

GIT & GITHUB

Do Básico ao Avançado

Domine Git e GitHub com este material completo. Com ele você aprenderá a gerenciar fluxos de trabalho de maneira simples e dinâmica.



Git push!

Edite seu perfil do GitHub antes de começar...

Como você já deve saber, muitas pessoas e empresas dão uma olhada geral no GitHub de nós desenvolvedores, e por conta disso, vale muito a pena preenchermos algumas informações a mais, como:

- Seu Nome
- Sua Bibliografia
- Sua Foto (principalmente)
- A URL do seu site (caso tiver)
- E a sua Localização (que por padrão já vem preenchida)

Para alterar tais dados, entre no <https://github.com>, clique no ícone da sua foto de perfil (no canto superior direito), vá em **Settings** e depois em **Public Profile**.

Public profile

Name

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email

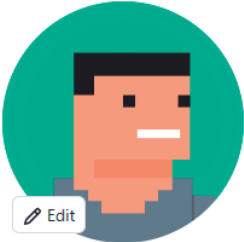
 You can manage verified email addresses in your email settings.

Bio

You can @mention other users and organizations to link to them.

URL

Profile picture



E por fim, não se esqueça de rolar a tela para baixo e clicar na opção **"Update Profile"**, para que suas alterações surtam efeito.

Update profile



O que são Repositórios?

Quando criamos um novo projeto na qual iremos utilizar o git, obviamente nós fazemos isso dentro de uma pasta, que no caso está identificada com o próprio nome do projeto.



Meu Projeto



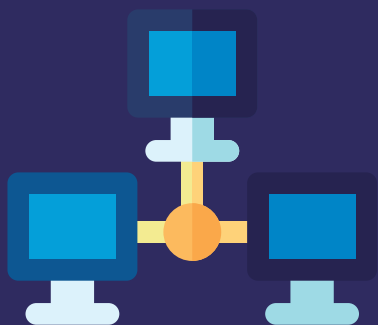
Loja de Roupas

Já no GitHub, o nome dessa pasta também existe, só que lá ela é conhecida como um **Repositório**, que nada mais é do que um local aonde podemos armazenar todo o código-fonte do nosso projeto, e disponibilizá-lo através de uma URL pública:

Repositório = Pasta do Projeto

E nesse caso, você pode ter vários projetos na sua conta do GitHub.

Assim como a pasta `.git` existente dentro da pasta do seu projeto, a mesma acontece no seu repositório do github, logo, o versionamento do seu código também pode ser encontrado dentro desse repositório.

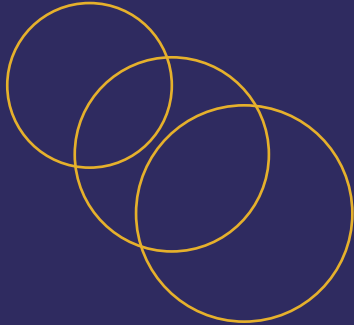


Basicamente existem duas formas de criarmos um repositório no github, a primeira delas é fazendo isso de forma totalmente visual, ou seja, abrimos o site do github e de lá criamos um novo repositório. Ou como também podemos fazer isso de forma manual, usando o próprio terminal do Git.

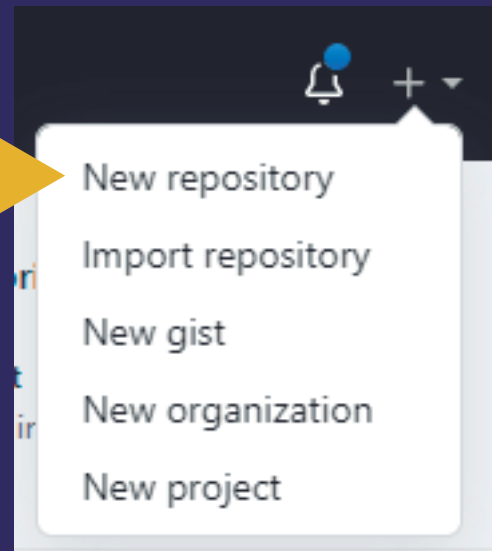


Criando um novo repositório (Visual)

Para criar um novo repositório diretamente pelo github, é bem simples, primeiro abra a página inicial do GitHub, e entre na sua conta.




Clique no botão de +, ao lado do ícone da sua foto, após isso selecione a primeira opção aonde diz: 'New repository'.




Você será levado a uma página aonde poderá preencher algumas informações referentes ao projeto que você quer criar:


Owner * Repository name *

 micilini /

Great repository names are short and memorable. Need inspiration? How about [symmetrical-octo-barnacle?](#)

Description (optional)

 **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.](#)

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

Repository Name

Neste ponto, você coloca o nome do seu repositório que você deseja criar.

Geralmente aqui informamos o nome do nosso projeto que estamos criando. (Ex: *Meu projeto* ou *Syntethical Blunde* ou etc...)

Description: (Optional)

Uma breve descrição do que se trata seu projeto. Lembre-se que outras pessoas podem querer entender sobre o que este projeto faz.

Public / Private

Aqui você pode tornar seu repositório público, aonde qualquer pessoa poderá achar seu projeto e ver seu código-fonte, seja por meio de uma URL ou uma simples busca na plataforma do github. Ou você também pode deixá-lo privado, mas lembre-se que essa opção é voltada a contas específicas, e talvez seja necessário assinar um plano.

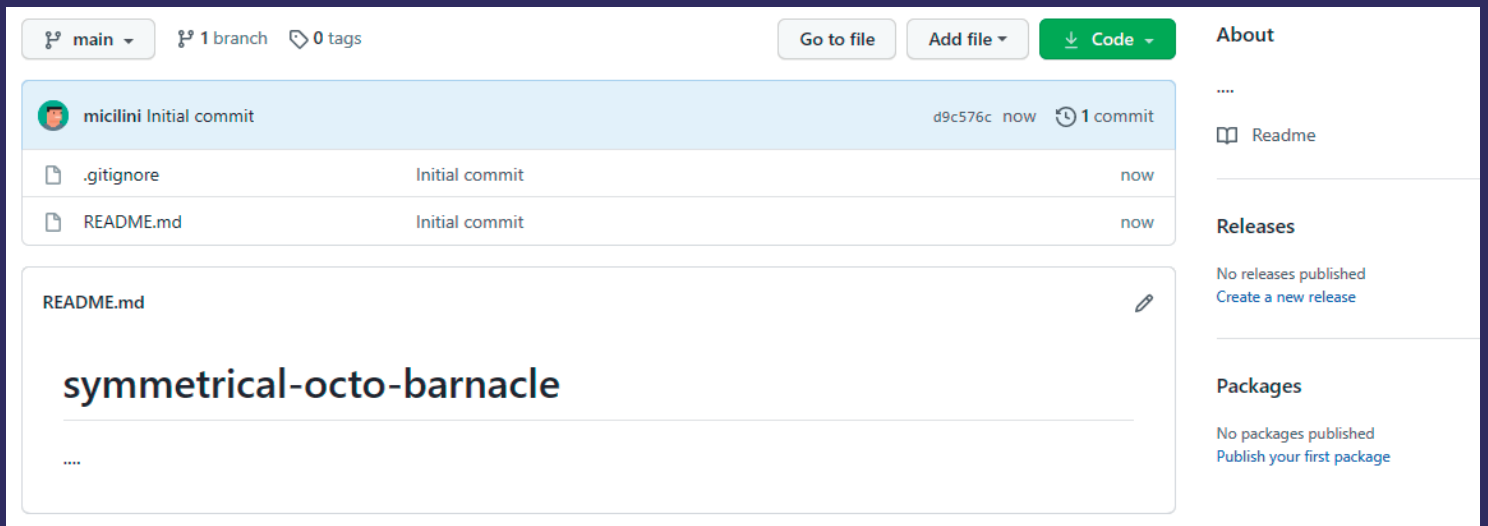
Initialize this Repository With

Permite você inicializar o repositório usando um arquivo readme, .gitignore ou uma licença.

Basicamente é um documento que você insere no seu repositório, e o github o executa como se fosse um documento/testamento/apresentação em formato de texto.

Quando for preencher as configurações iniciais do seu repositório, deixe marcado a visibilidade como **público**, e também deixe marcado a opção **add README file** e **add .gitignore**.

Após clicar em 'Create Repository', você será redirecionado para a tela inicial do seu projeto na plataforma:



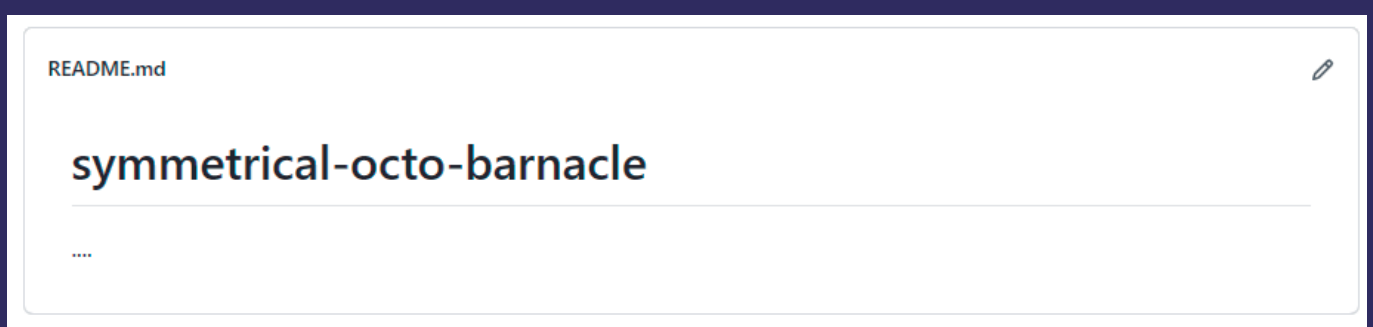
The screenshot shows a GitHub repository interface. At the top, there are navigation elements: 'main' branch, '1 branch', and '0 tags'. Below this, there are buttons for 'Go to file', 'Add file', and 'Code'. The main content area shows a commit by 'micilini' with the message 'Initial commit'. Below the commit, there is a table listing files: '.gitignore' and 'README.md', both from the 'Initial commit'. The 'README.md' file is expanded, showing the text 'symmetrical-octo-barnacle'. On the right side, there are sections for 'About', 'Releases', and 'Packages', each with a 'No releases/packages published' message and a link to create or publish.



Entendendo o Arquivo README.MD

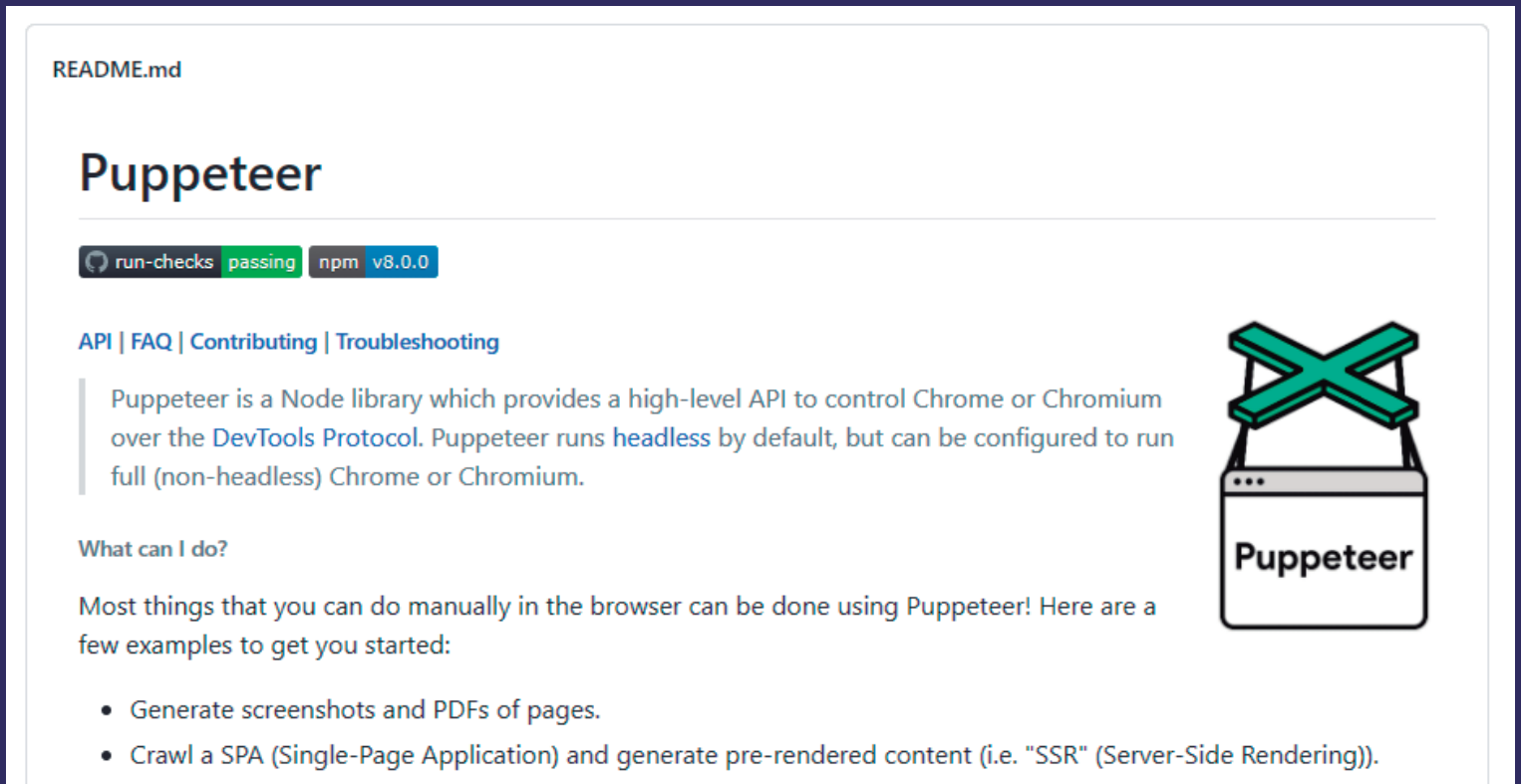
O GitHub recomenda fortemente a criação e o uso de um arquivo leia-me na raiz do seu projeto (então mesmo quando você não opta por criá-lo nas configurações iniciais, a plataforma te alerta de que ele ainda é necessário).

Esse arquivo nada mais é do que o arquivo de boas-vindas, e é por meio dele que iremos especificar a porque da existência do nosso projeto, o que ele faz, porque ele foi criado, como ele pode te ajudar e afins...



This screenshot is a zoomed-in view of the README.md file content from the previous image. It shows the text 'symmetrical-octo-barnacle' in a large, bold font, followed by a horizontal line and some ellipses below it.

Esse arquivo nada mais é do que a prévia textual do seu projeto, veja outro exemplo de um arquivo README.md:

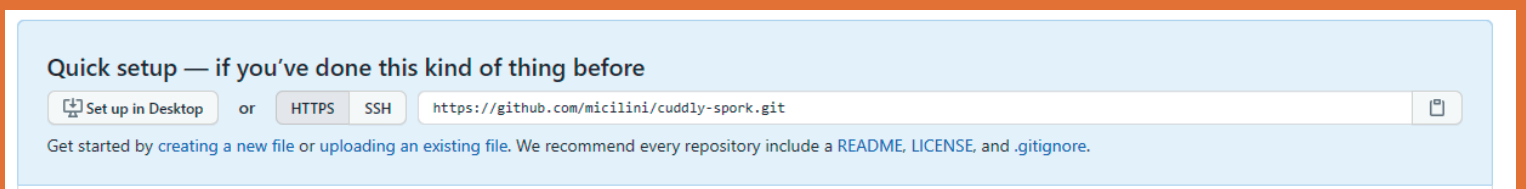


The screenshot shows a GitHub README.md file for the Puppeteer project. At the top, it says "README.md". Below that is the title "Puppeteer" in a large font. There are two status badges: "run-checks passing" and "npm v8.0.0". Below the badges are links for "API | FAQ | Contributing | Troubleshooting". The main text describes Puppeteer as a Node library for controlling Chrome or Chromium. To the right of the text is a logo for Puppeteer, which is a stylized green 'X' on top of a white box with the word "Puppeteer" inside. Below the main text, there is a section titled "What can I do?" followed by a list of examples: "Generate screenshots and PDFs of pages." and "Crawl a SPA (Single-Page Application) and generate pre-rendered content (i.e. 'SSR' (Server-Side Rendering))."



Fui para uma tela diferente...

Caso você tenha criado um novo repositório sem as opções de inicialização, você será redirecionado para a tela de Quick Setup, aonde o GitHub te instrui a criar um arquivo README.MD:



The screenshot shows the GitHub Quick Setup screen. It has a title "Quick setup — if you've done this kind of thing before". Below the title are two buttons: "Set up in Desktop" and "HTTPS SSH". To the right of these buttons is a text input field containing the URL "https://github.com/micilini/cuddly-spork.git". Below the input field is a small icon of a document. At the bottom of the screen, there is a line of text: "Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore."

Nesse caso, clique no link chamado README, e no final da próxima tela, basta clicar no botão chamado 'Commit New File'.

Pronto, após isso você conseguirá visualizar a mesma tela inicial do repositório como foi mostrada nas imagens anteriores.

No GitHub, todos os arquivos que terminam com a extensão `.md` (como é o caso do `README.md`), possuem uma marcação específica pela qual é administrado pelo github.

MD é a abreviação de **MarkDown**, cuja tradução é Marcador.

Basicamente ele é um arquivo de texto, na qual podemos estilizá-lo, de modo a:

- + Criar e formatar palavras e textos;
- + Adicionar negrito, itálico;
- + Adicionar imagens;
- + Colocar cores;
- + Importar links, criar listas e entre outros;



E sim, você pode ter mais de um único arquivo `.md` dentro do seu repositório do GitHub.

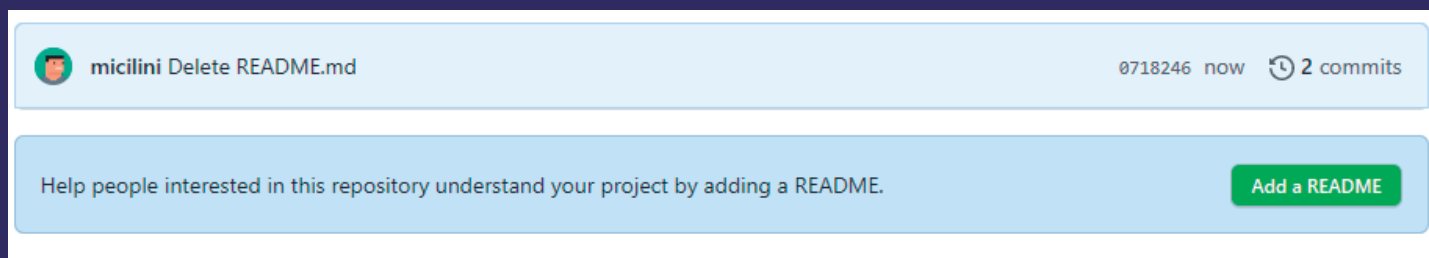


Mas entenda que:

Para o GitHub mostrar esses textos de boas-vindas que acabamos de visualizar, primeiro:

- + O arquivo `readme.md` deve estar na pasta raiz do seu projeto;
- + E o mais importante: ele deve ser chamado de `readme.md`;

Até porque se criamos outro arquivo com a extensão `.md`, o github não o reconhece como um marcador padrão, e mesmo assim ele pede para que você crie um arquivo com esse nome em específico se quiser criar uma tela de boas-vindas:

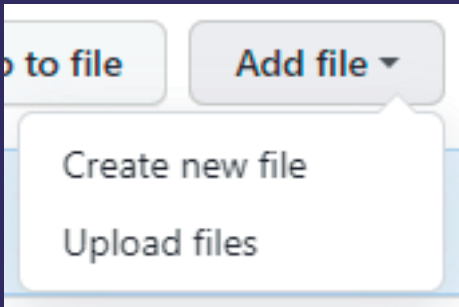


“Ajude as pessoas interessadas neste repositório a entender seu projeto adicionando um README.”

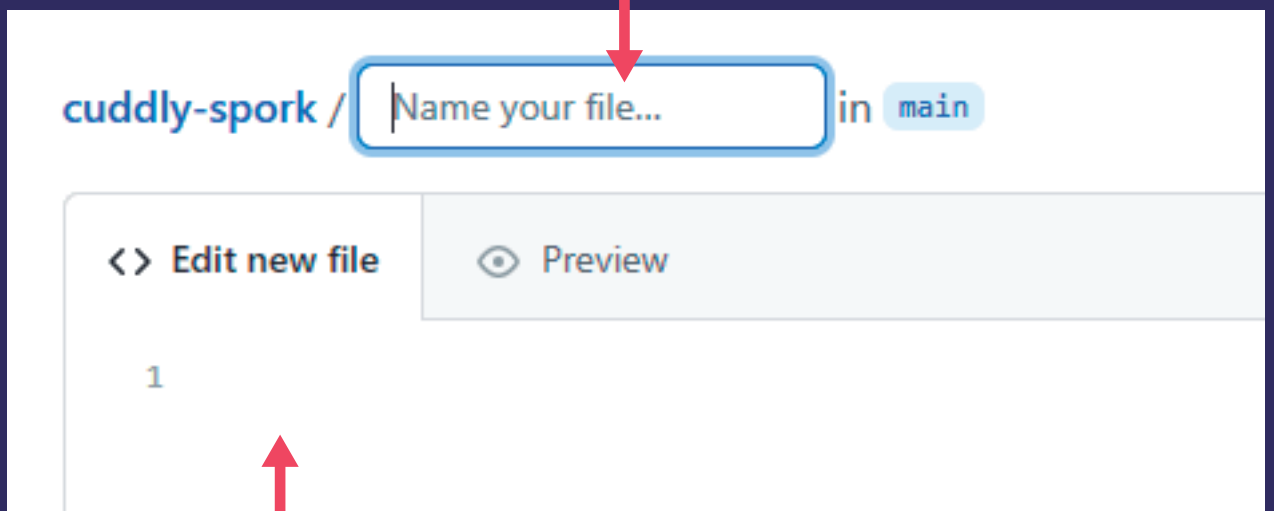


Editando e Criando Arquivos no GitHub

A criação e edição de arquivos diretamente pelo GitHub se dão de uma maneira bem simples. Para criar um arquivo, vá até a página inicial do seu repositório, e clique na opção **Add New File > Create New File**.



Nome e Extensão do Arquivo



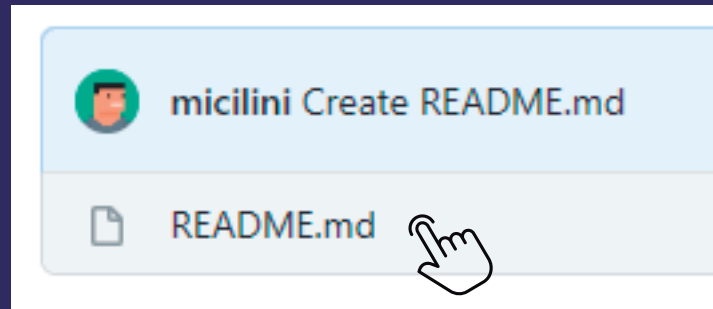
Conteúdo do Arquivo

Após isso, basta editar o texto, e no final da página clicar no botão commit.

E olha que interessante, mesmo sendo um arquivo de marcação, o GitHub já começa a controlar o versionamento dos arquivos, inclusive as alterações que você faz nesse arquivo.

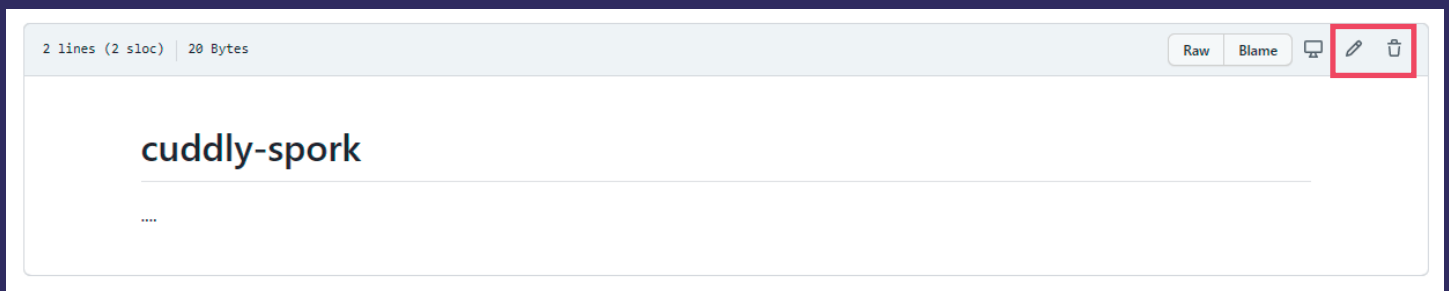
Bem diferente do terminal, não acha?

Já a edição acontece de uma forma bem simples também, basta apenas selecionar o arquivo desejado naquela estrutura de arquivos na página inicial do seu repositório, clicar nele:



E por fim, basta clicar no ícone de editar, para editar o arquivo, ou se preferir, clique no ícone da lixeira para excluir.

Não se esqueça de dar um commit no final para salvar as alterações.



Observação: README.MD

Tenha em mente que você pode criar, sim, o arquivo readme.md na pasta do seu projeto antes de enviá-lo ao GitHub, só que a maioria dos profissionais costuma criar e editar esse arquivo somente na versão visual do GitHub, isso devido a praticidade e a facilidade, no modo como é feita a edição diretamente pela plataforma do GitHub.





Entendendo o Arquivo .gitignore

Você sabia que é possível dizer ao Git para ignorar certos arquivos que não queremos versionar?

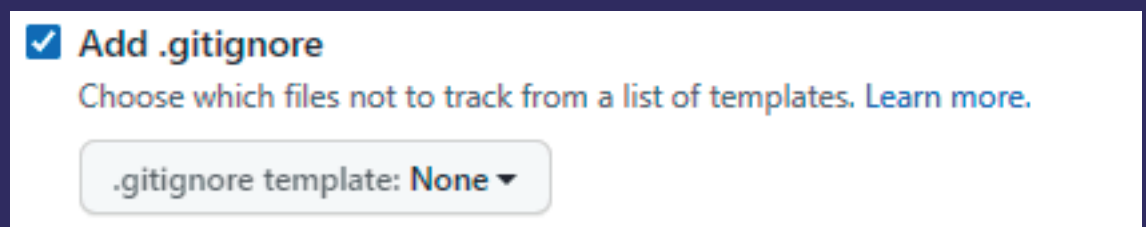
Ou seja, é possível dizer ao Git quais arquivos e diretórios devem ser ignorados ao fazer um commit.

Mas como alguém não quer versionar todos os arquivos do seu projeto, essa pessoa é louca?

Não, até porque, pode se tratar de arquivos temporários, como logs, configurações do editor de código, arquivos criados por meio do processo de compilação (que não precisamos versionar), e muitas outras coisas que podemos decidir não fazer o backup.

Para que o Git ignore certos arquivos, você precisa primeiro criar um arquivo chamado **.gitignore** na pasta raiz do seu projeto, de modo que fique na mesma hierarquia dos arquivos **readme.md** e da pasta **.git**.

Como estamos trabalhando diretamente no GitHub, nós podemos fazer isso por lá, assim que estivermos configurando o nosso repositório:



Tenha em mente que fazendo por este método, a plataforma te dá opções de template, aonde conta com uma lista de linguagens de programação que podemos selecionar.

Mas como funciona esses templates?

O entendimento deles é bem simples, isto é, caso você saiba um pouco da estrutura da linguagem daquele template.

Mas basicamente ele cria caminhos, seleciona locais, adiciona arquivos que geralmente não são versionados, observe esse exemplo do template do CakePHP:

```
1 # CakePHP 3
2
3 /vendor/*
4 /config/app.php
5
6 /tmp/cache/models/*
7 !/tmp/cache/models/empty
8 /tmp/cache/persistent/*
9 !/tmp/cache/persistent/empty
```

Pudemos ver que ele já pré seleciona alguns arquivos e pastas que são geralmente dedicadas a arquivos em cache, ou seja, arquivos temporários que mais tardar serão removidos.

No caso desse template ele fez uma configuração padrão, para o framework cakePHP, neste caso, estou assumindo que meu projeto usa o cakePHP, ok?

Mas, você não precisa selecionar um template, até porque você mesmo poderá especificar o que não será versionado.

E mesmo que você não tenha criado um arquivo de `.gitignore` na configuração inicial do repositório, você ainda pode criar esse arquivo dentro do repositório indo em **add file > create new file**.

E no nome do arquivo basta colocar `.gitignore`, e ser feliz =)

o arquivo `.gitignore` não é exclusivo do GitHub, e nesse caso, ele pode ser adicionado manualmente na pasta do seu projeto, e com isso ignorar o commit de certos arquivos de maneira offline diretamente pelo git.

END (y/n)

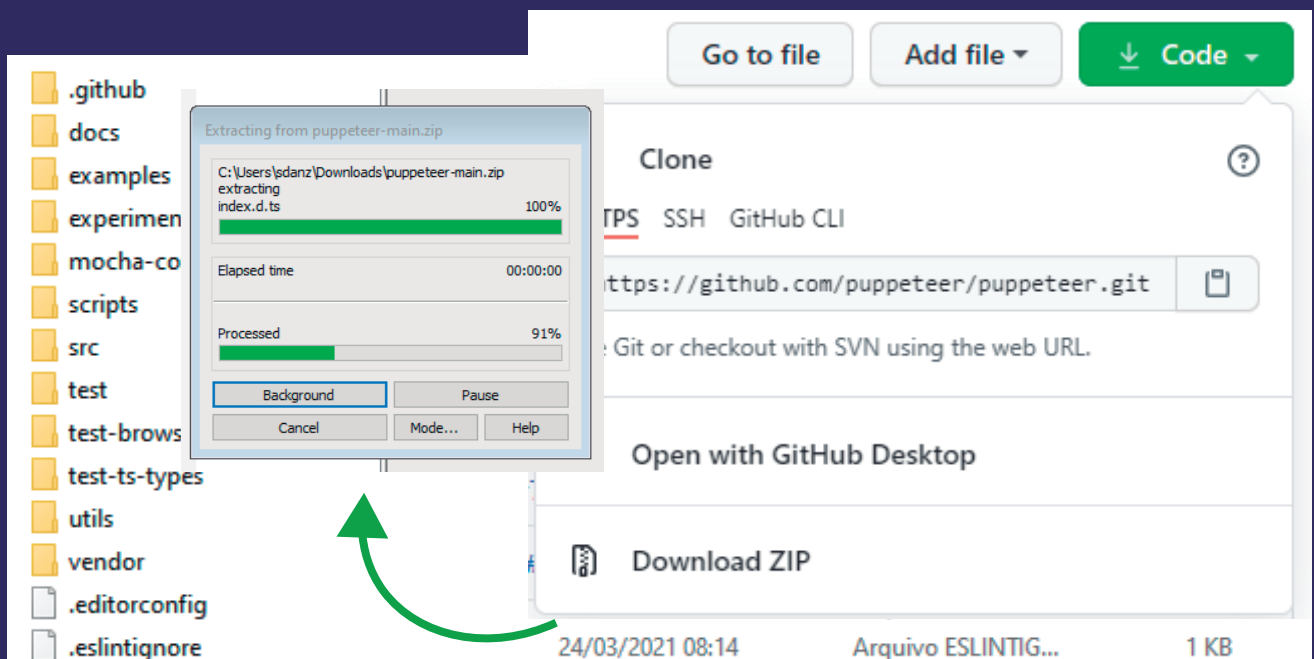


Fazendo Download de Repositórios (clone)



Como todos nós já sabemos, o GitHub é uma plataforma pública de projetos online, e por vezes pode acontecer de nos interessarmos por algum projeto ali existente, e com isso haver a necessidade de fazer download desse código-fonte, e trabalhar com ele dentro da nossa máquina.

A primeira opção que nós fazemos é abrir a URL do repositório, selecionar a opção **clone or download > Download Zip**, e no final descompactar os arquivos dentro da pasta do nosso projeto, e colocar para mão na massa.



Apesar de ser uma grande tentação fazer esse tipo de procedimento, nós não faremos isso dessa forma, uma vez que o intuito desse curso é fazer você usar as ferramentas integradas do Git (que está instalado em nosso computador) junto ao GitHub (Plataforma Online).

Então, para fazermos o download de um repositório diretamente na nossa máquina, nos utilizamos um comando especial para isso:

```
git clone
```

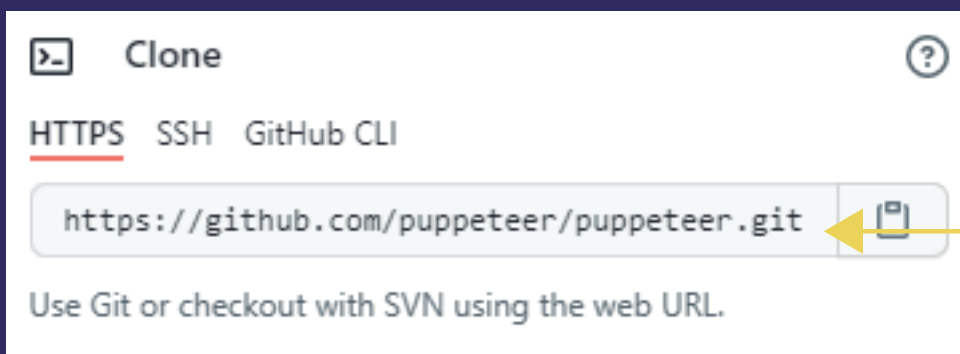
Diz ao Git para clonar um repositório, trazendo todas as informações locais e remotas deste repositório para dentro da sua máquina.

É importante ressaltar que o Git Clone quando executado, herda algumas configurações de conexão entre o projeto (instalado na sua máquina), e o repositório online, de modo que quando executamos um git commit e posteriormente um git push, não precisaremos fazer nenhuma conexão adicional ou configuração de SSH.



Praticando o Git Clone

Para clonarmos um repositório, é bem simples, com a URL aberta, basta selecionar a opção 'clone or download', e copiar a URL que aparece dentro do campo de texto escrito 'Clone With HTTPS'..



Você precisa copiar este link

Em seguida, crie uma pasta no seu computador, onde estarão localizados todos os projetos que você irá clonar no GitHub (isso para manter a organização).

Não é necessário criar uma pasta com o nome do repositório que você está clonando, uma vez que o Git faz isso automaticamente para você.

Com a pasta devidamente criada, basta entrarmos nela, e executarmos o terminal do git (Windows: Git Bash Here).

Após isso, basta executar o seguinte comando:

```
git clone urladorepositorioquevocecopiou
```

No meu caso, ficou assim:

```
git clone https://github.com/puppeteer/puppeteer.git
```

Agora se entrarmos na pasta do projeto que clonamos, veremos que todos os arquivos estarão listados lá =)



IMPORTANTE !!!

Quando nos fazemos o clone de qualquer projeto existente no GitHub, não precisamos mais realizar o comando **git init**, uma vez que ele já vem iniciado e pré-configurado para isso.

E se eu criar um novo repositório totalmente vazio, e mesmo assim realizar o clone, será que nesse caso eu preciso executar o **git init**?

Na verdade não, e você pode tirar suas próprias conclusões fazendo esse teste.

Você vai perceber que assim que o clone é finalizado, a pasta invisível (.git) virá junto, e isso significa que o git já foi inicializado desde o GitHub.

Qual a diferença entre baixar via zip ou fazer isso pelo terminal e comando?

A diferença principal é que com o terminal (`git clone`), conseguimos de baixar certos arquivos que nos permitem sincronizar as alterações que estamos realizando no nosso computador com o repositório online do projeto no GitHub.

Então não precisaríamos configurar conexões remotas do tipo SSH.

Eu posso editar qualquer repositório clonado, usando o código que aprenderemos a anteriormente?

Não, isso só funciona quando o repositório é nosso (ou o dono nos da permissão), ok? Para adicionar, alterar, ou remover o código-fonte de repositórios que não foram criados por você, é necessário permissão de quem criou o repositório, tá bom?



Enviando nosso projeto para o GitHub



Chegado a hora mais esperada desse conteúdo, agora é o momento de aprendermos a jogar nossos projetos para dentro da plataforma do GitHub, e o comando responsável por isso é o:

Com ele, nos somos capazes de sincronizar todos os arquivos e versionamentos existentes e que fizemos na pasta do projeto para o repositório online do GitHub.

```
git push
```



Praticando o Git Push (clone)

Se formos dentro da pasta Meu Projeto (criada no último e-book, se lembra?), abriremos o terminal do Git por lá e digitarmos o comando **git push**, olha só o que acontece:

```
sdanz@DESKTOP-UHDBDVB MINGW64 ~/Desktop/Meu Projeto (master)
$ git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote repository using

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

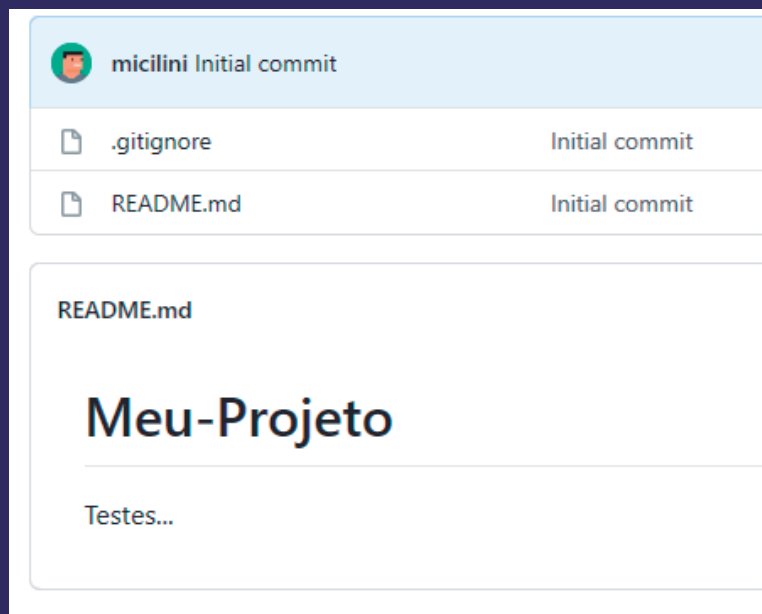
Aparece um erro alegando de que não configuramos nenhum destino...

Como estamos querendo enviar um projeto da nossa máquina para dentro do GitHub, nós precisamos primeiro, ter um repositório configurado com o mesmo nome do projeto, e que ele esteja de preferência vazio, isto é, somente com os arquivos de inicialização que podemos criar a partir do github, que são: **.readme.md** e **.gitignore**, pois estes sincronizaremos mais tarde.

Sendo assim a primeira coisa que devemos fazer é criar um repositório no GitHub, e que este tenha o mesmo nome do nosso projeto que estamos criando:

(Com relação ao template do .gitignore, pode ser qualquer um, uma vez que não fará diferença agora)

Após setar seu repositório, chegou a hora de executar alguns passos...

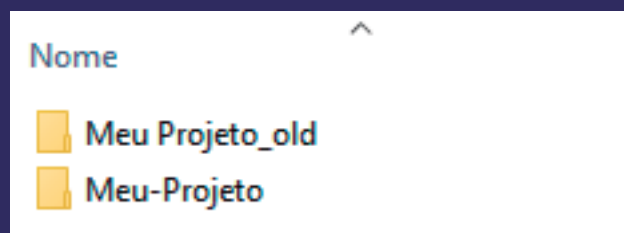


#0: Renomeando a sua pasta do projeto para `..._old`

Sabe aquela pasta que você criou no e-book anterior? (Meu Projeto), então... renomeie ela para `Meu Projeto_old`, mais tarde você vai entender o motivo.

#1: Clonando seu repositório para seu computador

Na mesma pasta raiz, disco local, ou subpasta onde está localizada a pasta `Meu Projeto_old`, você irá abrir o terminal do Git, e fazer um clone do repositório que acabamos de criar, de modo que agora existam duas pastas:



#2: Copiando os arquivos para a nova pasta

O segundo passo, consiste em você abrir a pasta `Meu Projeto_old` e copiar todos os arquivos de código-fonte que você criou (menos a pasta `.git`) para dentro da nova pasta que foi clonada '`Meu-Projeto`':

#3: Realizando o Primeiro Git Push

Após fazer isso, feche o terminal do git (que voce abriu para fazer o git clone), abra a pasta '`Meu-Projeto`' e execute o terminal por lá.

Ou se preferir, insira o código no terminal do Git '`cd/Meu-Projeto`' que também funciona da mesma forma.

Lá dentro basta executar os comandos padrão:

```
git add .
git commit -m 'commit inicial'
git push
```


```
sdanz@DESKTOP-UHBDVB MINGW64 ~/Desktop/Projetos Git/Meu-Projeto (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 359 bytes | 119.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/micilini/Meu-Projeto.git
 132118d..8c25602  main -> main
```

Como estamos vendo, aparecerá uma mensagem alegando que os dados foram sincronizados com nosso repositório no GitHub.


Se isso não acontecer, certifique-se de que o `user.name` e o `user.email` existentes no `git config`, são os mesmos do seu GitHub.

Agora se você abrir o seu repositório, verá que um novo arquivo foi criado, que no caso é o arquivo `exemplo.txt`:

 main ▾  1 branch  0 tags

 micilini commit inicial

 .gitignore Initial commit

 README.md Initial commit

 exemplo.txt commit inicial

Isso significa que eu sempre tenho que clonar a pasta para fazer o push?

Sim..... e não!

Esse método que vimos, é o método mais fácil, e que é a primeira forma que nos é apresentada para aqueles que estão iniciando no Git e no GitHub.

Existe sim outro método, na qual não precisamos clonar nosso repositório e fazer todos os passos que vimos anteriormente, nesse caso, nós faremos isso diretamente na pasta do projeto original. *(No Meu Projeto_old por assim dizer)*

Seguindo por esse método de copiar e colar os arquivos na pasta clone, como fica caso eu tenha feito alguns versionamentos no projeto anterior?

Ai neste caso, a princípio você perderia esses versionamentos, visto que, só estamos enviando os arquivos e não o conteúdo existente dentro da pasta .git (que contém o backup).

 [Como copiar commits de um repositório Git para outro?](#)



Praticando o Git
Push (sem clone)

Todos nos sabemos que o Git é um sistema VCS que roda em nosso computador, e que é capaz de versionar nossos projetos.

Sabemos também que o GitHub é a plataforma que consegue hospedar nossos projetos com os versionamentos que fizemos pelo Git, até aí tudo bem...

Sabemos também que podemos enviar nossos projetos versionados pelo Git diretamente para o GitHub, e isso já nos foi explicado em e-books anteriores.

```
git config --global
```

```
user.name micilini
```

```
user.email mi@mi.com
```

Se você se lembra, a única configuração que fizemos foi o nome e o e-mail do desenvolvedor que ficaria responsável por versionar os códigos, e que esses dados deveriam ser os mesmos que estão na sua conta do GitHub, certo?

Junto a isso, em nenhum momento precisamos fazer uma conexão entre o nosso Git e a nossa conta do GitHub, até porque usando o método anterior (`git clone`), ele meio que já trouxe por de baixo dos panos, as configurações de conexão, e por conta disso, bastávamos que déssemos um simples `git push`.

```
git push
```



Mas supondo que não queiramos usar o método do `git clone`, uma vez que ele é um processo demorado e um pouco redundante.

O que será que acontece quando executamos um `git push`, na pasta `Meu Projeto_old`?

```
sdanz@DESKTOP-UHDBDVB MINGW64 ~/Desktop/Meu Projeto (master)
$ git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote repository using

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

Como podemos ver, o Git nos alerta de que não há nenhum destino. E por conta disso, devemos configurá-lo manualmente.

E para fazer esta conexão, precisamos usar o comando:

Será através desse comando que poderemos visualizar para aonde o nosso projeto local esta apontando.

```
git remote
```

```
sdanz@DESKTOP-UHDBDVB MINGW64 ~/Desktop/Projetos Git/Meu Projeto_old (master)
$ git remote
```

E no nosso caso, ele não retornou absolutamente nada, e isso significa que não esta havendo conexão nenhuma, e isso é óbvio, uma vez que não configuramos esse tipo de coisa na pasta do nosso projeto.

Para estabelecer uma conexão, primeiro: Devemos nos certificar que existe um repositório no GitHub aonde iremos fazer o upload dos nossos arquivos (o que já foi feito em passos anteriores).

Em segundo lugar devemos utilizar um comando responsável por criar um destino para este projeto, aonde nele informaremos o nosso nome de usuário e o repositório:

```
git remote add origin
git@github.com:<seunomedeusuariodogit>/<nomedorepositorio>.git
```

(o código acima só tem uma linha, ok? Não precisa dar uma quebra)

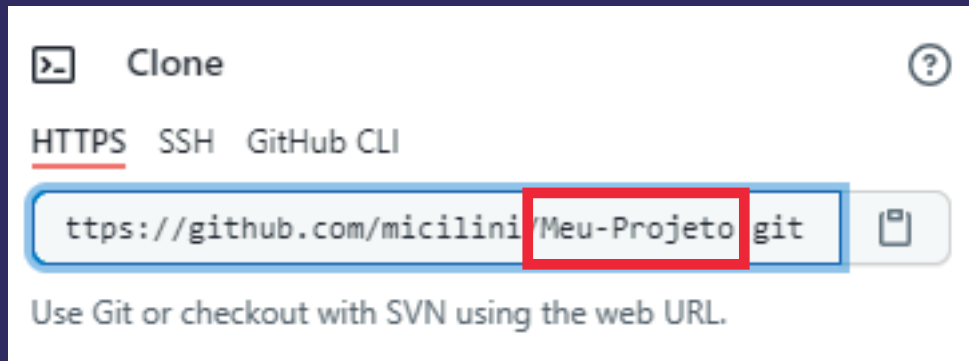
Este comando, cria uma conexão remota dentro do parâmetro Origin. O parâmetro origin especifica a origem dessa conexão, ou seja, "Meu Projeto tem uma origem online e ela aponta para o seguinte repositório..."

E esse repositório deve ser especificado com o seguinte parâmetro:

```
git@github.com:<seunomedeusuariodogit>/<nomedorepositorio>.git
```

Aonde no **<seunomedeusuariodigit>** você coloca o nome do seu usuário no github, e aonde está escrito **<nomedorepositorio>** você coloca o nome do repositório criado pelo GitHub (Ele sempre vem em formato de URL).

Para saber qual é o nome do seu repositório, você pode ir à página inicial dele, clicar em Code, e copiar a última linha do campo clone:



Veja como ficou o meu:

```
git remote add origin git@github.com:micilini/Meu-Projeto.git
```

Se executarmos `git remote` novamente, veremos que o origin foi retornado para nós, e isso significa que existe uma configuração de conexão remota.

Para obtermos mais detalhes dessa configuração, nos podemos executar o comando `git remote show origin`, para obter mais detalhes dessa conexão:

```
$ git remote show origin
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Como podemos ver, ocorreu um erro ao estabelecer uma conexão, isso aconteceu devido a um problema com sua chave pública do github, e que ainda não foi criada (ssh).

Mesmo que executemos o `git push` a mensagem de erro persistirá.



Conexões Remotas chave SSH

Se você está aqui, é porque provavelmente está tendo problemas para fazer o tal do git push do seu projeto local para o repositório online.

Aquele erro de chave pública acontece, porque nós não configuramos ainda, uma chave SSH para a sua máquina, e outra chave SSH que deve ser configurada no site do GitHub.

Mas eu não precisei configurar isso naquela vez que fizemos o clone.

Sim, pois como dito anteriormente, quando o clone é executado, o Git acaba herdando algumas configurações de conexão diretamente do GitHub, por isso, quando realizamos um simples push, ele consegue estabelecer uma conexão sem apresentar esse problema de chave pública.

E como posso corrigir isso?

Para corrigir este problema é bem simples, basta seguir os passos a seguir:

#1 Abra o Terminal do Git

Abra o terminal do Git na sua área de trabalho do seu sistema operacional, e digite o seguinte comando (E dê enter, sem preencher nenhum dado):

```
ssh-keygen  
ssh-agent
```

O comando que acabamos de executar é responsável, configurar uma chave do tipo SSH, que servirá para validarmos na plataforma do GitHub.

Tenha em mente que, esse processo não é exclusivo do Git ou do GitHub, visto que você pode executar ambos os comandos pelo terminal do seu sistema operacional.

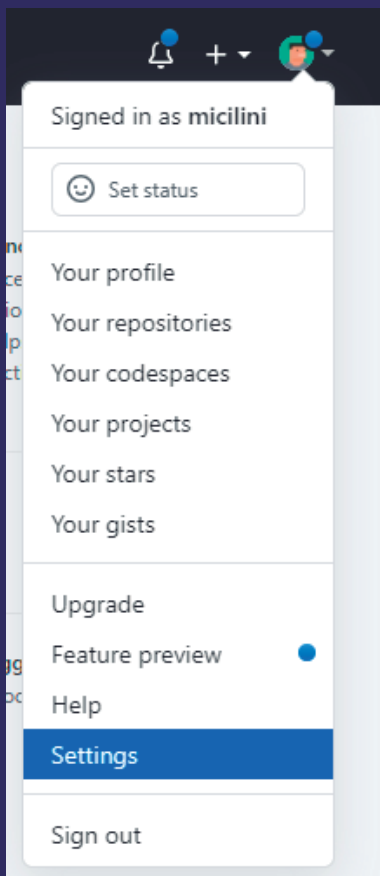
```
Enter same passphrase again:  
Your identification has been saved in C:\Users\sdanz/.ssh/id_rsa.  
Your public key has been saved in C:\Users\sdanz/.ssh/id_rsa.pub.  
The key fingerprint is:
```

Como podemos ver, a nossa identificação foi salva dentro de uma pasta `.ssh`, localizada na pasta do meu usuário, mais especificamente dentro do arquivo `id_rsa.pub`

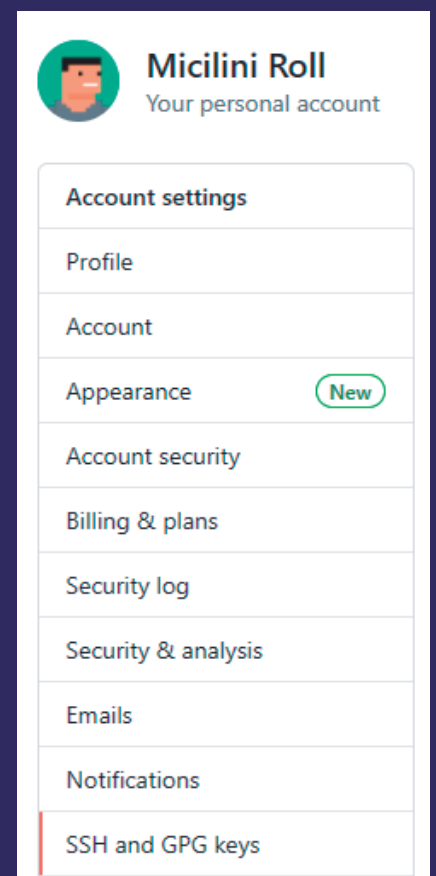
Vá até essa pasta, abra esse arquivo com qualquer editor de texto (pode ser o bloco de notas), e copie todo o código que aparece ali dentro.

#2 Criando uma chave SSH no GitHub

O segundo passo, é abrir a plataforma do GitHub, fazer login, abrir o menu, e selecionar a opção de `'settings'`:



Após isso, no menu a esquerda, selecione a opção chamada `SSH and GPG Keys`.



SSH keys

New SSH key

There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or [troubleshoot common SSH Problems](#).

Em seguida clique no botão 'New SSH Key':

SSH keys / Add new

Title

Key

Add SSH key

E basta que você coloque um título para essa nova chave SSH, eu coloquei uma identificação de que era meu computador, e no campo Key, cole ali o código que você copiou daquele arquivo `id_rsa.pub`

Por fim clique no botão **Add SSH Key**.

Lembrando que você pode criar quantas chaves SSH você desejar, ok? Então se algum dia você precisar fazer um git push de outra máquina, basta que você repita esse processo, ok?

A partir de agora, sempre quando o git push for sincronizar seus arquivos no seu repositório, por de baixo dos panos, ele vai lá naquele arquivo `id_rsa`, e vai validar com o(s) código(s) que está(ão) inserido(s) na sua conta do GitHub.





Voltando a praticar o git push (sem clone)

Agora que já temos uma chave SSH configurada, acredito que o Git não terá problemas para fazer o **git push** para o repositório, vamos tentar?

```
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 533 bytes | 106.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/micilini/Meu-Projeto/pull/new/master
remote:
To github.com:micilini/Meu-Projeto.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

E se recarregarmos nosso repositório, veremos que o arquivo foi atualizado e ele está lá =)



Atualizando a pasta local com o repositório online

Existe um comando muito utilizado, que nos permite atualizar o nosso repositório local (pasta do nosso projeto), com o repositório online (GitHub). E esse comando é: conhecido como:

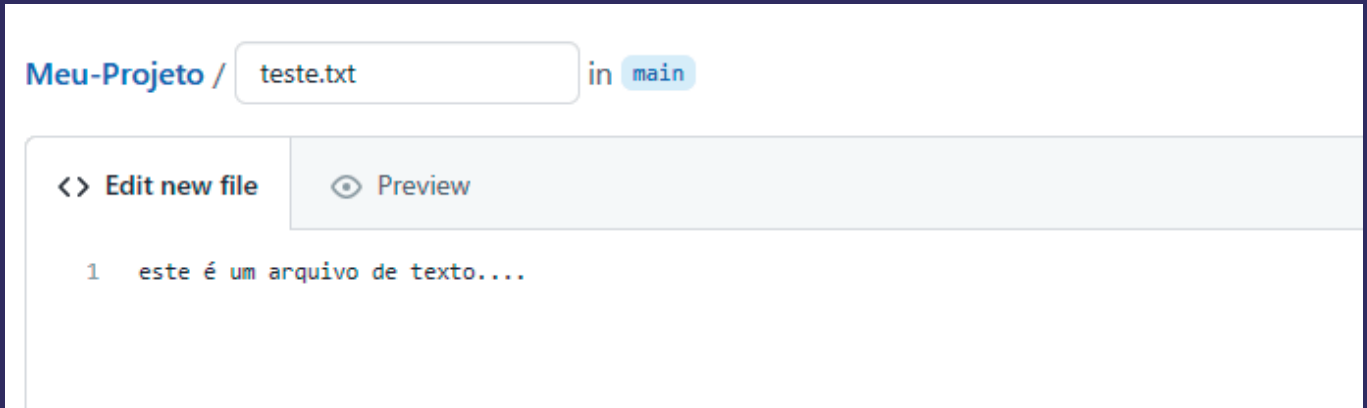
```
git pull
```

Para que ele funcione, nós precisamos abrir o terminal dentro da pasta que baixamos via clone, ou dentro da pasta na qual adicionamos uma conexão via SSH com o Git Remote.



Praticando o git pull

Para fazermos um teste, vá até o seu repositório do GitHub, e crie um arquivo de texto simples por lá, no meu caso eu criei um arquivo chamado `teste.txt`.



Após fazer o commit desse arquivo, vá até a pasta do seu projeto, execute o terminal do Git por lá, executando o comando do git pull.

Automaticamente o git começara a sincronização dos arquivos que estão em nuvem (repositório online), com aqueles que estão instalados na sua máquina.

```
sdanz@DESKTOP-UHDBDVB MINGW64 ~/Desktop/Projetos Git/Meu-Projeto (n
$ git pull
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 8 (delta 1), reused 5 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), 1.09 KiB | 26.00 KiB/s, done.
From https://github.com/micilini/Meu-Projeto
   8c25602..d4f253c  main       -> origin/main
  * [new branch]   master     -> origin/master
Updating 8c25602..d4f253c
Fast-forward
 teste.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 teste.txt
```



Após a finalização, você verá no seu repositório local, um novo arquivo chamado `teste.txt`, viu como é fácil?

Esse é um processo muito comum de acontecer, principalmente quando estamos trabalhando em equipes, uma vez que poderão ser muitos desenvolvedores trabalhando em um mesmo projeto, e toda vez que precisarmos baixar novas atualizações (até mesmo arquivos novos), precisamos sincronizar o repositório online com o local.



Recapitulando o fluxo de push até o pull

Acredito que a partir de agora você já tenha mais ou menos uma ideia para jogar seus projetos online e fazer suas devidas sincronizações, certo?

Mas antes de passarmos para a próxima etapa, vamos fazer uma recapitulação rápida, ok?

```
git clone
```

Diz ao Git para clonar um repositório, trazendo todas as informações locais e remotas deste repositório para dentro da sua máquina.

- Precisamos informar o caminho do Clone.
- Não precisamos executar o `git init`, pois dessa forma o repositório já vem inicializado.

Diz ao Git para sincronizar todos os arquivos e versionamentos existentes e que fizemos na pasta do projeto para o repositório online do GitHub.

```
git push
```

- É necessário que isso seja feito numa pasta clonada, ou que tenha o caminho origin já configurado.



```
git remote
```

Diz ao Git para visualizar as conexões remotas que o repositório local está apontando.

- Podemos configurar a conexão remota alterando ou adicionando um origin.



Diz ao Git para que ele pegue todos os arquivos do repositório online, e os atualize com o repositório local.

```
git pull
```

- Ele pode trazer novos arquivos, excluir alguns ou até mesmo vier editados com novas versões.





E agora, o que fazer?

Neste modulo do e-book você aprendeu um pouco sobre:

- * O que são repositórios;
- * Como criar repositórios;
- * Como criar, editar ou excluir arquivos no GitHub;
- * Como clonar repositórios;
- * Como enviar seu repositório local para o GitHub;
- * Como criar chaves públicas SSH;
- * Como atualizar seu repositório local a partir do online;



E então, animado para saber um pouco mais sobre branches?

END (y/n)

Gostou desse material?



Então não deixe de acessar a nossa seção de [Git & GitHub do básico ao avançado](#).

ACESSAR



MICILINI.COM

<Seu Portal de CODE>