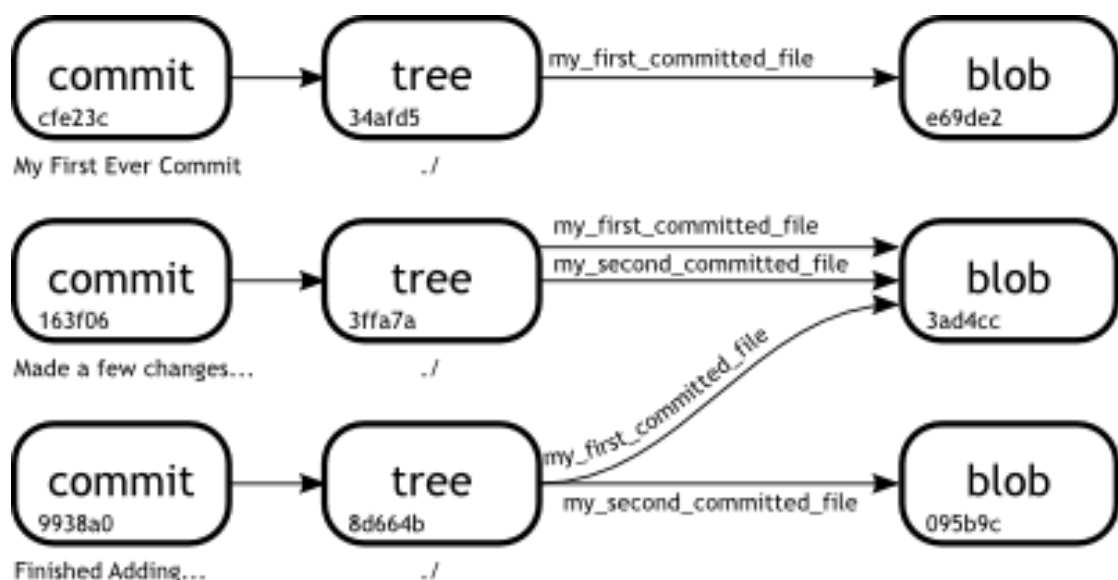
- **Commit**

Um commit é um objeto que representa uma versão específica do repositório em um determinado momento no tempo. Cada commit contém informações como:

- Um ponteiro para a tree que representa o estado do projeto naquele momento.
- O autor e o committer (pessoa que fez o commit e a pessoa que o confirmou).
- Mensagem de commit que descreve as alterações realizadas.
- Um ou mais hashes de commits pai, que formam a linha do tempo e indicam a sequência de commits que levaram àquele estado do repositório.

Os commits são imutáveis, o que significa que, uma vez criados, não podem ser alterados. Isso garante a integridade do histórico do projeto e fornece uma trilha de auditoria completa para todas as alterações realizadas.

Figura 1 – Gráfico de comportamento do Git



Fonte: <https://i.stack.imgur.com/sl3bm.png>

.1 Principais características do Git:

- **Distribuído:** Diferentemente de sistemas de controle de versão centralizados, no Git, cada desenvolvedor possui uma cópia completa do repositório em seu computador. Isso permite que o desenvolvimento possa continuar mesmo quando não há conexão com o servidor central, proporcionando maior autonomia e flexibilidade.
- **Eficiente:** O Git foi projetado para ser extremamente rápido e eficiente, tanto em relação ao armazenamento das alterações quanto à velocidade de acesso aos dados. Isso torna o Git adequado mesmo para projetos grandes e com históricos extensos.
- **Rastreamento de Histórico:** O Git mantém um histórico completo de todas as alterações feitas em cada arquivo ao longo do tempo, permitindo que os desenvolvedores possam navegar pelas diferentes versões e, se necessário, retornar a um estado anterior do código.
- **Ramificação e Fusão (Branching e Merging):** O Git torna fácil e seguro criar ramificações (branches) para desenvolver recursos ou correções independentes do código principal. Depois de testadas, essas ramificações podem ser integradas de volta ao projeto principal por meio de fusões (merges).
- **Controle de Conflitos:** Quando várias pessoas trabalham em partes diferentes do mesmo arquivo ou alteram a mesma linha de código, podem ocorrer conflitos. O Git possui mecanismos para resolver esses conflitos de forma organizada, garantindo que as alterações não sejam perdidas.
- **Compatibilidade:** O Git é compatível com várias plataformas e sistemas operacionais, o que o torna uma escolha versátil para equipes de desenvolvimento que utilizam diferentes ambientes.
- **Integração com Serviços de Hospedagem:** O Git se integra facilmente a serviços populares de hospedagem de código, como GitHub, GitLab e Bitbucket, permitindo que os desenvolvedores compartilhem seus projetos, colaborem e revisem o código em equipe.

1.2 Estados de um arquivo no Git

No Git, os arquivos do seu repositório podem existir em diferentes estados, também conhecidos como "estados de um arquivo". Esses estados refletem o ciclo de vida de um arquivo enquanto você trabalha nele e os gerencia com o Git. Existem três estados principais para os arquivos no Git:

- **Untracked (Não rastreado):**

Quando um arquivo é criado no diretório do seu projeto, mas ainda não foi adicionado ao controle de versão do Git, ele é considerado "não rastreado" (untracked). Isso significa que o Git não está ciente do arquivo e não está monitorando suas alterações. Arquivos não rastreados não fazem parte do histórico do Git até que você os adicione ao controle através do comando `git add`.

- **Tracked (Rastreado):**

Depois de adicionar um arquivo ao controle de versão do Git usando o comando `git add`, ele passa para o estado "rastreado" (tracked). Isso significa que o Git está ciente do arquivo e está monitorando suas alterações. Os arquivos rastreados podem estar em dois subestados diferentes:

- a. Unmodified (Não modificado):** Quando um arquivo rastreado não sofreu alterações desde o último commit, ele é considerado "não modificado". Isso significa que o conteúdo do arquivo é idêntico ao do último commit realizado.

- b. Modified (Modificado):** Quando um arquivo rastreado sofre alterações após o último commit, ele é considerado "modificado". Isso significa que o conteúdo do arquivo foi alterado e ainda não foi confirmado em um novo commit.

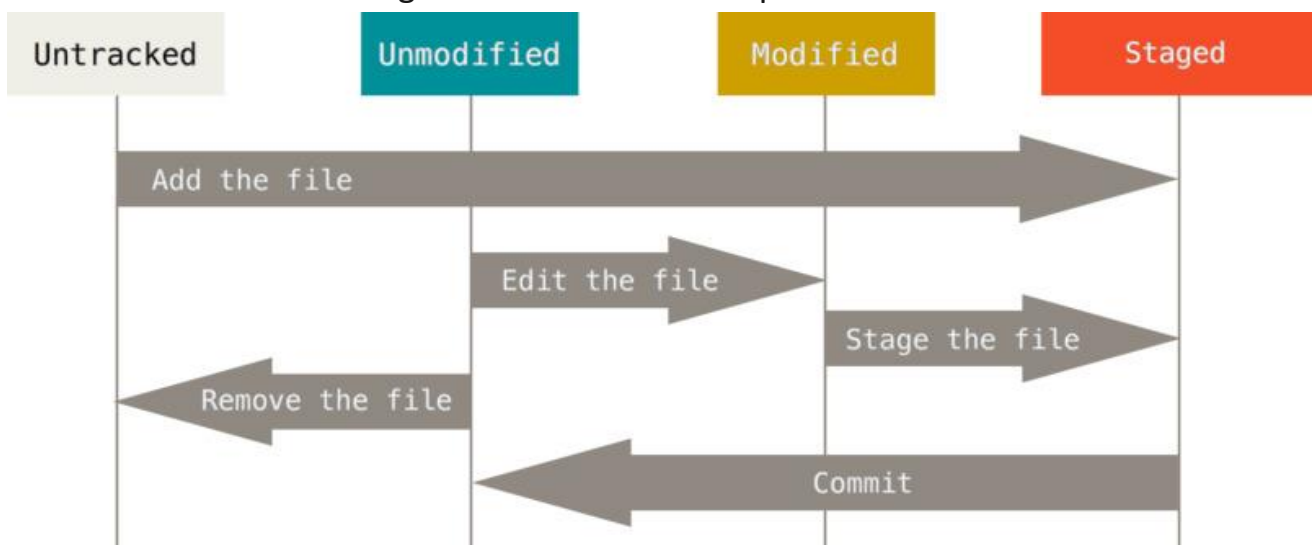
- **Staged (Preparado):**

Após fazer alterações em um arquivo modificado, você pode prepará-lo para ser incluído no próximo commit, adicionando-o à área de preparação (também conhecida como "staging area") usando o comando `git add`. Quando um arquivo está na área de preparação, ele está no estado "preparado" (staged). Isso significa que suas alterações foram selecionadas para fazer parte do próximo commit, mas ainda não foram efetivamente gravadas no histórico do Git.

Para resumir, os três estados de um arquivo no Git são:

- Untracked (Não rastreado): Arquivo não adicionado ao controle de versão.
- Tracked (Rastreado): Arquivo que está sendo monitorado pelo Git.
 - Unmodified (Não modificado): Arquivo rastreado que não sofreu alterações.
 - Modified (Modificado): Arquivo rastreado que sofreu alterações.
- Staged (Preparado): Arquivo modificado que foi selecionado para entrar no próximo commit, mas ainda não foi confirmado no histórico.

Figura 2 – Estados dos arquivos no Git



Fonte: https://www.alura.com.br/artigos/assets/iniciando-repositorio-git/imagen_git.jpg

1.3 Alguns dos principais comandos do Git

O Git possui uma ampla variedade de comandos para gerenciar repositórios, controlar versões, trabalhar com ramificações, colaborar com outros desenvolvedores e muito mais. Abaixo estão alguns dos principais comandos do Git:

- **git add:** Adiciona arquivos ao estágio de preparação (staging area) para que possam ser incluídos no próximo commit.
- **git branch:** Lista as ramificações disponíveis no repositório e mostra a ramificação atualmente ativa.

-
- **git checkout:** Permite alternar entre ramificações existentes ou criar novas ramificações.
 - **git clone:** Clona (faz uma cópia) de um repositório Git existente para o seu computador a partir de um URL remoto.
 - **git commit:** Cria um novo commit com as alterações preparadas na staging area, registrando uma nova versão do projeto.
 - **git fetch:** Obtém as referências de um repositório remoto, mas não mescla as alterações no seu repositório local.
 - **git init:** Inicializa um novo repositório Git em um diretório vazio ou converte um diretório existente em um repositório Git.
 - **git log:** Exibe o histórico de commits do projeto, mostrando informações como autor, data e mensagem de commit.
 - **git merge:** Combina as alterações de uma ramificação com outra, fundindo-as em uma nova versão.
 - **git pull:** Obtém as alterações de um repositório remoto e mescla-as no seu repositório local.
 - **git push:** Envia os commits locais para um repositório remoto, atualizando-o com as suas alterações.
 - **git remote:** Exibe informações sobre os repositórios remotos configurados e permite adicionar ou remover repositórios remotos.
 - **git reset:** Desfaz alterações no repositório, revertendo para um estado anterior.
 - **git stash:** Armazena temporariamente alterações não preparadas para que você possa alternar para outra ramificação sem realizar um commit.

- **git status:** Mostra o estado atual do repositório, incluindo arquivos não rastreados, modificados e preparados para commit.

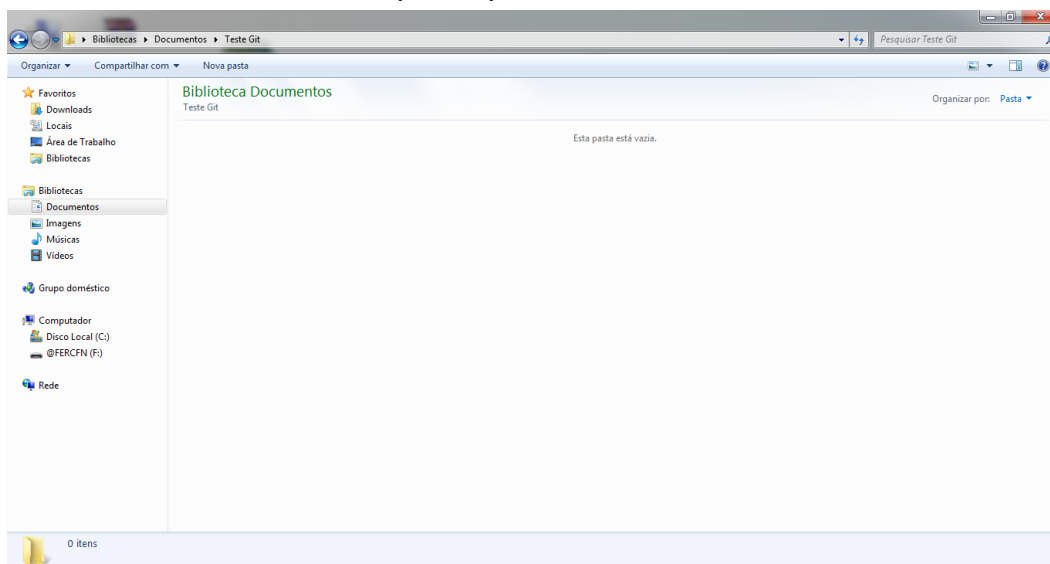
Esses são apenas alguns dos principais comandos do Git. Existem muitos outros comandos e opções disponíveis para realizar tarefas específicas de acordo com suas necessidades. É recomendável consultar a documentação oficial do Git ou usar o comando **git --help** para obter informações detalhadas sobre cada comando e suas opções.

1.4 Git na prática parte 1

Vamos praticar? Estou considerando que você já tem o Git instalado e configurado na sua máquina, bem como uma conta no GitHub, porém, se você ainda não instalou o Git e não criou sua conta no GitHub, [clique aqui](#) e assista a esse tutorial.

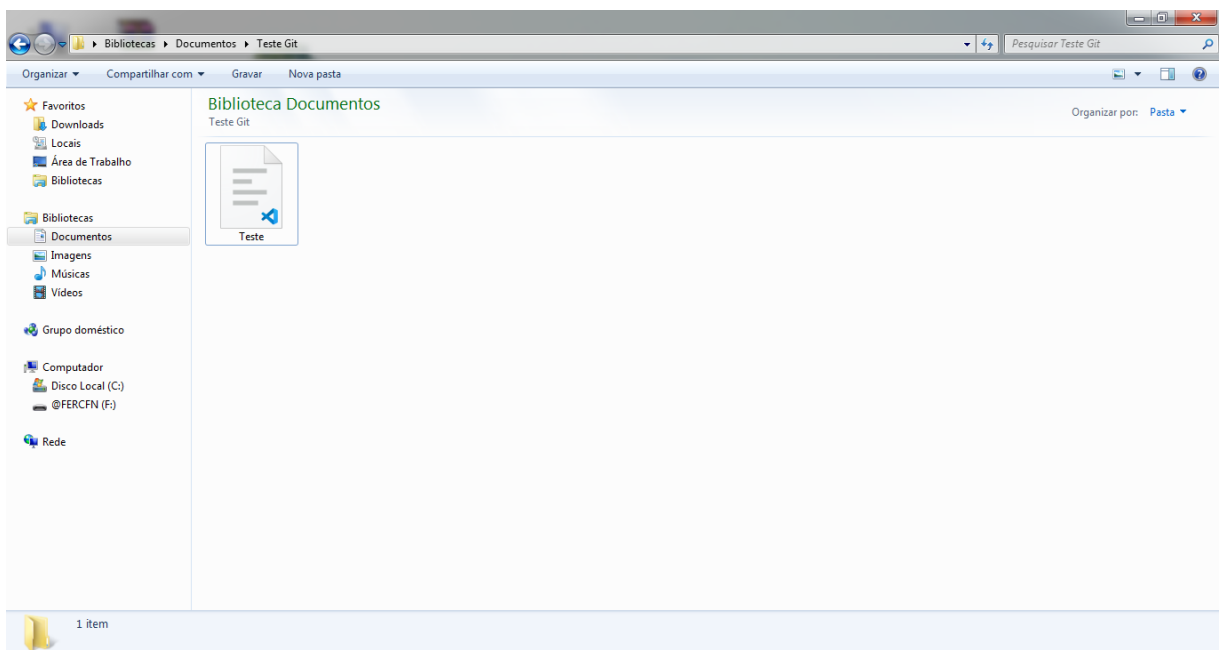
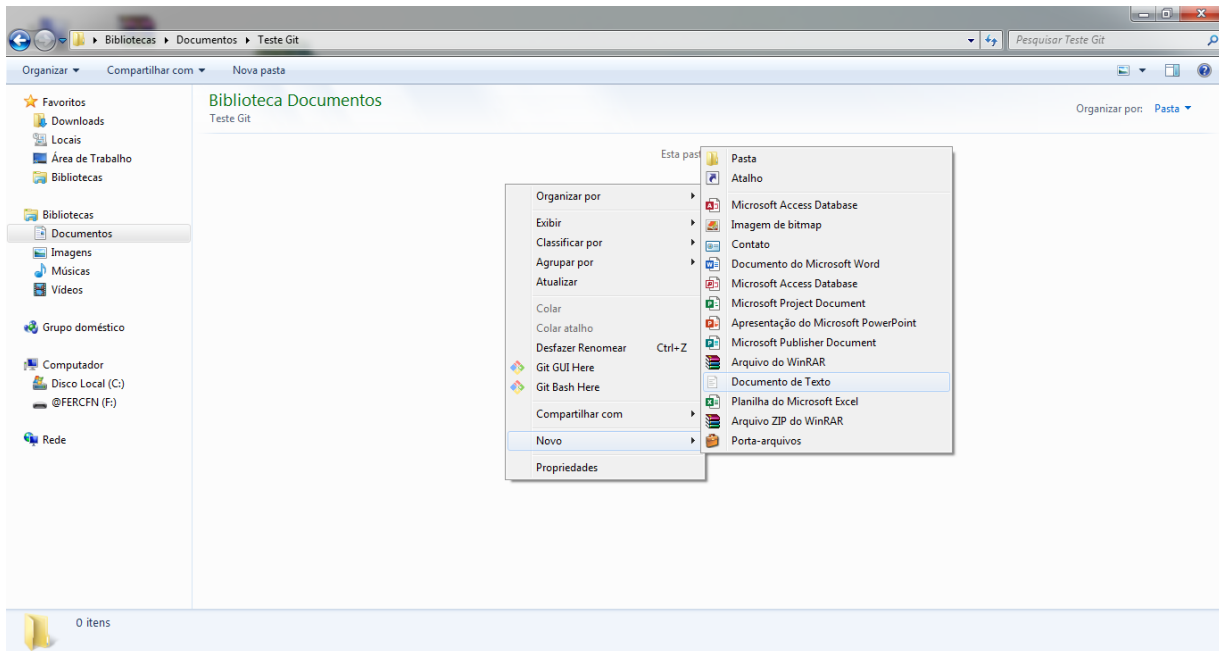
Se você já tem tudo pronto, vamos testar alguns dos comandos que foram apresentados acima.

Primeiro será criada uma pasta para realizar os testes

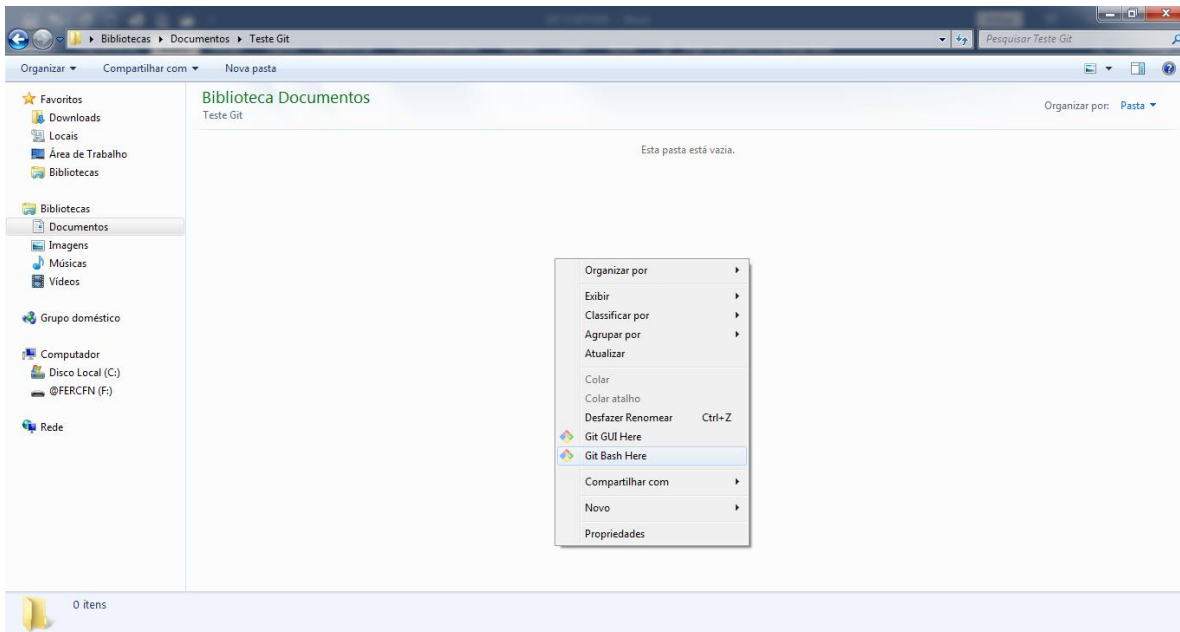


Nesse caso foi criada uma pasta com o nome **Teste Git** em biblioteca > documentos.

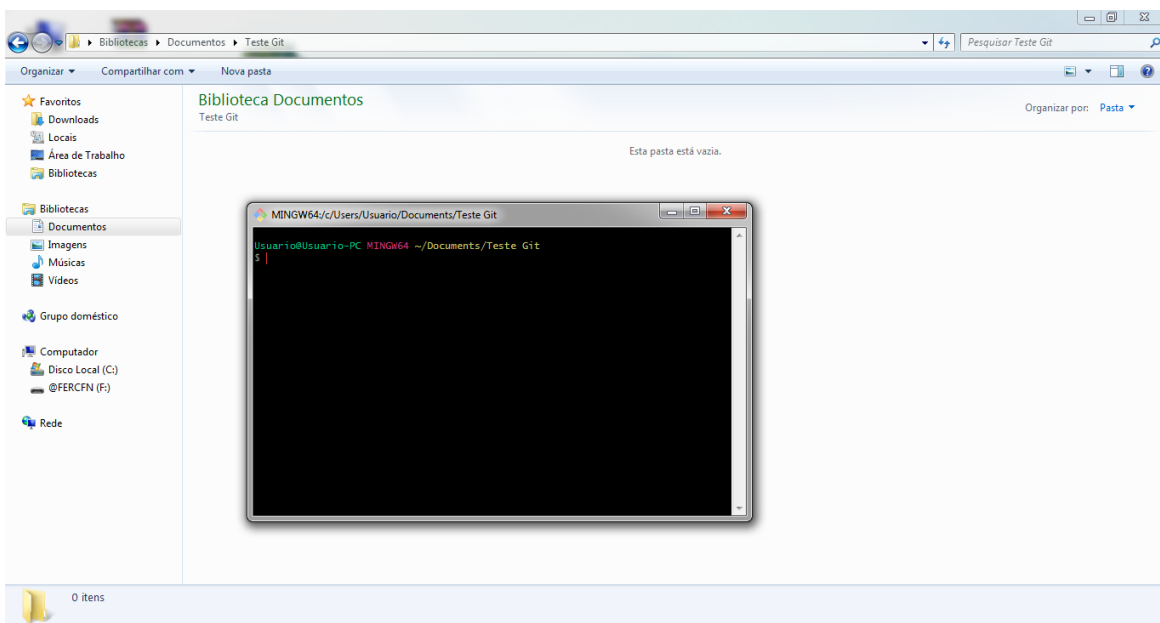
Depois foi criado um arquivo de texto clicando com o botão direito, direcionando o mouse sobre a palavra novo e depois na caixa à direita clicando em documento de texto. Eu nomeei o arquivo de Teste, mas fique à vontade para nomear de acordo com a sua preferência.



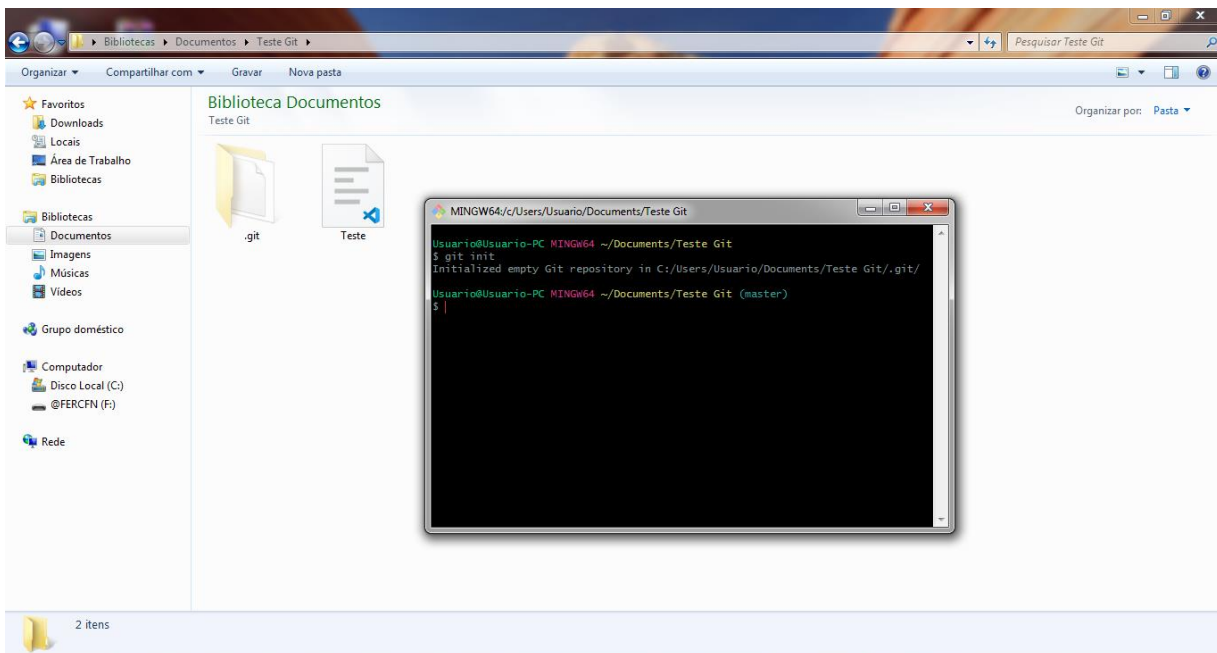
Clique com o botão direito do mouse e procure a opção **Git Bash Here**



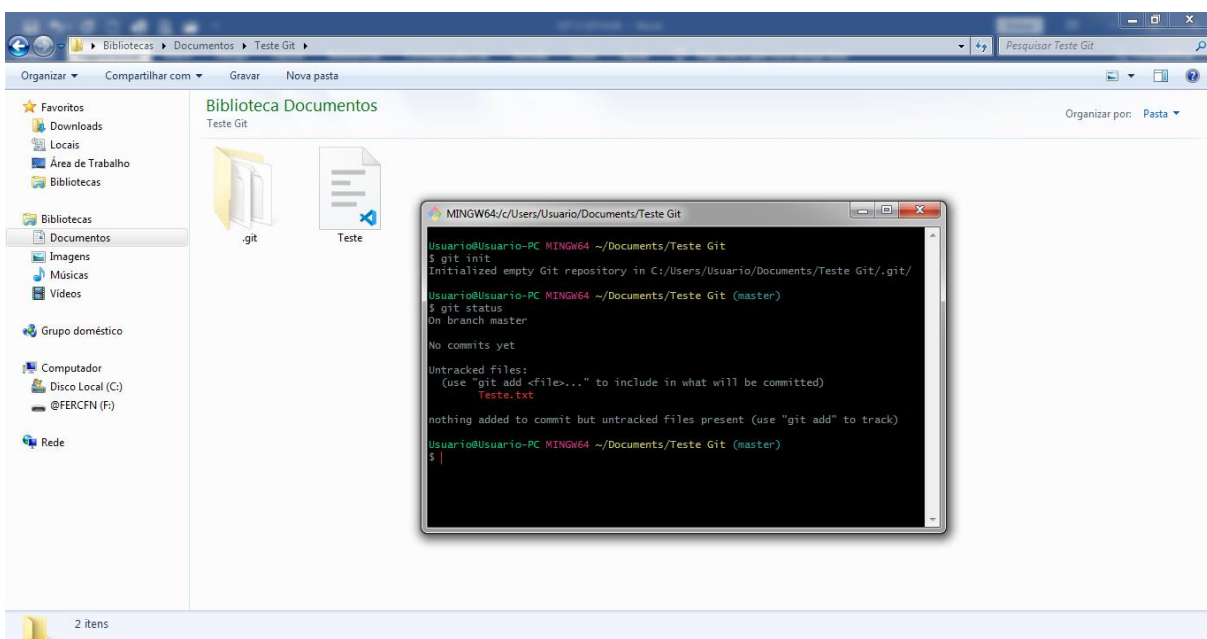
Ao clicar nela será aberto o terminal do Git. Deverá aparecer uma tela parecida com a tela abaixo



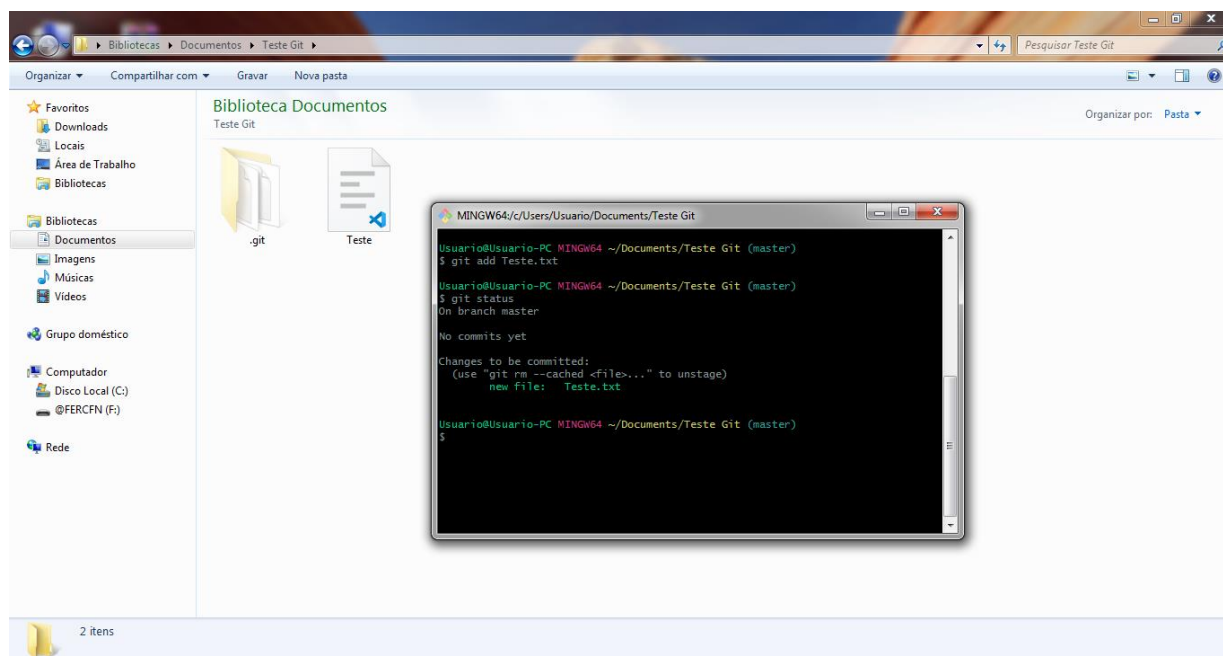
Através do comando **Git Init** inicializamos um diretório Git na pasta que foi criada, a partir de agora todas as alterações poderão ser salvas e versionadas via Git.



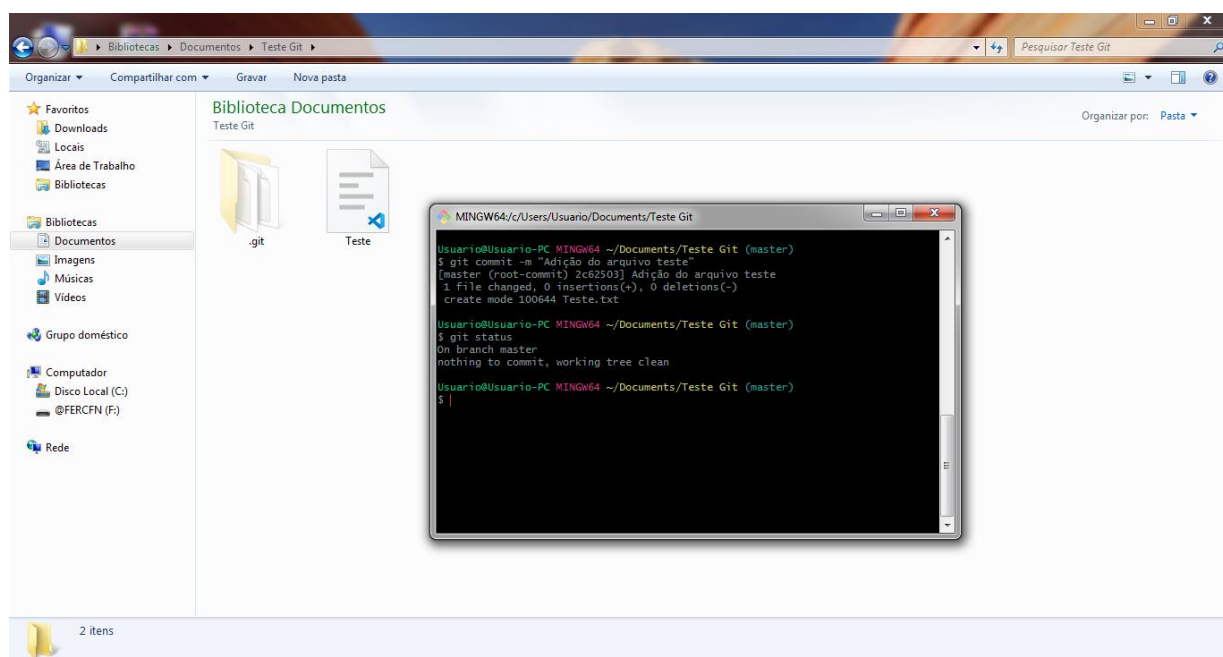
Utilizando o comando **Git Status** podemos verificar o estado dos arquivos. Nesse caso, como ainda não foi realizado nenhum commit, o arquivo criado aparece como untracked (não rastreado), o que significa que alterações feitas nesse arquivo não serão versionadas pelo Git, ou seja, você não poderá retornar a um estado anterior a uma alteração caso necessite. Para resolver esse problema vamos mover o arquivo para a área de staged



Utilizando o comando **Git Add** seguido pelo nome do arquivo com sua extensão o arquivo foi adicionado ao estado de staged (preparado) e está pronto para ser commitado. O comando **Git status** foi utilizado novamente para confirmar que o arquivo estava pronto para o commit.



Utilizando o comando **Git Commit -m "mensagem do commit"** foi realizado o commit do arquivo e criada sua primeira versão, após o commit foi verificado novamente através do **Git Status** que o arquivo havia sido commitado e que não haviam outras alterações, ou, como diz o Git, working tree clean.



Antes de testar mais alguns comandos vamos transferir essa pasta para o repositório remoto no GitHub, mas antes, vamos aprender um pouco sobre o GitHub.

2. O que é o GitHub?

O GitHub é uma plataforma de hospedagem de código e colaboração baseada em nuvem lançada em 2008. Ela se tornou uma das plataformas mais populares para desenvolvedores compartilharem, colaborarem e contribuírem para projetos de software de código aberto e privados.

Principais recursos e funcionalidades do GitHub:

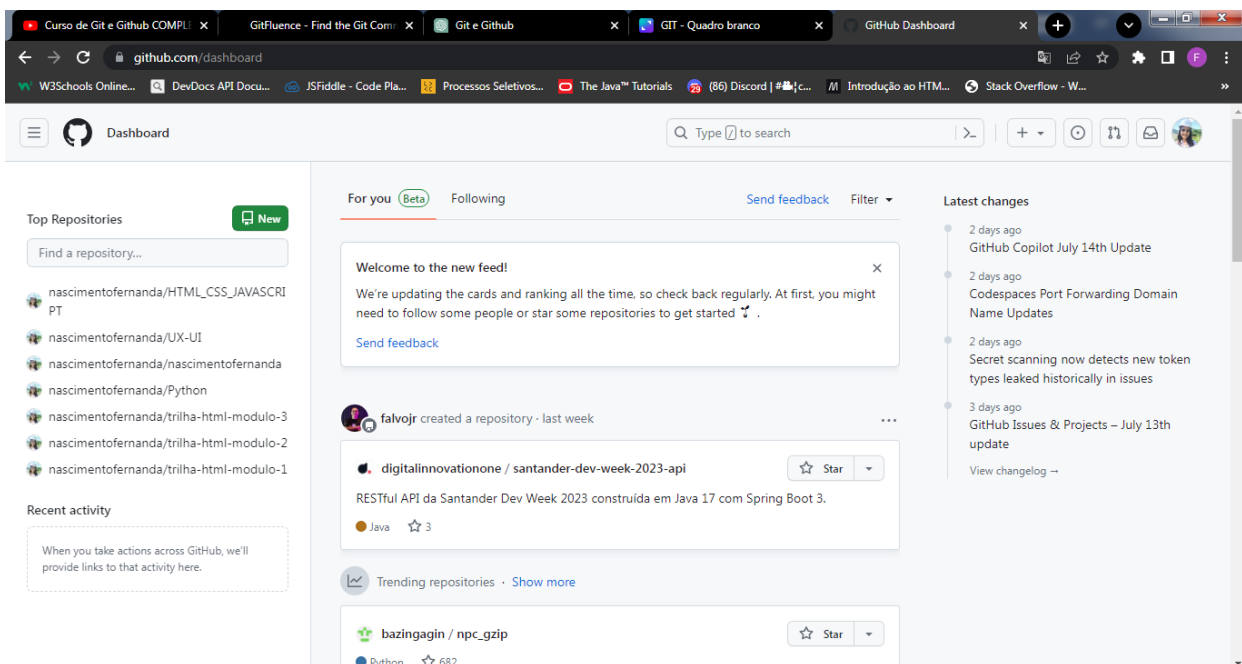
- **Repositórios:** Os projetos no GitHub são armazenados em repositórios, que contêm todo o código-fonte, arquivos, histórico de commits e outras informações relacionadas ao projeto.
- **Controle de Versão:** O GitHub utiliza o sistema de controle de versão Git para rastrear e gerenciar as alterações em projetos. Ele permite que os desenvolvedores criem ramificações, façam alterações independentes e, em seguida, integrem suas alterações ao projeto principal por meio de fusões.
- **Colaboração:** O GitHub facilita a colaboração entre desenvolvedores, permitindo que eles contribuam com projetos enviando "pull requests" (solicitações de inclusão de código) para propor mudanças ou correções. Os mantenedores do projeto podem revisar as solicitações e, se aprovadas, incorporá-las ao código principal.
- **Issues e Projeto:** O GitHub oferece recursos de rastreamento de problemas (issues), onde os usuários podem relatar bugs, solicitar novos recursos ou discutir melhorias no projeto.
- **Integração Contínua:** O GitHub permite que os projetos implementem integração contínua, que automatiza o processo de construção, testes e implantação de código, tornando-o mais eficiente e confiável.

-
- **Comunidade:** O GitHub é uma comunidade ativa de desenvolvedores, com milhões de projetos disponíveis para serem explorados e contribuídos. Os usuários podem seguir desenvolvedores, projetos e temas de interesse para receber atualizações e participar de discussões.
 - **Pull Requests:** As "pull requests" (solicitações de inclusão de código) são usadas para propor alterações de código para um repositório. Os desenvolvedores podem criar uma pull request para solicitar que suas alterações sejam revisadas e incorporadas ao projeto principal.
 - **Code Review:** O GitHub oferece recursos integrados para revisão de código. Os revisores podem comentar em trechos específicos do código e discutir as alterações propostas antes de aprová-las e incorporá-las ao projeto.
 - **Wikis:** Cada repositório pode conter uma wiki para documentar informações adicionais sobre o projeto. É uma maneira útil de fornecer documentação, guias de uso e outras informações relevantes para os colaboradores e usuários.
 - **Projetos:** Os painéis de projetos permitem que os mantenedores do repositório organizem e gerenciem tarefas e issues em um quadro Kanban. É uma forma eficaz de planejar, acompanhar e priorizar o trabalho do projeto.
 - **Gists:** Gists são pequenos trechos de código, textos ou arquivos que podem ser compartilhados independentemente do repositório principal. São úteis para compartilhar exemplos de código, notas ou informações rápidas.
 - **Actions:** O GitHub Actions é uma ferramenta de automação poderosa que permite criar, testar e implantar projetos automaticamente. É amplamente utilizado para integração contínua e implantação contínua (CI/CD).
 - **Packages:** O GitHub Packages permite que os desenvolvedores publiquem, gerenciem e compartilhem pacotes de código, como bibliotecas, módulos ou imagens de contêiner.

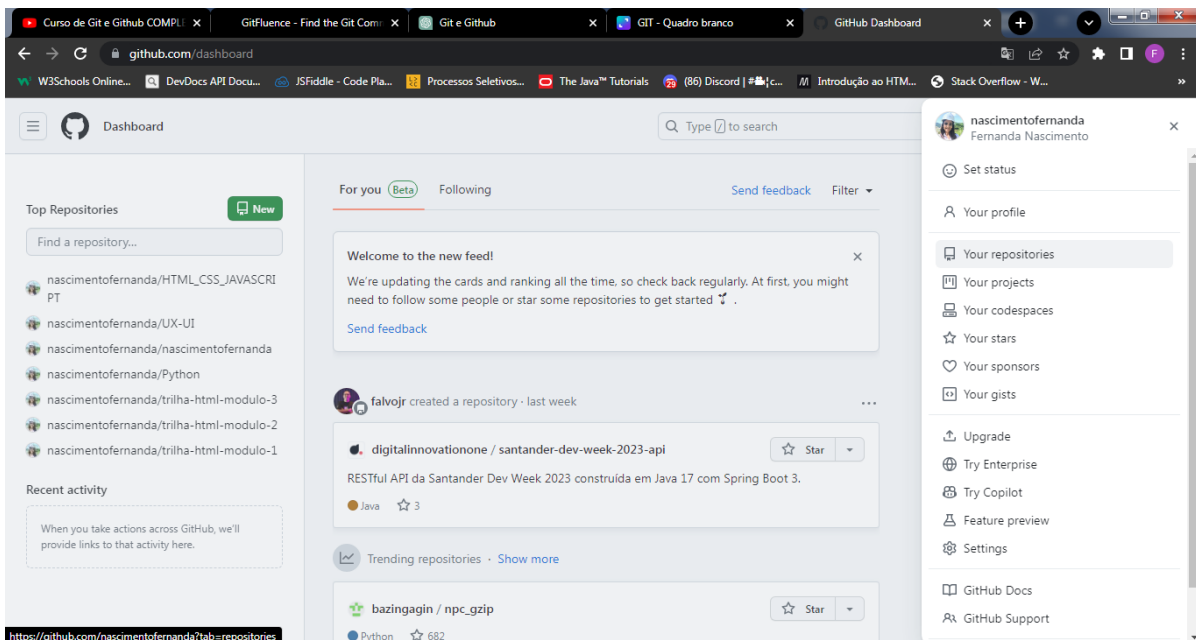
- **Security:** O GitHub oferece recursos para análise de segurança de código, alertando os desenvolvedores sobre possíveis vulnerabilidades em suas dependências.

2.1 GitHub na prática

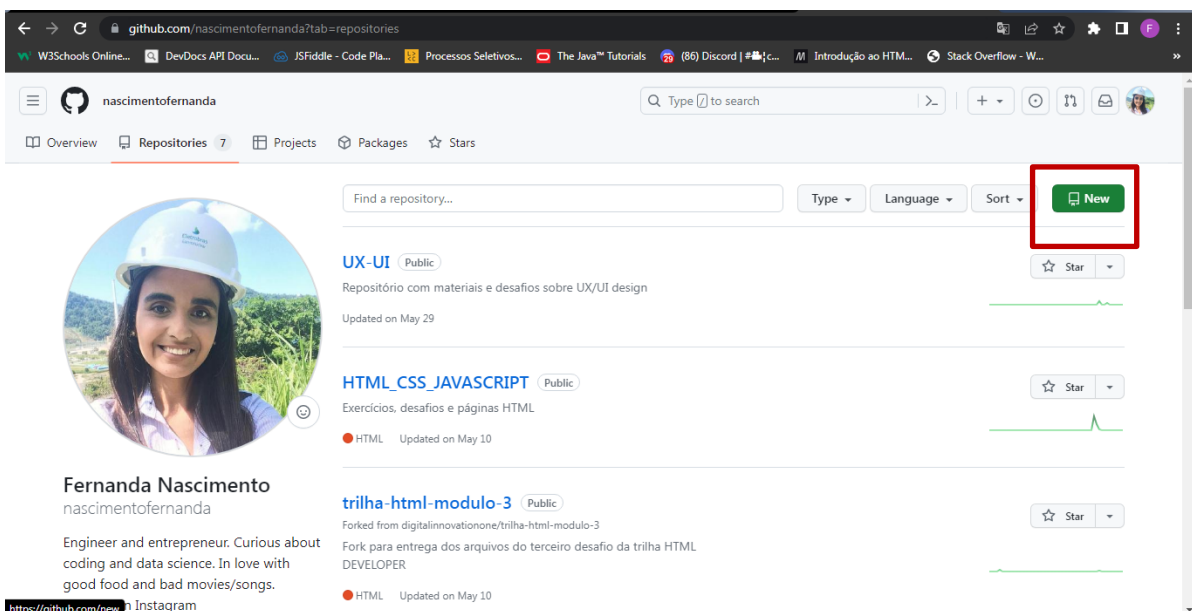
Agora vamos criar um repositório no GitHub para sincronizar com a pasta criada no nosso repositório local. Primeiro vou entrar na minha conta do GitHub.



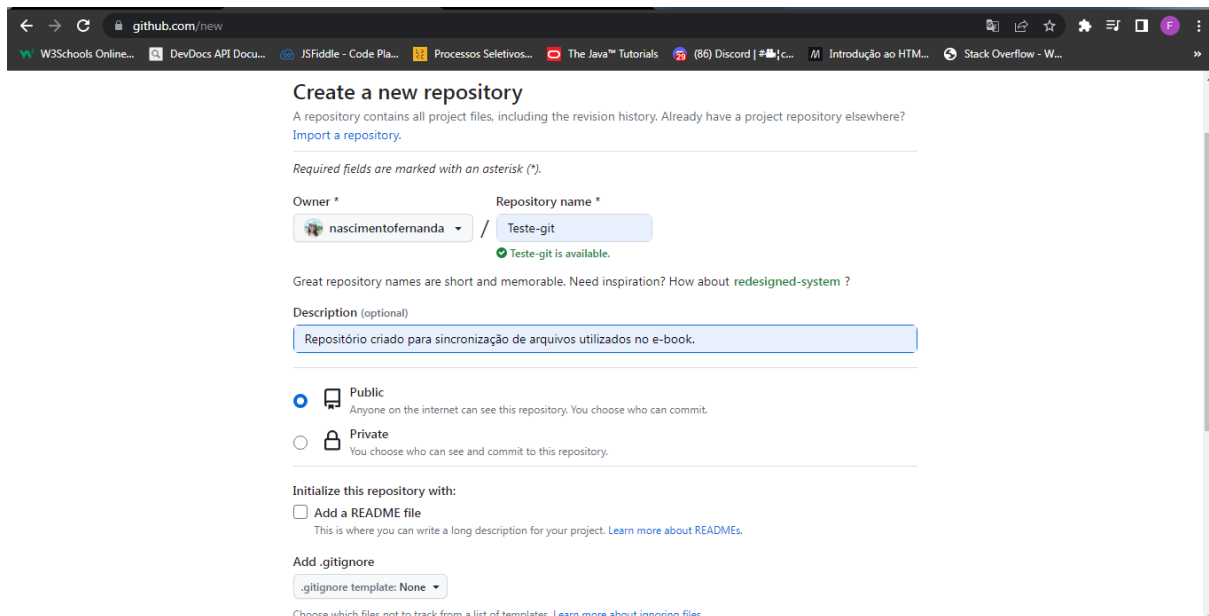
Após entrar no GitHub, você encontrará uma página inicial parecida com essa, clique no ícone no canto direito e depois em **your repositories**.



Novamente no canto direito, clique no botão verde **New**



Na nova página iremos preencher algumas informações



github.com/new

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * / Repository name *

nascimentofernanda / Teste-git

Teste-git is available.

Great repository names are short and memorable. Need inspiration? How about redesigned-system ?

Description (optional)

Repositório criado para sincronização de arquivos utilizados no e-book.

☒ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

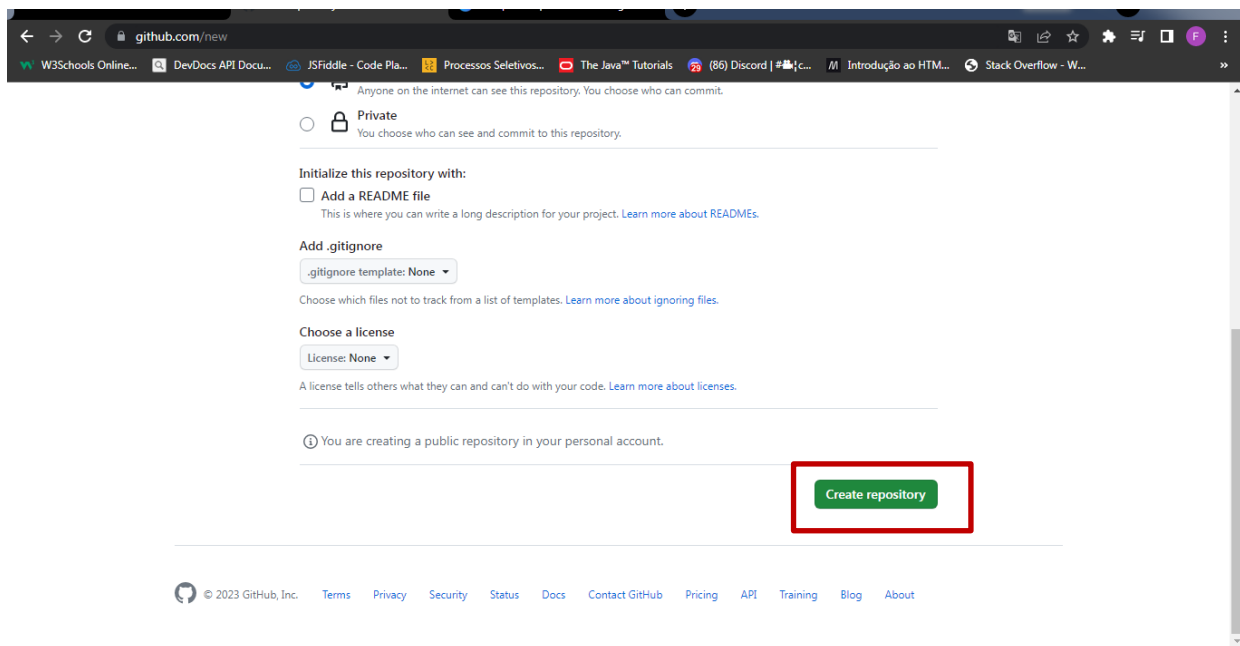
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Em **repository name** iremos colocar o mesmo nome da pasta do nosso repositório local (O nome no GitHub não permite adicionar espaços entre palavras, por isso, iremos alterar o nome do repositório local para Teste-git)

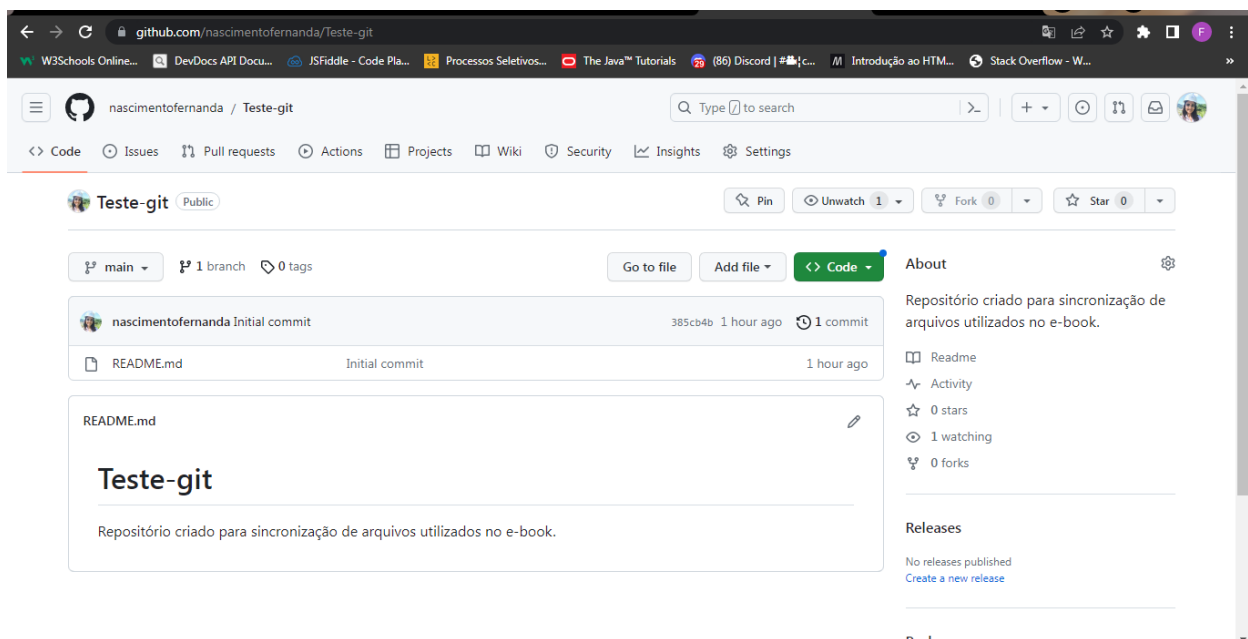
Em **description** iremos adicionar uma breve descrição sobre o que se trata o repositório, essa descrição pode ser alterada depois.

Iremos marcar se o repositório será Public ou Private, nesse caso deixarei como público para caso vocês queiram acessar.

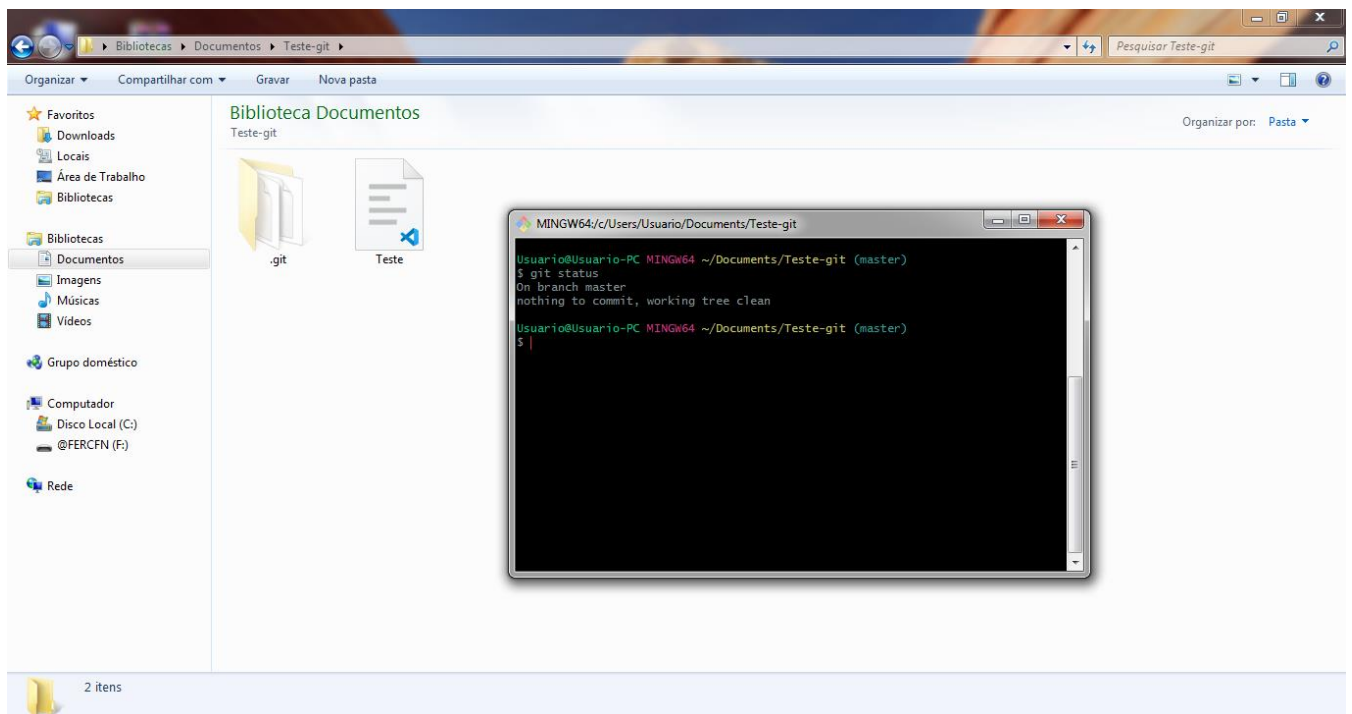
Não iremos alterar as outras configurações e vou criar o repositório criando em **Create repository**



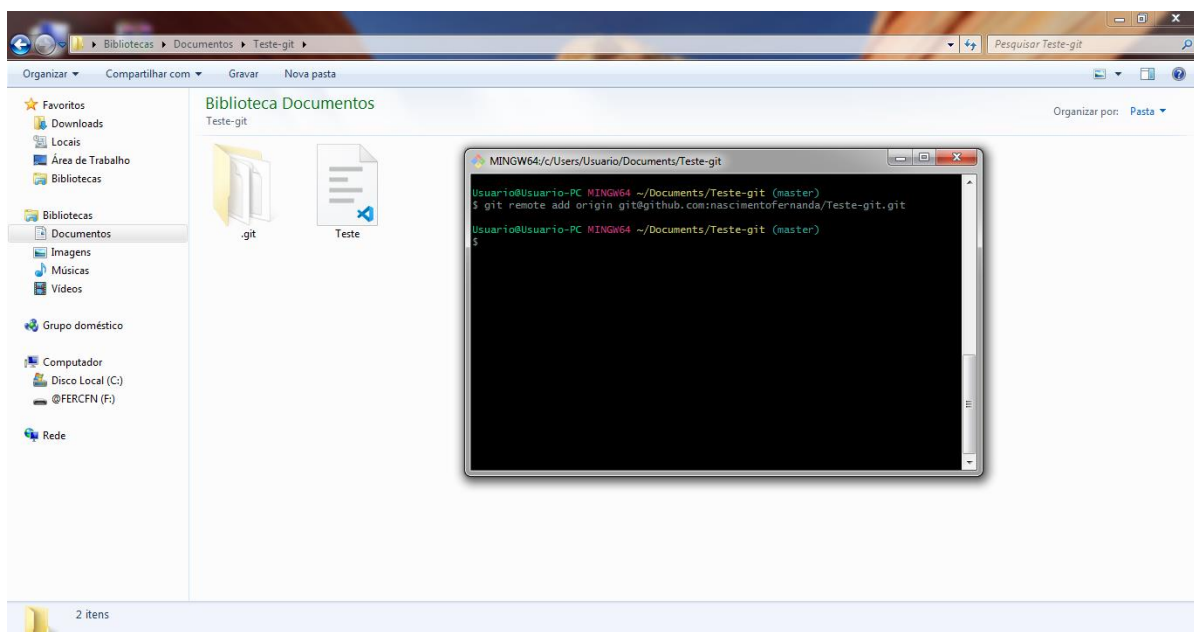
Após criar o repositório ele já fica disponível no perfil do GitHub

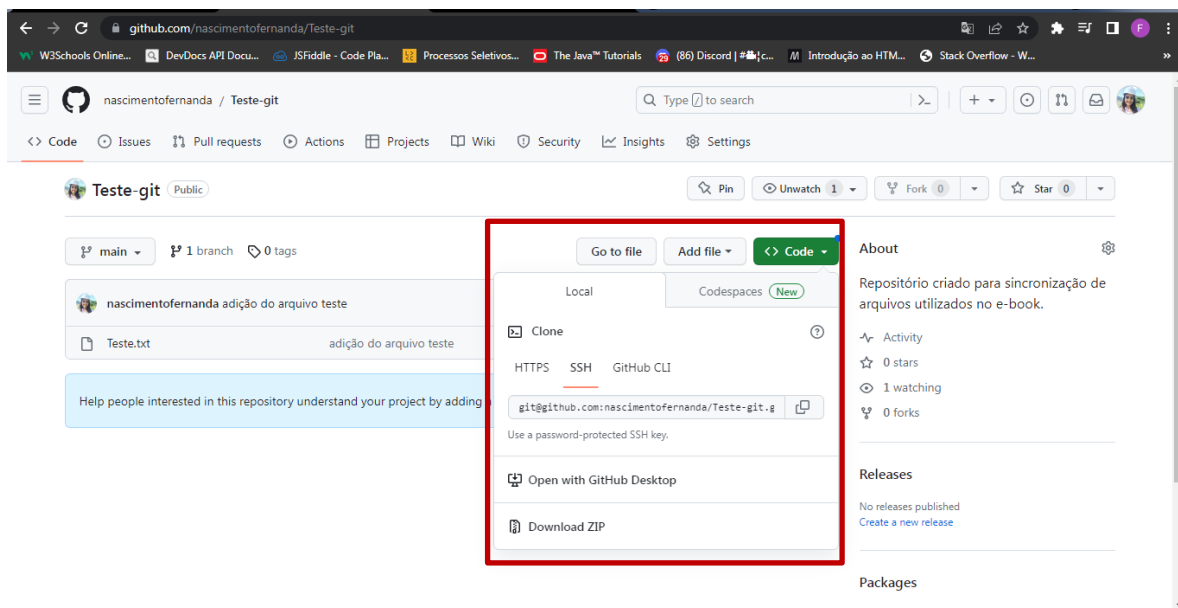


Agora vamos sincronizar o repositório do GitHub com o repositório local, para isso, primeiro precisamos abrir nosso repositório local e o terminal do Git através do Git Bash Here como fizemos anteriormente



Após abrir o terminal e executar um git status para verificar o status dos arquivos verificamos que os arquivos estão prontos para serem enviados ao repositório remoto, para isso vamos utilizar o comando git remote add origin seguido do código SSH disponível no repositório





Para utilizar a opção SSH é necessário ter configurado a opção de segurança de chave SSH nas configurações de segurança do GitHub.

Depois de adicionar o repositório remoto ao repositório local, vamos utilizar o comando `git branch -M main` para criar uma branch main no repositório local e torná-la a branch principal.

Ainda não sabe o que é uma branch? Então vamos aprender antes de seguir em frente.

2.2 O que é uma branch?

No Git, uma "branch" (ramificação) é uma linha de desenvolvimento independente que permite que você trabalhe em uma versão alternativa do seu projeto sem afetar o código principal. É como se você estivesse criando uma cópia do projeto em um momento específico para realizar alterações ou adicionar novos recursos sem interferir diretamente no trabalho dos outros desenvolvedores ou na versão estável do projeto.

Cada repositório Git começa com uma branch padrão chamada "master" (ou "main", dependendo da configuração). Quando você inicia um novo projeto ou repositório, o Git cria automaticamente essa branch para você. Essa branch principal contém o código estável e funcional do projeto.

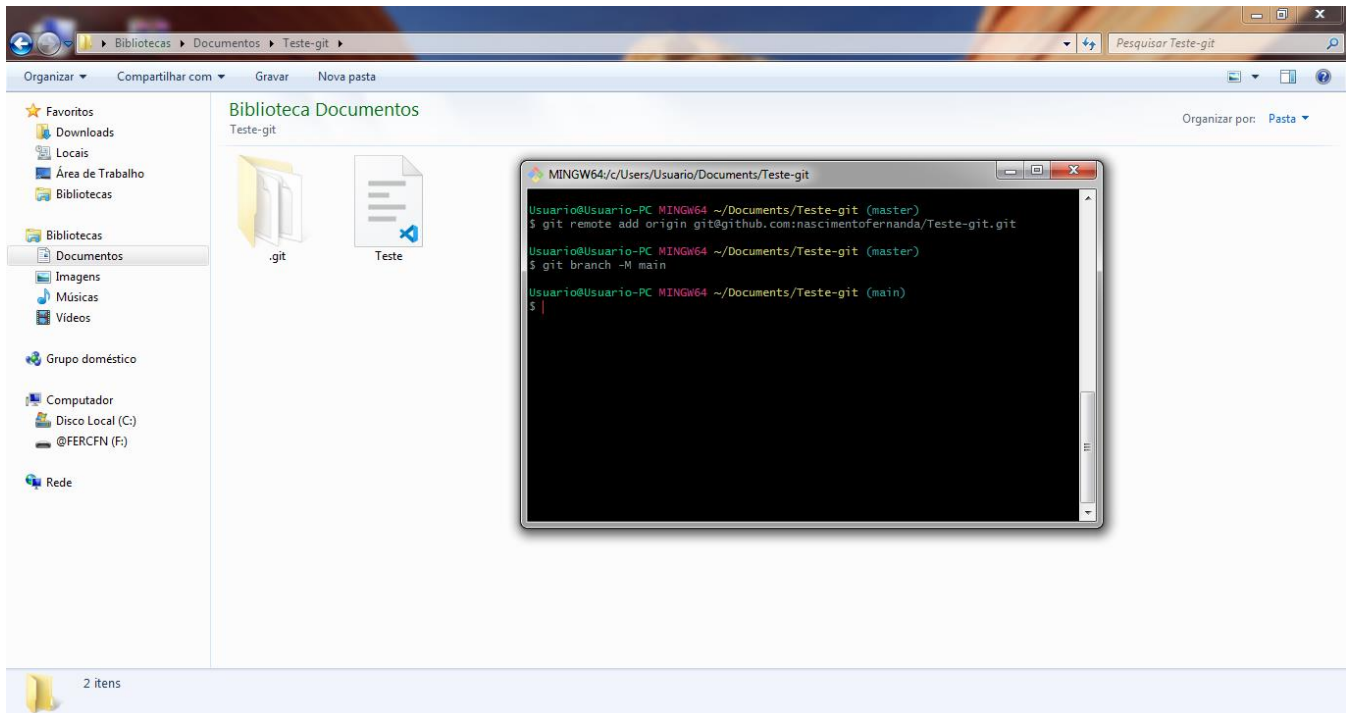
Ao criar uma nova branch, você pode iniciar seu trabalho a partir de um ponto específico do histórico do projeto (como o último commit na branch master). A partir desse ponto, todas as alterações que você fizer serão registradas na nova branch, mantendo-as separadas do código da branch principal.

As branches são úteis em vários cenários:

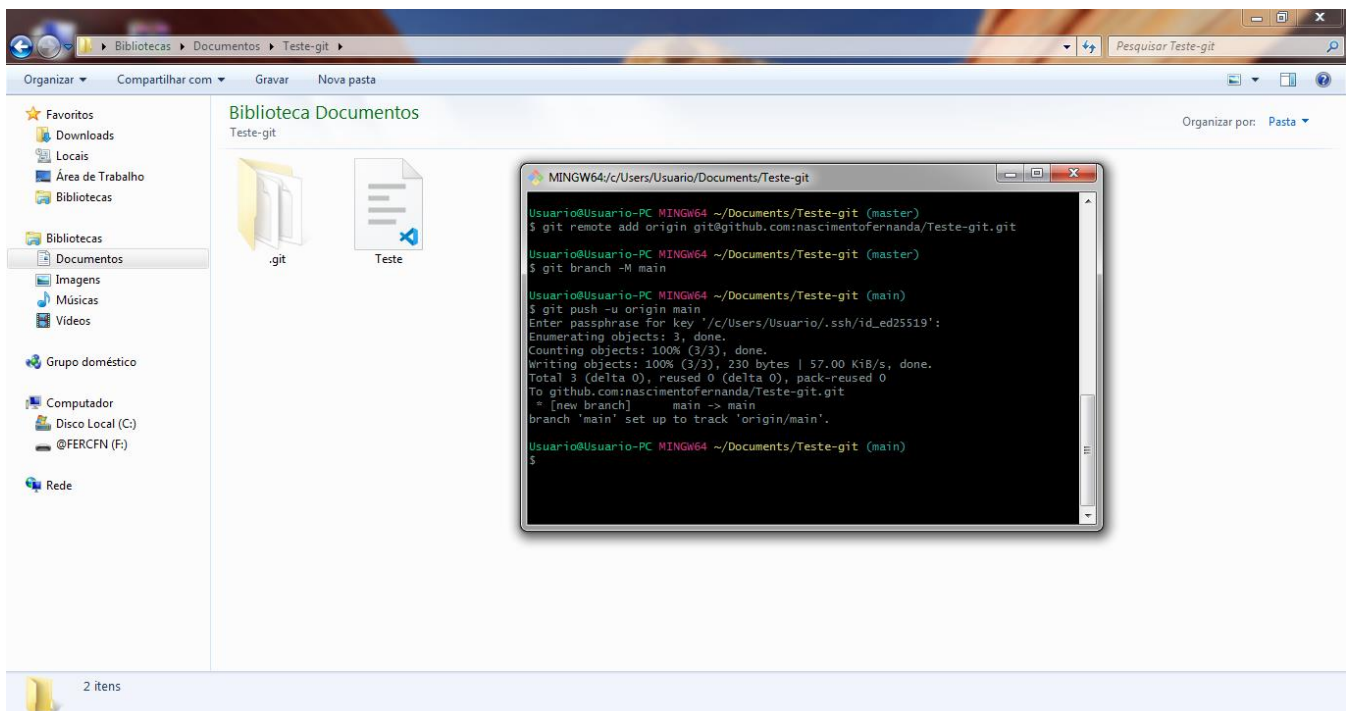
- **Desenvolvimento de Recursos:** Você pode criar uma branch para cada novo recurso ou funcionalidade que está sendo desenvolvido. Isso permite que você trabalhe nesses recursos isoladamente e os integre ao projeto principal somente quando estiverem prontos.
- **Correção de Bugs:** Quando você precisa corrigir um bug ou problema específico, pode criar uma branch dedicada para resolvê-lo sem afetar o código principal até que a correção seja testada e validada.
- **Experimentação:** Se você deseja testar novas ideias ou implementações sem modificar diretamente o código principal, uma branch é uma ótima opção para essa experimentação.
- **Colaboração:** Equipes de desenvolvimento podem usar branches para trabalhar em recursos ou correções de bugs separadamente antes de fundir as alterações no projeto principal.

Uma vez que você tenha concluído o trabalho em uma branch, pode mesclar (merge) as alterações de volta à branch principal (geralmente a "master") para incorporar suas mudanças ao projeto principal. O Git facilita a criação, comutação e gerenciamento de branches, permitindo que você mantenha um histórico completo de todas as alterações realizadas no projeto. Isso ajuda a manter o desenvolvimento organizado, colaborativo e flexível.

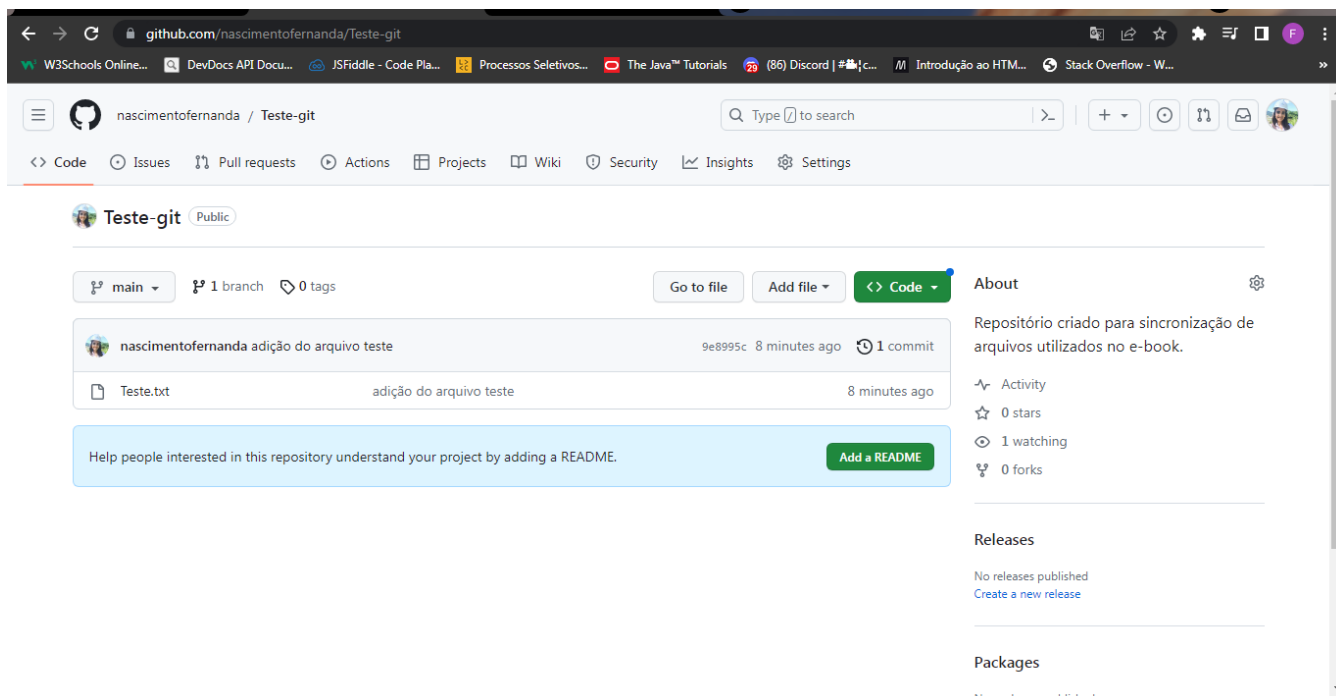
Agora que já sabemos o que é uma branch, vamos retornar ao exemplo.



Após executar o comando `git branch -M main` nossa branch principal não é mais a master e sim a main, como podemos ver na imagem acima. Após esses passo já podemos enviar os arquivos para o GitHub através do comando `git push -u origin main`



Após executar o comando e colocar a senha cadastrada para a chave SSH os diretórios remoto e local já estão sincronizados.



Podemos verificar no GitHub que o arquivo Teste que foi criado localmente já se encontra no repositório remoto, bem como a descrição do commit.

Agora que você já sabe o básico sobre como criar um repositório local e remoto e enviar arquivos entre eles que tal dar uma lida na documentação do Git e testar mais alguns exemplos por você mesmo? O link para a documentação está [aqui](#). Bons estudos e até o próximo e-book 😊.

Referências:

https://www.alura.com.br/artigos/o-que-e-git-github?gclid=CjwKCAjwh8mlBhB_EiwAsztdBFn4RzxxbqmyM7Jb4IHjr3Hwh8aOQz8sJU48V6bMda5bQMwkHpkR1BoCgb0QAvD_BwE

https://www.youtube.com/watch?v=kB5e-gTAl_s

https://git-scm.com/docs/git/pt_BR

<https://gitfluence.com/>