

# Aula #6: Primeiro Commit

---

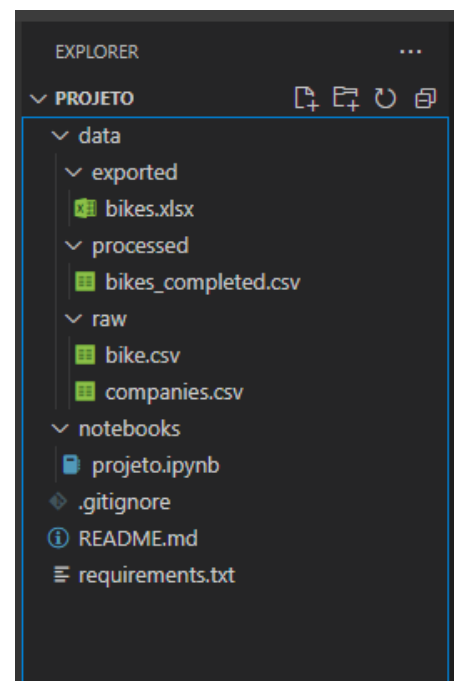
<a href="#">Estrutura do Projeto</a>	1
<a href="#">Inicialização do Repositório Local</a>	2
<a href="#">O Arquivo .gitignore</a>	3
<a href="#">Fazendo o Primeiro commit</a>	5
<a href="#">Exercícios</a>	7
<a href="#">Próxima Aula</a>	8
<a href="#">Fontes e Links Complementares</a>	8

---

## Estrutura do Projeto

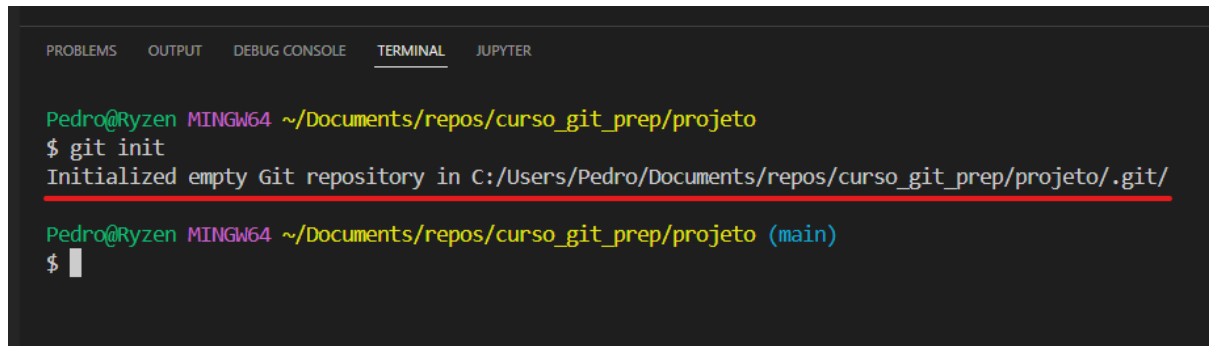
O projeto contém a seguinte estrutura:

- **data**: diretório que contém os dados não tratados, tratados e exportados;
- **notebook**: Diretório que contém os arquivos notebooks;
- **requirements.txt**: Arquivo com as dependências do projeto;
- **.gitignore**: Arquivo com arquivos, diretórios e extensões que devem ser ignorados pelo git;
- **README.md**: Arquivo de markdown com a explicação do projeto



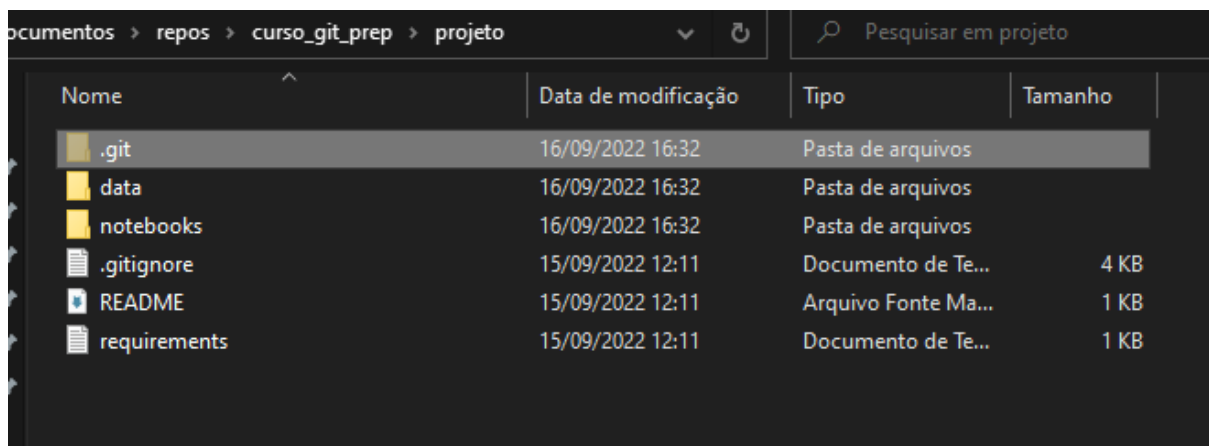
# Inicialização do Repositório Local

Para inicializar um repositório local, basta utilizar o comando `git init` no terminal, estando dentro do diretório raiz do projeto.

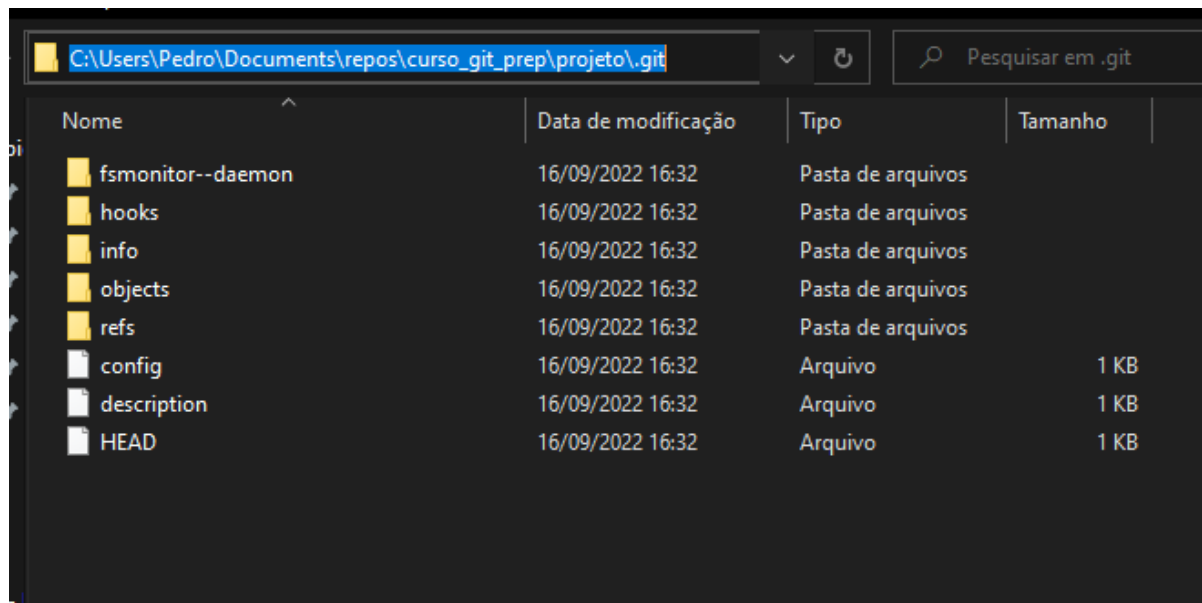


```
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto
$ git init
Initialized empty Git repository in C:/Users/Pedro/Documents/repos/curso_git_prep/projeto/.git/
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$
```

Com isso, será criado uma pasta chamada `.git`, que conterá todos os arquivos e diretórios necessários para que o projeto seja versionado. Embora não seja uma boa prática, caso necessário, você pode copiar o projeto todo, junto com a pasta `.git`, e fazer o backup dela em outro disco.

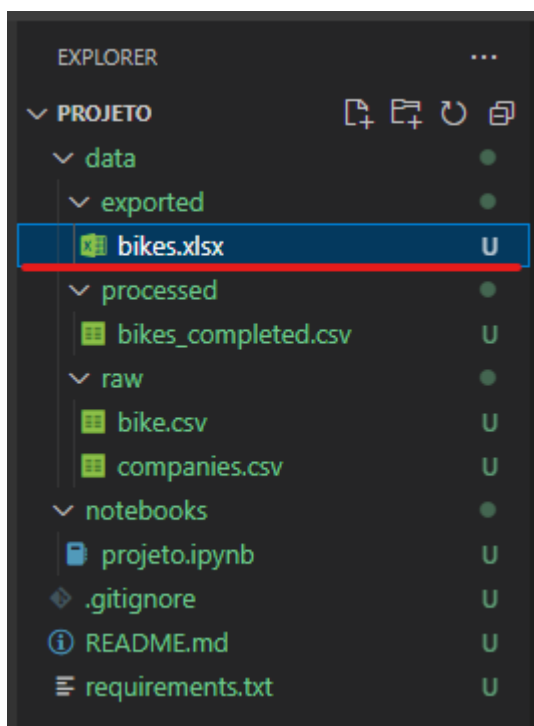


Nome	Data de modificação	Tipo	Tamanho
.git	16/09/2022 16:32	Pasta de arquivos	
data	16/09/2022 16:32	Pasta de arquivos	
notebooks	16/09/2022 16:32	Pasta de arquivos	
.gitignore	15/09/2022 12:11	Documento de Te...	4 KB
README	15/09/2022 12:11	Arquivo Fonte Ma...	1 KB
requirements	15/09/2022 12:11	Documento de Te...	1 KB



## O Arquivo `.gitignore`

Imagine a seguinte situação: Dentro do projeto, temos arquivos que não desejamos versionar, como arquivos de configuração local, ou, no caso da linguagem Python, o diretório `__pycache__`, que é gerado pela linguagem quando executamos uma aplicação. E é aí que o arquivo `.gitignore` entra: Ele nos possibilita informar ao Git quais arquivos, pastas ou extensões não desejamos versionar, evitando assim de armazenarmos dados que não sejam necessários.

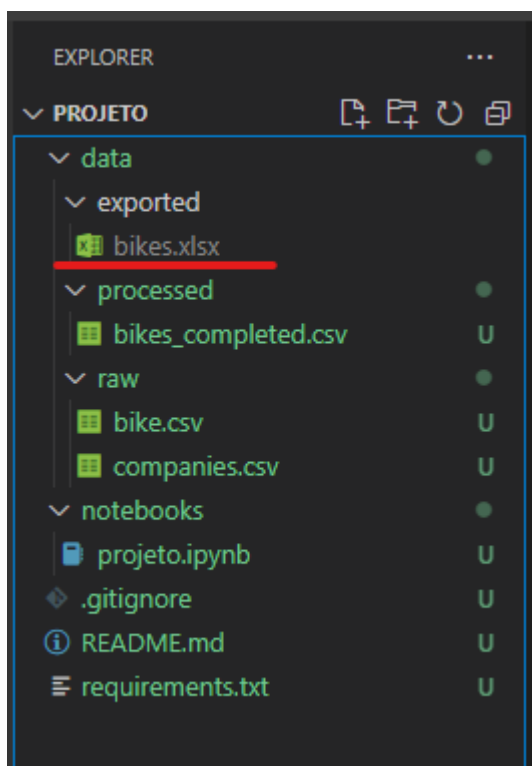


Existem muitos *templates* de arquivos `.gitignore` conforme a linguagem e, geralmente, quando criamos um repositório do zero, o arquivo `.gitignore` é um dos primeiros arquivos que criamos.

Quando inicializamos o repositório local no nosso projeto, podemos observar que os nomes dos arquivos na IDE VS Code ficaram verdes e com a letra “U” na frente de seus nomes.

Isso significa que esses arquivos foram reconhecidos pelo Git, mas que ainda não foram versionados. Imagine que não desejemos versionar todo e qualquer arquivo de planilha de excel. Ou seja, todo e qualquer arquivo que possua a extensão `.xlsx`. Para fazermos isso, basta adicionarmos essa extensão dentro do arquivo `.gitignore`:

```
.gitignore U x
.gitignore
155
156 # PyCharm
157 # JetBrains specific template is maintained in a separate JetBrains.gitignore that can
158 # be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
159 # and can be added to the global gitignore or merged into this file. For a more nuclear
160 # option (not recommended) you can uncomment the following to ignore the entire idea folder.
161 #.idea/
162 *.xlsx
```



Uma vez que a extensão é inserida no arquivo e a alteração é salva, podemos observar que agora o nome do arquivo que contém a extensão `.xlsx` não está mais verde e nem com a letra “U” na frente do seu nome.

Por tanto, sempre que precisarmos fazer com que o Git ignore arquivos, extensões ou pastas, adicionamos elas dentro do arquivo `.gitignore` para que elas sejam ignoradas no processo de versionamento.

# Fazendo o Primeiro *commit*

Agora que vimos como ignorar arquivos, extensões e pastas, vamos fazer o nosso primeiro `commit`. Um `commit` nada mais é que dizermos ao Git que todas as nossas alterações e modificações nos arquivos do projeto estão “prontas” e devem ser salvas. Ou seja, desejamos criar uma “versão” das nossas modificações.

Todas as vezes que fazemos alterações nos arquivos do projeto, e essas alterações são terminadas e ficam “prontas”, criamos um `commit`. Um `commit` deve possuir uma mensagem informando o que foi feito de alterações e a boa prática nos diz para fazermos `commits` “atômicos”, ou seja, `commits` com poucas alterações e que sigam uma linha ou lógica de alterações, para que, se necessário, possamos voltar para um `commit` anterior sem que sejam desfeitas muitas alterações.

O processo para fazer um `commit` é simples: Primeiro, temos que verificar o estado dos arquivos versionados no nosso repositório local. Para isso, utilizamos o comando `git status`:

```
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        README.md
        data/
        notebooks/
        requirements.txt

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

Esse comando irá nos informar várias coisas: Qual a branch que estamos, quais arquivos estão prontos para serem `commitados`, quais arquivos ainda não foram

marcados como prontos e etc. Caso isso esteja confuso, não se preocupe, iremos rever esses conceitos de forma mais aprofundada nas próximas aulas.

Como podemos observar, a saída do comando `git status` nos diz que todos os arquivos em vermelho ainda não estão sendo versionados ou monitorados pelo Git. Para marcarmos um arquivo como pronto para ser `commitado`, utilizamos o comando `git add <nome_arquivo>`. Porém, existe um atalho caso seja necessário marcar todos os arquivos de uma única vez, esse atalho é o caractere de ponto (`.`). Como esse é o nosso primeiro `commit`, iremos inserir todos os arquivos de uma única vez:

```
$ git add .
warning: in the working copy of 'data/processed/bikes_completed.csv', CRLF will be replaced by LF the next time Git touches it
warning: in the working copy of 'data/raw/bike.csv', CRLF will be replaced by LF the next time Git touches it

Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .gitignore
    new file:   README.md
    new file:   data/processed/bikes_completed.csv
    new file:   data/raw/bike.csv
    new file:   data/raw/companies.csv
    new file:   notebooks/projeto.ipynb
    new file:   requirements.txt
```

Como podemos observar pelo comando `git status`, agora todos os arquivos estão marcados como prontos para serem `commitados`! Para realizar o `commit`, utilizamos o comando `git commit -m "Mensagem"`:

```
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git commit -m "Primeiro Commit"
[main (root-commit) 085ae1c] Primeiro Commit
 7 files changed, 8883 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 data/processed/bikes_completed.csv
 create mode 100644 data/raw/bike.csv
 create mode 100644 data/raw/companies.csv
 create mode 100644 notebooks/projeto.ipynb
 create mode 100644 requirements.txt

Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git status
On branch main
nothing to commit, working tree clean
```

Feito isso, foi criado uma versão inicial para todos os arquivos do projeto! Se executarmos o comando `git status` novamente, podemos ver que nos é retornada uma mensagem dizendo que não há nada novo para ser `commitado`. Agora, todas as alterações que fizemos, serão rastreadas e poderemos não somente ver as alterações feitas, mas também retornar para uma versão anterior!

Em resumo, os comandos que utilizamos para realizar um commit são:

```
$ git add <nome_arquivo> | .
```

```
$ git commit -m "Mensagem"
```

## Exercícios

Link para o formulário de exercícios de fixação de conteúdo: [Exercícios](#)

# Próxima Aula

Na próxima aula, entender melhor o que é um versionador, como o Git funciona e qual o problema que ele resolveu.

## Fontes e Links Complementares

[Livro - Capítulo 2.2 - Gravando alterações em um repositório](#)