

# Aula #10: Áreas de Trabalho do Git

---

<b>Áreas de trabalho do Git</b>	<b>1</b>
Área de Repositório (Repository)	2
Working Directory	2
Área de Staging (Staging Area)	3
<b>Interação entre as Áreas de Trabalho do Git</b>	<b>3</b>
Arquivos Novos	9
Arquivos Já Versionados	9
<b>Exercícios</b>	<b>10</b>
<b>Próxima Aula</b>	<b>10</b>
<b>Fontes e Links Complementares</b>	<b>11</b>

---

## Áreas de trabalho do Git

Conforme vimos na aula anterior, os arquivos versionados pelo Git possuem um ciclo de vida. Ou seja, transitam entre estados dentro da ferramenta quando um processo de preparação e armazenamento das alterações é feito.

Porém, esse ciclo de vida ocorre dentro de 3 grandes áreas do Git, que são:

- Área de Repositório
- Working Directory
- Área de Staging

Vamos ver em mais detalhes como essas áreas de Git funcionam e quais etapas do ciclo de vida compreendem essas áreas.

## Área de Repositório (Repository)

A **Área de Repositório** é a área onde estão todos os arquivos estão sendo versionados e que não possuem nenhuma alteração. É também a área onde todos os arquivos que foram alterados e tiveram o seu estado commitados vão.

Fazendo um paralelo com as seções do ciclo de vida de um arquivo versionado, vistos na aula passada, essa área abrange a seção **Unmodified**. Ou seja, sempre quando fizermos alguma alteração em um arquivo, ele irá sair da **Área de Repositório**, irá para a **Working Directory**, depois para a **Área de Staging** e por fim, voltará para a **Working Directory**.

Veremos melhor esse processo ao longo desta aula.

## Working Directory

A **Working Directory** é onde estão todos os arquivos que estão sendo trabalhados ou modificados, mas que ainda não estão prontos. Ou seja, são os arquivos que estamos alterando para melhorar/criar uma funcionalidade ou resolver um bug.

Novamente, se traçarmos um paralelo com a aula anterior, a seção de **Modified** está contida dentro desta área do Git.

Caso as alterações sejam “aceitas” e marcadas como prontas, os arquivos marcados irão dessa área para a **Área de Staging**. Caso as alterações sejam descartadas, os arquivos marcados voltarão para a **Área de Repositório**.

## Área de Staging (Staging Area)

Por fim, a **Área de Staging** é a área onde ficam os arquivos que possuem alterações e estão marcados como prontos para terem essas alterações salvas com um commit.

Em paralelo, a **Área de Staging** representa a seção de mesmo nome no ciclo de vida de um arquivo versionado: a seção **Stage**.

Dessa área, caso seja decidido que ainda há alterações nos documentos a serem feitas, podemos voltá-los para a **Working Directory**. Ou, enviá-los para a **Área de Repositório**, fazendo o commit das alterações.

## Interação entre as Áreas de Trabalho do Git

Conforme vimos na seção anterior, o Git possui 3 áreas principais de trabalho: **Área de Repositório**, **Working Directory** e **Área de Staging**. Vamos verificar como uma área interage com a outra.

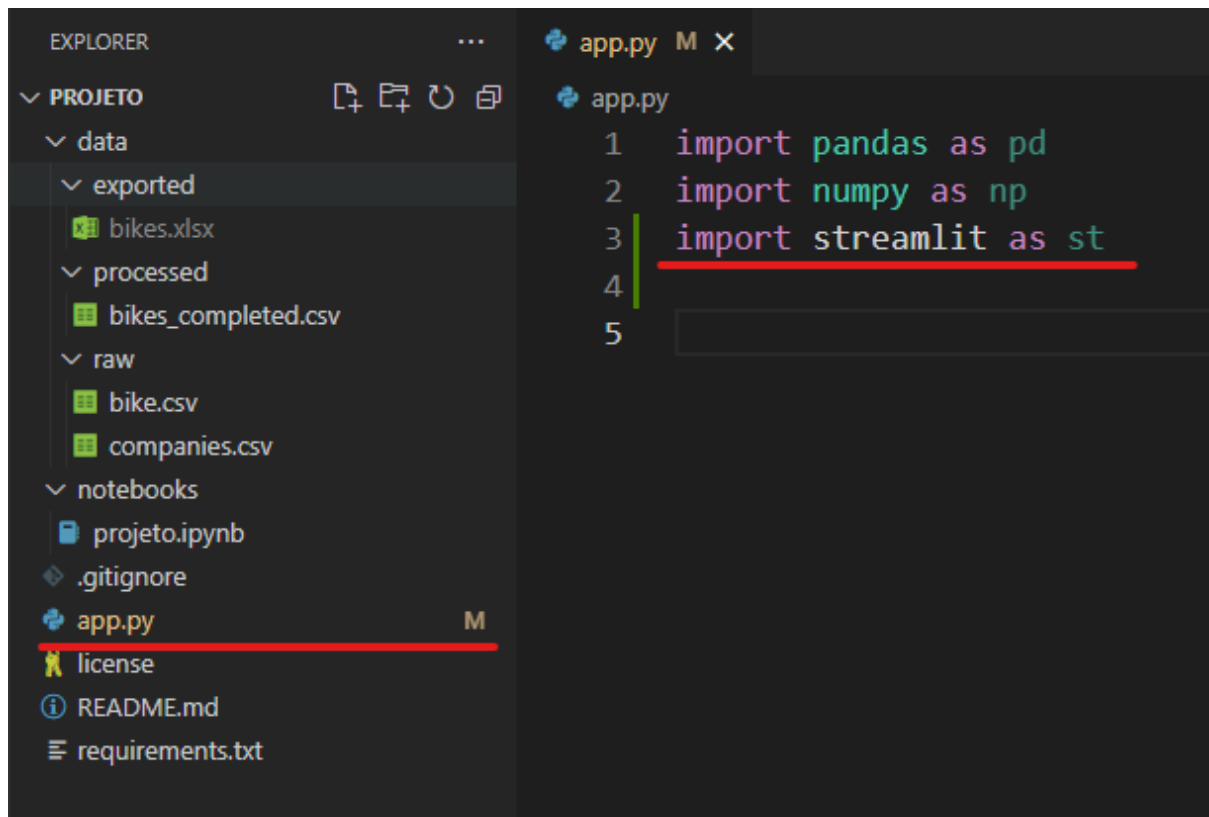
Antes de fazermos qualquer coisa, vamos utilizar o comando **git status** para verificarmos como está o nosso repositório do Git:

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

Como podemos observar, não temos nenhuma alteração feita e não estamos trabalhando com nenhum de nossos arquivos. Para começar, primeiro vamos

instalar o `Streamlit` dentro do nosso ambiente virtual do `Anaconda`. O processo é o mesmo que foi feito para a instalação da biblioteca `Pandas` na aula 5.

Com a biblioteca do `Streamlit` instalado, vamos adicioná-la dentro do arquivo `app.py`:



```
EXPLORER
PROJETO
  data
    exported
      bikes.xlsx
    processed
      bikes_completed.csv
    raw
      bike.csv
      companies.csv
  notebooks
    projeto.ipynb
  .gitignore
  app.py
  license
  README.md
  requirements.txt

app.py
1 import pandas as pd
2 import numpy as np
3 import streamlit as st
4
5
```

Feito isso, se utilizarmos o comando `git status` novamente, veremos que o arquivo `app.py` está marcado como arquivo modificado.

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

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

Ou seja, nesse momento, o arquivo `app.py` está dentro da `Working Directory` do Git. Podemos observar que, além disso, o Git nos dá duas opções: um comando para adicionar o arquivo para a `Área de Staging`, e um outro comando para voltar o arquivo para a `Área de Repositório`:

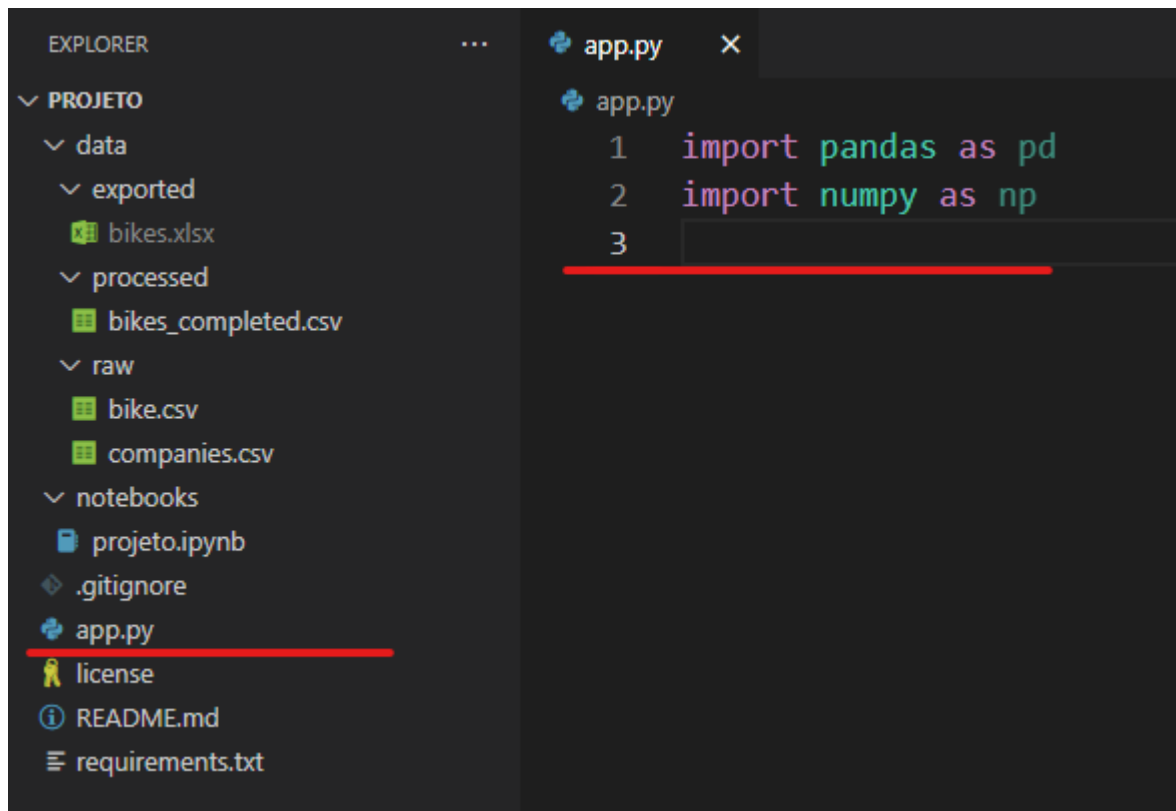
```
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   app.py

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

Portanto, se desejarmos descartar as alterações feitas no arquivo, utilizamos o comando `git restore <nome_arquivo>`. Se desejamos marcar as modificações como prontas, utilizamos o comando `git add <nome_arquivo>`. Vamos desfazer as alterações feitas e retornar o arquivo para a `Área de Repositório`, utilizando o comando `git restore app.py`:

```
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git restore app.py
```

Embora não tenhamos tido nenhum retorno se a operação foi realizada com sucesso, podemos observar no arquivo que as linhas adicionadas foram removidas, e que o arquivo voltou “ao normal” dentro da área de exploração de arquivos da IDE VS Code:



Ou seja, se utilizamos o comando `git restore <nome_arquivo>`, podemos desfazer as alterações feitas em um arquivo que esteja com elas marcadas como prontas. Além disso, esse comando faz com que o arquivo alterado, e que estava inicialmente dentro da **Working Directory** do Git, volte para a **Área de Repositório**, que é onde estão os arquivos versionados em suas últimas versões.

Vamos retornar as linhas removidas e adicionar o arquivo para a **Área de Staging**, utilizando o comando `git add app.py`:

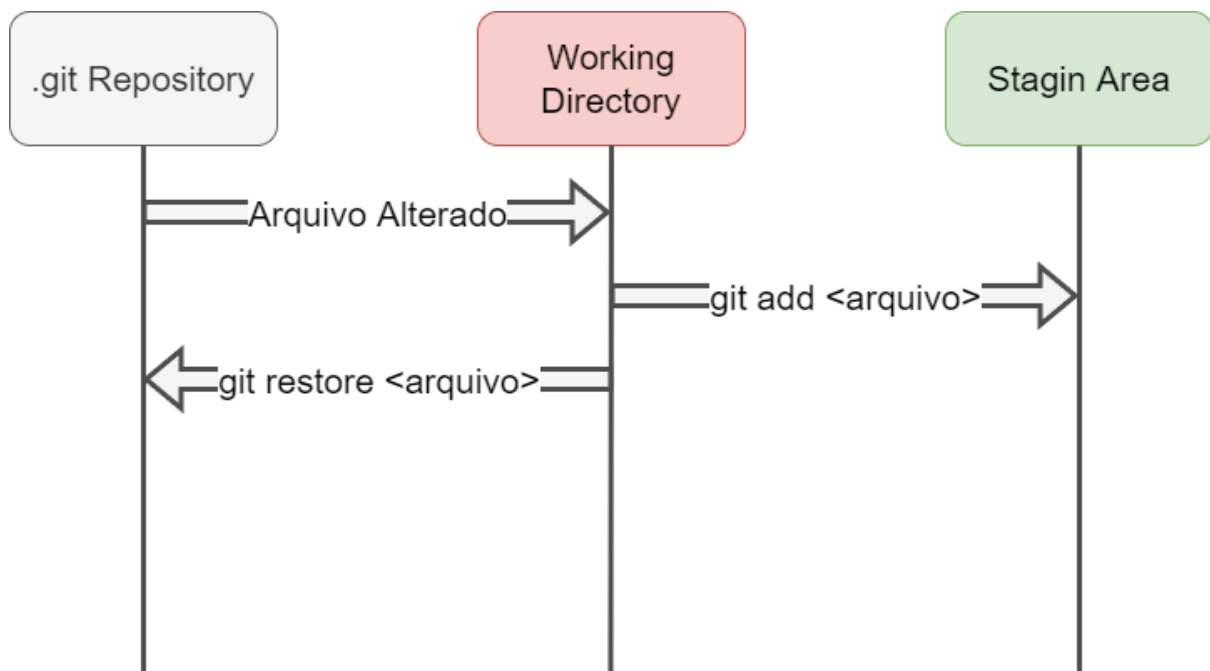
```
EXPLORER
PROJETO
├── data
│   ├── exported
│   │   └── bikes.xlsx
│   ├── processed
│   │   └── bikes_completed.csv
│   └── raw
│       ├── bike.csv
│       └── companies.csv
├── notebooks
│   ├── projeto.ipynb
│   ├── .gitignore
│   └── app.py
├── license
├── README.md
└── requirements.txt

app.py
1 import pandas as pd
2 import numpy as np
3 import streamlit as st
4

TERMINAL
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git add app.py

Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   app.py
```

Com esses comandos, transitamos entre as áreas de trabalho do Git com o arquivo `app.py`. Ou seja, quando fizemos uma modificação neste arquivo, ele foi retirado da **Área de Repositório (Repository)** e ele foi posto na **Working Directory**.

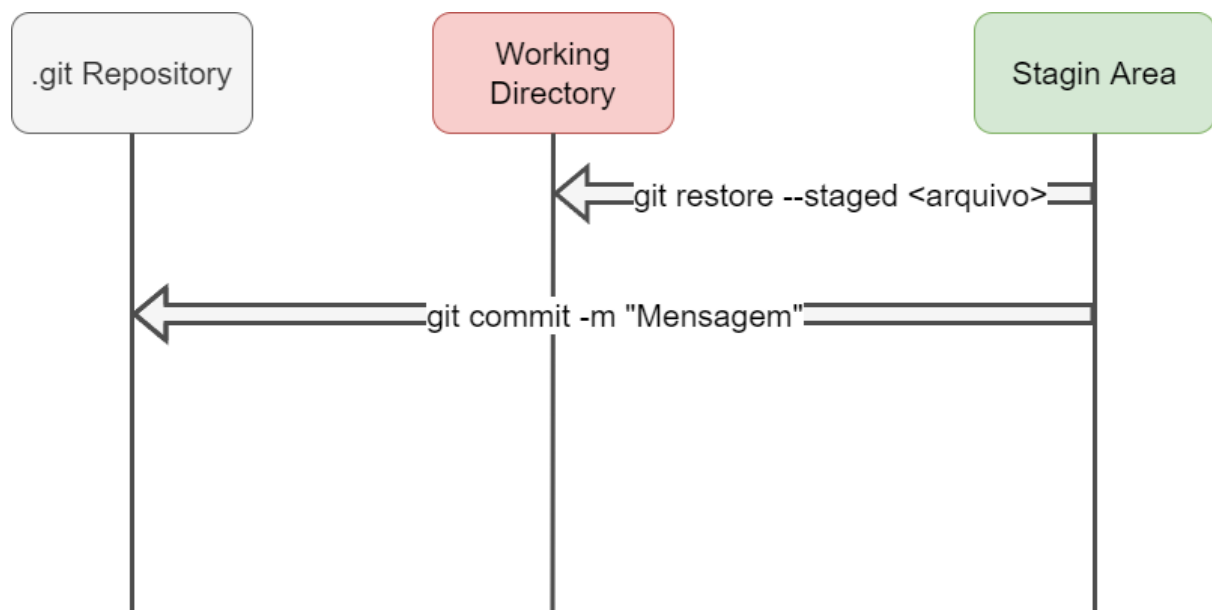


O arquivo estando dentro da **Área de Staging**, podemos utilizar o comando `git restore --staged <nome_arquivo>` para remove-lo da **Área de**

**Staging** e retorná-lo para a **Working Directory**, conforme pode ser visto na saída do comando `git status`:

```
Pedro@Ryzen MINGW64 ~/Documents/repos/curso_git_prep/projeto (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   app.py
```

Ou podemos usar o comando `git commit`, colocando uma mensagem de identificação com o parâmetro `-m` para salvar as alterações no repositório e enviar o arquivo novamente para a **Área de Repositório**.



Feito o commit, o arquivo retorna para a **Área de Repositório**, que é a área onde ficam os arquivos que estão versionados e sem alterações.

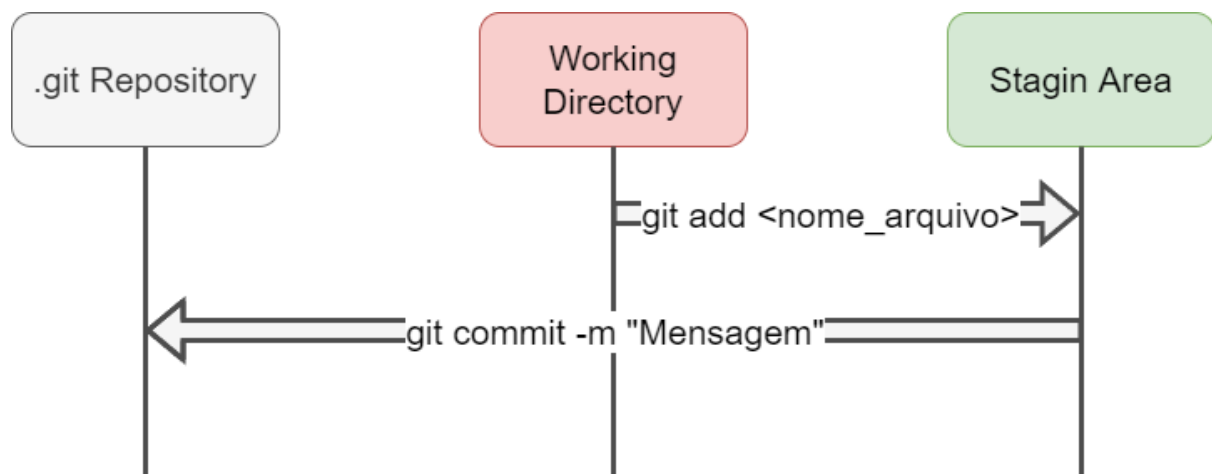
Dessa forma, o fluxo padrão de trabalho do Git, utilizando as três áreas de trabalho pode ser dividido de duas formas: **Arquivos Novos** e **Arquivos Já Versionados**.



## Arquivos Novos

Quando trabalhamos com novos arquivos, ou seja, arquivos que ainda não estão devidamente versionados, quando eles são inseridos dentro do projeto, temos o seguinte fluxo:

1. O arquivo adicionado entra na **Working Directory**
2. O arquivo é adicionado na **Área de Staging** com o comando `git add <nome_arquivo>`
3. O arquivo é adicionado na **Área de Repositório** com o comando `git commit -m "Mensagem"`

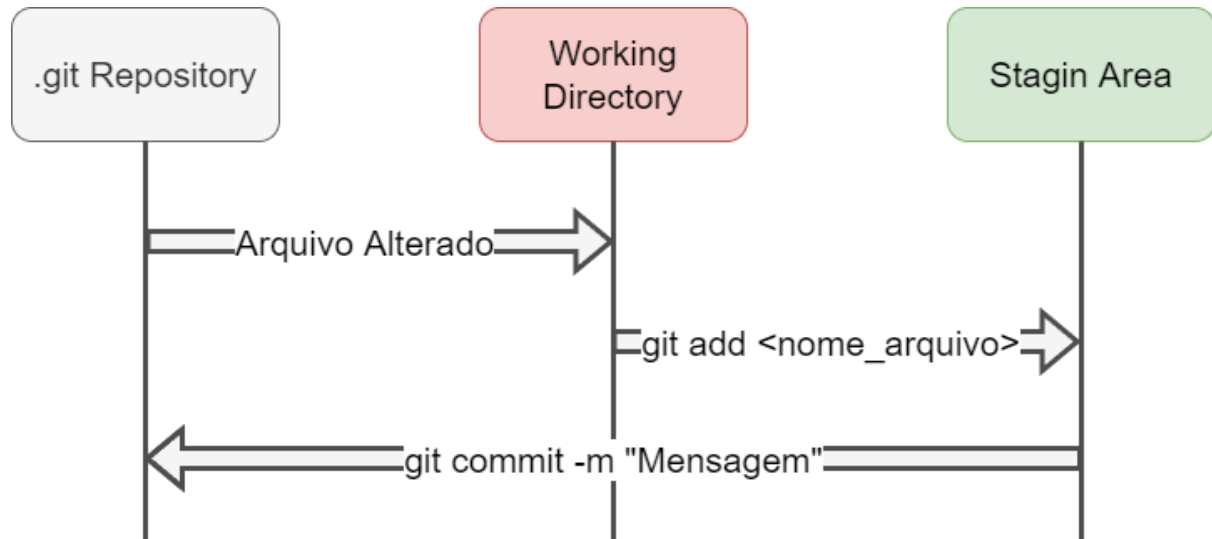


## Arquivos Já Versionados

Quando trabalhamos com arquivos que já são versionados, o fluxo de trabalho é o seguinte:

1. O arquivo que está dentro da área de **Área de Repositório** é alterado e vai para a **Working Directory**
2. O arquivo é marcado como pronto para ser commitado com o comando `git add <nome_arquivo>` e vai para a **Área de Staging**

3. O arquivo é retornado para a **Área de Repositório** novamente através do comando `git commit -m "Mensagem"`, tendo as suas alterações salvas



## Exercícios

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

Link para os exercícios de revisão de ciclo: [Revisão de Ciclo](#)

## Próxima Aula

Esta é a última aula do Ciclo 02, que tem a função de demonstrar os Fundamentos do Git, que serão utilizados ao longo do curso. A próxima aula, que abre o ciclo 03 - Trabalhando com Repositórios Locais, tem como objetivo fazer uma recapitulação dos principais comandos e conceitos que vimos neste ciclo.

# Fontes e Links Complementares

[Livro Pro Git - Capítulo 1.3 - O que é o Git](#)

[Livro Pro Git - Capítulo 2.2 - Salvando alterações no repositório](#)