

Tutorial de Git e GitHub

O que é Git?

Git é um sistema de código aberto e gratuito para controle de versão, criado em 2005 por [Linus Torvalds](#) enquanto desenvolvia o [Kernel do Linux](#). **Controle de versão** é um sistema que permite gravar alterações em arquivos ao longo do tempo, portanto, podemos visualizar versões específicas desses arquivos posteriormente.

Geralmente, acontece de termos muitos desenvolvedores trabalhando numa mesma base de códigos, então um sistema de controle de versão como o Git é necessário para diminuir conflitos entre o código de cada desenvolvedor.

O que é GitHub?

Se você vai utilizar Git, é necessário armazenar seu repositório em algum lugar. Existem duas formas de se fazer isso, offline, no seu próprio computador, ou online, em alguma plataforma como [GitHub](#), [BitBucket](#) ou [GitLab](#).

É exatamente para isso que o GitHub serve, para armazenar seus repositórios, ele é o “Google Drive dos códigos”.

Configuração do ambiente

Primeiramente, é necessário possuir o Git instalado no seu computador e uma conta no GitHub. Você pode baixar o Git no [site oficial](#), basta selecionar o seu sistema operacional e seguir o guia de instalação.

No Linux, recomenda-se instalar o Git via gerenciador de pacotes, nesse caso, você pode ir até [essa página](#) e executar no seu terminal o comando específico da sua distribuição.

Depois de instalar o Git, é necessário configurar o seu nome e e-mail que serão exibidos em seus *commits*, para isso é só digitar no terminal os comandos abaixo:

```
1 git config --global user.name "Seu Nome"
2 git config --global user.email "seu-email@buteco.tech"
```

*Observação: Caso você esteja no Windows, não se esqueça de utilizar o **git bash**, pois ele simula um ambiente Unix, facilitando a forma de se trabalhar com Git.*

Agora iremos criar uma chave de SSH, com ela provaremos ao GitHub que nós somos os donos da nossa conta. Caso já tenha uma chave SSH, podemos verificar seu diretório **.ssh** listando o conteúdo:

```
1 cd ~/.ssh
2 ls
```

Caso o comando **ls** retorne algo com a extensão **.pub**, você não precisa executar o comando **ssh-keygen**. Caso contrário, digite esse comando no seu terminal:

```
1 ssh-keygen -t rsa -b 4096 -C "seu-email@buteco.tech"
```

O comando acima irá pedir um nome do arquivo para salvar a chave, não escreva nada e aperte enter (será gerada uma com o nome **id_rsa**). Depois, informe uma senha (*passphrase*) e confirme ela. Pronto, você gerou uma chave SSH.

Vamos agora adicionar a chave ao ssh-agent, um programa que fará a autenticação da sua máquina local, com o servidor remoto, que nesse caso seria o GitHub, execute então o comando **ssh-agent**:

```
1 eval $(ssh-agent -s)
```

A mensagem **Agent pid 1234** será exibida.

Executar o comando **ssh-add**, onde o **id_rsa** é o nome da sua chave SSH, informe a *passphrase* utilizada no comando **ssh-keygen**:

```
1 ssh-add ~/.ssh/id_rsa
```

A mensagem **Identity added : id_rsa (seu-email@buteco.tech)** será exibida.

Adicionando a chave SSH à sua conta no GitHub

Primeiramente, você deve criar uma conta no GitHub.

No diretório `~/.ssh` existem dois arquivos, **id_rsa** e **id_rsa.pub**. O com a extensão **.pub** é a chave pública, ou seja, a chave que você irá informar ao GitHub. O arquivo sem o **.pub** é a sua chave privada.

Digite o seguinte comando no terminal para exibir o conteúdo da chave gerada:

```
1 cat ~/.ssh/id_rsa.pub
```

Copie todo o conteúdo para a sua área de transferência (lembre-se de copiar desde “ssh-rsa” até o seu e-mail).

Acesse a página de SSH and GPG keys no GitHub para associar a nova chave SSH a sua conta. Clique então no botão **New SSH Key**.

O título você pode escolher um (ex: chave seu-email@buteco.tech), e em Key, você informará o conteúdo que copiou anteriormente.

Clique em *Add SSH Key* para salvar. Pronto, sua configuração está feita.

```
MINGW64:/c/Users/leonardo.dalcegio/.ssh

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/.ssh
$ cd ~/.ssh

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/.ssh
$ ls
id_rsa id_rsa.pub known_hosts

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/.ssh
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQDDxn07SfGSKn0t4ovnReWpDOV5PIxsqDDsY8Jg50h0
anav8N7bxZwb823o2ZbEMe/H4veP70zHXcaY+018k4+fTjGFPREbVH1076ESg1KYAa1RIOT/Cgpwdq81
PYu3+h2QVgJdfs8I40g9Z4acfQrQ08o0RTbeoMuHxGSVXrX+oGt11GrqX2Ev2JtZ8bzAz2qNb8/C0n7
AgHunhVtbITWjZV+9cXIofRKZZ1e3/yJtmqNEuf66+hQN++RR/pSjyXy8Bwr/f06G7DwrjK2Y11sYrQm
pJDwisF1UF+p1X2CmjVvyikr1wtgwxMHUK1QCuBhgyM433R1ZAniPnpG2pcw138mwI70fX6EMk6PJ+9m
rPG7uCP7U0twrCZT7rksvK/72sPGv4YCT+LfSbyULbHX51QcXLY87+Y3Hrt2FQeCrjzC8y0ArYMcA77w
QgkZxxmmZ38kUhya0Lfn1Gz0MgrrCgiLCLWqKtspk3i1ar3vDkdD419Z6ggWUCcLgSARShu04McH/OA
RMhIM9yoY8Kjq55SwxNnNeBEKCorWDMaqZg0tWuF1J4qxYs+kQyyXzjKJYXb25JvV7VBoV4RTF8uEkpz
M0wMDXqb0iPqvJytmE14mw23Z/Pncfct12fbk27Y+bngPbwFAeYT+orOzsDYdh7sSFuImqIIBond9c+1
ww== leoodalcegio@hotmail.com

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/.ssh
$ eval $(ssh-agent -s)
Agent pid 1616

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/.ssh
$ ssh-add id_rsa
Enter passphrase for id_rsa:
Identity added: id_rsa (leoodalcegio@hotmail.com)

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/.ssh
$ |
```

Fluxo de trabalho 1

Geralmente, o fluxo de trabalho com Git e GitHub é realizando um *fork* de um determinado repositório de outra pessoa. Dessa forma, vamos criar uma cópia deste repositório no nosso perfil. Após fazer o *clone* do *fork* para a nossa máquina local, podemos realizar alterações, e publicar as alterações feitas em seu repositório *fork* no GitHub. Por fim, poremos criar um *pull request* do seu *fork* para o repositório principal (*upstream*).

Fork

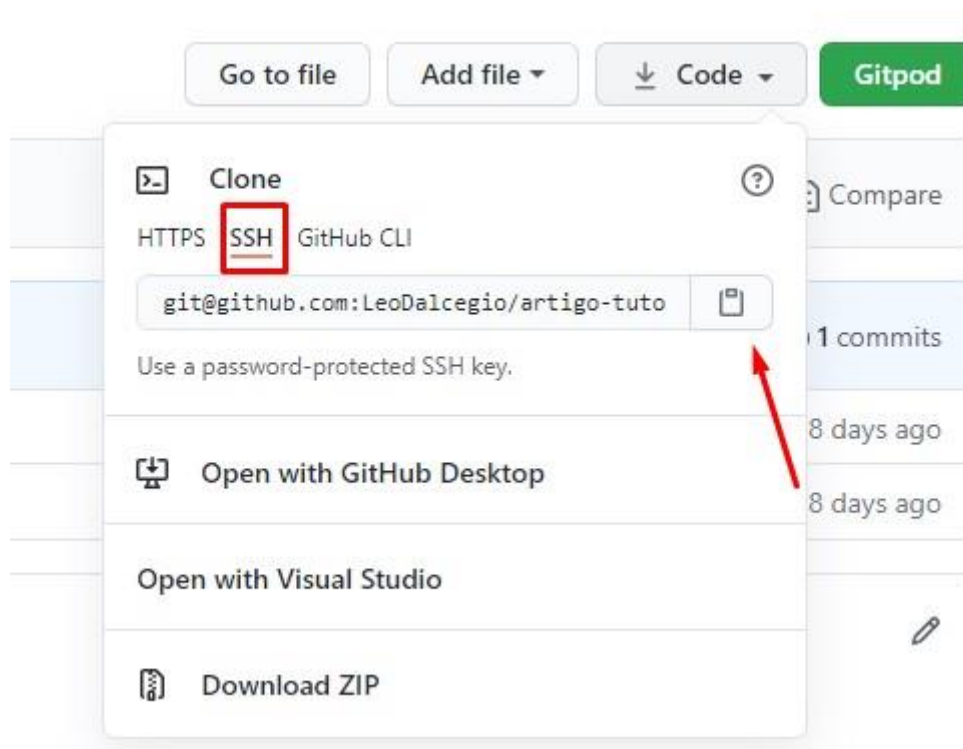
Vamos começar realizando o *fork* de um repositório base no GitHub. Veja nosso repositório [artigo-tutorial-git](#) e clique no botão **Fork** no canto superior direito. Ao clicar, será feita uma cópia do repositório do Buteco em seu perfil no GitHub.

Agora em seus repositórios você terá uma cópia de **artigo-tutorial-git**. Lembre-se, você não está no repositório principal, você está no seu, as alterações feitas ali, não terão impacto no principal.

Você deve criar um *Pull Request* para enviar suas alterações, que é o que iremos ver daqui a pouco.

Git Clone

Vamos clonar o repositório para a sua máquina local para que você possa realizar alterações. Na página do seu repositório no GitHub, existe um botão chamado **Code**, clique nele, selecione a opção **SSH**, e depois clique no botão de copiar.



Agora abra o seu terminal no local que deseja baixar o repositório, e execute (use o endereço que você copiou):

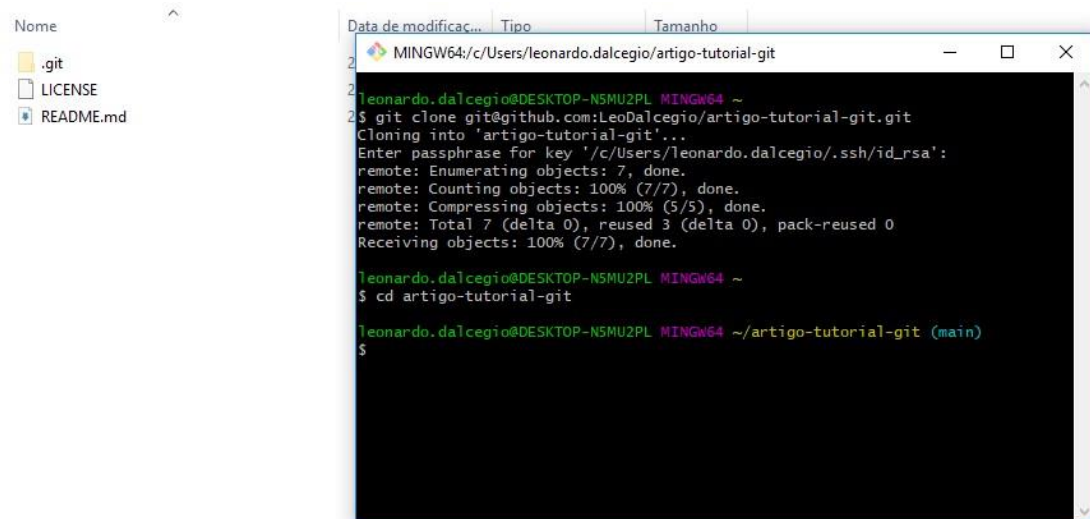
```
1 git clone git@github.com:buteco-tech/artigo-tutorial-git.git
```

Será pedido para que você informe a sua *passphrase* de quando criou a sua chave SSH. Após, o clone do repositório você terá uma nova pasta chamada **artigo-tutorial-git**.

Entre na nova pasta com o comando:

```
1 cd artigo-tutorial-git
```

Você deverá ter algo similar a imagem abaixo:



Perceba que existe ali, uma pasta chamada **.git**, é nela que o Git guarda todas as informações do seu repositório, todas as alterações, o endereço do repositório remoto e tudo o que é relacionado ao controle de versão.

No Windows talvez você não consiga visualizar esta pasta pois ela é um arquivo oculto, então é só seguir esse tutorial para alterar as configurações.

Git Status e Git Add

Vamos criar um novo arquivo para que possamos adicionar ele depois, então, crie um arquivo de texto qualquer dentro do diretório do repositório.

```
1 echo "Tutorial de Git e GitHub" > arquivo.txt
```

Voltando para o terminal, digite **git status**. Você verá essa mensagem:



```
MINGW64:/f/artigo-tutorial-git
leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 /f
$ git clone git@github.com:buteco-tech/artigo-tutorial-git
Cloning into 'artigo-tutorial-git'...
Enter passphrase for key '/c/Users/leonardo.dalcegio/.ssh/id_rsa':
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
Receiving objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 /f
$ cd artigo-tutorial-git

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 /f/artigo-tutorial-git (main)
$ echo "Tutorial de Git e GitHub" > "Leonardo Luis Dalcegio.txt"

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 /f/artigo-tutorial-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)
    "Leonardo Lu\303\255s Dalcegio.txt"

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

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 /f/artigo-tutorial-git (main)
$
```

Untracked files são as alterações pendentes, nós precisamos adicionar elas à área de *staging*. As alterações que estão no *staging* são as que serão *commitadas* futuramente.

Para adicionar apenas um arquivo execute o comando:

```
1 git add nome-do-arquivo
```

Ou para adicionar todos os arquivos execute o comando:

```
1 git add .
```

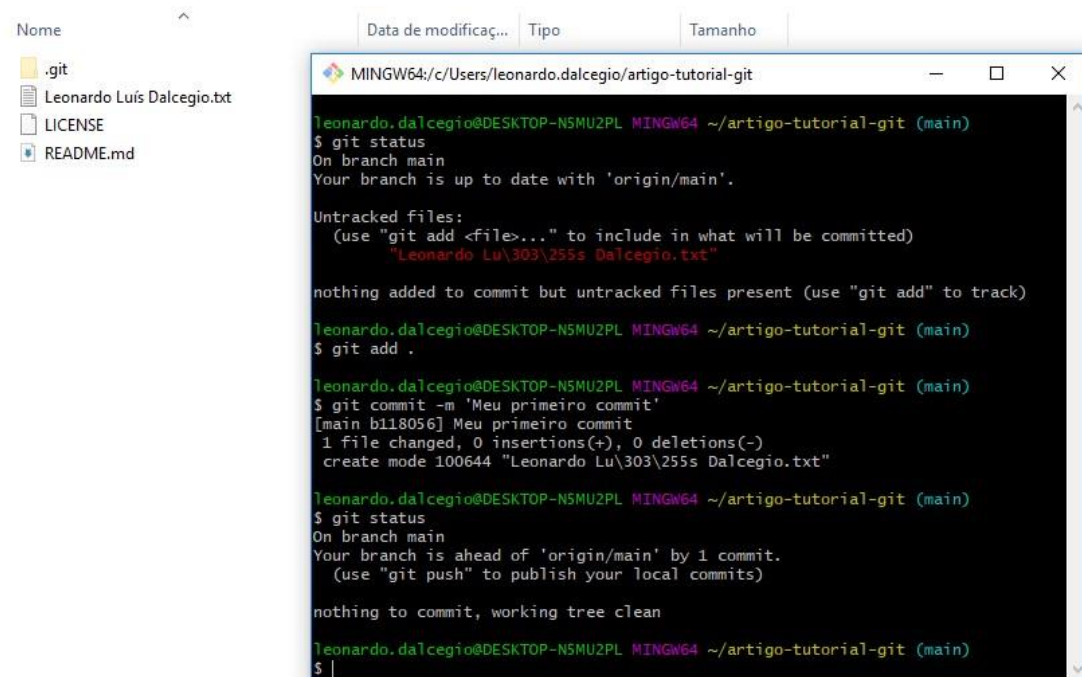
O `git add` não afeta diretamente o repositório, pois ele ainda não “salvou” as alterações, apenas as moveu para a área de *staging*, quando executamos `git commit` essas as alterações serão salvas.

Agora sim, podemos confirmar as alterações, para isso execute o comando:

```
1 git commit -m "Meu primeiro commit"
```

O `-m` é para informar uma mensagem que será gravada junto ao *commit*.

Se você executar o comando `git status` novamente, o seu terminal deverá estar parecido com o abaixo.



The image shows a file explorer window on the left and a terminal window on the right. The file explorer shows a directory with files: .git, Leonardo Luis Dalcegio.txt, LICENSE, and README.md. The terminal window shows the output of the following commands:

```
leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/artigo-tutorial-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)
  "Leonardo Lu\303\255s Dalcegio.txt"

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

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/artigo-tutorial-git (main)
$ git add .

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/artigo-tutorial-git (main)
$ git commit -m 'Meu primeiro commit'
[main b118056] Meu primeiro commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "Leonardo Lu\303\255s Dalcegio.txt"

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/artigo-tutorial-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)

nothing to commit, working tree clean

leonardo.dalcegio@DESKTOP-N5MU2PL MINGW64 ~/artigo-tutorial-git (main)
$
```

Agora é hora de subir as alterações feitas no seu repositório local para o seu repositório remoto, nesse caso, o do GitHub.

Git Push

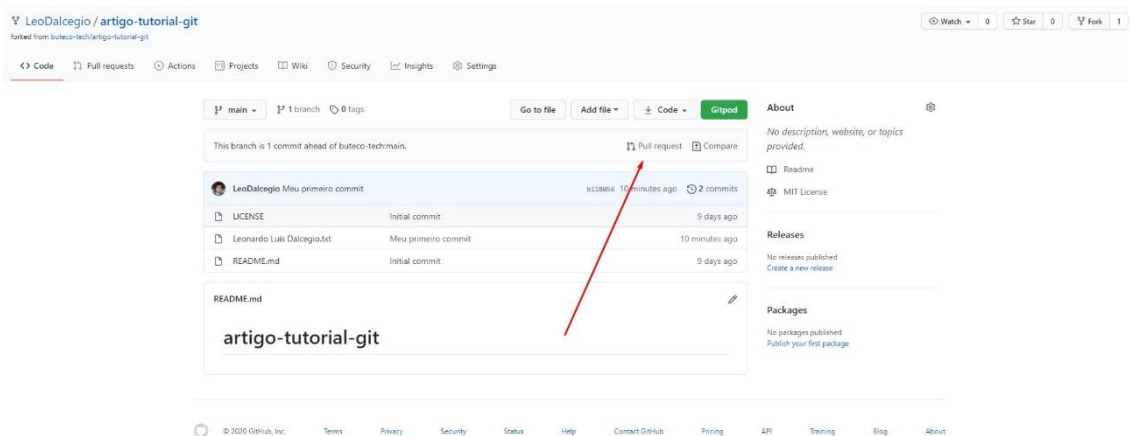
Quando estamos com alguma alteração *commitada* localmente e queremos subir elas para o repositório remoto, precisamos publicá-las, para isso execute o comando:

```
1 git push
```

Após, visite a página do seu repositório no GitHub. Você perceberá que as alterações feitas localmente, já estão lá, mas atenção, quando você fez o *push*, as alterações foram para o seu *fork*, não para o repositório principal.

Criando um Pull request

No seu repositório, após feito um *push*, você verá um botão escrito **Pull request**. Clique nele.



LICENSE

Leonardo Luis Dalcegio.txt

README.md

artigo-tutorial-git

About

No description, website, or topics provided.

Readme

MIT License

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Isso irá abrir uma página mostrando os arquivos alterados e um botão **Create pull request**. Clique nele, informe o título e um comentário para esse *pull request* e clique no botão **Create pull request**.

Pronto, seu PR foi criado com sucesso.

Se você for até a página de [pull requests do repositório principal](#), o seu PR estará lá. Agora, para a sua alteração entrar de fato no repositório principal, o seu *pull request* deve ser aceito por algum mantenedor do repositório. Esse processo de aceitar um *pull request* é chamado de *merge*.

O que é Pull Request?

Desde o começo desse artigo, tudo girou em torno deste momento, em que você cria o seu *pull request*, mas afinal, o que é um *pull request*?

Para entendermos o que é um *pull request* precisamos entender o que é um *pull*. Quando existe alguma alteração no repositório

remoto, e você quer baixar essa alteração para o seu repositório local, ou seja, o seu repositório local está atrasado em relação ao remoto, você digita `git pull` no terminal (olha só, mais um comando novo). Um *pull request* é uma sugestão de uma alteração, ou seja, você está pedindo para que aquele repositório que você enviou o *pull request* faça um *pull* com as suas alterações.

Fluxo de trabalho 2

Como dito anteriormente, vamos ver outra maneira de se utilizar o Git e GitHub.

Desta vez, vamos começar com o comando `git init` para criar o nosso repositório localmente, depois iremos conectar ele ao GitHub e enviar os nossos *commits*.

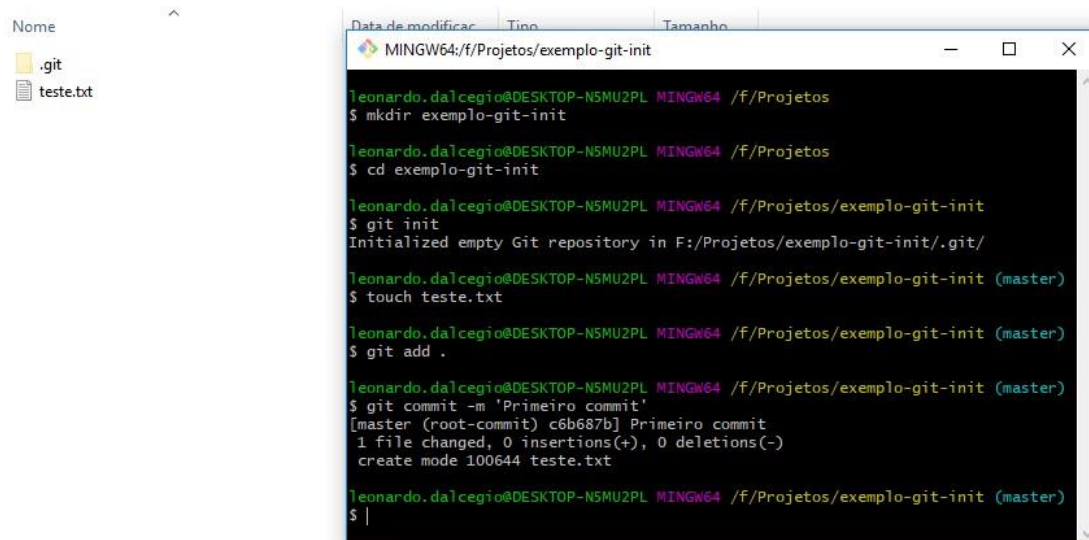
Git Init

Crie uma pasta em qualquer lugar do seu computador e navegue até ela com o seu terminal. Digite então dentro dela, o comando:

```
1 git init
```

Você perceberá que foi criado uma pasta **.git**, que mapeia as alterações e *commits* do repositório como mencionado anteriormente.

Realize agora alguma alteração. Crie um arquivo de texto, por exemplo. Depois, execute o comando `git add` que foi falado anteriormente e `git commit -m`, agora você já está craque nisso.



Feito isso, é hora de subirmos essas alterações para o repositório remoto (que ainda não foi criado).

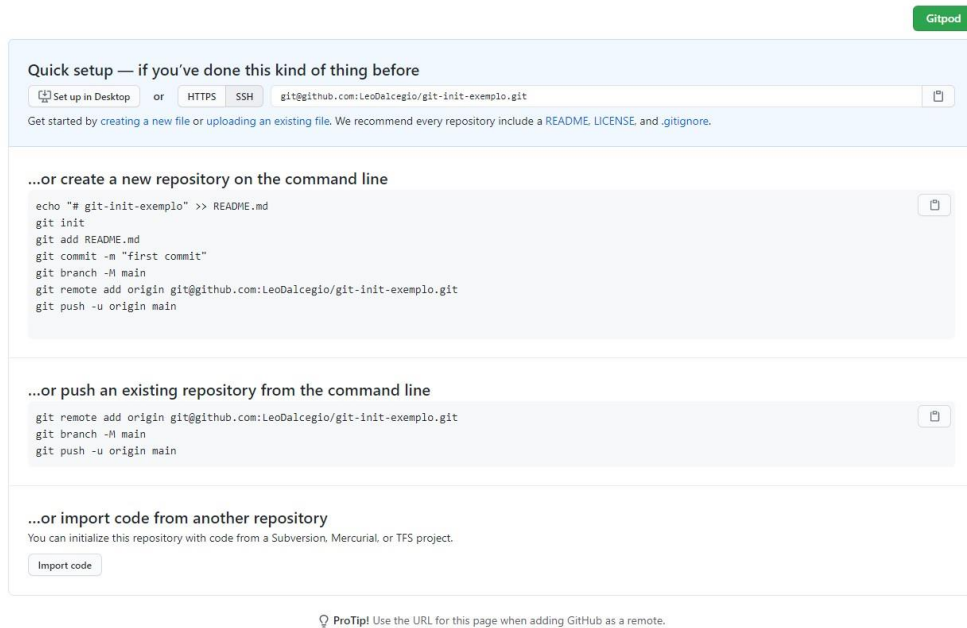
Precisamos criar um repositório no GitHub. Nesta página informe algumas informações sobre o repositório, como o nome, descrição, e clique em **Create repository**.

Pronto, um novo repositório foi criado.

Vinculando o repositório local, ao remoto

Veja que existem ali três opções:

- ***...or create a new repository on the command line***
- ***...or push an existing repository from the command line***
- ***...or import code from another repository***



A primeira opção, é para criar um repositório local, vincular ele ao GitHub, e fazer um *push* das alterações. A segunda é para apenas conectar um repositório local já existente, e fazer um *push* dele. E a terceira, é para importar o código de algum repositório que esteja utilizando um sistema de versionamento de código diferente do Git.

Nós já temos o nosso repositório criado, queremos apenas subir ele ao GitHub, portanto, vamos realizar os passos da segunda opção.

Com o seu terminal aberto na pasta do seu repositório local, execute então os seguintes comandos:

```
1 git remote add origin git@github.com:seu-nome/git-init-exemplo.git
```

Esse comando define para onde enviaremos nossos *commits*, nesse caso o GitHub.

```
1 git branch -M main
```

Esse comando cria uma *branch* com o nome **main**, padrão do GitHub.

```
1 git push -u origin main
```

Esse comando envia as alterações *commitadas* ao seu repositório remoto no GitHub.

Pronto, se você for até a página do seu repositório no GitHub as alterações já estarão lá.

O fim, se existe, está muito distante ainda

Se você chegou até aqui, então aprendeu bastante coisa.

Você aprendeu como fazer um *fork* de um repositório no GitHub, como *clonar* ele para a sua máquina local com `git clone`, adicionar as suas alterações a ele com `git add`, *commitar* elas com `git commit`, subir as suas alterações ao repositório remoto com `git push` e como abrir um `pull request` com as alterações.

Mas não acabou por aí, o próximo passo é estudar sobre *branches*, entendendo como elas funcionam e são usadas. E claro, contribuir com software livre.