

# **Noções básicas sobre Git e GitHub**

**Lucas Matheus Casarotti Rodrigues**

## Sumário

1. Introdução .....	4
1.1. O que é controle de versão? .....	4
1.2. O que é Git? .....	4
1.3. O que é um repositório? .....	4
1.4. O que é Github? .....	4
2. Git fundamental .....	5
2.1. Criando um repositório remoto no GitHub .....	5
2.2. Inicializando um repositório local .....	6
2.3. Enviando arquivos pela primeira vez ao repositório remoto .....	7
2.4. Enviando arquivos pela segunda vez ao repositório remoto .....	8
2.4.1. Verificando mudanças do projeto .....	8
2.4.2. Adicionando arquivos ao projeto .....	9
2.4.3. Salvando alterações .....	9
2.4.4. Enviando código para o repositório .....	11
2.5. Recebendo alterações .....	11
2.6. Clonando repositórios .....	13
2.7. Removendo arquivos do repositório remoto .....	14
2.8. Histórico de alterações .....	15
2.9. Renomeando arquivos .....	16
2.9.1. Movendo arquivos para outra pasta .....	16
2.9.2. Renomeando arquivos .....	17
2.10. Desfazendo alterações com git checkout .....	18
2.11. Desfazendo alterações com git reset .....	19
2.11.1. Utilizando o comando 'git reset --hard HEAD~1' .....	19
2.11.2. Utilizando o comando 'git reset --soft HEAD~1' .....	21
2.12. Desfazendo alterações com git revert .....	21
2.13. Ignorando arquivos e diretórios em um projeto .....	23
2.14. Limpando arquivos untracked .....	26
2.15. Limpando arquivos staging .....	27
3. Trabalhando com branches .....	29
3.1. O que são branches? .....	29
3.2. Criando e visualizando uma branch .....	29
3.3. Deletando branches .....	30
3.4. Mudando de branch .....	30

3.4.1. Trocando de branch.....	30
3.4.2. Trocando e criando branch ao mesmo tempo .....	30
3.5. Unindo branches .....	31
3.6. Utilizando stash.....	33
3.7. Recuperando stash .....	34
3.8. Removendo a stash .....	35
3.9. Criando tags.....	36
3.10. Exibindo informações.....	38
3.11. Verificando diferenças .....	39

## **1. Introdução**

### **1.1. O que é controle de versão?**

É uma maneira de gerenciar o código-fonte de uma aplicação (software), tendo controle de todas as alterações do código e permitindo a reversão das mesmas. É possível criar várias versões de um mesmo software, cada uma contendo simultaneamente diferentes ou até mesmo mudanças diferentes para clientes específicos da empresa. Em empresas de software, é comum que cada desenvolvedor trabalhe em uma versão diferente, para isso é utilizada uma ferramenta, como o Git.

### **1.2. O que é Git?**

É uma das ferramentas utilizadas para controle de versão. O Git é baseado em repositórios que contêm todas as versões do código, bem como as cópias de trabalho de cada desenvolvedor.

Instalação git Windows

Acessar o site: <https://git-scm.com/downloads>

Fazer o download do executável;

E seguir as instruções

### **1.3. O que é um repositório?**

É onde o código é armazenado. Na maioria das vezes, é criado um repositório para cada projeto, e esses repositórios são hospedados em um servidor responsável pelo gerenciamento, como o GitHub.

### **1.4. O que é Github?**

É um servidor responsável por gerenciar os repositórios de forma gratuita. Nele, é possível enviar e disponibilizar projetos para outras pessoas, assim como baixá-los em outras máquinas. Além disso, é possível colaborar com outros desenvolvedores na criação de projetos em conjunto.

Criar conta no GitHub

Acessar o site: <https://github.com/>

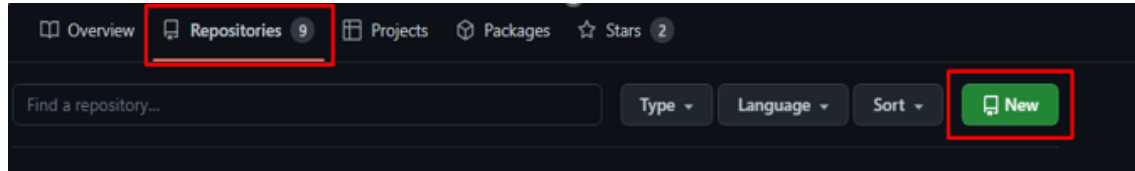
E realizar o cadastro de usuário

## 2. Git fundamental

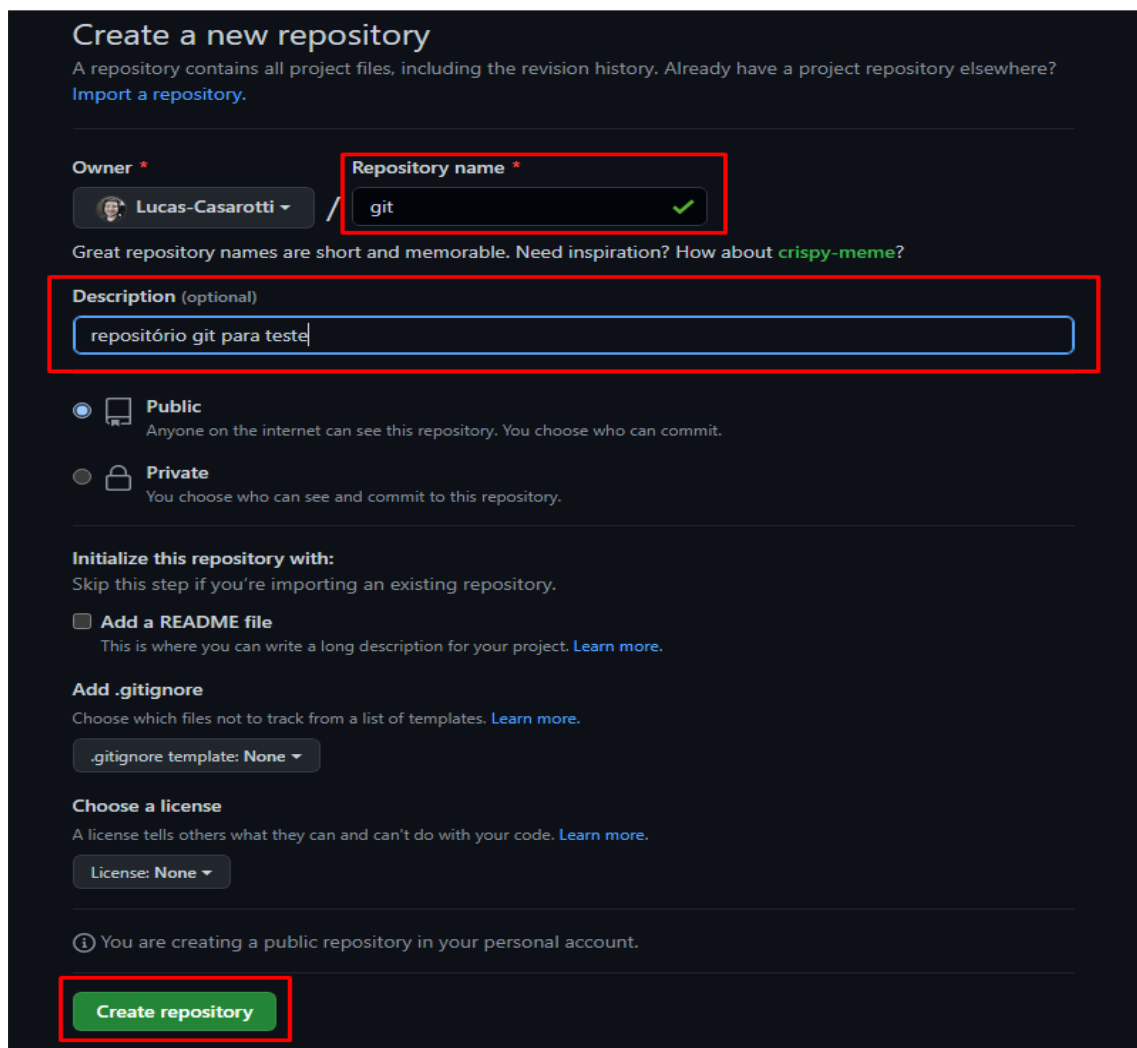
### 2.1. Criando um repositório remoto no GitHub

1 – Acesse o site do GitHub com seu nome de usuário e senha.

2 – Clique em 'Repositories' e em seguida, em 'New'.



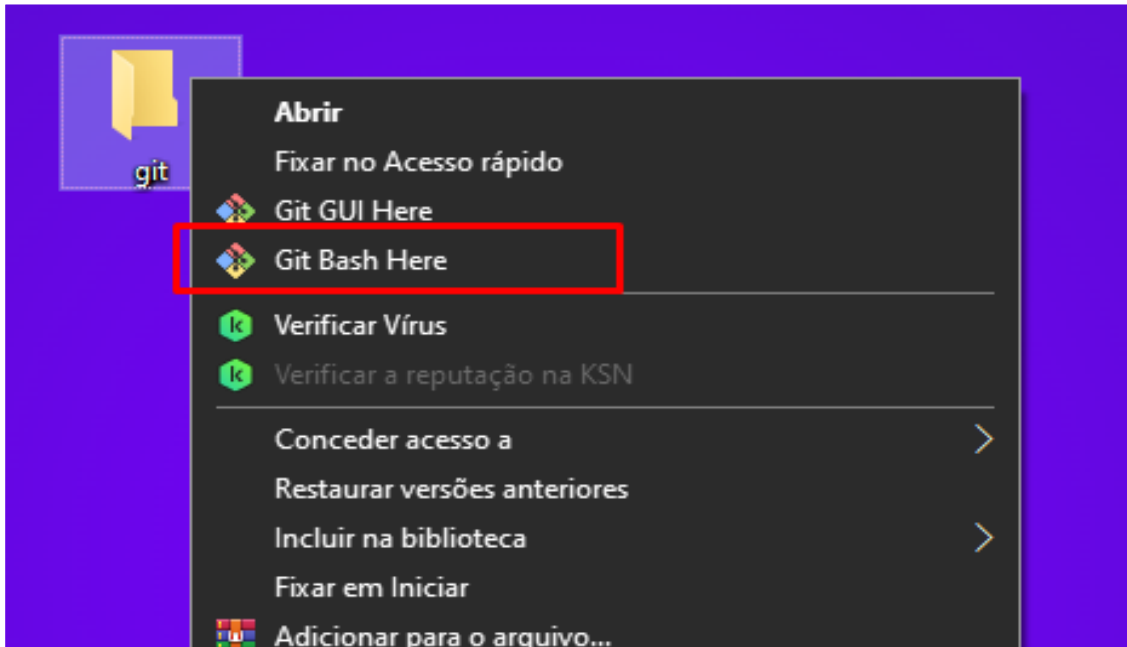
3 – Digite o nome do repositório, adicione uma descrição (opcional), selecione a opção 'Public' para tornar o repositório visível para todas as pessoas e clique em 'Create repository'.

A screenshot of the 'Create a new repository' page on GitHub. The form contains several fields and options: the 'Owner' is set to 'Lucas-Casarotti'; the 'Repository name' field contains 'git' and is highlighted with a red box; the 'Description' field contains 'repositório git para teste' and is also highlighted with a red box; the 'Public' radio button is selected; the 'Initialize this repository with' section has 'Add a README file' checked; the '.gitignore' template is set to 'None'; and the 'License' is set to 'None'. At the bottom, a green 'Create repository' button is highlighted with a red box. A note at the bottom states: 'You are creating a public repository in your personal account.'

## 2.2. Inicializando um repositório local

1 – Crie uma pasta na sua máquina.

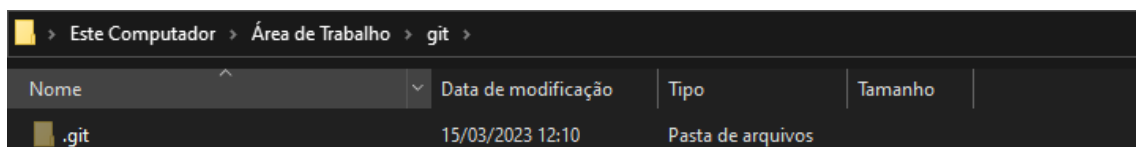
2 – Clique com o botão direito na pasta e, em seguida, clique em 'Git Bash Here' para abrir o terminal do Git.



3 – No terminal do Git, execute o comando 'git init' para inicializar o repositório local.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git
$ git init
Initialized empty Git repository in E:/Usuários/Lucas/Área de Trabalho/git/.git/
```

4 – Logo em seguida, uma pasta oculta será automaticamente criada dentro da pasta que criamos. Isso indica que um repositório local foi criado.



## 2.3. Enviando arquivos pela primeira vez ao repositório remoto

1 – Adicione um arquivo chamado '01teste.txt' à pasta do repositório local.

Nome	Data de modificação	Tipo	Tamanho
.git	13/02/2023 23:14	Pasta de arquivos	
01teste	07/02/2023 23:58	Documento de Te...	0 KB

2 – No terminal do Git, execute o comando 'git status'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    01teste.txt

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

3 – No terminal do Git, execute o comando 'git add 01teste.txt'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (master)
$ git add 01teste.txt

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (master)
$
```

4 – No terminal do Git, execute o comando 'git commit -m "nome do commit"'.  
"v1"

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (master)
$ git commit -m "v1"
[master (root-commit) c128f32] v1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 01teste.txt
```

5 – No terminal do Git, execute o comando 'git branch -m main'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (master)
$ git branch -m main
```

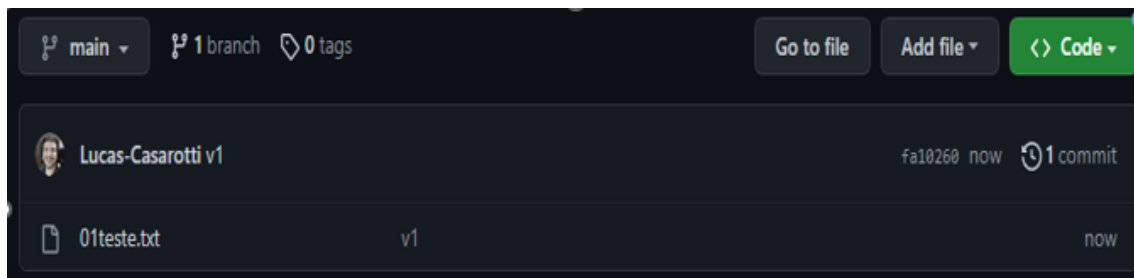
6 – No terminal do Git, execute o comando 'git remote add origin link.repositorio'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git remote add origin https://github.com/Lucas-Casarotti/git.git
```

7 – No terminal do Git, execute o comando 'git push -u origin main'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 212 bytes | 212.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Lucas-Casarotti/git.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

8 – Após executar o comando 'git push -u origin main', podemos observar que o arquivo '01teste.txt' foi adicionado ao repositório remoto. Dessa forma, podemos acessar nosso repositório e seus arquivos de qualquer outra máquina.



## 2.4. Enviando arquivos pela segunda vez ao repositório remoto

### 2.4.1. Verificando mudanças do projeto

As mudanças no projeto podem ser verificadas com o comando 'git status'. Esse comando irá mapear todas as alterações realizadas no projeto, como arquivos novos e arquivos modificados.

Exemplo:

1 – Adicione três arquivos .txt na pasta do repositório local: '02teste.txt', '03teste.txt' e '04teste.txt'.

Nome	Data de modificação	Tipo	Tamanho
.git	15/03/2023 11:05	Pasta de arquivos	
01teste	15/03/2023 10:33	Documento de Te...	0 KB
02teste	15/03/2023 10:33	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
04teste	15/03/2023 10:33	Documento de Te...	0 KB

2 – No terminal do Git, execute o comando 'git status' para verificar os arquivos criados ou alterados que não estão sendo mapeados pelo Git.

```
MINGW64:/e/Usuários/Lucas/Área de Trabalho/git
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    02teste.txt
    03teste.txt
    04teste.txt

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



### 2.4.2. Adicionando arquivos ao projeto

Para adicionar arquivos novos a um projeto, utilizamos o comando 'git add'. Podemos adicionar um arquivo específico ou vários arquivos. É importante lembrar que todos os arquivos que forem modificados devem ser adicionados, caso contrário, eles não serão monitorados pelo Git e não serão enviados para o repositório remoto.

Exemplo:

1 – No terminal do Git, execute o comando 'git add .' para adicionar todos os arquivos criados/alterados de uma única vez. É importante lembrar que dessa maneira todos os arquivos criados/alterados serão adicionados, portanto, se não for o caso, é recomendado utilizar o comando 'git add nome do arquivo' apenas para os arquivos que devem ser adicionados ao repositório e posteriormente enviados para o repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git add .
```

2 – No terminal do Git, execute o comando 'git status' novamente. Logo em seguida, você poderá ver que os novos arquivos estão com a cor verde, o que significa que estão no estágio de 'stage', ou seja, estão prontos para serem commitados e enviados para o repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   02teste.txt
        new file:   03teste.txt
        new file:   04teste.txt
```

### 2.4.3. Salvando alterações

Para salvar as alterações realizadas no repositório, utilizamos o comando 'git commit'. É possível efetuar commits em arquivos específicos ou em vários arquivos de uma vez usando a flag '-a'. Também é considerada uma boa prática incluir uma mensagem com a descrição das alterações em cada commit, para isso utilizamos a flag '-m'.

Exemplo:

#### 2.4.3.1. Salvando apenas um arquivo

1 – No terminal do Git, execute o comando 'git commit -m "nome do commit"' para enviar somente um arquivo ao repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit 02teste.txt -m "v2 - enviando um arquivo"
[main d28cc7b] v2 - enviando um arquivo
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 02teste.txt
```

2 – No terminal do Git, execute o comando 'git status' novamente. Observe que o arquivo '01teste.txt' não está sendo listado, o que significa que ele já foi commitado e pode ser enviado ao repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   03teste.txt
        new file:   04teste.txt
```

#### 2.4.3.2. Salvando todos os arquivos

1 – No terminal do Git, execute o comando 'git commit -a -m "nome do commit"' para enviar todos os arquivos ao repositório remoto.

A flag -m (abreviação de "message") é usada para adicionar uma mensagem de commit diretamente na linha de comando, sem abrir o editor de texto padrão. Por exemplo, o comando "git commit -m 'Adicionando nova funcionalidade'" adicionaria um commit com a mensagem "Adicionando nova funcionalidade".

Já a flag -a (abreviação de "all") é usada para adicionar automaticamente as mudanças em todos os arquivos que foram modificados (ou deletados) desde o último commit. Isso evita a necessidade de usar o comando "git add" separadamente para cada arquivo modificado. No entanto, é importante lembrar que a flag -a não adiciona novos arquivos que foram criados.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit -a -m "v2 - enviando todos os arquivos"
[main 900d611] v2 - enviando todos os arquivos
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 03teste.txt
create mode 100644 04teste.txt
```

2 – No terminal do Git, execute o comando 'git status' novamente. Observe que nenhum outro arquivo está sendo listado, o que significa que todos os arquivos foram commitados.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
```

#### 2.4.4. Enviando código para o repositório

Quando terminamos de alterar/criar um novo arquivo, precisamos enviar o código para o repositório remoto seguindo os passos mostrados anteriormente. A ação de enviar as alterações para o repositório remoto é feita pelo comando 'git push'.

Exemplo:

1 – No terminal do Git, execute o comando 'git push'. Repare que o comando 'git push' está diferente em comparação com a primeira vez, pois na segunda vez já está apontando para qual branch iremos enviar, no caso, a branch principal.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 445 bytes | 445.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/Lucas-Casarotti/git.git
 3848261..900d611  main -> main
```

2 – Após executar o comando 'git push', é possível verificar que os arquivos foram adicionados ao repositório remoto.

Observe que foram criados dois commits: um com o nome de 'v2 - enviando um arquivo' e outro com 'v2 - enviando todos os arquivos'. Isso ocorreu porque no tópico anterior fizemos dois commits separados, um para cada ocasião.

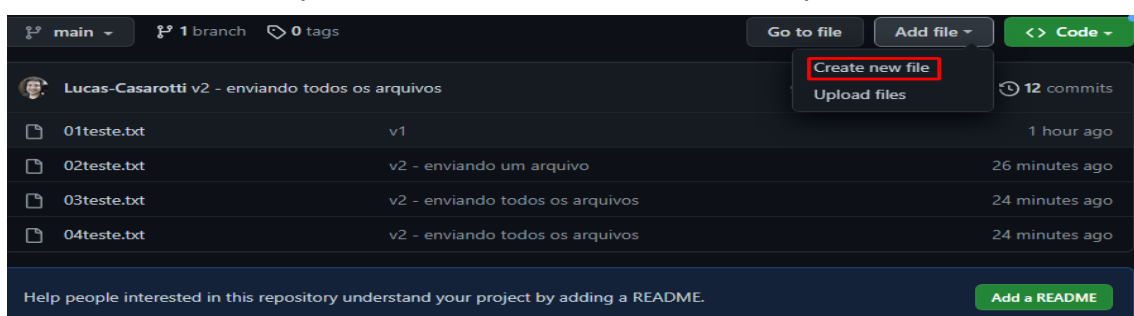
01teste.txt	v1	1 hour ago
02teste.txt	v2 - enviando um arquivo	21 minutes ago
03teste.txt	v2 - enviando todos os arquivos	20 minutes ago
04teste.txt	v2 - enviando todos os arquivos	20 minutes ago

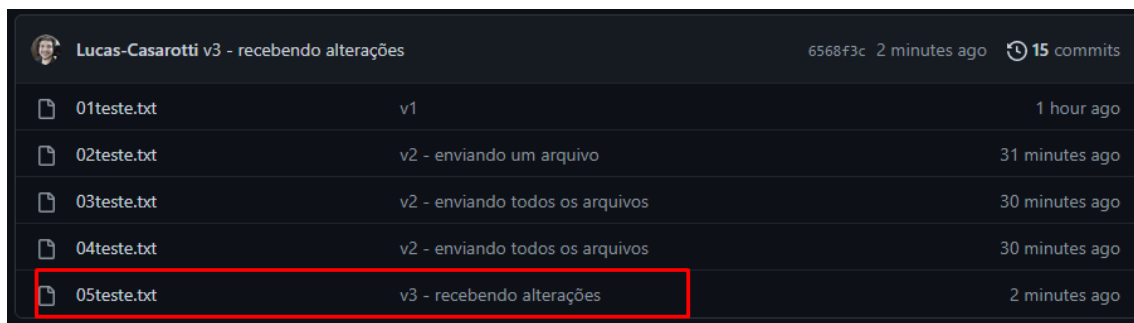
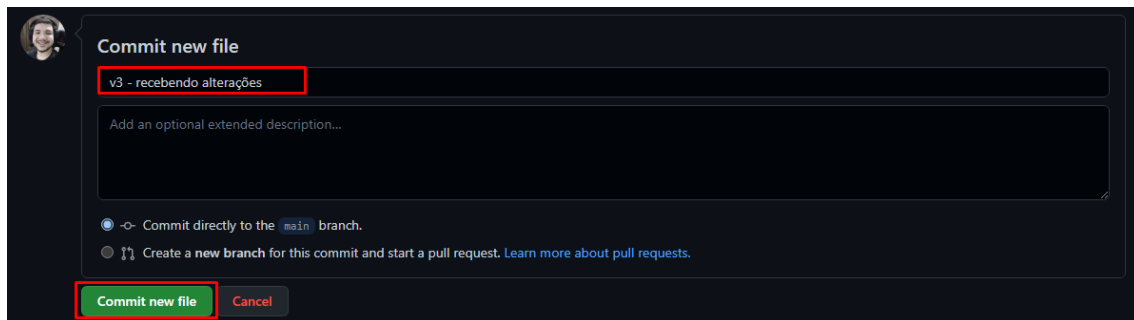
#### 2.5. Recebendo alterações

É comum trabalharmos em equipe e, em algumas situações, um dos desenvolvedores pode ter enviado uma nova funcionalidade para o repositório remoto. A partir desse momento, nosso repositório local fica desatualizado e não terá as novas alterações dessa funcionalidade. Para sincronizar nosso repositório local com as mudanças do remoto, é necessário utilizar o comando 'git pull', que buscará as atualizações e as juntará com o código atual do repositório local.

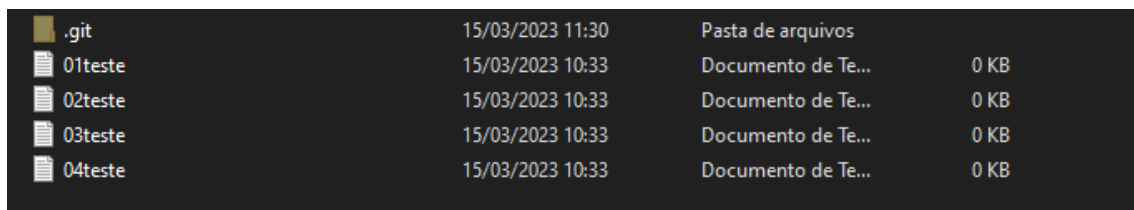
Exemplo:

1 – Crie um novo arquivo chamado de '05teste.txt' no repositório remoto.





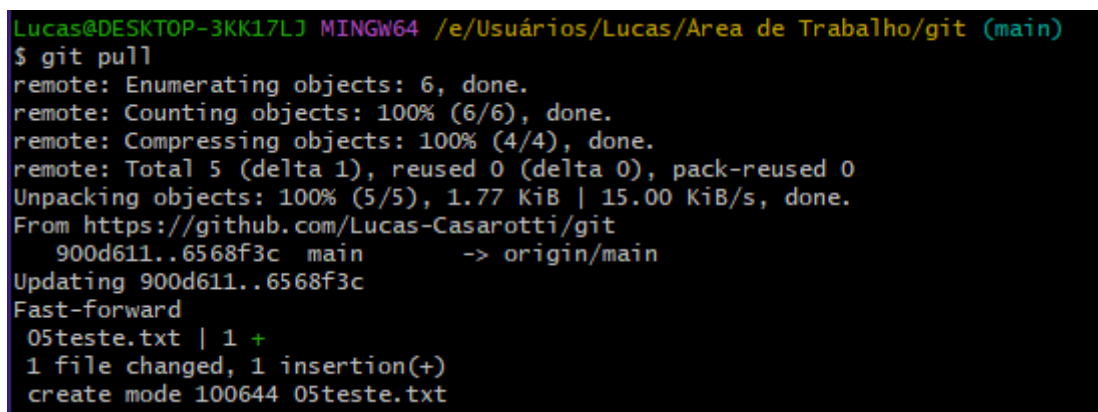
2 – Logo em seguida, vamos verificar se esse arquivo existe em nosso repositório local.



2.1 – Podemos observar que não há nenhum arquivo chamado "05teste.txt", o que indica que nosso repositório local está desatualizado em relação ao repositório remoto.

2.2 – Para atualizar o nosso repositório local com as mudanças do repositório remoto, utilizamos o comando "git pull".

3 – No terminal do Git, execute o comando 'git pull'.



4 – Agora, ao verificarmos a pasta do nosso repositório local, é possível ver que o arquivo "05teste.txt" foi baixado e está presente no diretório.

Nome	Data de modificação	Tipo	Tamanho
.git	15/03/2023 12:02	Pasta de arquivos	
01teste	15/03/2023 10:33	Documento de Te...	0 KB
02teste	15/03/2023 10:33	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
04teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	15/03/2023 12:02	Documento de Te...	1 KB

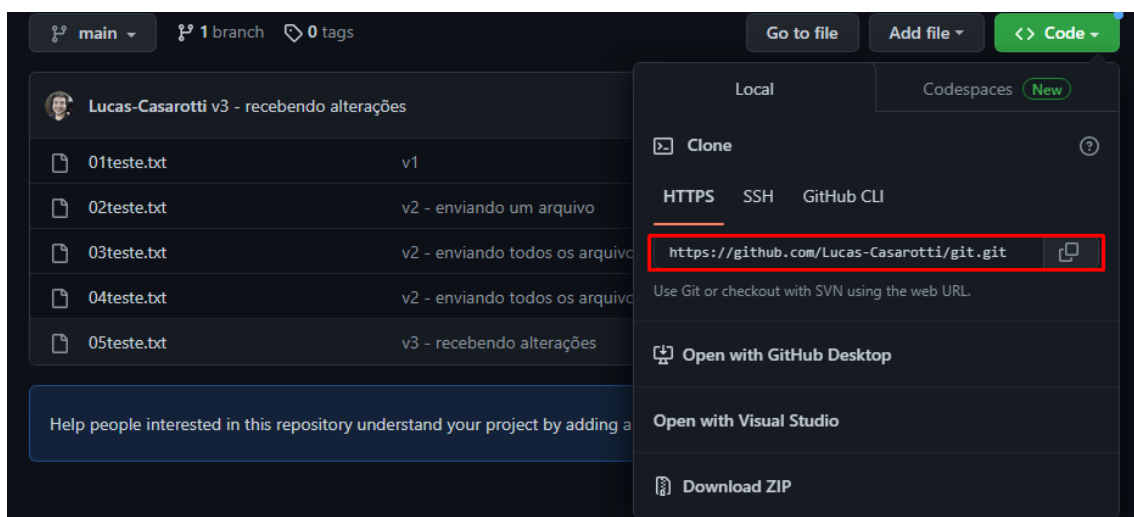
## 2.6. Clonando repositórios

O ato de baixar um repositório de um servidor remoto é chamado de “clonar repositório”. Em uma empresa, quando entramos em um novo projeto, é comum termos que clonar o repositório que contém o projeto em questão. Para isso, utilizamos o comando ‘git clone’.

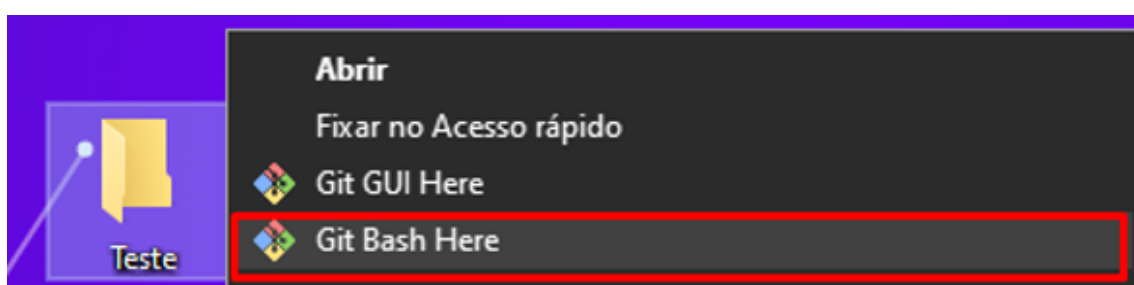
Exemplo:

1 – Podemos clonar o repositório diretamente na nossa área de trabalho ou em uma pasta específica. Neste exemplo, iremos criar uma nova pasta chamada “Teste”.

2 – Para realizar o clone de um determinado repositório é necessário ter acesso ao link do mesmo.



3 – Logo em seguida, abriremos o terminal do Git na pasta que criamos.



4 – Após copiar o link do repositório, abra o terminal do Git e execute o comando "git clone link\_do\_repositório" na pasta em que deseja clonar o repositório.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/Teste
$ git clone https://github.com/Lucas-Casarotti/git.git
Cloning into 'git'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 16 (delta 5), reused 5 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (5/5), done.
```

5 – Logo em seguida, podemos ver que na pasta que criamos foi criada outra pasta com o nome do repositório e com todo o conteúdo atual do repositório remoto.

Nome	Data de modificação	Tipo	Tamanho
.git	15/03/2023 12:05	Pasta de arquivos	
01teste	15/03/2023 12:05	Documento de Te...	0 KB
02teste	15/03/2023 12:05	Documento de Te...	0 KB
03teste	15/03/2023 12:05	Documento de Te...	0 KB
04teste	15/03/2023 12:05	Documento de Te...	0 KB
05teste	15/03/2023 12:05	Documento de Te...	1 KB

## 2.7. Removendo arquivos do repositório remoto

É possível deletar arquivos do repositório local e do repositório remoto usando o comando "git rm".

Exemplo:

1 – No terminal do Git, execute o comando 'git rm' seguido do nome do arquivo que desejamos excluir.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git rm 04teste.txt
rm '04teste.txt'
```

2 – Logo em seguida, podemos ver que na pasta do repositório local o arquivo "04teste.txt" foi excluído.

Nome	Data de modificação	Tipo	Tamanho
.git	15/03/2023 12:08	Pasta de arquivos	
01teste	15/03/2023 10:33	Documento de Te...	0 KB
02teste	15/03/2023 10:33	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	15/03/2023 12:02	Documento de Te...	1 KB

3 – No terminal do Git, execute o comando 'git status' para verificar quais arquivos foram excluídos.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    04teste.txt
```

4 – No terminal do Git, execute o comando 'git commit -m "nome do commit"' para subir essa nova alteração para o repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit -a -m "deletando arquivo desnecessário"
[main 33d1868] deletando arquivo desnecessário
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 04teste.txt
```

5 – No terminal do Git, execute o comando 'git push'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 247 bytes | 247.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
   6568f3c..33d1868  main -> main
```

6 – Após executar o comando 'git push', podemos ver que o arquivo "04teste.txt" foi removido do nosso repositório remoto.

Lucas-Casarotti deletando arquivo desnecessário			33d1868 1 minute ago	🕒 16 commits
📄 01teste.txt	v1			1 hour ago
📄 02teste.txt	v2 - enviando um arquivo			44 minutes ago
📄 03teste.txt	v2 - enviando todos os arquivos			42 minutes ago
📄 05teste.txt	v3 - recebendo alterações			14 minutes ago

## 2.8. Histórico de alterações

É possível ter acesso ao histórico de modificações feitas no projeto. O comando para este recurso é 'git log'. Com ele, você pode visualizar as informações dos commits realizados no projeto.

Exemplo:

1 – No terminal do Git, execute o comando 'git log'.



Observe que estão sendo listados os dois últimos commits realizados nos tópicos anteriores. Para sair do terminal git log, digite “q”.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git log
commit 33d1868397e189a2b6588b94fd399e785fc8c9 (HEAD -> main, origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date: Wed Mar 15 12:10:37 2023 -0300

    deletando arquivo desnecessário

commit 6568f3cd8beda362213540250a6998cf4ab4a4fc
Author: Lucas Casarotti <53920361+Lucas-Casarotti@users.noreply.github.com>
Date: Wed Mar 15 11:57:23 2023 -0300

    v3 - recebendo alterações
```

## 2.9. Renomeando arquivos

É possível renomear arquivos utilizando o comando ‘git mv’. O mesmo comando também pode ser usado para mover um arquivo para outra pasta.

Exemplo:

### 2.9.1. Movendo arquivos para outra pasta

1 – Para isso, vamos criar uma nova pasta chamada "Arquivos" dentro da pasta do repositório local.

Nome	Data de modificação	Tipo	Tamanho
.git	15/03/2023 12:10	Pasta de arquivos	
Arquivos	15/03/2023 14:23	Pasta de arquivos	
01teste	15/03/2023 10:33	Documento de Te...	0 KB
02teste	15/03/2023 10:33	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	15/03/2023 12:02	Documento de Te...	1 KB

2 – No terminal do Git, execute o comando ‘git mv 01teste.txt Arquivos/01teste.txt’. Com esse comando, o arquivo “01teste.txt” será movido para a pasta “Arquivos”.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git mv 01teste.txt Arquivos/01teste.txt
```

Este Computador > Área de Trabalho > git > Arquivos			
Nome	Data de modificação	Tipo	Tamanho
01teste	15/03/2023 10:33	Documento de Te...	0 KB

3 – No terminal do Git, após executar o comando ‘git status’, podemos ver que o arquivo 01teste.txt foi movido para a pasta "Arquivos" e renomeado para "Arquivos/01teste.txt".



```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    01teste.txt -> Arquivos/01teste.txt
```

4 – No terminal do Git, execute o comando ‘git commit -a -m “nome do git”’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit -a -m "v4 - movendo arquivos para outro pasta"
```

5 – No terminal do Git, execute o comando ‘git push’ para enviar as alterações ao repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 331 bytes | 331.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
    33d1868..36e056f  main -> main
```

6 – Logo em seguida, no repositório remoto, podemos ver que foi criada uma nova pasta e dentro desta pasta está o arquivo chamado de “01teste.txt”.

Lucas-Casarotti v4 - movendo arquivos para outra pasta 36e056f 6 minutes ago 17 commits		
Arquivos	v4 - movendo arquivos para outra pasta	6 minutes ago
02teste.txt	v2 - enviando um arquivo	3 hours ago
03teste.txt	v2 - enviando todos os arquivos	3 hours ago
05teste.txt	v3 - recebendo alterações	3 hours ago

Lucas-Casarotti v4 - movendo arquivos para outra pasta 36e056f 7 minutes ago History		
01teste.txt	v4 - movendo arquivos para outra pasta	7 minutes ago

### 2.9.2. Renomeando arquivos

1 – No terminal do Git, execute o comando ‘git mv Arquivos/01teste.txt Arquivos/01\_teste.txt’. Resumidamente, o arquivo “01teste.txt” que está dentro da pasta “Arquivos” será renomeado para “01\_teste.txt”.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git mv Arquivos/01teste.txt Arquivos/01_teste.txt
```

Este Computador > Área de Trabalho > git > Arquivos			
Nome	Data de modificação	Tipo	Tamanho
01_teste	15/03/2023 10:33	Documento de Te...	0 KB

2 – No terminal do Git, após executar o comando 'git status', podemos ver que o arquivo 01teste.txt foi renomeado para 01\_teste.txt.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:   Arquivos/01teste.txt -> Arquivos/01_teste.txt
```

3 – No terminal do Git, execute o comando 'git commit -a -m "nome do git"'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git commit -a -m "v5 - renomeando arquivos"
[main 0f6ca5f] v5 - renomeando arquivos
1 file changed, 0 insertions(+), 0 deletions(-)
rename Arquivos/{01teste.txt => 01_teste.txt} (100%)
```

4 – No terminal do Git, execute o comando 'git push' para enviar as alterações ao repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 309.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
   36e056f..0f6ca5f  main -> main
```

5 – Logo em seguida, no repositório remoto, podemos ver que o nome do arquivo "01teste.txt" foi alterado para "01\_teste.txt".

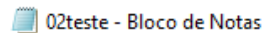
Lucas-Casarotti v5 - renomeando arquivos		0f6ca5f 1 minute ago	History
..			
01_teste.txt	v5 - renomeando arquivos	1 minute ago	

## 2.10. Desfazendo alterações com git checkout

Às vezes é necessário desfazer algumas alterações realizadas em um arquivo específico. Mesmo que apaguemos parte por parte, é possível que ainda fiquem algumas alterações indesejadas, como uma linha em branco. Para evitar isso, utilizamos o comando 'git checkout', que é responsável por fazer o arquivo modificado voltar ao estado original do repositório remoto.

Exemplo:

1 – Primeiro iremos alterar o arquivo “02teste.txt” inserindo um texto dentro.



Arquivo Editar Formatar Exibir Ajuda  
02teste

2 – No terminal do Git, execute o comando ‘git status’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   02teste.txt

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

3 – No terminal do Git, execute o comando ‘git checkout “nome do arquivo alterado”’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git checkout 02teste.txt
Updated 1 path from the index
```

4 – Ao executar o comando ‘git status’ novamente, podemos verificar que não há mais arquivos modificados.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

## 2.11. Desfazendo alterações com git reset

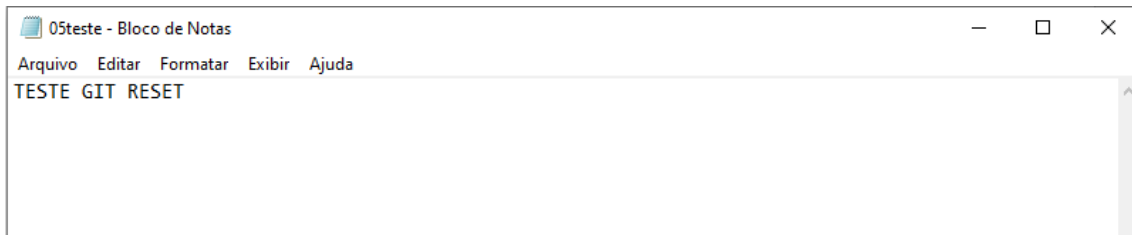
É possível desfazer o commit e todas as alterações que foram commitadas utilizando o comando ‘git reset --hard HEAD~1’. No entanto, é importante lembrar que todas as alterações serão excluídas, incluindo novos arquivos que foram criados. Portanto, é necessário ter cuidado ao usar este comando.

Também podemos utilizar o comando ‘git reset --soft HEAD~1’, a diferença desse comando para o anterior é que esse desfaz apenas os commits, mantendo as alterações e os novos arquivos no repositório local.

Exemplo:

### 2.11.1. Utilizando o comando ‘git reset --hard HEAD~1’

1 – Primeiro iremos alterar o arquivo “05teste.txt” inserindo um texto dentro.



2 – No terminal do Git, execute o comando ‘git status’.

```
MINGW64:/e/Usuários/Lucas/Área de Trabalho/git

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   05teste.txt

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

3 – No terminal do Git, execute o comando ‘git add .’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git add .
```

4 – No terminal do Git, execute comando ‘git commit -a -m “nome do commit”’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git commit -a -m "testando o git reset"
[main c35c51e] testando o git reset
1 file changed, 1 insertion(+), 1 deletion(-)
```

5 – No terminal do Git, execute o comando ‘git reset --hard HEAD~1’ para desfazer todas as alterações, inclusive o commit.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git reset --hard HEAD~1
HEAD is now at f58d08b adicionando gitignore
```

6 – Se executarmos o comando ‘git log’ podemos ver que não existe o último commit realizado.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git log
commit f58d08bbb087d0cf1813db63e4946ca470283a27 (HEAD -> main, origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Fri Mar 17 17:35:04 2023 -0300

    adicionando gitignore
```

7 – E se verificarmos o arquivo “05teste.txt” podemos ver que a alteração foi desfeita.



### 2.11.2. Utilizando o comando ‘git reset --soft HEAD~1’

1 – Primeiro iremos realizar o passo a passo anterior do tópico 1 ao 4. Logo em seguida, no terminal do git, executar o comando ‘git reset --soft HEAD~1’.

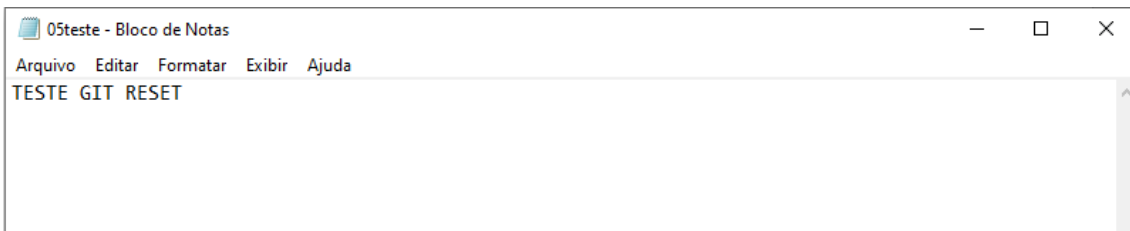
```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git reset --soft HEAD~1
```

2 – Se executarmos o comando ‘git log’ podemos ver que não existe o último commit realizado.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git log
commit f58d08bbb087d0cf1813db63e4946ca470283a27 (HEAD -> main, origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date: Fri Mar 17 17:35:04 2023 -0300

    adicionando gitignore
```

3 – E se verificarmos o arquivo “05teste.txt” podemos ver que as alterações foram mantida.



## 2.12. Desfazendo alterações com git revert

Assim como o ‘git reset’, o ‘git revert’ é utilizado para desfazer alterações, a diferença entre eles é que no ‘git revert’ faz um novo commit do que foi revertido. Por exemplo, se quisermos reverter o commit "abc123", podemos executar o comando ‘git revert abc123’ no terminal do Git. O Git irá criar um novo commit que desfaz as alterações introduzidas pelo commit "abc123". Isso é muito útil quando queremos desfazer alterações em um commit específico, sem afetar os commits posteriores.

Exemplo:

1 – Primeiro iremos alterar o arquivo “05teste.txt” inserindo um texto dentro.



2 – No terminal do Git, execute o comando 'git status'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   05teste.txt

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

3 – No terminal do Git, execute o comando 'git add .'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git add .
```

4 – No terminal do Git, execute comando 'git commit -a -m "nome do commit"'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit -a -m "testando git revert"
[main e8f7c9f] testando git revert
1 file changed, 1 insertion(+), 1 deletion(-)
```

5 – No terminal do Git, execute comando 'git push'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 281 bytes | 281.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
   8d1ad93..e8f7c9f  main -> main
```

6 – Suponhamos que essa alteração enviada tenha dado algum erro para o cliente e precise ser desfeita.

7– No terminal do Git, execute o comando 'git log' para copiar o nome do commit que vai ser desfeito.

```
$ git log
commit e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8 (HEAD -> main, origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Wed Apr 5 14:09:04 2023 -0300

    testando git revert
```

8 – No terminal do Git, execute o comando 'git revert nome do commit'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git revert e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8
```

9 – No terminal do Git, execute o comando 'git log' novamente, logo em seguida é possível ver que foi criado um novo commit para reverter as alterações do commit.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git log
commit c7b5ed92b73a1fac9b3830c9ce9ee71d8e9922ff (HEAD -> main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Wed Apr 5 14:14:29 2023 -0300

    Revert "testando git revert"

    This reverts commit e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8.

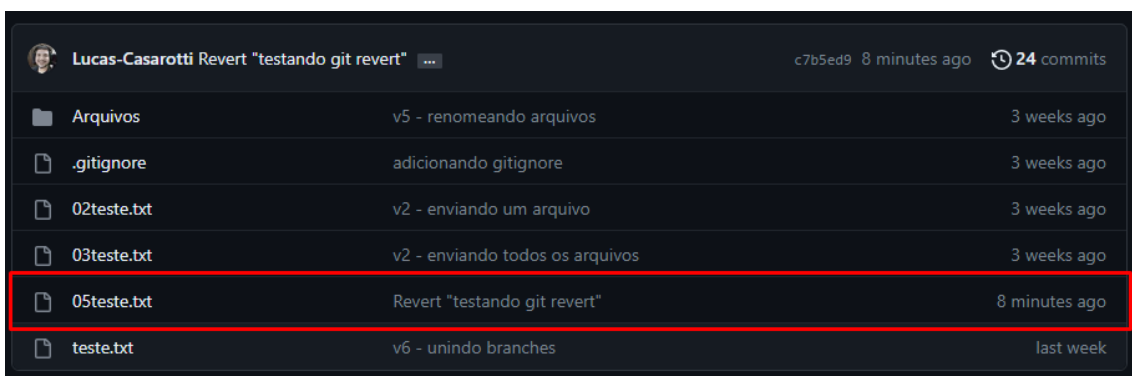
commit e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8 (origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Wed Apr 5 14:09:04 2023 -0300


    testando git revert
```

10 – No terminal do Git, execute o comando 'git push' para enviar ao repositório remoto o novo commit gerado pelo revert.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 289 bytes | 289.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
   e8f7c9f..c7b5ed9  main -> main
```

11 – Logo em seguida no repositório remoto, podemos ver que foi enviado um revert do commit anterior.



	Lucas-Casarotti Revert "testando git revert" ...	c7b5ed9 8 minutes ago	🕒 24 commits
📁	Arquivos	v5 - renomeando arquivos	3 weeks ago
📄	.gitignore	adicionando gitignore	3 weeks ago
📄	02teste.txt	v2 - enviando um arquivo	3 weeks ago
📄	03teste.txt	v2 - enviando todos os arquivos	3 weeks ago
📄	05teste.txt	Revert "testando git revert"	8 minutes ago
📄	teste.txt	v6 - unindo branches	last week

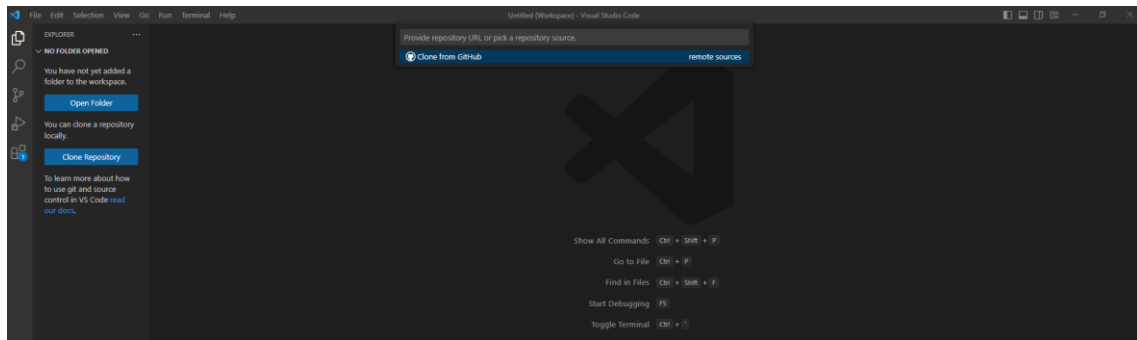
## 2.13. Ignorando arquivos e diretórios em um projeto

Uma técnica muito utilizada é ignorar alguns arquivos no projeto, para isso, devemos inserir um arquivo chamado ".gitignore" na raiz do projeto. Nele

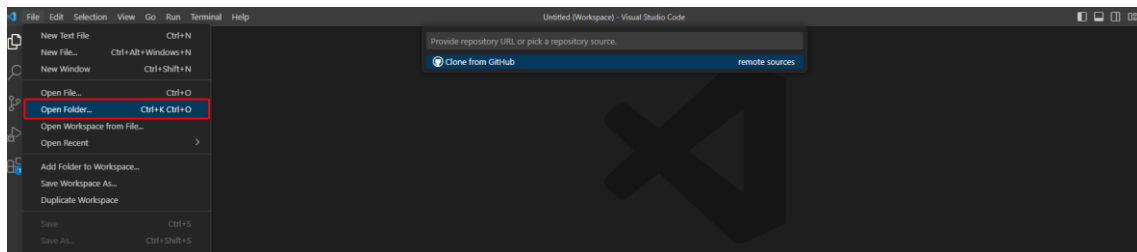
podemos inserir todos os arquivos ou pastas com arquivos que não devem ser enviados ao repositório remoto. Isso é muito útil para arquivos gerados automaticamente ou arquivos que contêm informações sensíveis, como senhas e chaves de acesso.

Exemplo:

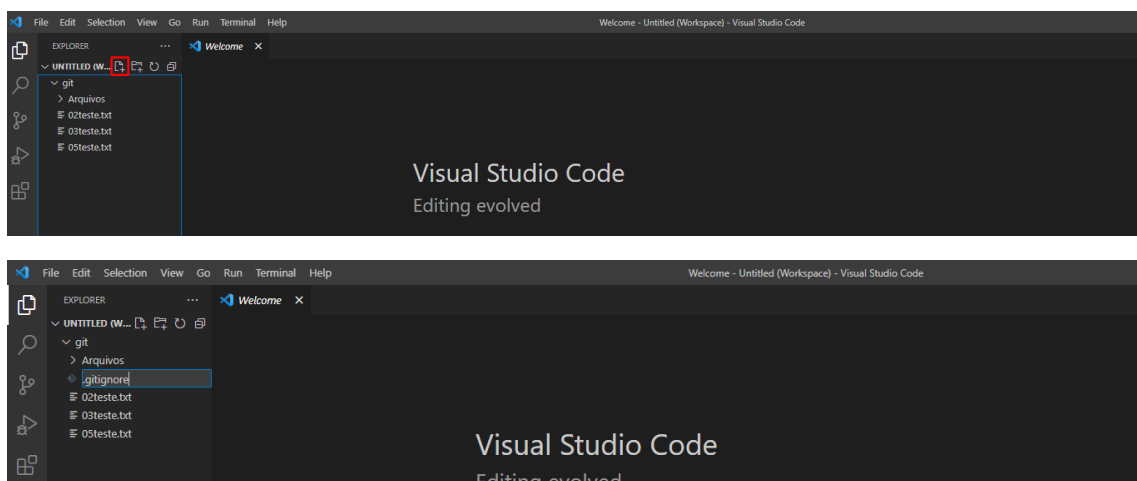
1 – Para criarmos o arquivo “.gitignore” iremos precisar do editor de código Visual Studio Code.



2 – Abrir a pasta onde está localizado o repositório local.



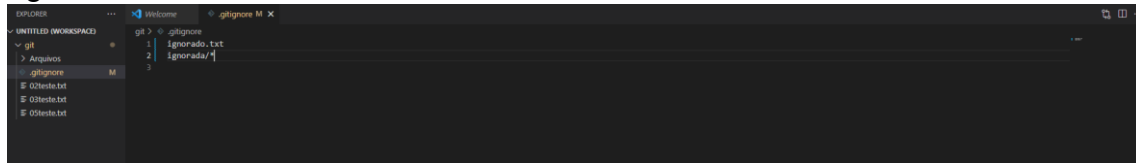
3 – Criar um novo arquivo chamado “.gitignore”.



4 – No arquivo “.gitignore”, é necessário adicionar o nome do(s) arquivo(s) e/ou pasta(s) que não devem ser enviados ao repositório remoto. Neste exemplo, iremos adicionar um arquivo chamado “ignorado” e uma pasta chamada



“ignorada”.



5 – No terminal do Git ou do Visual Studio Code, execute o comando ‘git status’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

6 – No terminal do Git ou do Visual Studio Code, execute o comando ‘git add .’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git add .
```

7 – No terminal do Git ou do Visual Studio Code, execute o comando ‘git commit -a -m “nome do commit”’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit -a -m "adicionando gitignore"
[main 0bf826a] adicionando gitignore
1 file changed, 1 insertion(+)
create mode 100644 .gitignore
```

8 – No terminal do Git ou do Visual Studio Code, execute o comando ‘git push’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 251 bytes | 251.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
   0f6ca5f..0bf826a  main -> main
```

9 – Logo em seguida, se adicionarmos o arquivo chamado "ignorado" e a pasta chamada "ignorada" no repositório local e executarmos o comando ‘git status’, podemos ver que o arquivo e a pasta não são listados.

Nome	Data de modificação	Tipo	Tamanho
git	17/03/2023 17:36	Pasta de arquivos	
Arquivos	15/03/2023 15:23	Pasta de arquivos	
ignorada	17/03/2023 17:36	Pasta de arquivos	
gitignore	15/03/2023 17:33	Documento de Te...	1 KB
0teste	15/03/2023 16:23	Documento de Te...	0 KB
0teste	15/03/2023 10:33	Documento de Te...	0 KB
0teste	15/03/2023 12:02	Documento de Te...	1 KB
ignorado	17/03/2023 17:36	Documento de Te...	0 KB

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

## 2.14. Limpando arquivos untracked

O comando 'git clean' é utilizado para remover arquivos que não estão sendo rastreados, ou seja, aqueles que você ainda não adicionou com o comando 'git add'. Ele é geralmente usado para limpar arquivos gerados automaticamente que não devem ser versionados. Além disso, podemos usar esse comando para desfazer todas as alterações desnecessárias que não serão enviadas ao repositório remoto.

Exemplo:

1 – Adicione um arquivo chamado '06teste.txt' à pasta do repositório local.

.git	19/04/2023 14:30	Pasta de arquivos	
Arquivos	15/03/2023 15:23	Pasta de arquivos	
ignorada	17/03/2023 17:36	Pasta de arquivos	
.gitignore	05/04/2023 10:48	Documento de Te...	1 KB
02teste	30/03/2023 09:28	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	14/04/2023 16:36	Documento de Te...	1 KB
06teste	19/04/2023 14:37	Documento de Te...	0 KB
teste tag	14/04/2023 16:36	Documento de Te...	0 KB
teste	19/04/2023 09:32	Documento de Te...	1 KB

2 – No terminal do Git, execute o comando 'git status', podemos ver que existe um arquivo untracked, ou seja, ele não será enviado no próximo commit pois não está na área de staging.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  06teste.txt

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

3 – Para que essas alterações sejam desfeitas utilize o comando 'git clean -f'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git clean -f
Removing 06teste.txt
```

4 – Na pasta do repositório local, podemos ver que o arquivo '06teste' foi removido.

.git	19/04/2023 14:30	Pasta de arquivos	
Arquivos	15/03/2023 15:23	Pasta de arquivos	
ignorada	17/03/2023 17:36	Pasta de arquivos	
.gitignore	05/04/2023 10:48	Documento de Te...	1 KB
02teste	30/03/2023 09:28	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	14/04/2023 16:36	Documento de Te...	1 KB
teste tag	14/04/2023 16:36	Documento de Te...	0 KB
teste	19/04/2023 09:32	Documento de Te...	1 KB

## 2.15. Limpando arquivos staging

O comando 'git restore' é utilizado para remover um ou mais arquivos da área de staging, ou seja, arquivos que foram adicionados com o comando 'git add .' para serem commitados futuramente.

Exemplo:

1 – Adicione um arquivo chamado '06teste.txt' à pasta do repositório local.

.git	19/04/2023 14:30	Pasta de arquivos	
Arquivos	15/03/2023 15:23	Pasta de arquivos	
ignorada	17/03/2023 17:36	Pasta de arquivos	
.gitignore	05/04/2023 10:48	Documento de Te...	1 KB
02teste	30/03/2023 09:28	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	14/04/2023 16:36	Documento de Te...	1 KB
06teste	19/04/2023 14:37	Documento de Te...	0 KB
teste tag	14/04/2023 16:36	Documento de Te...	0 KB
teste	19/04/2023 09:32	Documento de Te...	1 KB

2 – No terminal do Git, execute o comando 'git status'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    06teste.txt

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

3 – No terminal do Git, execute o comando 'git add .'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git add .
```

4 – No terminal do Git, execute o comando 'git status' novamente, podemos ver que agora o arquivo foi adicionado a stage, ou seja, está pronto para ser commitado.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   06teste.txt
```

5 – No terminal do Git, execute o comando 'git restore - -staged 06teste.txt' para remover o arquivo da área de stage, para remover todos os arquivos execute o comando 'git restore - -staged .'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git restore --staged 06teste.txt
```

6 – No terminal do Git, execute o comando 'git status .', logo em seguida podemos ver que o arquivo está untracked, ou seja, foi removido da área de staging.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    06teste.txt

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

### 3. Trabalhando com branches

#### 3.1. O que são branches?

Traduzido do inglês, branches são ramificações, ou seja, são criados branches para separar as versões do projeto, quando um projeto é criado por padrão ele inicia na branch main. Geralmente quando vamos criar uma nova feature (funcionalidade) do projeto, utilizamos uma branch separada, após a finalização das alterações as branches são unidas para ter o código-fonte final em uma única branch, geralmente na main.

#### 3.2. Criando e visualizando uma branch

É muito comum um desenvolvedor ter que criar e visualizar varias branches, para isso podemos utilizar os seguintes comandos, para criarmos uma branch utilizamos o comando 'git branch nome da branch' e para visualizarmos 'git branch'.

Exemplo:

1 – No terminal do Git, execute o comando 'git branch nome da branch' para criar uma nova branch.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git branch primeira_branch
```

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git branch segunda_branch
```

2 – No terminal do Git, execute o comando 'git branch' para verificar as branches existentes no repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git branch
* main
  primeira_branch
  segunda_branch
```

3 – Podemos ver que existe duas branches, uma delas é a main, que por padrão já é criada no momento em que criamos um repositório remoto e a outra que acabamos de criar. Lembrando que essas branches só existem no repositório local, as mesmas só passam a existir no repositório remoto depois de ser enviado um push.

### 3.3. Deletando branches

Não é comum deletar uma branch, geralmente se usa o delete quando a branch foi criada errada. Para isso utilizamos o comando 'git branch -d' ou 'git branch --delete'.

Exemplo:

1 – No terminal do Git, execute o comando 'git branch -d nome da branch' para deletar a branch desejada.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git branch -d segunda_branch
Deleted branch segunda_branch (was f58d08b).
```

2 – No terminal do Git, se executarmos o comando 'git branch' para listar todas as branches, podemos ver que a branch chamada de "segunda\_branch" foi deletada.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git branch
* main
  primeira_branch
```

### 3.4. Mudando de branch

Podemos mudar de branch utilizando o comando 'git checkout nome\_da\_branch'. Também é possível criar uma nova branch e mudar para ela ao mesmo tempo utilizando o comando 'git checkout -b nome\_da\_branch'. É importante lembrar que ao utilizar esse comando, podemos levar alterações desnecessárias para outras branches. Para evitar esse problema, devemos commitar nossas alterações ou desfazê-las caso não sejam necessárias. Portanto, é necessário ter cuidado ao utilizar esse comando.

Exemplo:

#### 3.4.1. Trocando de branch

1 – No terminal do Git, execute o comando 'git checkout nome da branch' para trocar de branch.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git checkout primeira_branch
Switched to branch 'primeira_branch'
```

#### 3.4.2. Trocando e criando branch ao mesmo tempo

1 – No terminal do Git, executar o comando 'git checkout -b nome da branch nova' para criarmos uma nova branch e logo em seguida entrar na mesma.

Obs: se utilizarmos o comando 'git checkout -b nome da branch', a branch vai ser criada a partir da branch que estamos. Neste caso a branch segunda\_branch foi criada a partir da primeira\_branch então consequentemente vai herdar tudo que tem nela, mesma coisa aconteceu quando criamos a primeira\_branch a partir da master.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (primeira_b
ranch)
$ git checkout -b segunda_branch
Switched to a new branch 'segunda_branch'
```

### 3.5. Unindo branches

É possível juntar o código de duas branches distintas utilizando o comando 'git merge nome da branch'. Essa é uma operação muito comum no dia a dia de um desenvolvedor, geralmente feita na finalização de uma tarefa. Resumidamente, o desenvolvedor cria uma nova funcionalidade no sistema e, após a aprovação e testes, envia a mesma para a branch final do cliente.

Exemplo:

1 – No terminal do Git, execute o comando 'git checkout nome da branch' para entrar na nossa branch de teste.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git checkout primeira_branch
Switched to branch 'primeira_branch'
```

2 – Logo em seguida, vamos criar um arquivo chamado "teste.txt" na branch.

Nome	Data de modificação	Tipo	Tamanho
.git	30/03/2023 17:17	Pasta de arquivos	
Arquivos	15/03/2023 15:23	Pasta de arquivos	
ignorada	17/03/2023 17:36	Pasta de arquivos	
.gitignore	17/03/2023 17:33	Documento de Te...	1 KB
02teste	30/03/2023 09:28	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	30/03/2023 11:00	Documento de Te...	1 KB
ignorado	17/03/2023 17:36	Documento de Te...	0 KB
teste	30/03/2023 17:17	Documento de Te...	0 KB

3 – Vamos imaginar que esse arquivo foi validado e testado pela equipe de teste da empresa. Logo em seguida, precisamos enviar essa alteração para branch final do cliente, por exemplo a branch master.

Para isso:

1 – No terminal do Git, execute o comando 'git add .'.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (primeira_b
ranch)
$ git add .
```

2 – No terminal do Git, execute o comando 'git commit -a -m "v6 – unindo branches"'.  
 3 – Logo em seguida, vamos enviar a alteração para a branch final do cliente.

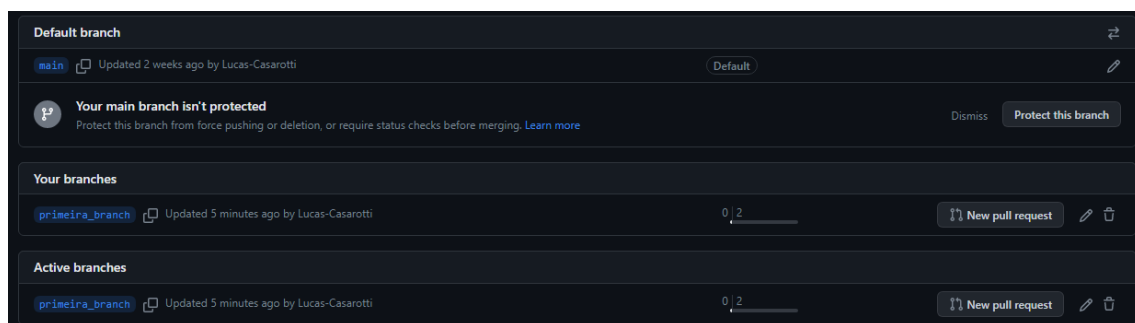
```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (primeira_b
ranch)
$ git commit -a -m "v6 - unidando branches"
[primeira_branch 4223e58] v6 - unidando branches
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 teste.txt
```

3 – No terminal do Git, execute o comando 'git push origin nome da branch' para que a branch passe a existir no repositório remoto.

Obs: Para as demais vezes podemos somente utilizar o comando 'git push', pois a branch já está disponível no repositório remoto.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (primeira_b
ranch)
$ git push origin primeira_branch
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 245 bytes | 245.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'primeira_branch' on GitHub by visiting:
remote:   https://github.com/Lucas-Casarotti/git/pull/new/primeira_branch
remote:
To https://github.com/Lucas-Casarotti/git.git
 * [new branch]      primeira_branch -> primeira_branch
```

4 – No repositório remoto, é possível ver que foi criada uma nova branch.



5 – No terminal do Git, execute o comando 'git checkout main' para voltarmos para a branch final.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (primeira_b
ranch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

6 – No terminal do Git, execute o comando 'git merge primeira\_branch' para juntar as novas alterações da primeira\_branch com a main.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git merge primeira_branch
Updating f58d08b..8d1ad93
Fast-forward
 teste.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 teste.txt
```

7 – No terminal do Git, execute o comando 'git push' para enviar as alterações a branch main remota.



```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Lucas-Casarotti/git.git
f58d08b..8d1ad93 main -> main

```

8 – Logo em seguida, podemos ver que na branch main existem novas alterações. Também é possível verificar na pasta do repositório local.

Lucas-Casarotti v6 - unindo branches		8d1ad93 18 minutes ago	🕒 22 commits
📁 Arquivos	v5 - renomeando arquivos	2 weeks ago	
📄 .gitignore	adicionando gitignore	2 weeks ago	
📄 02teste.txt	v2 - enviando um arquivo	2 weeks ago	
📄 03teste.txt	v2 - enviando todos os arquivos	2 weeks ago	
📄 05teste.txt	v3 - recebendo alterações	2 weeks ago	
📄 teste.txt	v6 - unindo branches	18 minutes ago	

Nome	Data de modificação	Tipo	Tamanho
📁 .git	30/03/2023 17:48	Pasta de arquivos	
📁 Arquivos	15/03/2023 15:23	Pasta de arquivos	
📁 ignorada	17/03/2023 17:36	Pasta de arquivos	
📄 .gitignore	17/03/2023 17:33	Documento de Te...	1 KB
📄 02teste	30/03/2023 09:28	Documento de Te...	0 KB
📄 03teste	15/03/2023 10:33	Documento de Te...	0 KB
📄 05teste	30/03/2023 11:00	Documento de Te...	1 KB
📄 ignorado	17/03/2023 17:36	Documento de Te...	0 KB
📄 teste	30/03/2023 17:48	Documento de Te...	1 KB

### 3.6. Utilizando stash

É utilizado para salvar as modificações atuais para prosseguir com outra abordagem. Por exemplo, estamos desenvolvendo uma tarefa, porém a maneira que estamos desenvolvendo não é totalmente correta, mas queremos guardar as alterações realizadas no código sem a necessidade de commitar. Para isso, utilizamos o comando 'git stash'. Após o comando ser executado, a branch será resetada para a sua versão de acordo com o repositório remoto, ou seja, as alterações são desfeitas e 'guardadas'.

Exemplo:

1 – No terminal do Git, execute o comando 'git checkout nome da branch'.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git checkout segunda_branch
Switched to branch 'segunda_branch'

```

2 – Logo em seguida, iremos alterar o arquivo "05teste.txt".

3 – No terminal do Git, execute o comando ‘git stash’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash
Saved working directory and index state WIP on segunda_branch: 0bf826a adicionan
do gitignore
```

4 – Após executar o comando “git stash”, é possível verificar que as alterações realizadas foram “desfeitas”, ou seja, foram guardadas.

### 3.7. Recuperando stash

É possível verificar as stashes criadas utilizando o comando ‘git stash list’, também podemos recuperar a stash com o comando ‘git stash apply número da stash’. Desta maneira podemos continuar de onde paramos com os arquivos adicionados na stash.

Exemplo:

1 – No terminal do Git, execute o comando ‘git stash list’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash list
stash@{0}: WIP on segunda_branch: 0bf826a adicionando gitignore
```

2 – No terminal do Git, execute o comando ‘git stash apply número da stash’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash apply 0
On branch segunda_branch
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:   05teste.txt

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

3 – Desta forma, as alterações guardadas na stash foram recuperadas.

4 – Também é possível verificar o que foi alterado em cada stash utilizando o comando ‘git stash show -p número da stash’.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash show -p 0
diff --git a/05teste.txt b/05teste.txt
index 8b13789..05bda68 100644
--- a/05teste.txt
+++ b/05teste.txt
@@ -1,1 @@
-
+TESTE STASH
\ No newline at end of file

```

### 3.8. Removendo a stash

Para deletar uma stash especifica utilizamos o comando 'git stash drop número da stash', caso seja necessário também podemos limpar totalmente a stash de uma branch utilizando o comando 'git stash clear' para deletar todas as stachs.

Exemplo:

1 – No terminal do Git, execute o comando 'git stash list' para listar as stachs criadas nos tópicos anteriores.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash list
stash@{0}: WIP on segunda_branch: 0bf826a adicionando gitignore
stash@{1}: WIP on segunda_branch: 0bf826a adicionando gitignore
stash@{2}: WIP on segunda_branch: 0bf826a adicionando gitignore
stash@{3}: WIP on segunda_branch: 0bf826a adicionando gitignore

```

2 – No terminal do Git, execute o comando 'git stash drop número da stash'.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash drop 3
Dropped refs/stash@{3} (3fe4ef14b5613ca6643639a8d3d20845ce17d6ff)

```

3 – No terminal do Git, execute o comando 'git stash list' para verificarmos se a stash foi deletada.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash list
stash@{0}: WIP on segunda_branch: 0bf826a adicionando gitignore
stash@{1}: WIP on segunda_branch: 0bf826a adicionando gitignore
stash@{2}: WIP on segunda_branch: 0bf826a adicionando gitignore

```

4 – Para excluirmos todas as stash, no terminal do Git, execute o comando 'git stash clear'.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_br
anch)
$ git stash clear

```

5 – No terminal do Git, execute o comando ‘git stash list’ para verificarmos se as stashes foram deletadas.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (segunda_branch)
$ git stash list
```

### 3.9. Criando tags

Podemos criar tags nos branches por meio do comando ‘git tag -a versão -m “nome”’. A tag é diferente do stash, sendo utilizada como um checkpoint de uma branch. É utilizada para marcar estágios do desenvolvimento de algum recurso do código fonte. Por exemplo, podemos criar uma tag para “marcar” commits, permitindo que tenhamos acesso a tudo o que foi criado ou alterado até aquele commit específico.

Exemplo:

1 – Adicione um arquivo chamado “teste tag” na pasta do repositório local.

.git	06/04/2023 16:15	Pasta de arquivos	
Arquivos	15/03/2023 15:23	Pasta de arquivos	
ignorada	17/03/2023 17:36	Pasta de arquivos	
.gitignore	05/04/2023 10:48	Documento de Te...	1 KB
02teste	30/03/2023 09:28	Documento de Te...	0 KB
03teste	15/03/2023 10:33	Documento de Te...	0 KB
05teste	05/04/2023 14:14	Documento de Te...	1 KB
teste tag	06/04/2023 16:11	Documento de Te...	0 KB
teste	05/04/2023 10:48	Documento de Te...	1 KB

2 – No terminal do Git, execute o comando ‘git add .’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git add .
```

3 – No terminal do Git, execute o comando ‘git commit -a -m “nome do commit”’.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git commit -a -m "v7 - testando tag"
[main 11d8c56] v7 - testando tag
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 teste tag.txt
```

4 – No terminal do Git, execute o comando ‘git push’.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 239 bytes | 239.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lucas-Casarotti/git.git
    b2f7145..11d8c56  main -> main

```

5 – Agora que enviamos um novo commit ao repositório remoto, podemos criar um tag para o mesmo, desta maneira criamos um checkpoint para esse commit. Para isso, no terminal do Git, execute o comando 'git tag -a versão -m "nome"'.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git tag -a v1 -m "v7 - testando tag"

```

6 – No terminal do Git, execute o comando 'git tag' para listar todas as tags criadas.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git tag
v1

```

7 – No terminal do Git, execute o comando 'git push origin v1' para enviar a tag criada ou 'git push origin - -tags' para enviar todas as tags.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git push origin v1
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 165 bytes | 165.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Lucas-Casarotti/git.git
 * [new tag]          v1 -> v1

```

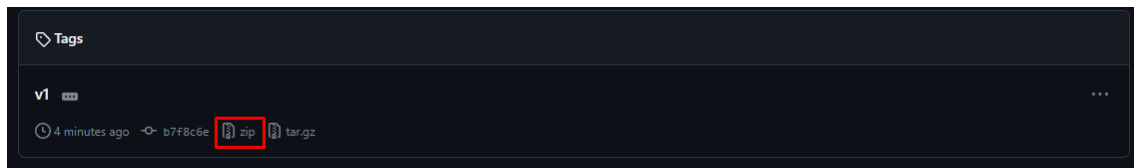
8 – No repositório remoto, é possível verificar que foi criada uma nova tag.



The screenshot shows the GitHub repository page for 'Lucas-Casarotti v7 - testando tag'. The 'tags' tab is selected and highlighted with a red box, showing '1 tag'. Below the header, a table lists the repository's files and their commit history:

Arquivos	Commit	Tempo
Arquivos	v5 - renomeando arquivos	3 weeks ago
.gitignore	adicionando gitignore	3 weeks ago
02teste.txt	v2 - enviando um arquivo	3 weeks ago
03teste.txt	v2 - enviando todos os arquivos	3 weeks ago
05teste.txt	Revert "testando git revert"	yesterday
teste tag.txt	v7 - testando tag	1 minute ago
teste.txt	v6 - unindo branches	last week

9 – Desta maneira, podemos baixar essa tag e ter acesso a todas as funcionalidades que foram criadas até o commit que foi marcado pela tag.



### 3.10. Exibindo informações

O comando 'git show' nos dá diversas informações úteis, ele nos dá informações sobre o branch atual e também sobre seus commits. As modificações e arquivos entre cada commit também são exibidos. Além disso, podemos exibir as informações de tags com o comando 'git show nome\_tag'.

#### Exemplo:

1 – No terminal do Git, execute o comando 'git log' para listar todos os commits.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Área de Trabalho/git (main)
$ git log
commit b7f8c6ef8b950c77a412f8e0237c534a71efb32b (HEAD -> main, tag: v1, origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date: Thu Apr 6 17:27:05 2023 -0300

    v7 - testando tag

commit c7b5ed92b73a1fac9b3830c9ce9ee71d8e9922ff
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date: Wed Apr 5 14:14:29 2023 -0300

    Revert "testando git revert"

    This reverts commit e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8.

commit e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date: Wed Apr 5 14:09:04 2023 -0300

    testando git revert
```

2 – Logo em seguida, podemos escolher um commit no qual queremos ver o que foi criado/alterado. No terminal do Git, execute o comando 'git show nome gerado do commit'.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git show e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8
commit e8f7c9f46097b251eac0e5c8cd3ee269f5f55aa8
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Wed Apr 5 14:09:04 2023 -0300

    testando git revert

diff --git a/05teste.txt b/05teste.txt
index 8b13789..c698da2 100644
--- a/05teste.txt
+++ b/05teste.txt
@@ -1,1 @@
-
+TESTE GIT REVERT

```

3 – Podemos ver que é exibido o que foi criado/alterado no commit, o mesmo serve para tags. No terminal do Git, execute o comando 'git tag'.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git tag
v1

```

4 – No terminal do Git, execute o comando 'git show nome tag' para exibir o que foi criado/alterado nas tags.

```

Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git show v1
tag v1
Tagger: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Thu Apr 6 17:28:11 2023 -0300

v7 - testando tag

commit b7f8c6ef8b950c77a412f8e0237c534a71efb32b (HEAD -> main, tag: v1, origin/main)
Author: Lucas-Casarotti <casarotti@hotmail.com>
Date:   Thu Apr 6 17:27:05 2023 -0300

    v7 - testando tag

diff --git a/teste tag.txt b/teste tag.txt
new file mode 100644
index 0000000..e69de29

```

### 3.11. Verificando diferenças

O comando 'git diff' serve para exibir as diferenças entre arquivos e/ou commits em um repositório git. Quando utilizado sem nenhum parâmetro, exibe as diferenças entre a cópia de trabalho (working copy) e o último commit. É possível também verificar a diferença entre branches e commits específicos utilizando o nome da branch ou o hash do commit como parâmetro.

Exemplo:

1 – Primeiro, vamos alterar o arquivo "teste.txt", inserindo um novo teste.



2 – No terminal do Git, execute o comando 'git diff' para exibir as diferenças entre a branch local e remota.

```
Lucas@DESKTOP-3KK17LJ MINGW64 /e/Usuários/Lucas/Area de Trabalho/git (main)
$ git diff
diff --git a/teste.txt b/teste.txt
index e80773e..cb5ce98 100644
--- a/teste.txt
+++ b/teste.txt
@@ -1,3 @@
-unidando branches
\ No newline at end of file
+unidando branches
+
+teste diff
\ No newline at end of file
```