

Relatório 26 - Git e Github para Iniciantes

Guilherme Loan Schneider

Descrição da atividade

1. Entendendo o que é Git e GitHub

Para que seja possível entender o que é o Git e GitHub, é importante comentar sobre a diferença entre eles. O primeiro é a ferramenta local que fará o controle das versões de arquivos que você está alterando, tendo o papel de controlador total dos commits, stages, status, dentre outros, do seu projeto. Já o segundo, é a plataforma online onde os seus arquivos de código podem se tornar públicos e serem salvos de forma remota; é basicamente a plataforma onde o Git opera.

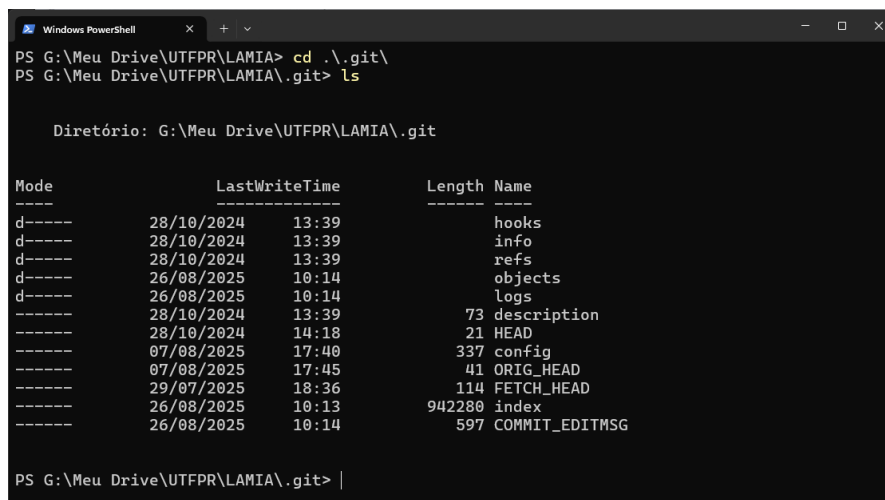
É interessante citar também o motivo da criação desse sistema open-source para versionamento de projetos, que se deu por conta de uma empresa gerenciadora de arquivos do Linux que começou a exigir o pagamento do seu serviço. Linus Torvalds então se sentiu na obrigação de criar um novo, dado que esse gerenciado pela empresa contratada era muito ruim e com baixa qualidade, surge então o Git como ferramenta de versionamento de arquivos.

2. Configurando o Git

Esse tópico foi bem rápido, só mostrando a instalação e configuração inicial do Git, que consistia em inserir um nome de usuário e o e-mail de uso no Git, juntamente com a escolha do editor de texto que o Git usará.

3. Essencial do Git

Aqui começamos com a inicialização de um repositório local com o Git, que consistiu em criar uma pasta e utilizar o comando “git init”. Esse comando fará a criação dos arquivos base que o Git usará para efetuar o controle dos arquivos. Na imagem abaixo, conseguimos verificar os arquivos que são criados com o Git (note que esse repositório é o meu atual do Bootcamp LAMIA, por isso existem mais arquivos aqui), são eles: “HEAD”, “branches”, “config”, “description”, “hooks”, “info”, “objects” e “refs”.



```
Windows PowerShell
PS G:\Meu Drive\UTFPR\LAMIA> cd .\git\
PS G:\Meu Drive\UTFPR\LAMIA\git> ls

Diretório: G:\Meu Drive\UTFPR\LAMIA\git

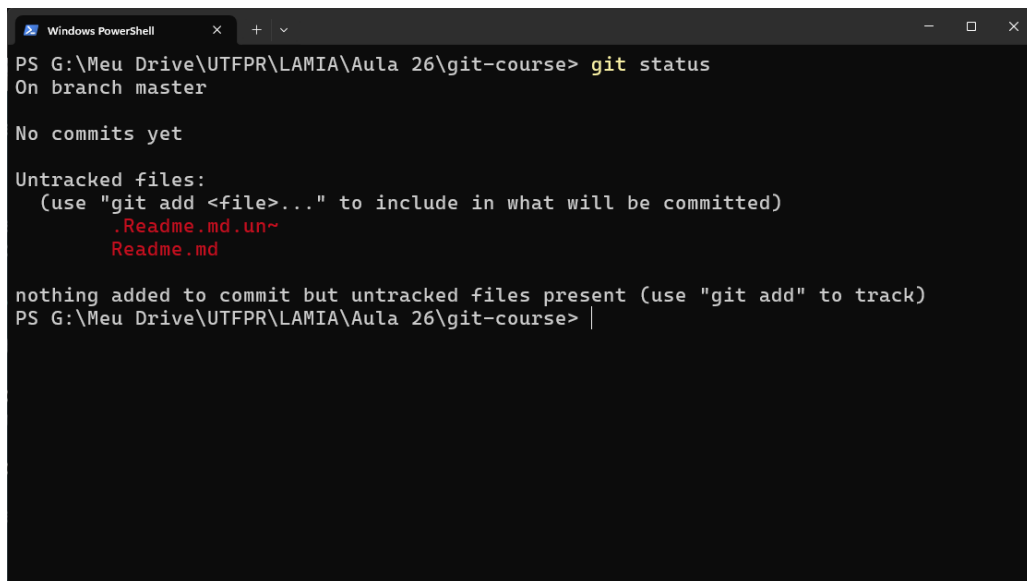
Mode                LastWriteTime         Length Name
----                -
d-----          28/10/2024   13:39             hooks
d-----          28/10/2024   13:39             info
d-----          28/10/2024   13:39             refs
d-----          26/08/2025   10:14           objects
d-----          26/08/2025   10:14             logs
-----          28/10/2024   13:39             73 description
-----          28/10/2024   14:18             21 HEAD
-----          07/08/2025   17:40           337 config
-----          07/08/2025   17:45             41 ORIG_HEAD
-----          29/07/2025   18:36           114 FETCH_HEAD
-----          26/08/2025   10:13        942280 index
-----          26/08/2025   10:14           597 COMMIT_EDITMSG

PS G:\Meu Drive\UTFPR\LAMIA\git> |
```

Nós temos quatro estágios de arquivos no Git, sendo eles:

- Untracked – Arquivos que existem no diretório de trabalho, mas o Git ainda não está rastreando (não fazem parte do versionamento).
- Unmodified – Arquivos que já estão sendo rastreados pelo Git e não sofreram alterações desde o último commit.
- Modified – Arquivos rastreados que foram alterados no diretório de trabalho, mas ainda não foram adicionados para o próximo commit.
- Staged - Arquivos que tiveram suas alterações adicionadas à área de preparação (via git add) e estão prontos para serem incluídos no próximo commit.

Na imagem abaixo conseguimos analisar o estágio “Untracked”, a partir do comando “git status”, sobre o arquivo Readme adicionado.



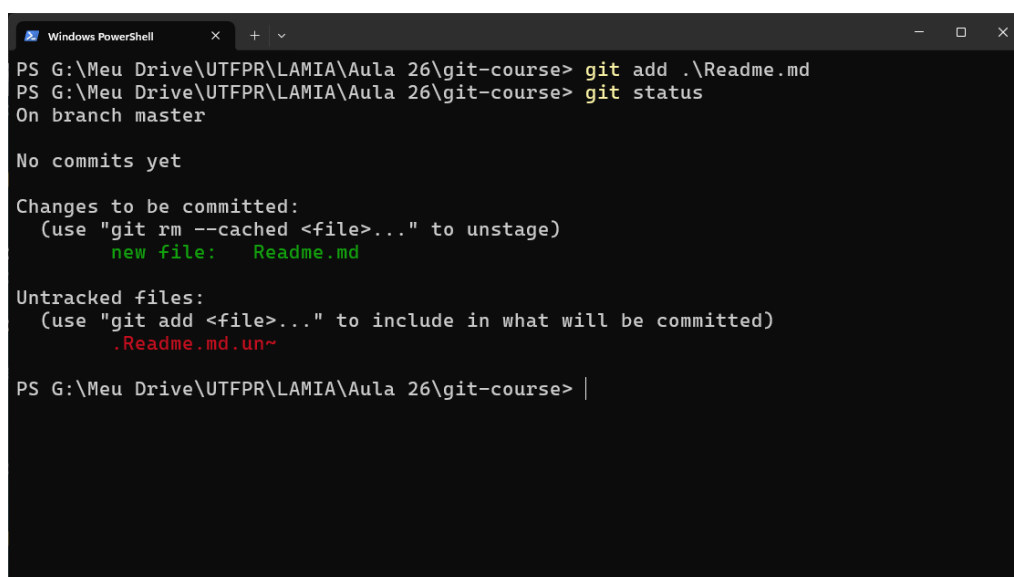
```
Windows PowerShell
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> |
```

Já na imagem abaixo, com o comando “git add <file>”, conseguimos perceber que o arquivo já passou para “Staged”. Caso seja feita alguma alteração no arquivo Readme, o Git automaticamente passará o arquivo para “Modified”.



```
Windows PowerShell
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> git add .\Readme.md
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Readme.md

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

PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> |
```

Na imagem abaixo, após alterar o arquivo Readme, verificamos que está sendo mostrado como “Modified”.


```
Windows PowerShell
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> git log
commit a8d26c07d7fd2158dc8fa8de007ed19754d206c3 (HEAD -> master)
Author: Guilherme Schneider <guilhermeloa@alunos.utfpr.edu.br>
Date:   Fri Aug 29 12:13:33 2025 -0300

    Add readme.md
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26\git-course> |
```

Abaixo está outro exemplo do meu repositório do LAMIA, contendo mais commits realizados por mim, onde conseguimos ver o nome do autor, e-mail registrado no Git, data e a mensagem de commit.

```
Windows PowerShell
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26> git log
commit d76930935167eef8ca0f197e654fb64bb61ff319 (HEAD -> main, origin/main)
Author: Guilherme Schneider <guilhermeloa@alunos.utfpr.edu.br>
Date:   Tue Aug 26 10:13:21 2025 -0300

    Correção de um erro no arquivo SHAP_aula + adição de comentários explicativos

commit 37356f7b2271d49c5a055d2c0c305c9755b71fc5
Author: Guilherme Schneider <guilhermeloa@alunos.utfpr.edu.br>
Date:   Mon Aug 25 12:10:40 2025 -0300

    Adição dos arquivos da Aula 25 e remoção de arquivo temporário

commit 3de663fe9f7338a70158fbc4ff1fdca574f1780e
Author: Guilherme Schneider <guilhermeloa@alunos.utfpr.edu.br>
Date:   Sat Aug 23 19:08:37 2025 -0300

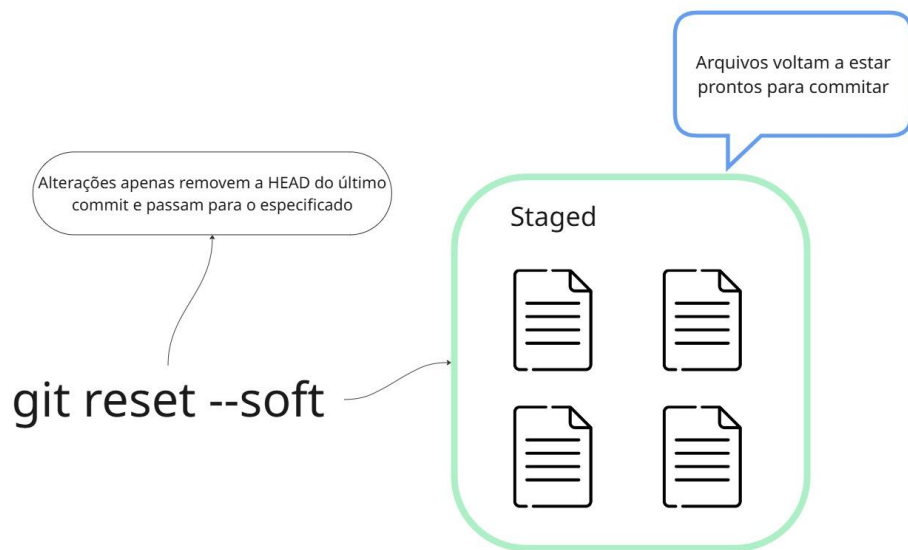
    Adicionando o relatório da Aula 23, juntamente com os arquivos da Aula 24 de NL
    P + Relatório

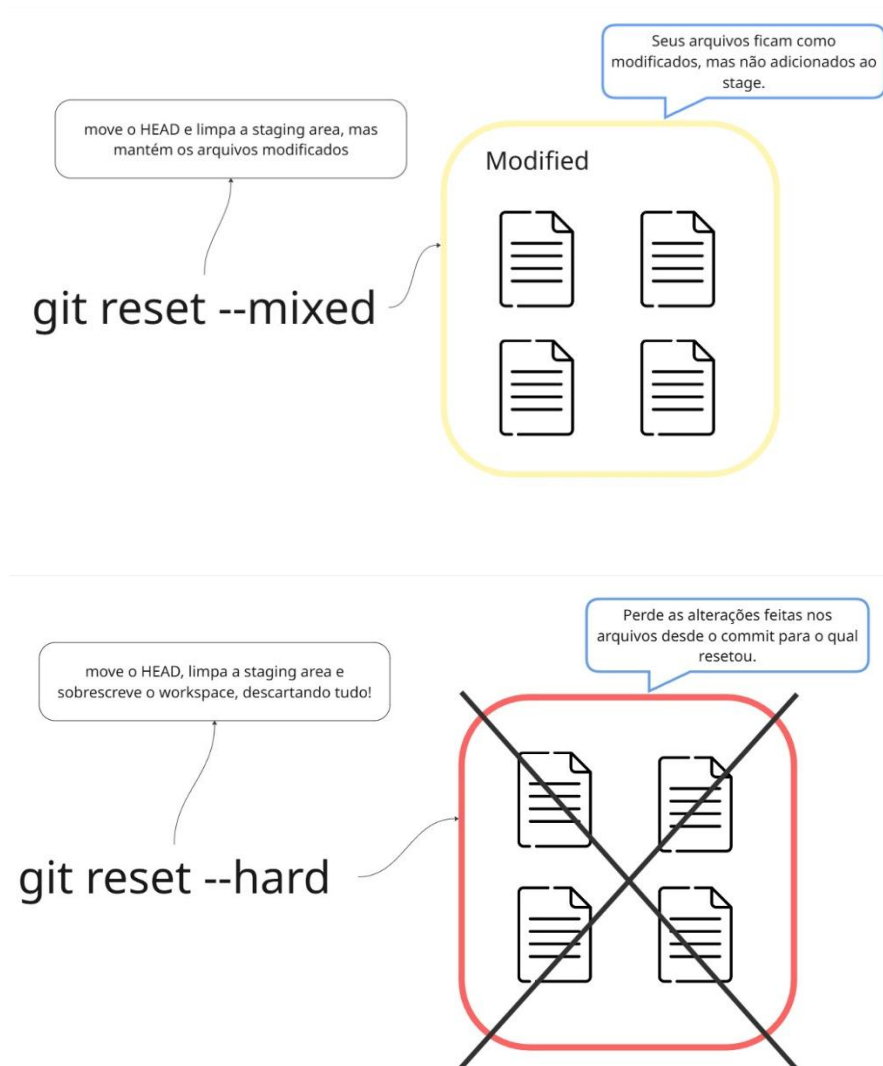
commit 63a1b40db537cb48a590df93165415af5ea30e70
Author: Guilherme Schneider <guilhermeloa@alunos.utfpr.edu.br>
:...skipping...
```

Além disso, é possível verificar também as diferenças dos arquivos a serem commitados, a partir do comando “git diff”. Na imagem abaixo, podemos ver que existe um arquivo que está modificado, mas não foi commitado ainda.

```
Windows PowerShell
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26> git diff
diff --git a/Aula 11/arquivos aula/airflow-materials b/Aula 11/arquivos aula/airflow-materials
--- a/Aula 11/arquivos aula/airflow-materials
+++ b/Aula 11/arquivos aula/airflow-materials
@@ -1,1 @@
-Subproject commit 23d9221c9f68d91a12795cd4c9b1880ca4deb978
+Subproject commit 23d9221c9f68d91a12795cd4c9b1880ca4deb978-dirty
PS G:\Meu Drive\UTFPR\LAMIA\Aula 26> |
```

Por fim, temos três principais estágios de reset que podem ser feitos, o `--soft`, `--mixed` e `--hard`. Cada um está descrito nas figuras abaixo.





4. Repositórios Remotos

Aqui começamos a trabalhar com o GitHub, onde o primeiro passo foi criar um repositório e gerar uma chave de criptografia SSH.

Essa chave SSH funcionará como um identificador único de quem estiver mexendo nos arquivos do repositório, chamado de “fingerprint” do usuário. Note que para alterar os arquivos de um repositório, é obrigatório a utilização dessa chave SSH.

Em seguida nos conectamos ao repositório a partir dos comandos que o próprio GitHub nos retorna para efetuar a conexão. No momento que nos conectamos, podemos realizar um commit inicial para verificar se a conexão foi feita corretamente.

Não vou entrar muito em detalhes nos comandos do GitHub dado que é algo que já tenho um pouco de familiaridade, e já estou utilizando o GitHub para os projetos que estou realizando no LAMIA.

Clonando repositórios

O GitHub permite que clonemos repositórios para gerar um estado idêntico localmente. Isso é interessante quando queremos utilizar um código de alguém para

realizar adaptações, etc, pegando o estado atual do projeto. Note que clonar o repositório copia todos os arquivos para a sua máquina.

Apenas citando um exemplo que precisei utilizar o “git clone”, que foi quando eu formatei o meu computador, onde precisei clonar o meu repositório do LAMIA dado que perdi esses arquivos. Os arquivos voltaram exatamente como os deixei e a conexão com o GitHub + Git foi praticamente instantânea para continuar os cards.

Utilizando o Fork em projetos

Esse caso funciona como um “git clone”, mas possui uma aplicação diferente, onde é mais voltado para quando existe algum erro ou melhoria que você percebeu em um código alheio e deseja sugerir a correção para o autor do repositório. Cabe ao autor acatar a sua correção ou não. É importante salientar que isso não é possível de ser feito com um repositório próprio.

5. Ramificação (Branch)

Nesse ponto começamos a trabalhar com ramificação, que é um dos benefícios do Git. Um branch pode ser entendido como uma “linha do tempo” de desenvolvimento dentro do repositório, que parte de um ponto específico e permite criar novas funcionalidades, correções ou experimentações sem interferir diretamente no código principal.

O uso de branches é fundamental para organizar o fluxo de trabalho em equipe, já que possibilita que cada desenvolvedor trabalhe de forma isolada, integrando suas mudanças ao projeto somente quando estiverem prontas.

Primeiramente, foi apresentada a definição do que é um branch e por que utilizá-lo, destacando que o Git, por padrão, cria o branch principal chamado main ou master. É a partir dele que outros branches podem ser criados para novas features ou correções. Em seguida, podemos criar um novo branch através do comando “git branch” e também como navegar entre branches utilizando “git checkout”. O autor também abordou como excluir branches que não são mais necessários, seja de forma segura (git branch -d <nome>) ou forçada (git branch -D <nome>).

Outro ponto importante foi entender o processo de merge, que é a forma de unir o histórico de commits de um branch ao outro. No merge, o Git cria um commit especial que junta as duas linhas de desenvolvimento, preservando os históricos de ambos. Também foi apresentado o conceito de rebase, que é uma alternativa ao merge. O rebase reorganiza o histórico, “reposicionando” os commits de um branch em cima do outro, melhorando a visualização do histórico de commits, mas que exige mais atenção em projetos maiores ou com desenvolvimento de várias pessoas.

Na prática, vimos as diferenças entre utilizar merge e rebase. O merge preserva a árvore de commits, enquanto o rebase reescreve a linha do tempo para que pareça que os commits foram feitos de forma sequencial, sem ramificações. Ambos têm suas vantagens, e a escolha depende da estratégia de versionamento da equipe.

Referencias

[Git e GitHub para iniciantes \(Seção 1 a 5\)](#)

[Como usar o Github Desktop](#)