

Relatório LAMIA 26

Prática: Git e Github para Iniciantes (III)

Kaique Medeiros Lima

1 Introdução

Esse card fornece uma visão geral sobre o uso do Git, abordando desde a inicialização de repositórios, até branches e repositórios remotos. O Git é um sistema de controle de versão distribuído e o GitHub facilita o armazenamento e colaboração em projetos. Muitíssimo importante na área de desenvolvimento, pois é a maneira a qual os desenvolvedores compartilham e colaboram em projetos.

2 Descrição da atividade

2.1 Entendendo o que é Git e GitHub

2.1.1 Introdução

O objetivo deste curso é compreender o conceito de controle de versão, aprender a utilizar os principais comandos do Git e entender como publicar repositórios no GitHub. Com isso, será possível gerenciar projetos de forma eficiente, acompanhando mudanças no código e colaborando com outros desenvolvedores.

2.1.2 Controle de versão

O controle de versão é um sistema que permite registrar e gerenciar modificações em um projeto ao longo do tempo. Com ele, é possível restaurar versões anteriores, comparar mudanças e trabalhar de forma colaborativa sem risco de perder progresso. Entre as diversas ferramentas disponíveis, o Git se destaca como a mais popular e eficiente.

2.1.3 História do Git

Inicialmente, o desenvolvimento do kernel do Linux utilizava um sistema de controle de versão chamado BitKeeper. No entanto, uma disputa entre a empresa responsável pelo BitKeeper e a Linux Foundation resultou na perda do acesso gratuito à ferramenta. Para evitar custos e manter a independência do projeto, Linus Torvalds, criador do Linux, desenvolveu o Git, um sistema de controle de versão distribuído, inspirado no BitKeeper, mas com diversas melhorias, especialmente em desempenho e segurança.

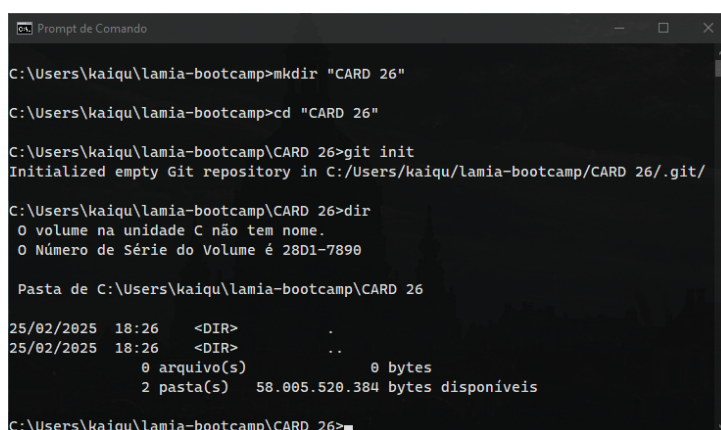
2.1.4 O que é GitHub

O GitHub é uma plataforma online que permite armazenar e compartilhar projetos versionados com Git. Ele oferece funcionalidades para colaboração entre desenvolvedores, como controle de permissões, integração com outras ferramentas e um ambiente para revisão de código. Os repositórios podem ser públicos, permitindo que qualquer pessoa visualize o código, ou privados, garantindo acesso restrito a determinados usuários.

2.2 Essencial do Git

2.2.1 Inicializando um repositório

Para começar a usar o Git em um projeto, é necessário inicializar um repositório. Sendo feito com o comando `git init`.



```
Prompt de Comando

C:\Users\kaiqu\lamia-bootcamp>mkdir "CARD 26"

C:\Users\kaiqu\lamia-bootcamp>cd "CARD 26"

C:\Users\kaiqu\lamia-bootcamp\CARD 26>git init
Initialized empty Git repository in C:/Users/kaiqu/lamia-bootcamp/CARD 26/.git/

C:\Users\kaiqu\lamia-bootcamp\CARD 26>dir
O volume na unidade C não tem nome.
O Número de Série do Volume é 28D1-7890

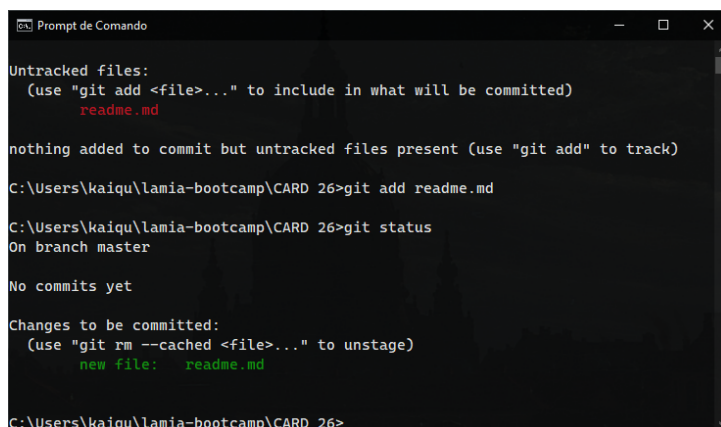
Pasta de C:\Users\kaiqu\lamia-bootcamp\CARD 26

25/02/2025  18:26    <DIR>        .
25/02/2025  18:26    <DIR>        ..
               0 arquivo(s)            0 bytes
               2 pasta(s)  58.005.520.384 bytes disponíveis

C:\Users\kaiqu\lamia-bootcamp\CARD 26>
```

2.2.2 O ciclo de vida dos status de seus arquivos

O Git organiza os arquivos em quatro categorias principais: **untracked** (não marcado), **unmodified** (não modificado), **modified** (modificado) e **staged** (preparado para commit). Inicialmente, quando um arquivo é adicionado ao repositório e ainda não foi reconhecido pelo Git, ele é classificado como **untracked**. Após ser reconhecido, o arquivo passa para o status **unmodified**, indicando que não há alterações desde o último commit. Quando o arquivo sofre modificações, ele é classificado como **modified**. Se você quiser preparar o arquivo para ser salvo no próximo commit, deve adicioná-lo à área **staged** com o comando `git add`. Após o commit, o arquivo retorna ao estado **unmodified**, indicando que ele foi registrado no histórico do repositório e não há mais mudanças pendentes.



```
Prompt de Comando

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

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

C:\Users\kaiqu\lamia-bootcamp\CARD 26>git add readme.md

C:\Users\kaiqu\lamia-bootcamp\CARD 26>git status
On branch master

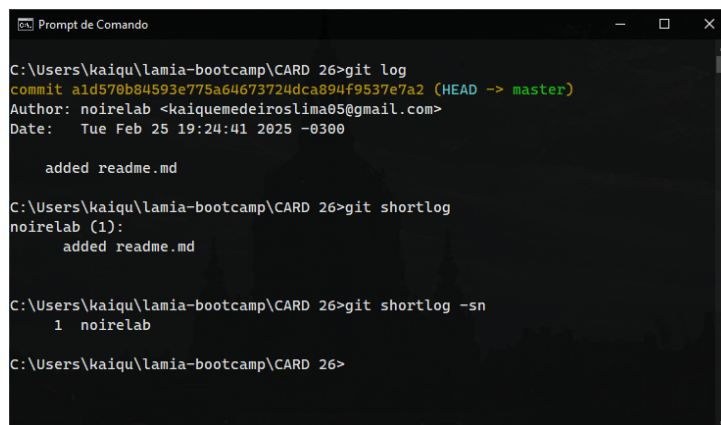
No commits yet

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

C:\Users\kaiqu\lamia-bootcamp\CARD 26>
```

2.2.3 Visualizando logs

O comando `git log` exibe informações detalhadas sobre os commits realizados em um repositório, como o ID do commit, o autor, a data e a descrição do que foi feito. Existem várias configurações adicionais que permitem personalizar a saída desse comando. Por exemplo, é possível filtrar os commits por autor utilizando a opção `-author=<nome>`, ou utilizar o `shortlog`, que exibe quantos commits cada autor fez e quais foram. A opção `shortlog -sn` mostra apenas o nome dos autores e a quantidade de commits realizados. A opção `-graph` exibe uma representação gráfica das branches e versões do repositório, facilitando a visualização das mudanças ao longo do tempo. Além disso, o comando `git show <id do commit>` fornece todas as informações detalhadas sobre um commit específico, permitindo uma análise mais profunda.



```
Prompt de Comando
C:\Users\kaiku\lamia-bootcamp\CARD 26>git log
commit a1d570b84593e775a64673724dca894f9537e7a2 (HEAD -> master)
Author: noirelab <kaikumedeiroslima05@gmail.com>
Date: Tue Feb 25 19:24:41 2025 -0300

    added readme.md

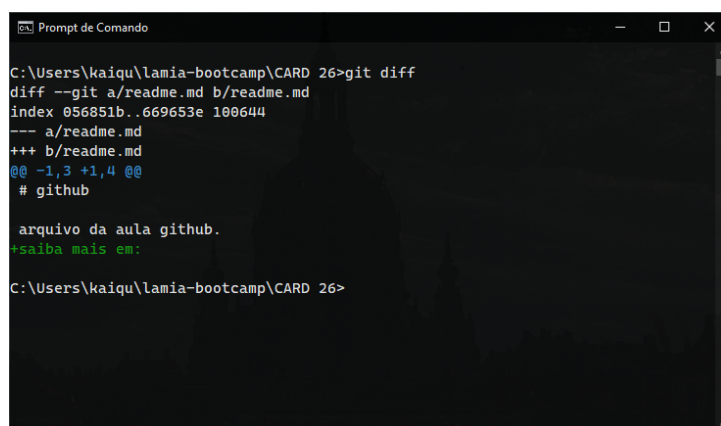
C:\Users\kaiku\lamia-bootcamp\CARD 26>git shortlog
noirelab (1):
    added readme.md

C:\Users\kaiku\lamia-bootcamp\CARD 26>git shortlog -sn
1 noirelab

C:\Users\kaiku\lamia-bootcamp\CARD 26>
```

2.2.4 Visualizando o diff

O comando `git diff` exibe as modificações feitas nos arquivos, mesmo que esses arquivos ainda não tenham sido commitados. Esse comando é útil para revisar as mudanças antes de realizar um commit, ajudando a evitar o envio de alterações indesejadas ou não finalizadas.



```
Prompt de Comando
C:\Users\kaiku\lamia-bootcamp\CARD 26>git diff
diff --git a/readme.md b/readme.md
index 056851b..669653e 100644
--- a/readme.md
+++ b/readme.md
@@ -1,3 +1,4 @@
 # github

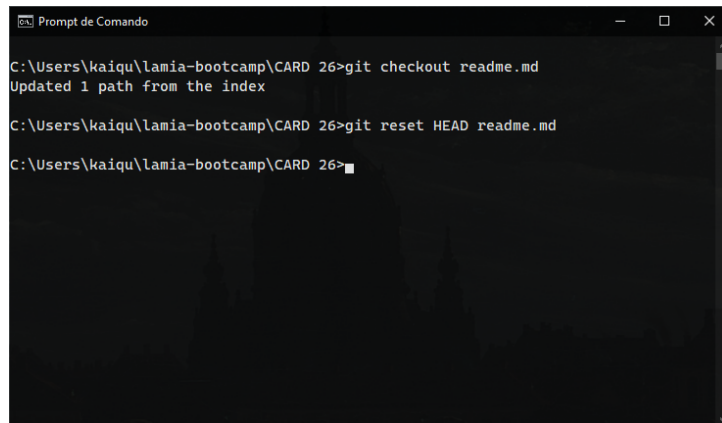
arquivo da aula github.
+saiba mais em:

C:\Users\kaiku\lamia-bootcamp\CARD 26>
```

2.2.5 Desfazendo coisas

Caso você tenha feito alterações em um arquivo e deseje revertê-las, é possível restaurar a versão anterior do arquivo utilizando o comando `git checkout <nome_arquivo>`. Esse comando descarta as mudanças locais e traz o arquivo de volta ao estado da última versão commitada. Se, após usar o comando `git add`, você decidir que não deseja mais incluir o

arquivo no próximo commit, pode removê-lo da área `staged` com o comando `git reset HEAD <nome_arquivo>`. Esse comando desfaz a adição do arquivo à área `staged`, mas mantém as alterações locais feitas no arquivo.



```
Prompt de Comando
C:\Users\kaiqu\lamia-bootcamp\CARD 26>git checkout readme.md
Updated 1 path from the index

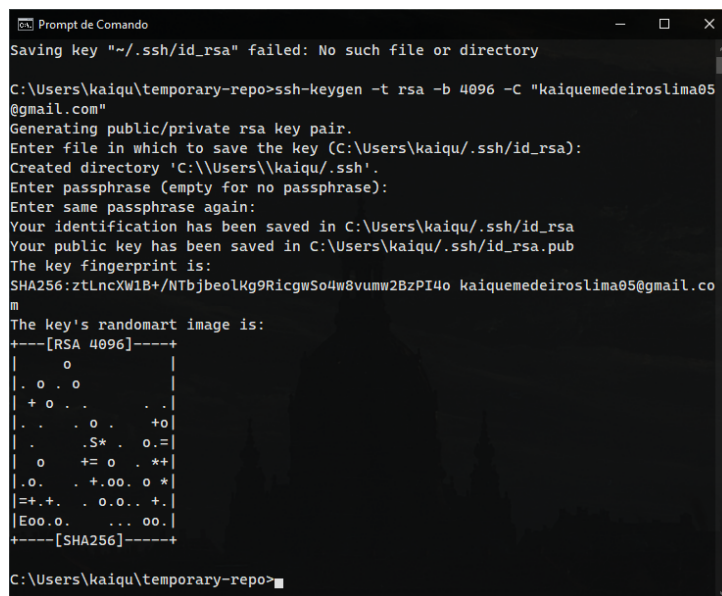
C:\Users\kaiqu\lamia-bootcamp\CARD 26>git reset HEAD readme.md

C:\Users\kaiqu\lamia-bootcamp\CARD 26>
```

2.3 Repositórios Remotos

2.3.1 Criando e Adicionando uma chave SSH

A chave SSH é uma forma de autenticação que estabelece uma conexão segura entre o repositório e o usuário, permitindo a comunicação direta entre a máquina e o GitHub. Ao gerar e adicionar uma chave SSH, pode-se enviar seus arquivos diretamente pela IDE sem precisar de autenticação manual a cada operação. Isso facilita o processo de push, pull e clone de repositórios no GitHub.



```
Prompt de Comando
Saving key "~/.ssh/id_rsa" failed: No such file or directory

C:\Users\kaiqu\temporary-repo>ssh-keygen -t rsa -b 4096 -C "kaiquedeiroslima05@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\kaiqu\.ssh/id_rsa):
Created directory 'C:\Users\kaiqu\.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\kaiqu\.ssh/id_rsa
Your public key has been saved in C:\Users\kaiqu\.ssh/id_rsa.pub
The key fingerprint is:
SHA256:ztLncXW1B+/NTbjbeolKg9RicgwSo4w8vumw2BzPI4o kaiquedeiroslima05@gmail.com
The key's randomart image is:
+---[RSA 4096]-----+
|  o  o  |
| . o . o |
| + o . . |
| . . . o . +o|
| . . .S* . o.=|
| o  += o . *+|
|.o. . +.oo. o *|
|=+.+. . o.o. +.|
|Eoo.o. . . oo.|
+---[SHA256]-----+

C:\Users\kaiqu\temporary-repo>
```

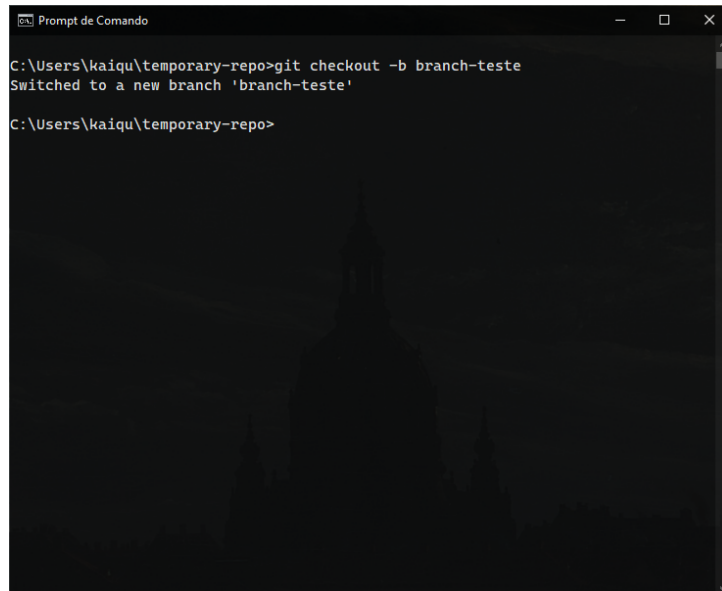
2.4 Ramificação (Branch)

2.4.1 O que é um branch?

Um **branch** é um ponteiro móvel que aponta para um commit específico em um repositório. Ele permite a criação de diferentes ramificações dentro do mesmo repositório. O uso de branches é fundamental para permitir que várias pessoas trabalhem simultaneamente em tarefas distintas, sem afetar o código principal.

2.4.2 Criando um branch

Para criar um novo branch no Git, basta utilizar o comando `git checkout -b <branch>`. Esse comando cria um branch com o nome especificado e, em seguida, muda automaticamente para ele.

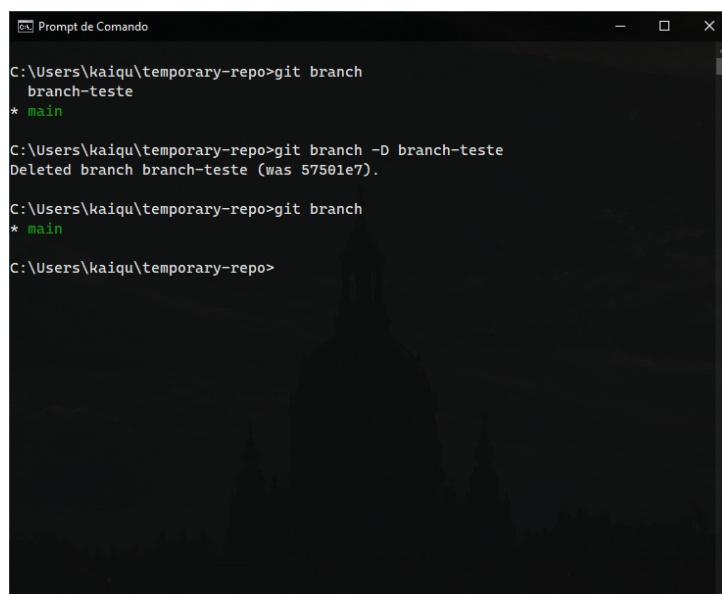
A screenshot of a Windows Command Prompt window titled "Prompt de Comando". The window has a black background with white text. The command prompt shows the following sequence of commands and output:

```
C:\Users\kaiku\temporary-repo>git checkout -b branch-teste
Switched to a new branch 'branch-teste'

C:\Users\kaiku\temporary-repo>
```

2.4.3 Movendo e deletando branches

Para navegar entre os branches no Git, basta usar o comando `git checkout <branch>`. Isso muda o branch atual para o especificado, permitindo que você trabalhe nas alterações daquele branch. Para deletar um branch, use o comando `git branch -D <branch>`, que remove o branch localmente. Esse comando deve ser utilizado com cautela, pois deletar um branch pode resultar na perda de alterações não comprometidas naquele branch.

A screenshot of a Windows Command Prompt window titled "Prompt de Comando". The window has a black background with white text. The command prompt shows the following sequence of commands and output:

```
C:\Users\kaiku\temporary-repo>git branch
  branch-teste
* main

C:\Users\kaiku\temporary-repo>git branch -D branch-teste
Deleted branch branch-teste (was 57501e7).

C:\Users\kaiku\temporary-repo>git branch
* main

C:\Users\kaiku\temporary-repo>
```

2.4.4 Entendendo o Merge

Quando é necessário unir dois branches, existem dois principais métodos: **merge** e **rebase**. Ambos têm o objetivo de integrar as alterações de um branch no outro, mas de formas diferentes. O **merge** cria um novo commit que une os dois branches, preservando o histórico de ambos. Esse método mantém a estrutura dos commits original, criando um "merge commit" que conecta as alterações feitas em cada branch.

2.4.5 Entendendo o Rebase

O **rebase**, por outro lado, altera a base de um branch, pegando todos os commits de um branch separado (como **feature**) e aplicando-os no topo de outro branch (geralmente o **master** ou **main**). Isso faz com que o histórico do branch seja reescrito, colocando os commits no início da fila, como se as alterações tivessem sido feitas a partir do último commit do branch principal. O rebase oferece um histórico mais linear, mas deve ser usado com cautela, especialmente em branches compartilhados, para evitar conflitos e reescrita de histórico de outros colaboradores.

2.5 Como usar o Github Desktop

O vídeo apresenta um tutorial sobre o uso do GitHub Desktop, explicando o processo de instalação, autenticação e clonagem de repositórios. Ele também mostra como criar um novo repositório, adicionar e modificar arquivos, e realizar commits com mensagens descritivas. Além disso, ensina a enviar mudanças para o repositório remoto usando o *push*. O tutorial menciona a função do arquivo **.gitignore** para excluir arquivos sensíveis do versionamento e destaca a importância de manter o repositório organizado para facilitar o trabalho colaborativo.

3 Conclusão

Concluindo, o uso do Git e GitHub é essencial para o gerenciamento eficaz de projetos de desenvolvimento de software. O Git permite rastrear e controlar alterações no código, enquanto o GitHub oferece uma plataforma colaborativa para o compartilhamento e controle de versões. Entendendo os conceitos básicos, como inicialização de repositórios, status de arquivos, branches e repositórios remotos.