

# Git e GitHub

## Sumário

Git e GitHub.....	1
O que é Git? .....	1
O que é um repositório?.....	1
O que é GitHub? .....	2
O que são Branches? .....	2
Workflow do Git. ....	3
Git Bash .....	4
Exemplo de utilização do Git e upload para o GitHub pelo Git Bash .....	4
Cheatsheet de Comandos Git .....	14
Comandos básicos .....	14

## O que é Git?



Git é um VCS — Version Control System, ou Sistema de Controle de Versão/Versionamento, uma ferramenta usada para realizar a manutenção e controle do histórico de alterações de um código fonte de um projeto de desenvolvimento de sistemas<sup>1</sup>.

Ao utilizarmos o Git em um projeto, transformamos a pasta em que ele está armazenado em um *repositório local*, e podemos criar rastreios para os arquivos ali presentes. Ele também nos permite trabalhar em diferentes versões (ou “raízes”) de um código-fonte, além de poder sincronizar diferentes versões do mesmo código base, como um código base presente no repositório local e outro em um repositório remoto.

## O que é um repositório?

Repositório é, em computação, o local central onde um dado é guardado e administrado, podendo esse local ser tanto físico quanto lógico.

Tratando-se do Git, utilizaremos dois tipos de repositórios: um *local* e outro *remoto*.

---

<sup>1</sup> O Git é majoritariamente utilizado no desenvolvimento de softwares, mas pode ser utilizado para fazer versionamento de qualquer tipo de arquivo digital. Há, por exemplo, livros sendo escritos no GitHub com o Git.

- **Local:** Ter um repositório local significa que o repositório está armazenado na sua máquina, como por exemplo, uma pasta chamada "Atividades" que possui um arquivo oculto `.git`.
- **Remoto:** Um repositório remoto significa um repositório que é acessado através da internet ou da rede,

Como já dito, o repositório local é criado utilizando o Git, que transforma uma pasta com arquivos em um repositório com o rastreamento e versionamento das atualizações que são feitas nos arquivos. Enquanto isso, o repositório remoto é criado em outro local, mais comumente sendo no GitHub.

## O que é GitHub?



GitHub é uma aplicação web muito utilizada para a criação e manutenção de repositórios remotos, além de ser considerada uma "rede social para desenvolvedores" e um grande requisito para qualquer desenvolvedor que queira entrar no mercado de trabalho dessa área.

Nele, podemos criar um portfólio dentro do próprio perfil, compartilhar documentos e dicas, postar projetos e interagir com repositórios de outras pessoas.

Um dos motivos para o GitHub ser tão requisitado pelas empresas é a facilidade de uma equipe inteira trabalhar no mesmo projeto sem grandes problemas, independentemente da localização de cada integrante. Isso se dá graças ao repositório remoto que tem o projeto original e pode ser passado para os computadores de toda a equipe, aos avisos de novas alterações e de quando e quem as fizeram, e à versão mais visual das *branches* do Git.

É importante ter em mente que, apesar de extremamente complementares, o Git não precisa do GitHub e vice-versa. Você pode usar um sem o outro sem problemas.

## O que são Branches?

Imagine uma branch como uma versão do seu projeto. Numa situação hipotética, seu repositório possui três arquivos `.py` na *master branch*<sup>2</sup> (ou branch mestre) com o conteúdo que já foi revisado e é compartilhado com toda a equipe, tendo certeza de seu funcionamento e de seu uso. Caso você queira adicionar algo novo, inserir diretamente no código base não é uma boa ideia, pois pode causar erros à parte que já estava correta e funcionando. Para contornar isso, criamos uma nova branch.

---

<sup>2</sup> *Master* usualmente é o nome da primeira branch criada, também sendo a principal. Ela também pode ter o nome *main* (principal).

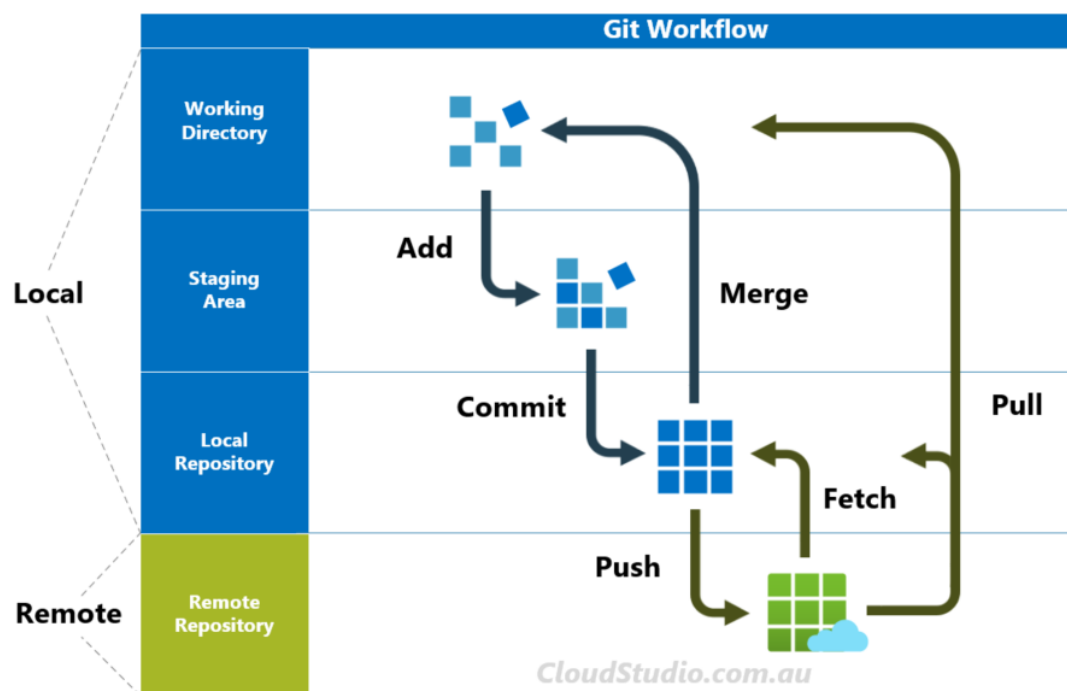
Essa nova branch será uma cópia da branch original, porém, ao alterá-la, não comprometemos o código base. Quando as alterações forem finalizadas, testadas e aprovadas, você pode fundir a nova branch à master branch.

De forma simples, pense em branches como cópias de um mesmo arquivo (por exemplo, um documento no bloco de notas) com novas adições de conteúdo que, após aprovadas, são lançadas no documento original. Apesar de não fazer sentido em um arquivo de texto, é uma forma extremamente útil e segura para trabalhar com códigos, que podem parar de funcionar por uma mínima alteração.

## Workflow do Git.

Workflow trata-se do **Fluxo de trabalho**: uma sequência de processos a qual um projeto passa desde o início até sua finalização.

O Git possui seu próprio Workflow, como mostrado a seguir:



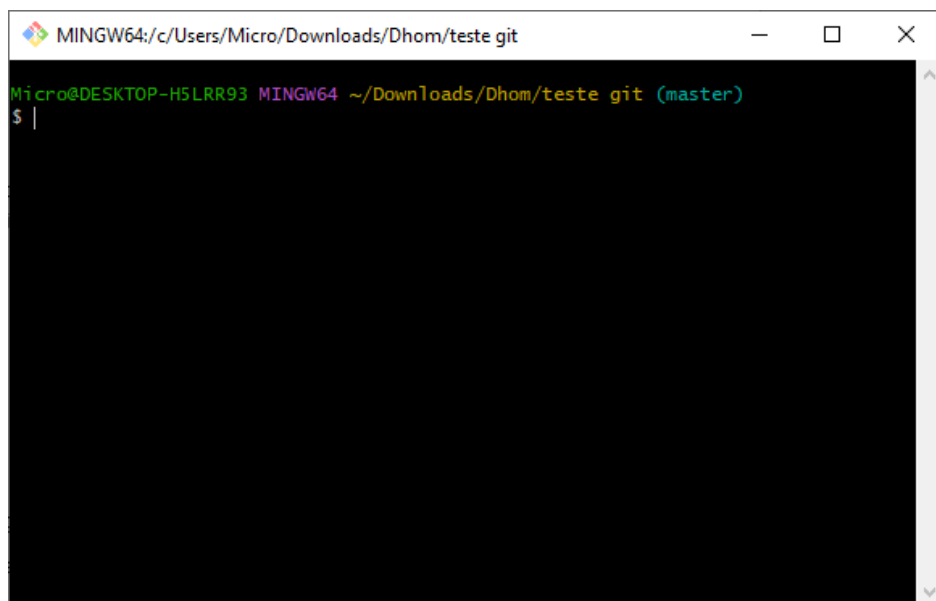
A imagem apresenta os estágios da relação entre arquivo sendo trabalhado, repositório local e repositório remoto, além dos principais comandos utilizados nesta relação.

- **Working Directory:** Working Directory, ou Diretório de trabalho, trata-se da pasta com todos os arquivos do seu projeto.
- **Staging Area:** A Staging Area é uma área entre o Diretório de trabalho e o Repositório Local utilizada para decidir quais alterações realizadas no Diretório de trabalho irão para o Repositório Local e, mais para frente, para o Repositório Remoto.

- **Local Repository:** O Repositório Local, como já dito, é a pasta *.git* presente no seu Diretório de trabalho, e é ele nele que ocorre o rastreamento e a comunicação entre local e remoto.
- **Remote Repository:** O Repositório Remoto é o armazenamento do projeto em um repositório em nuvem, geralmente utilizando a internet.

## Git Bash

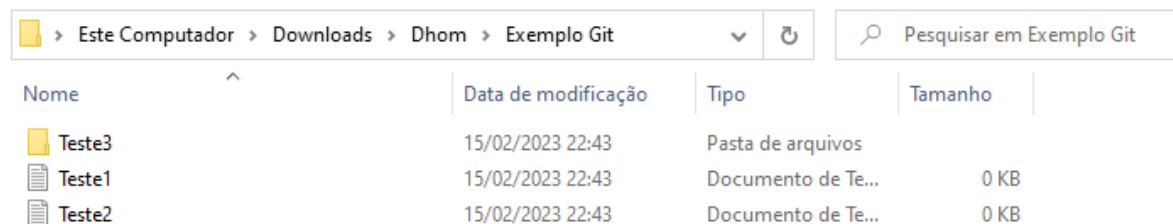
Git Bash é um aplicativo para ambientes do Microsoft que emula a experiência de linha de comando do Git. Nele, você trabalha com os comandos Git e consegue fazer o controle entre repositório local e remoto, arquivos, rastreamentos, branches e mais.



Git Bash aberto na pasta *teste git*

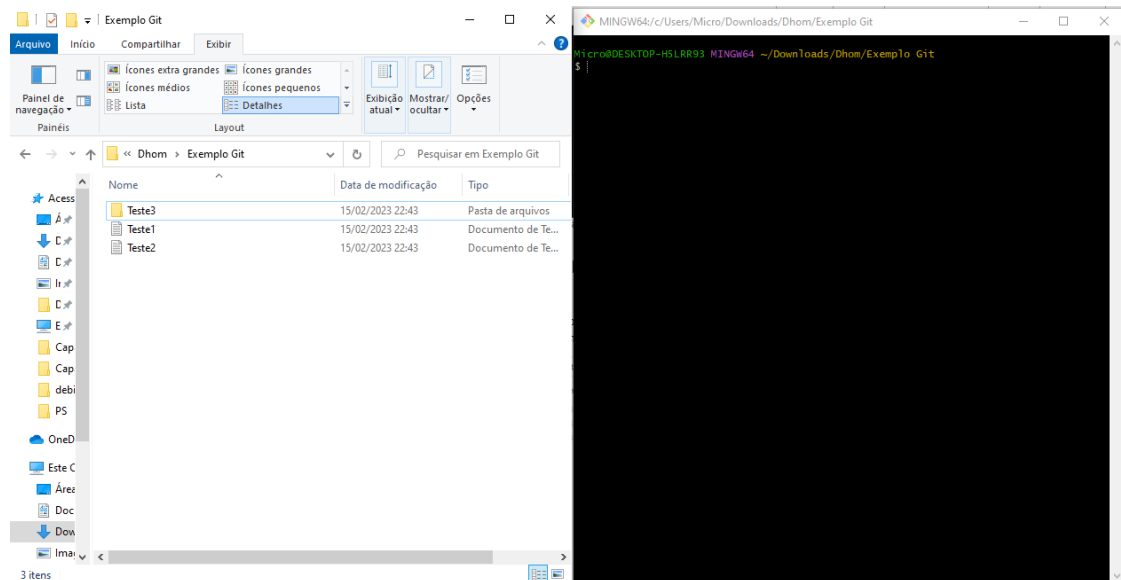
## Exemplo de utilização do Git e upload para o GitHub pelo Git Bash

Aqui teremos um exemplo de como utilizar o Git e realizar o upload para o GitHub através do Git Bash. Primeiro, criamos a seguinte pasta:



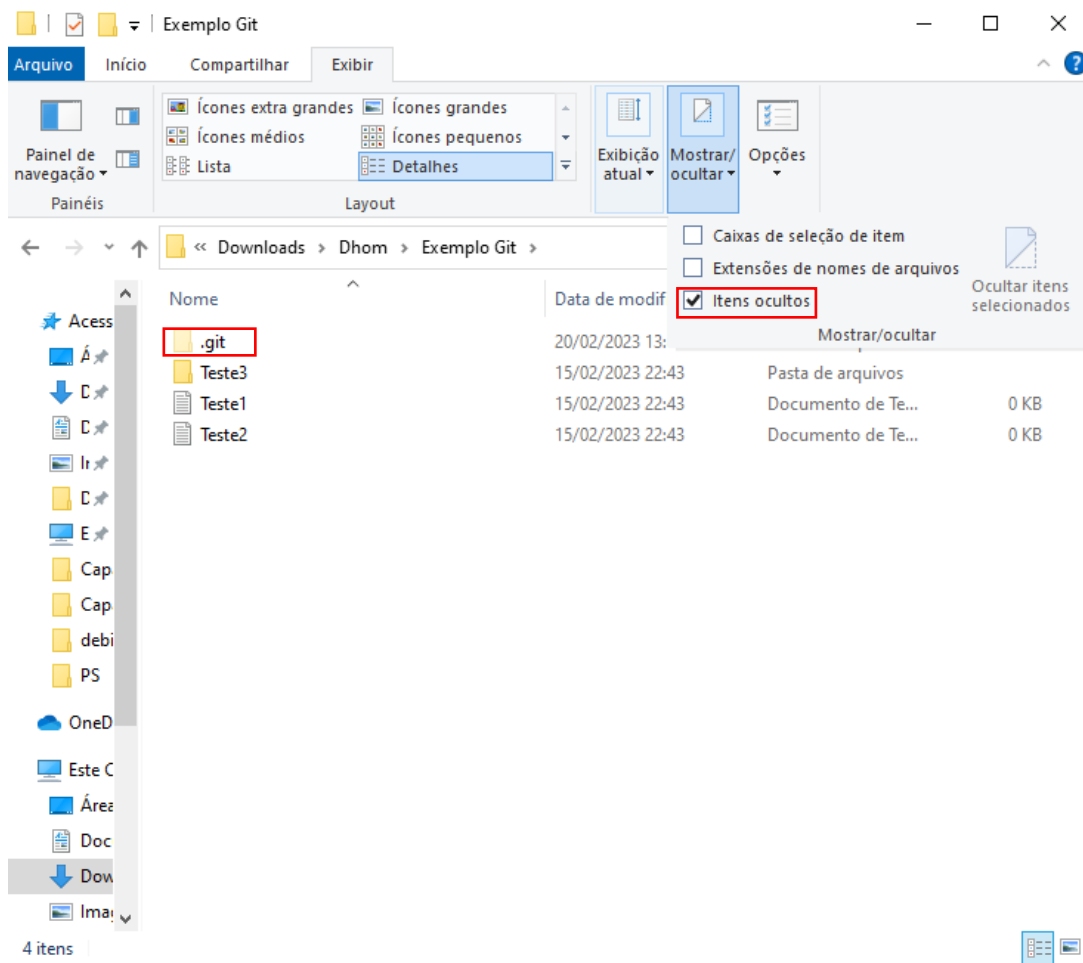
Nenhum dos arquivos possuem conteúdo a primeiro momento, e a pasta Testez também está vazia.

Primeiro, com o Git instalado no computador, abriremos o Git Bash nessa pasta clicando com o botão direito no mouse no espaço vazio e selecionar "*Git Bash Here*". Isso fará com que o Git Bash use este caminho para configurar o repositório local, sem necessidade de caminhar entre diretórios.



Para adicionarmos à pasta Exemplo Git um repositório local. Digitaremos ***git init*** no Git Bash. Para confirmarmos que o comando foi bem-sucedido, podemos exibir arquivos ocultos. A pasta ***.git*** é seu repositório local daquela pasta.





Em seguida, vemos quais arquivos não estão rastreados através do **git status**.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  Teste1.txt
  Teste2.txt

nothing added to commit but untracked files present (use "git add" to track)
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |
```

Notamos que apenas os arquivos .txt são mostrados como não rastreados. Isso acontece porque o Git não rastreia pastas vazias.

Para adicionarmos os arquivos à Staging area e posteriormente lançar no repositório local, utilizamos o **git add**. É possível utilizar o **git add** para adicionar todos os arquivos não rastreados, ou especificar o arquivo que queremos com **git add (nome do arquivo)**.

Primeiro, o **git add** .:

```

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git add .

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Teste1.txt
        new file:   Teste2.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |

```

Vemos que o Git agora mostra que os arquivos Teste1.txt e Teste2.txt são novos à Staging area. Isso significa que o **git add** deu certo.

Porém, caso você queira remover algum arquivo, você pode utilizar o **git rm --cached (nome do arquivo)**. Dessa maneira:

```

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git rm --cached Teste2.txt
rm 'Teste2.txt'

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Teste1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Teste2.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |

```

Vemos agora que o arquivo Teste2.txt não está mais rastreado pelo Git. Vamos adicioná-lo usando o **git add** e especificando seu nome.

```

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git add Teste2.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Teste1.txt
        new file:   Teste2.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$

```

Em seguida, vamos realizar nosso primeiro commit para o repositório local. Aqui, utilizamos o comando ***git commit -m "(mensagem)"***. A mensagem é obrigatória, e ela deve explicar o que há de diferente naquele commit de forma direta e simples. Como é nosso primeiro commit, podemos colocar “primeira commit”.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git commit -m "primeira commit"
[master (root-commit) 8884b5e] primeira commit
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Teste1.txt
create mode 100644 Teste2.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$
```

Agora, vamos verificar em qual branch estamos trabalhando com o comando ***git branch***. Ele é um comando que exibe todas as branches existentes e em qual estamos trabalhando.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git branch
* master

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |
```

Vemos aqui que existe apenas a master branch e que é nela que estamos trabalhando. Vamos criar outra branch para modificarmos os arquivos de texto com o comando ***git checkout -b [nome da branch]***. Utilizaremos o nome “alteracao” como nome da branch. Quando usamos esse comando, significa que criamos uma nova branch ao mesmo tempo que copiamos e colamos o conteúdo da branch anterior, basicamente uma “cópia” da branch.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git checkout -b alteracao
Switched to a new branch 'alteracao'

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$
```

No console está escrito “Switched to a new branch ‘alteracao’”. Traduzindo, significa que trocamos para uma nova branch com nome ‘alteracao’. Agora, iremos modificar o arquivo Teste1.txt adicionando qualquer texto a ele e em seguida salvar.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$ git status
On branch alteracao
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Teste1.txt

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

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$
```



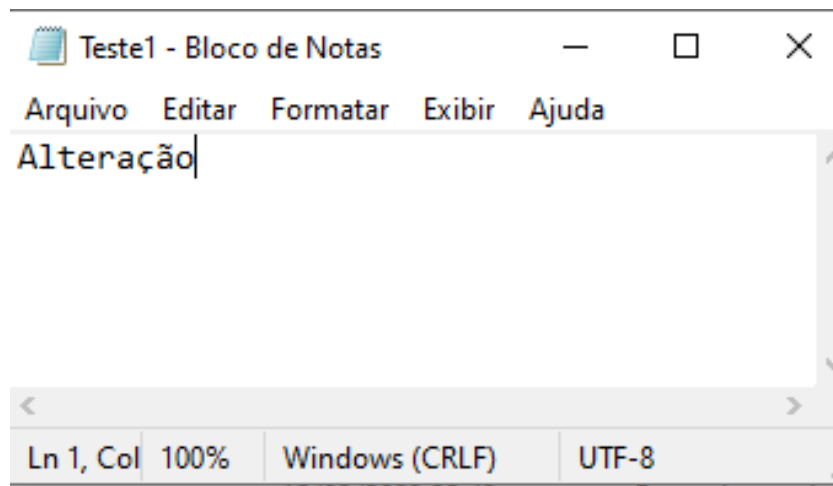
Utilizando o **git status**, vemos que o Git nos informa sobre o arquivo modificado. Vamos adicionar as modificações e comitá-las.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$ git add .

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$ git status
On branch alteracao
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Teste1.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$ git commit -m "alteração na nova branch"
[alteracao 18040a8] alteração na nova branch
1 file changed, 1 insertion(+)
```

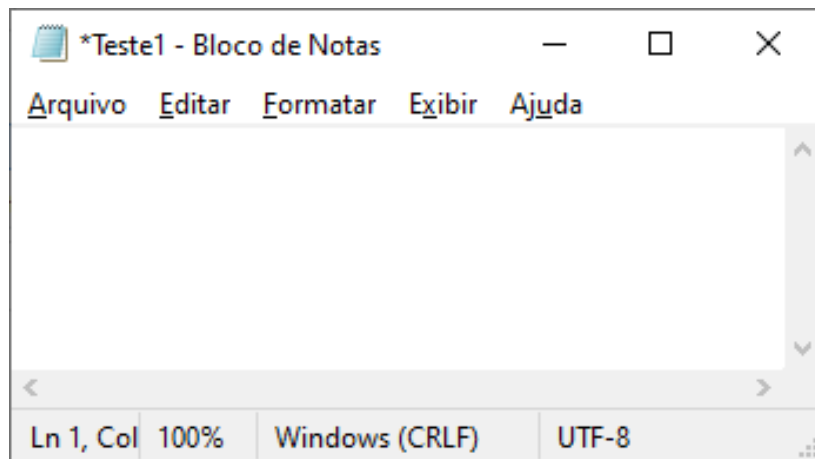
Com o commit bem-sucedido na nova branch, podemos fazer um teste de versão. Meu arquivo ficou assim após a modificação:



Vamos fechar este arquivo e trocar de branch com o comando **git checkout** para a master. Em seguida, abriremos o mesmo arquivo Teste1.txt.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (alteracao)
$ git checkout master
Switched to branch 'master'

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |
```



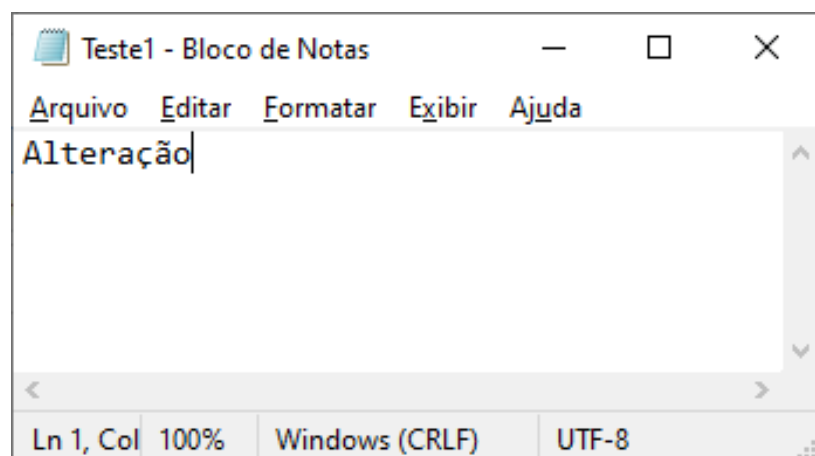
A modificação feita na branch 'alteracao' não aparece aqui. Esse é um exemplo do versionamento que o Git controla.

Para fundirmos ambas as branches e fazer com que a modificação vá para a master branch, utilizamos o comando ***git merge [nome da branch]***. É importante entender que para utilizar este comando, você precisa estar na branch para onde você quer transferir a modificação, e na branch onde foi feita a modificação.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git merge alteracao
Updating 8884b5e..18040a8
Fast-forward
 Teste1.txt | 1 +
 1 file changed, 1 insertion(+)

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$
```

Fechando o documento e o reabrindo, vemos que a modificação agora está no arquivo da master branch:





Agora, iremos lançar nosso repositório local para um repositório remoto no GitHub. Neste exemplo, criaremos o repositório remoto no perfil do GitHub primeiro.

## Create a new repository

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

---

**Owner \*** **Repository name \***


 dhomimonteiro / Exemplo-Git 


Great repository names are short and memorable. Need inspiration? How about [musical-bassoon?](#)

**Description (optional)**

Exemplo de git

---

☒  **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:**  
Skip this step if you're importing an existing repository.

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


**Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None**

**Choose a license**  
A license tells others what they can and can't do with your code. [Learn more.](#)

License: **None**

---

 You are creating a public repository in your personal account.

[Create repository](#)

O README é essencial para descrever seu projeto. Como já temos um repositório local e isto é apenas um exemplo, não adicionaremos um ao nosso repositório remoto. Após clicar em “*Create repository*”, a seguinte tela aparecerá:





dhomimonteiro / Exemplo-Git Public

Pin Unwatch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

---

**Quick setup — if you've done this kind of thing before**


 Set up in Desktop or  HTTPS  SSH <https://github.com/dhomimonteiro/Exemplo-Git.git> 

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

---

**...or create a new repository on the command line**


```
echo "# Exemplo-Git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/dhomimonteiro/Exemplo-Git.git
git push -u origin main
```



---

**...or push an existing repository from the command line**

```
git remote add origin https://github.com/dhomimonteiro/Exemplo-Git.git
git branch -M main
git push -u origin main
```



---

**...or import code from another repository**  
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

Nela, são explicadas as formas de adicionar um repositório local ao repositório remoto, seja um já existente ou um novo. Nós usaremos as linhas de comando do segundo exemplo, para repositórios locais já existentes.

O comando ***git remote add origin [URL do repositório remoto]*** irá dizer para o Git qual o repositório remoto que ele irá salvar o repositório local toda vez que realizarmos o push.

O comando ***git branch -M main*** trocará o nome da master para main, e não será necessário neste exemplo. Podemos pular ele.

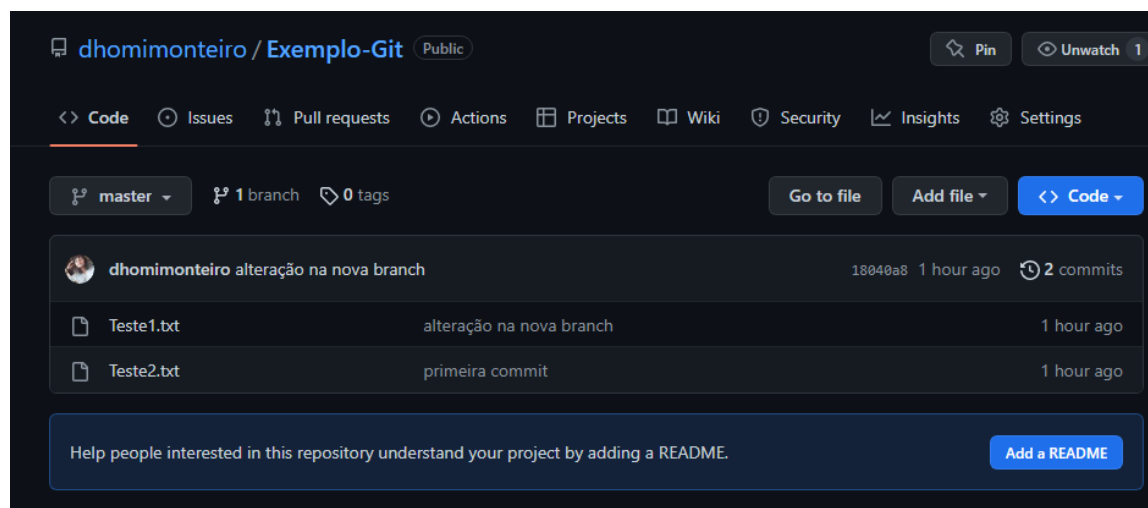
O comando ***git push -u origin master*** será o responsável de salvar o repositório local no repositório remoto.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git remote add origin https://github.com/dhomimonteiro/Exemplo-Git.git

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git push -u origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 486 bytes | 10.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/dhomimonteiro/Exemplo-Git.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |
```

Se recarregarmos a página no navegador, veremos que os arquivos estão presentes no repositório remoto.



Podemos notar que as mensagens escritas nos commits aparecem entre o nome do arquivo e quando foram modificadas. Por isso elas devem ser explicativas, simples e diretas.

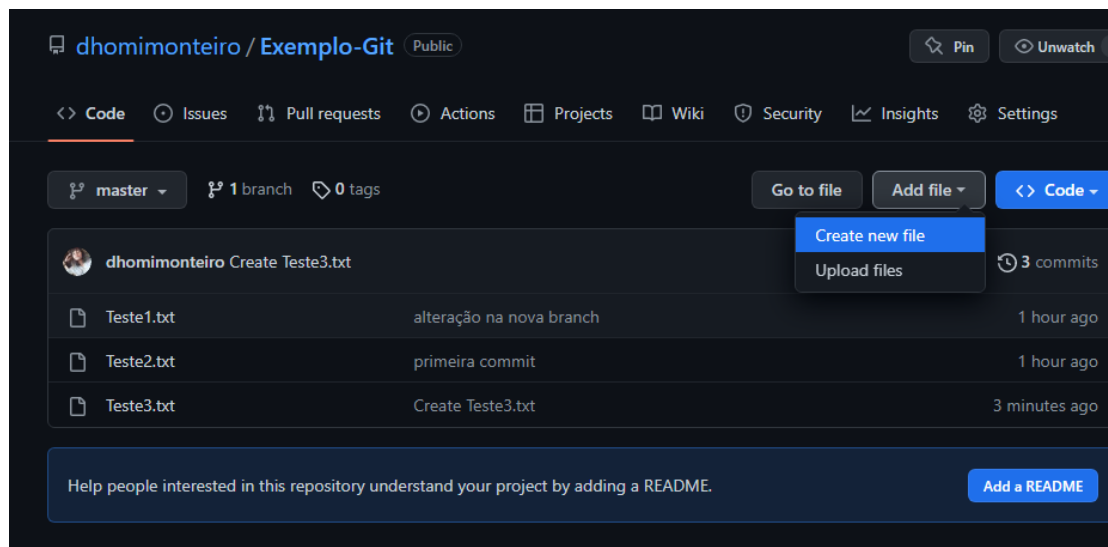
Também é possível pegar arquivos do repositório remoto para seu repositório local com o comando ***git pull***.

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git pull
Already up to date.

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |
```

No console aparece que o repositório local já está atualizado. Isso acontece porque não há diferença entre os arquivos presentes em cada um.

Para vermos funcionar o comando, adicionaremos um arquivo diretamente ao repositório remoto através do botão “Add file > Create file”.



Na imagem, já está criado um arquivo nomeado “Teste3.txt”, com o texto “Teste git pull”.

Realizando novamente o *git pull*, aparecerá:

```
Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 707 bytes | 1024 bytes/s, done.
From https://github.com/dhomimonteiro/Exemplo-Git
  18040a8..5b692e8 master    -> origin/master
Updating 18040a8..5b692e8
Fast-forward
 Teste3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Teste3.txt

Micro@DESKTOP-H5LRR93 MINGW64 ~/Downloads/Dhom/Exemplo Git (master)
$ |
```

Se verificarmos nossa pasta com o repositório local, veremos que agora temos o arquivo Teste3.txt no nosso Explorador de Arquivos:

<div> <div> <div>↑</div> <div> <div> <div>« Downloads »</div> <div>Dhom »</div> <div>Exemplo Git</div> </div> <div> <div>▼</div> <div>↺</div> </div> </div> <div> <div>🔍</div> <div>Pesquisar em Exemplo Git</div> </div> </div> </div>				
Nome	Data de modificação	Tipo	Tamanho	
.git	20/02/2023 16:24	Pasta de arquivos		
Teste3	15/02/2023 22:43	Pasta de arquivos		
Teste1	20/02/2023 15:26	Documento de Te...	1 KB	
Teste2	15/02/2023 22:43	Documento de Te...	0 KB	
Teste3	20/02/2023 16:24	Documento de Te...	1 KB	

## Cheatsheet de Comandos Git

### Comandos básicos

**Git init** → Inicializa um repositório local na pasta.

**Git status** → Exibe os arquivos não rastreados, arquivos na Staging area e arquivos modificados.

**Git add** → Adiciona os arquivos não rastreados à Staging area.

*Git add .* → Adiciona todos os arquivos não rastreados.

*Git add [nome do arquivo]* → Adiciona o arquivo especificado.

**Git rm --cached [nome do arquivo]** → Exclui o arquivo especificado da Staging area e ele volta a ser um arquivo não rastreado.

**Git commit -m "[mensagem]"** → Envia os arquivos da Staging area para o repositório local. É necessário inserir uma mensagem, e é ideal que você seja sucinto ao descrever o que aquele commit possui de diferente. No primeiro commit, é normal escrever "first commit" ou "primeiro commit".

**Git commit -am "[mensagem]"** → Junção do Git add com o Git commit.

**Git checkout [nome da branch]** → Usado para navegar entre diferentes branches e cria-las, caso a branch especificada não exista.

**Git branch** → Exibe todas as branches existentes no repositório local/remoto e especifica qual você está trabalhando com um asterisco (\*) antes do nome e a cor verde.

**Git merge [nome da branch]** → Funde a branch em que você está com a branch especificada no comando. Por exemplo, caso você queira fundir uma branch chamada "nova-função" à branch master, você precisa estar dentro da branch master e escrever "git merge nova-função".

**Git remote add origin [URL do repositório remoto]** → Adiciona o caminho na internet do repositório remoto que o Git deve vincular aquele repositório local.

*Git push -u origin [nome da branch no repositório remoto]* → Responsável por realizar o push do repositório local para o repositório remoto.

*Git pull* → Transfere arquivos presentes no repositório remoto para o repositório local, caso ele já não tenha.

### *Comandos especiais*

*Git config --global user.name "[primeiro-nome segundo-nome]"* → Configura um nome identificável do usuário para poder monitorar em reviews de histórico.

*Git config --global user.email "[seu email]"* → Configura o e-mail válido do usuário que aparecerá no histórico de versionamento.

*Git config --global color.ui auto* → Configuração padrão de cor para o console do Git.

*Git clone [URL do repositório remoto]* → Clona um repositório remoto inteiro para o computador, já criando um repositório local.

*Git log* → Mostra todos os commits realizados na branch atual.

*Git log [nome da branch1] [nome da branch2]* → Mostra os commits feitos na branch1 que não foram feitos na branch2.

*Git log --follow [arquivo]* → Mostra os commits que alteraram o arquivo especificado, mesmo após renomear.

*Git diff [nome da branch1] [nome da branch2]* → Mostra as diferenças entre a branch1 e a branch2.