

## **Material Didático - Git/GitHub**

### **TÓPICOS DO MATERIAL DIDÁTICO**

**Módulo 1: Introdução ao git**

**Módulo 2: Introdução ao github**

**Módulo 3: Criando um repositório**

**Módulo 4: Realizando os comandos**

**Módulo 5: Criando nova branch**

**Módulo 6: Pull Request**

**Módulo 7: Tags e Releases**

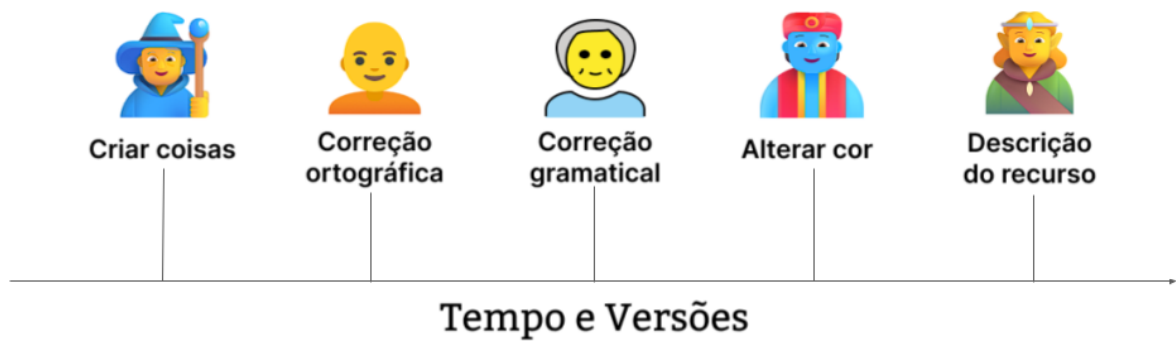
**Módulo 8: Issues**

## Tecnologias necessárias



- **Visual Studio Code:** para realizar os códigos.
- **Git:** para administração e versionamento dos códigos.
- **GitHub:** para armazenar os códigos de maneira colaborativa.
- **Github Desktop:** para efetuar comandos juntamente a uma interface gráfica.

## 1.1 O que é um Sistema de Controle de Versão?

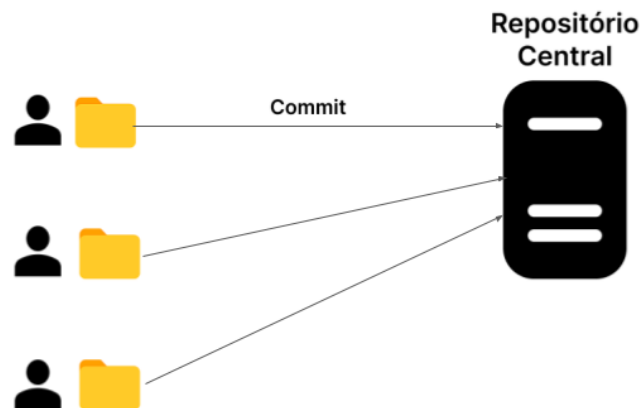


Primeiramente é importante saber o que é um Sistema de Controle de Versão, basicamente, os controladores de versão de código estão relacionados à capacidade de controlar e administrar qualquer tipo de alteração realizada em um arquivo ou conjunto de arquivos. Esse controle de versão permite você saber informações importantes como: “Houveram mudanças em um determinado arquivo?”, “Quem realizou essas mudanças?” e “Quando foram realizadas?”.

Sendo assim, para lidar com os problemas de controle de versões de um determinado arquivo, foram desenvolvidos os primeiros Sistemas de Controle de Versão Centralizado.

## 1.2 Arquiteturas de Sistemas de Controle de Versão: Centralizado

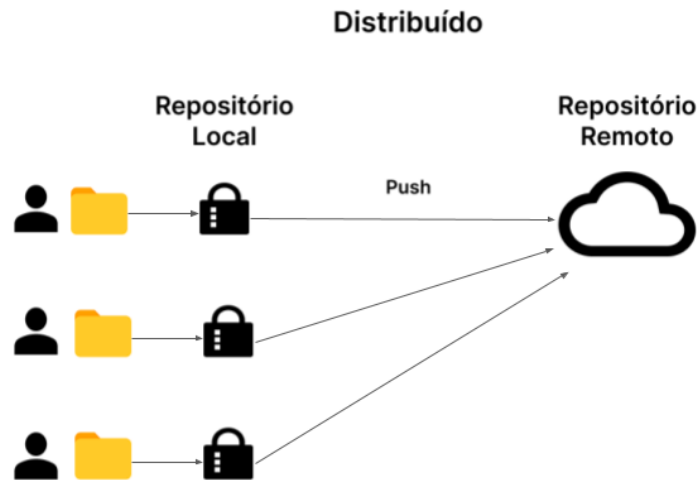
### Centralizado



Os primeiros Sistemas de Controle de Versão que surgiram foram os Sistemas de Controle de Versão Centralizado, nesses sistemas há um servidor único com todos os arquivos versionados, permitindo o acesso dos colaboradores envolvidos no projeto.

Porém, pelo fato de ser um servidor único, qualquer problema com o servidor pode comprometer o desenvolvimento do projeto, como por exemplo se o servidor cair, durante esse tempo ninguém poderá colaborar com o projeto ou realizar mudanças nos arquivos.

### 1.3 Arquiteturas de Sistemas de Controle de Versão: Distribuído



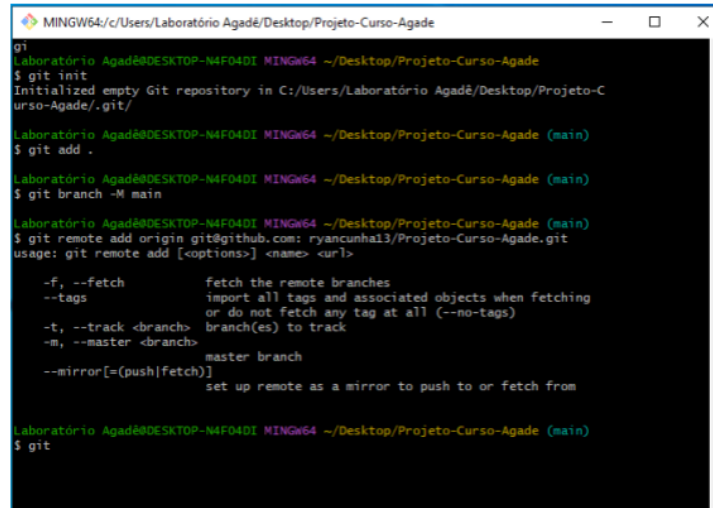
Surgiu então, os Sistemas de Controle de Versão Distribuídos, que diferente do centralizado, cada pessoa tem o seu repositório local, ou seja, o repositório em sua máquina, e após feitas as alterações, elas são enviadas para um repositório remoto, permitindo assim uma melhor segurança e controle de versão do projeto.

Assim, com esses sistemas os colaboradores clonam os repositórios remotos em suas máquinas locais, e com isso, qualquer problema com o servidor não afetaria o desenvolvimento do projeto.

#### **Vantagens dos Sistemas de Controle de Versão Distribuído:**

- Controle de histórico;
- Organização;
- Trabalho em equipe;
- Segurança;

### 1.3 O que é o Git?



```
g?
Laboratório Agadê@DESKTOP-N4F04DI MINGW64 ~/Desktop/Projeto-Curso-Agade
$ git init
Initialized empty Git repository in C:/Users/Laboratório Agadê/Desktop/Projeto-Curso-Agade/.git/

Laboratório Agadê@DESKTOP-N4F04DI MINGW64 ~/Desktop/Projeto-Curso-Agade (main)
$ git add .

Laboratório Agadê@DESKTOP-N4F04DI MINGW64 ~/Desktop/Projeto-Curso-Agade (main)
$ git branch -M main

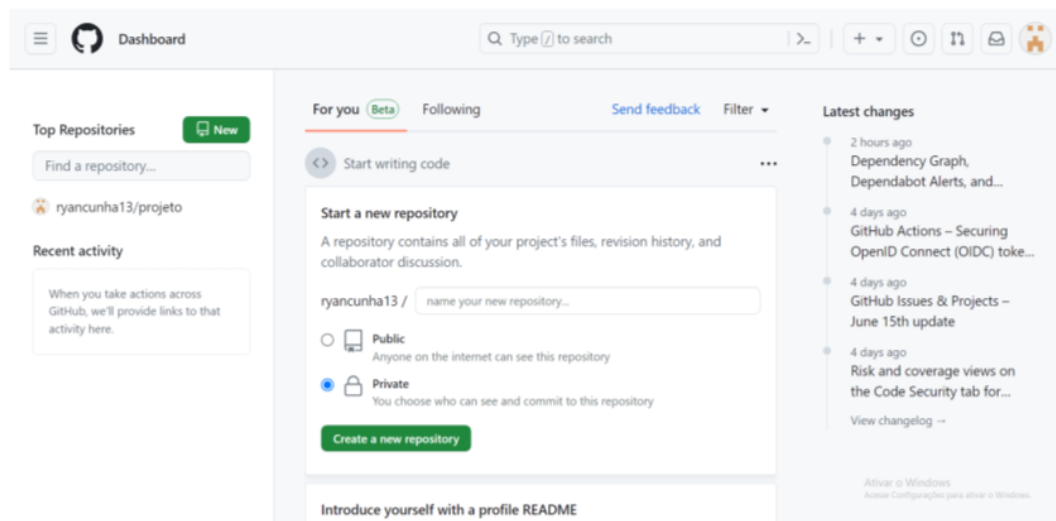
Laboratório Agadê@DESKTOP-N4F04DI MINGW64 ~/Desktop/Projeto-Curso-Agade (main)
$ git remote add origin git@github.com:ryaninha13/Projeto-Curso-Agade.git
usage: git remote add [<options>] <name> <url>

    -f, --fetch          fetch the remote branches
    --tags              import all tags and associated objects when fetching
                        or do not fetch any tag at all (--no-tags)
    -t, --track <branch> branch(es) to track
    -m, --master <branch> master branch
    --mirror[=(push|fetch)]
                        set up remote as a mirror to push to or fetch from

Laboratório Agadê@DESKTOP-N4F04DI MINGW64 ~/Desktop/Projeto-Curso-Agade (main)
$ git
```

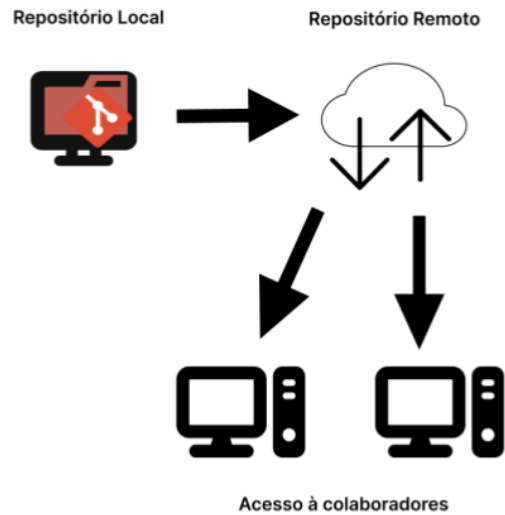
O git é um Sistema de Controle de Versão Distribuído, detentor de diversos comandos simples, que tornam a interação do usuário com a linha de comando bastante fluida ao longo da prática. Além disso, o git possibilita uma interação distribuída, ou seja, permite que pessoas de diferentes localizações geográficas façam alterações, inicialmente locais, nos seus computadores e, eventualmente, contribuam tais alterações de forma remota. Entre outras palavras, o ponto forte do git, além da múltipla interação entre diferentes usuários, é que ele permite visualizar todo o histórico do que foi feito e/ou alterado, e ter acesso às versões anteriores, possibilitando a comparação, o controle e consequentemente a evolução do projeto.

## 2.1 O que é o GitHub?



O github é uma ferramenta gratuita de hospedagem de repositórios git, ou seja, é o site onde ficará armazenado todos os códigos e arquivos enviados previamente utilizando os comandos do git, permitindo os usuários terem uma ideia visual daquilo que foi produzido, seja em termos de código, alterações, adições ou remoções. Ele é justamente um serviço baseado em nuvem que oferece essa colaboração e controle de versão do código desenvolvido. Para usufruir de tal ferramenta, basta criar uma conta no site oficial, e lá será possível visualizar todos os recursos e features disponíveis.

## 2.2 Para que serve um repositório compartilhado



### O que é um repositório?

Um repositório é onde estão todos os arquivos de um projeto e o histórico de revisão de cada arquivo desse projeto.

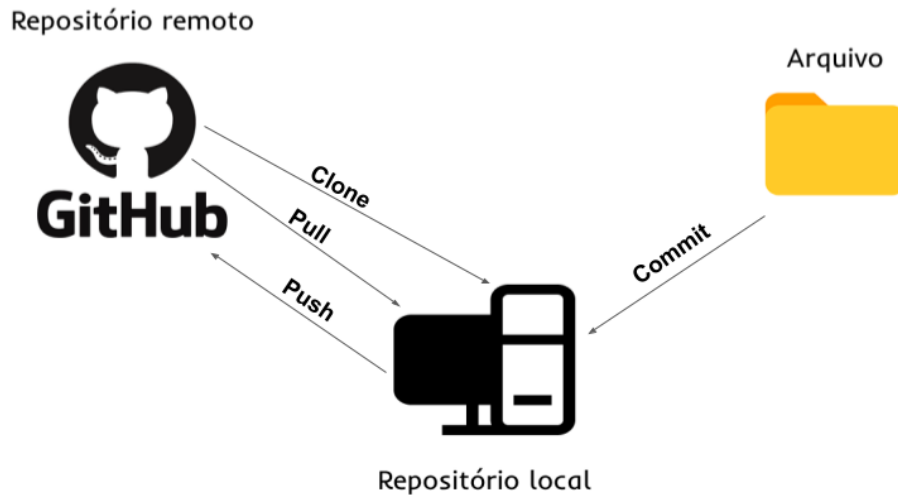
### Repositório Local vs Remoto

Um repositório local é o repositório que está armazenado em sua máquina. Enquanto que um repositório remoto está armazenado em um ambiente virtual, assim, facilitando o compartilhamento do mesmo e possibilitando outras pessoas a terem acesso ao repositório.

Sendo assim, um repositório compartilhado, tem como finalidade permitir que múltiplos usuários compartilhem ou peguem informações de um determinado repositório. A maneira como isso será realizada será definida pelo time e/ou empresa. Isso permite um maior dinamismo, controle e organização no ambiente de desenvolvimento.



### 2.3 Funcionamento do Git/GitHub

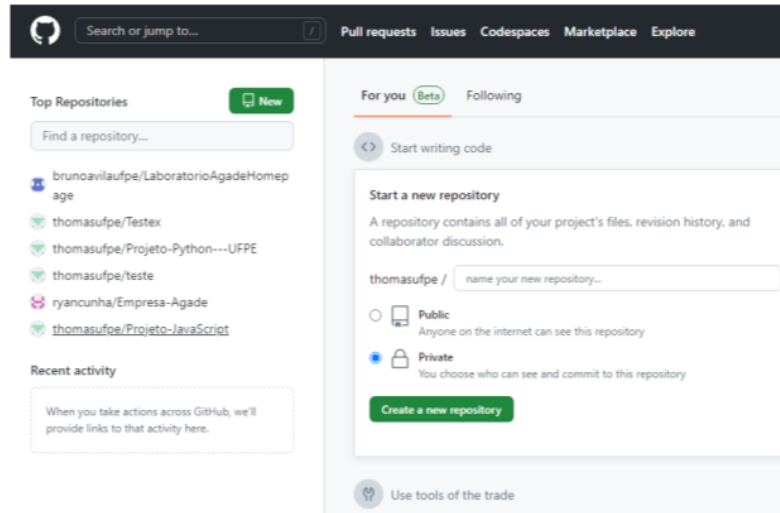


- **Clone:** o processo de clonagem permite que um usuário pegue os arquivos de um repositório remoto e baixe para o seu ambiente local.
- **Add:** o comando “git add” é utilizado somente no git, e serve apenas para informar ao git as alterações que você deseja fazer, no entanto, para realmente aplicá-las é necessário realizar o “commit”. Como dito anteriormente, neste curso iremos utilizar o GitHub Desktop então não utilizaremos o comando “git add”.
- **Commit:** após as alterações serem feitas em um código ou arquivo, é utilizado o comando commit para confirmar e aplicar aquelas alterações dentro do repositório.
- **Push:** o push nada mais é que o ato de enviar as alterações, ou seja, os commits, do seu ambiente local para o remoto, no repositório que você deseja.
- **Fetch:** o fetch tem como objetivo verificar se há novas alterações no repositório, ou seja, ele pega as informações mais recentes e informa

ao ambiente local que há novas atualizações, porém o fetch apenas avisa, a partir do fetch o usuário decide fazer o pull.

- **Pull:** o pull serve para importar as últimas alterações feitas no ambiente remoto, para o seu ambiente local.
- **Branch:** as branches funcionam como uma nova linha temporal alternativa dentro do desenvolvimento de um código. Quando é criado um novo repositório dentro do github, esse repositório automaticamente gera uma branch padrão, chamada de main ou master. A partir dessa branch main, geralmente são criadas outras branches com o intuito de criar uma linha temporal paralela, onde é possível realizar testes e correções de bugs. Tudo isso torna o processo de desenvolvimento mais seguro, evitando que os desenvolvedores façam alterações diretamente na branch main, o que não seria uma boa prática.
- **Merge:** o merge é o processo ao qual unificamos duas branches diferentes. No exemplo acima, foi citado a criação de uma branch alternativa somente para testes. Nesse exemplo, existem duas branches: a main e a de teste. Após verificar que tudo está funcional na branch de teste, realizamos o merge, que é juntar o que foi feito na branch teste, dentro da branch main, para seguir o fluxo temporal de desenvolvimento.

### 3.1 Como criar um repositório no site do GitHub



A criação do repositório é bem simples. Pode ser feita tanto no GitHub quanto no GitHub Desktop. No site do GitHub, após a conta ter sido criada, basta ir na home page, e clicar em “New”, no lado superior esquerdo. Depois, aparecerá uma caixa a qual será necessário atribuir um nome para o repositório, e definir se ele será público ou privado.

### 3.2 Como criar um repositório no GitHub Desktop

The image displays two side-by-side screenshots of the GitHub Desktop application's repository creation and publishing workflow.

The left screenshot shows the "Create a new repository" dialog box. It includes the following fields and options:

- Name:** A text input field with the placeholder "repository name".
- Description:** A text input field.
- Local path:** A text input field showing "C:\Users\Laboratório Agadê\Desktop" and a "Choose..." button.
- Initialize this repository with a README:** An unchecked checkbox.
- Git ignore:** A dropdown menu currently set to "None".
- License:** A dropdown menu currently set to "None".
- Buttons:** "Create repository" and "Cancel" at the bottom.

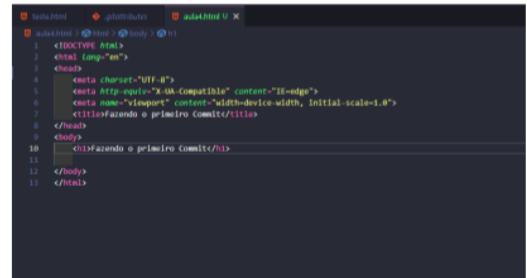
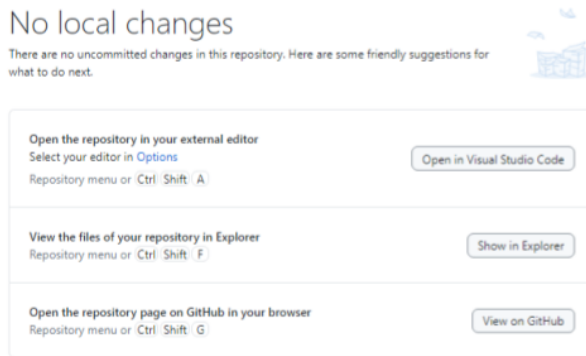
The right screenshot shows the "Publish repository" dialog box. It includes the following fields and options:

- Repository Selection:** Tabs for "GitHub.com" (selected) and "GitHub Enterprise".
- Name:** A text input field containing "privado".
- Description:** A text input field containing "teste".
- Keep this code private:** A checked checkbox.
- Buttons:** "Publish repository" and "Cancel" at the bottom.

Dentro do GitHub Desktop, o processo é basicamente o mesmo. Na aba file, clicar em “new repository” e, a partir daí, o procedimento é o seguinte:

- Dê um nome ao seu repositório.
- Coloque uma descrição do repositório.
- (Opcional) Escolha um caminho para salvar o repositório.
- Marque a opção “Initialize this repository with a README”.
- (Opcional) Crie uma pasta gitignore.
- (Opcional) Selecione uma licença para o repositório.

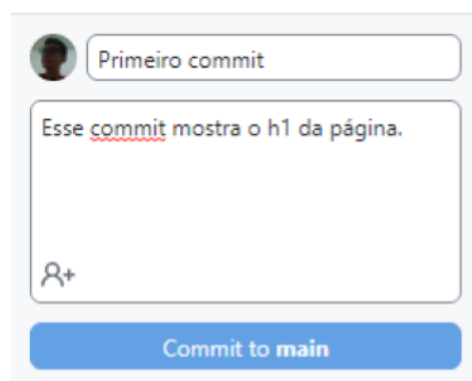
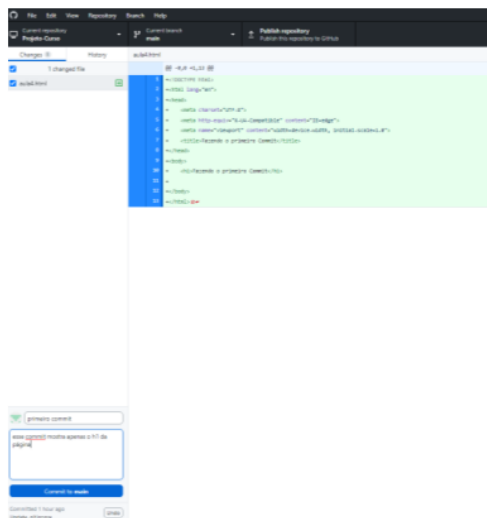
## 4.1 Realizando e entendendo o commit



Para realizar o primeiro commit, existem duas formas: por meio do command-line interface (Interface de Linha de Comando) do git, ou utilizando o Vs Code. No nosso curso, utilizaremos o VS Code juntamente com o Github Desktop. Então, dentro da interface inicial do GitHub Desktop, localize o “Open in Visual Studio Code”. O Vs Code auxiliará na criação dos códigos para realização dos commits.

- “Open in Visual Studio Code” serve para abrir o arquivo em questão.
- “Show in Explorer” serve para abrir a pasta do repositório local.
- “View on github” serve para abrir o repositório remoto.

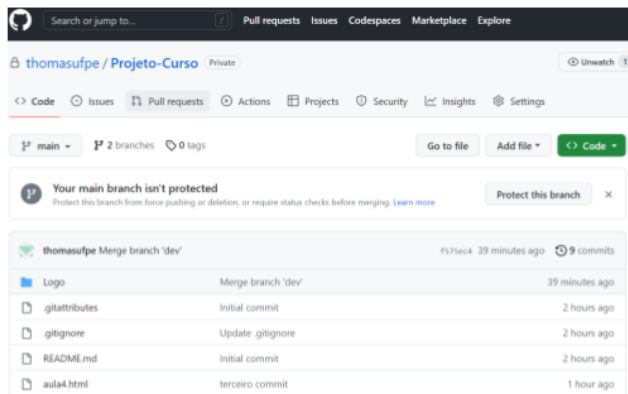
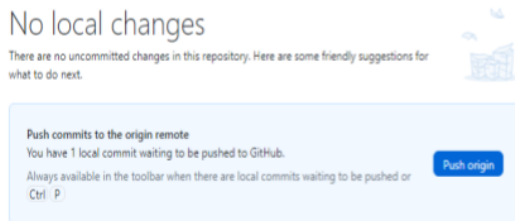
## 4.2 Realizando e entendendo o commit



Depois de criar o primeiro código no VS Code, salve-o (Ctrl + S). Agora, quando abrir novamente o GitHub Desktop, irá abrir a interface com: o código modificado, o histórico do que foi feito, e uma caixa para atribuir o nome e a descrição do commit.

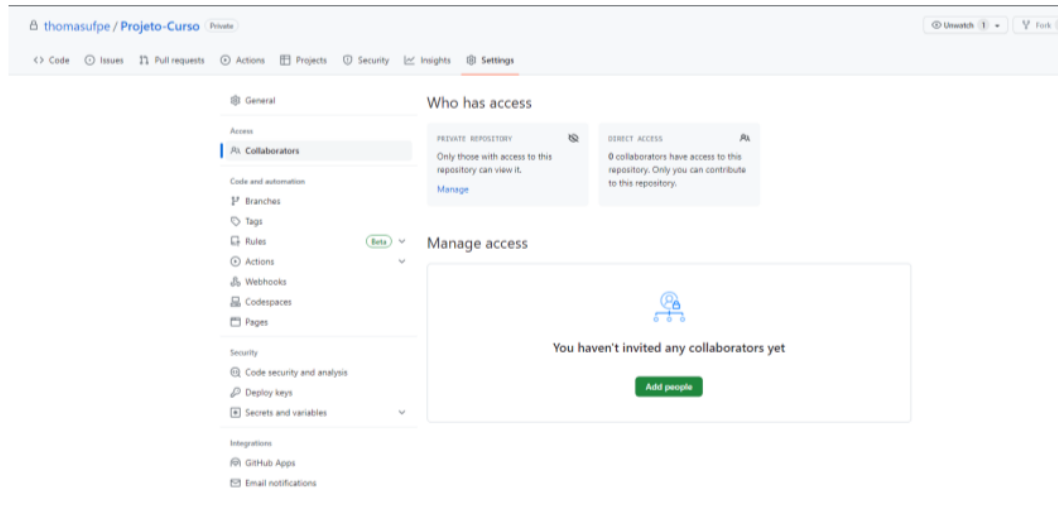
A caixa do commit (imagem da direita) é muito importante para as boas práticas de desenvolvimento. Com a possibilidade de nomear e descrever que tipo de commit foi feito, permite-se que toda equipe fique por dentro daquilo que foi alterado. Dentro de um cenário grande, isso define uma boa organização para a continuação do fluxo. Por exemplo, o desenvolvedor pode dizer se sua modificação foi para uma nova funcionalidade ou se foi alguma correção de bugs(erros).

### 4.3 Realizando o Push



Após criar o código e definir nome e descrição dos commits, está na hora de realizar o push que, como citado anteriormente, irá publicar os commits no repositório remoto.

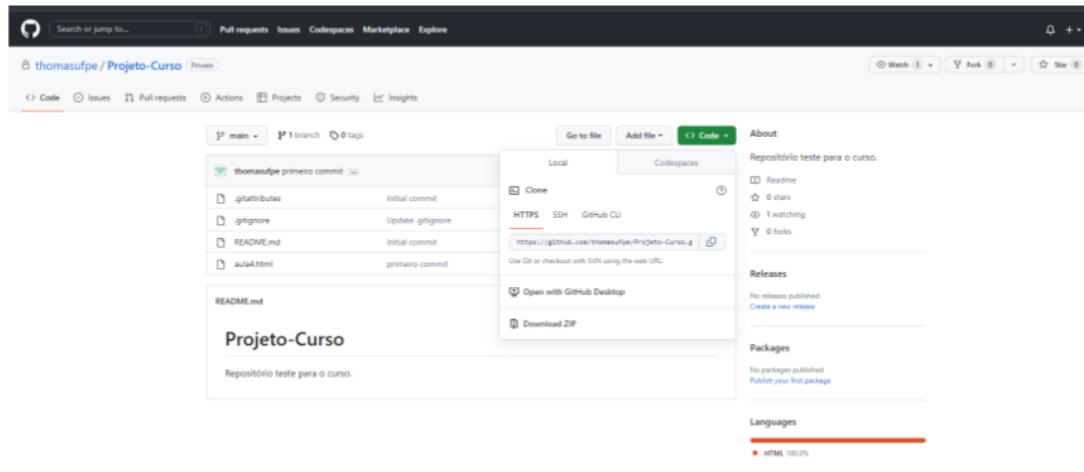
## 4.4 Adicionando um novo membro



Para iniciarmos em um ambiente de desenvolvimento colaborativo é necessário que o dono do repositório adicione um novo membro. Basta ir nas configurações e ir na aba de colaboradores e adicionar com o nome do usuário.

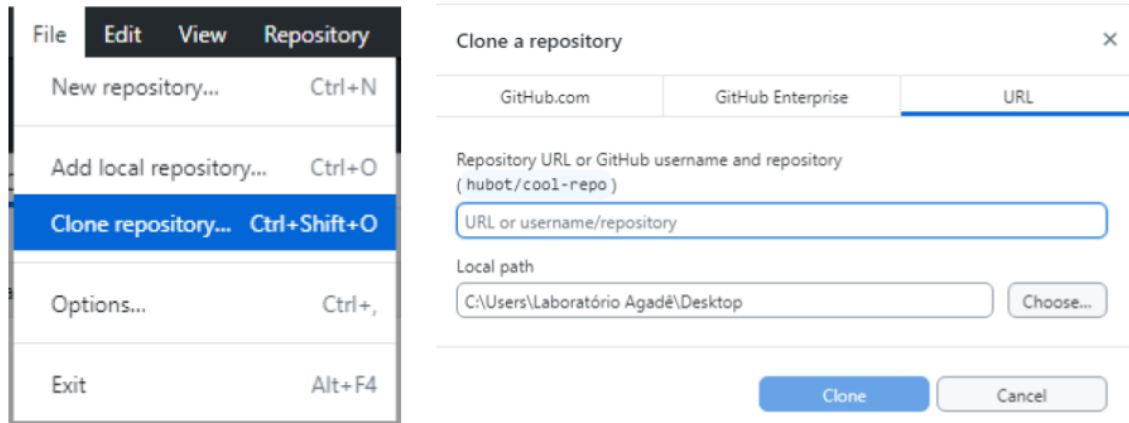


## 4.5 Copiando o link de clonagem do projeto no GitHub



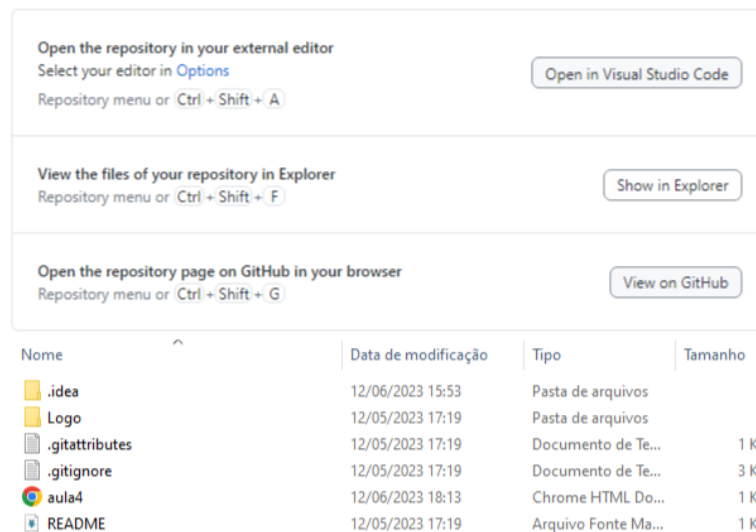
Depois que um novo membro foi adicionado, ele precisa clonar o repositório remoto, ou seja, fazer com que ele fique na sua máquina local. Para isso, basta ele ir no repositório remoto, e clicar no ícone verde “code”. Dentro dele, ele verá um link HTTPS. Basta copiar o link.

#### 4.6 Clonando o Projeto



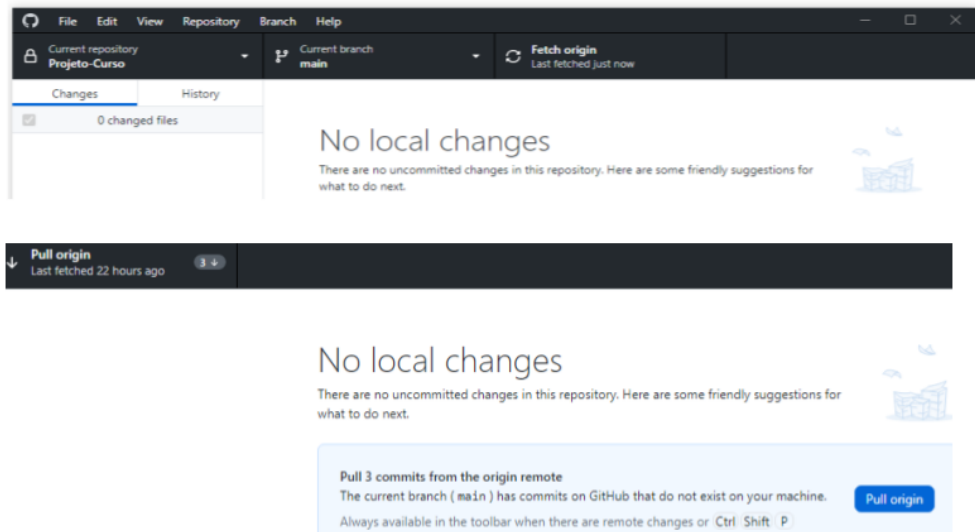
Em seguida, iremos no github desktop, na aba “File” e em “Clone repository”. Feito isso, dentro do “Clone Repository”, colamos o link previamente copiado na primeira caixa, onde tem “URL or username/repository”.

## 4.7 Encontrando o Projeto Clonado



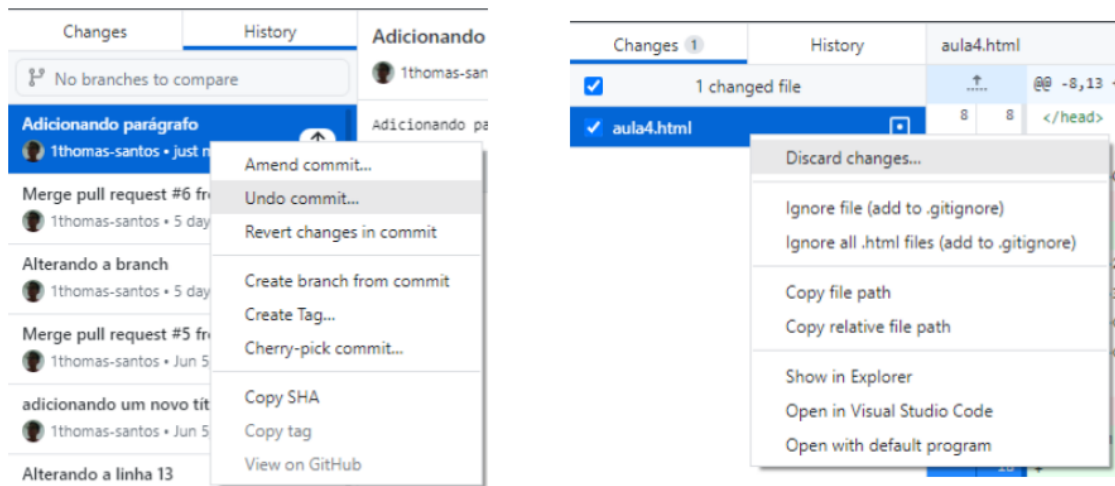
Após clonar, basta ir na tela inicial do GitHub Desktop e clicar na opção “Show in Explorer”, após isso irá abrir o local dos arquivos com a pasta em que estão os arquivos do repositório clonado. Em seguida, para abrir no VS Code vá novamente na tela inicial do GitHub Desktop e clique em “Open in Visual Studio Code”.

## 4.8 Realizando o Fetch e o Pull



Trabalhando em equipe o “Fetch” e o “Pull” serão utilizados frequentemente, o “Fetch” serve para buscar atualizações no repositório remoto, realizadas por outros colaboradores. E, caso haja alguma mudança, o GitHub Desktop sinaliza mostrando a opção de “Pull origin”, que serve para trazer essas alterações para o repositório local e mantê-lo atualizado.

#### 4.9 Alterando e revertendo um commit



Para alterar ou reverter um commit realizado, é necessário ir no histórico do projeto no github desktop e clicar com o botão direito do mouse no commit recente, então irá aparecer as seguintes opções: “Amend commit”, “Undo commit” e “Revert commit”.

O “Amend commit” é utilizado para modificar o último commit realizado, ou seja, você pode excluir ou alterar algo no commit mais recente, porém, mantendo-o no histórico do projeto.

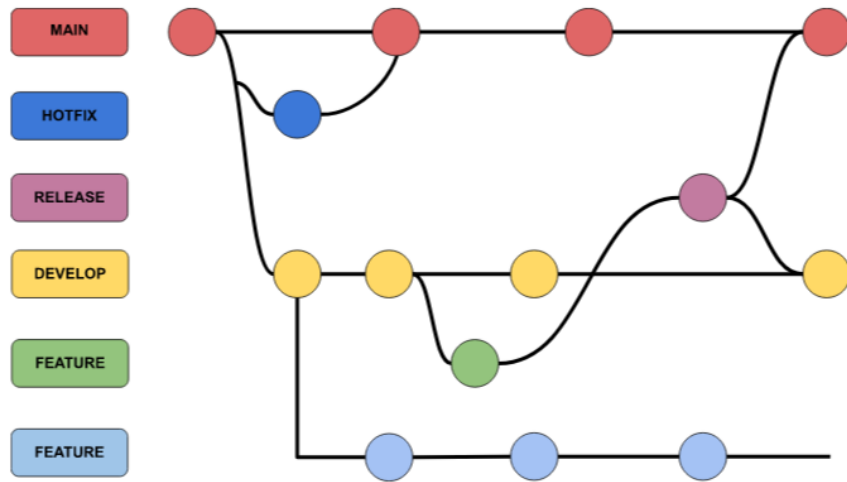
O “Undo commit” é utilizado para refazer completamente o último commit, inclusive removendo-o do histórico.

Já o “Revert commit” é utilizado para desfazer as alterações do commit mais recente, porém, mantendo o commit no histórico, ou seja, as mudanças são desfeitas completamente mas o commit fica registrado no histórico.

Sendo assim, como o objetivo do git é justamente registrar todas as mudanças do projeto, o “Revert commit” geralmente é o mais utilizado quando o intuito é desfazer algum commit.

No entanto, caso queira excluir completamente o commit (inclusive do histórico) a melhor maneira é escolhendo a opção “Undo commit” e, após isso, clicar com o botão direito do mouse no commit e clicar na opção “Discard changes”.

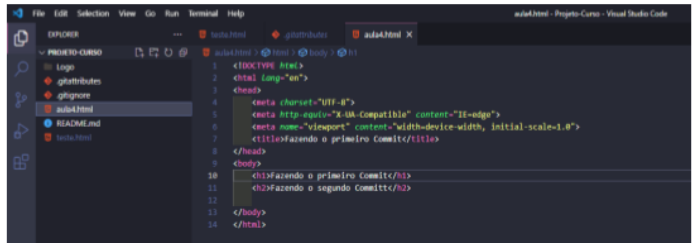
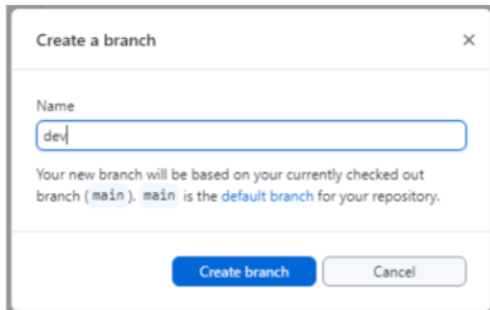
### 5.1 Entendendo uma Branch



Como citado anteriormente, as branches servem para realizar alterações no projeto sem alterar diretamente a main branch, pois, caso algo dê errado, não afetará o projeto principal.

As branches funcionam como linhas temporais diferentes de um mesmo projeto, então, quando as alterações estão finalmente de acordo com o desejado, elas são finalmente aplicadas ao projeto principal, ou seja, a main branch, esse processo é chamado de “merge”.

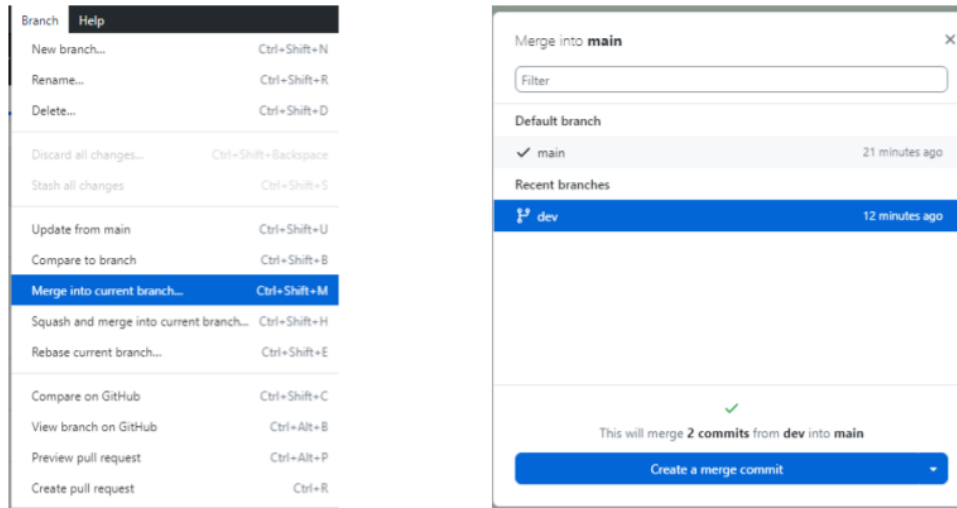
## 5.2 Criando uma nova Branch



Como visto anteriormente, para realizar o merge é necessário a existência de, no mínimo, 2 branches. Então, para isso, criaremos uma segunda branch chamada “dev”. Nota-se que, na branch dev, estão basicamente os mesmos arquivos da branch main, porém com a adição da pasta “Logo”. Agora, juntamos essa branch com a main.

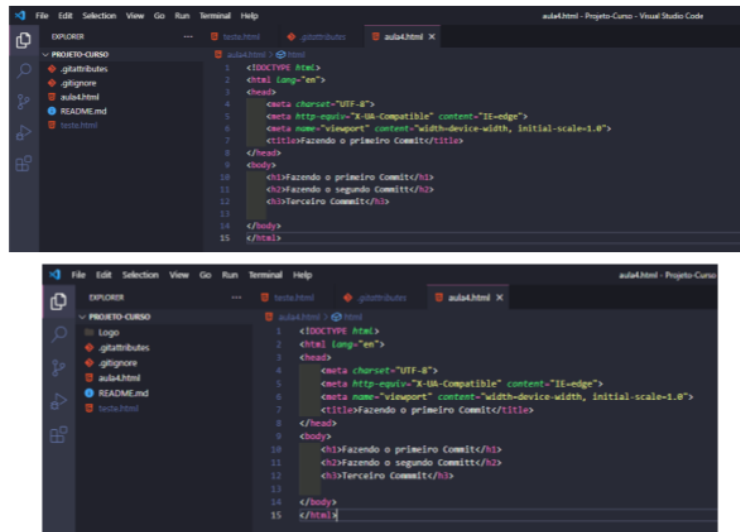


### 5.3 Processo de Merge



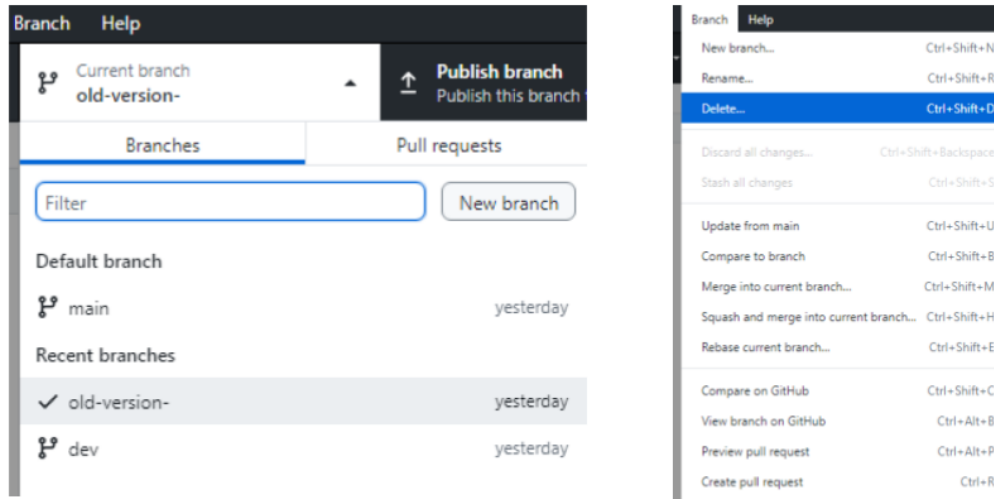
Primeiro, selecione a branch para qual você deseja trazer as alterações, nesse exemplo, deixamos a main branch selecionada, pois, queremos trazer as alterações da “dev” para a main. Então, com a main branch selecionada, basta clicar na aba branch e ir em “Merge into current Branch”, depois, selecionamos a branch “dev” criando o merge “De dev para a main”.

## 5.4 Processo de Merge



Esses são os arquivos da main antes do merge. Abaixo estão os arquivos da branch main após o merge. Repare que, além da adição da pasta “logo”, na branch dev não havia a tag “h3”, indicando que ambas as branches estavam em linhas temporais diferentes.

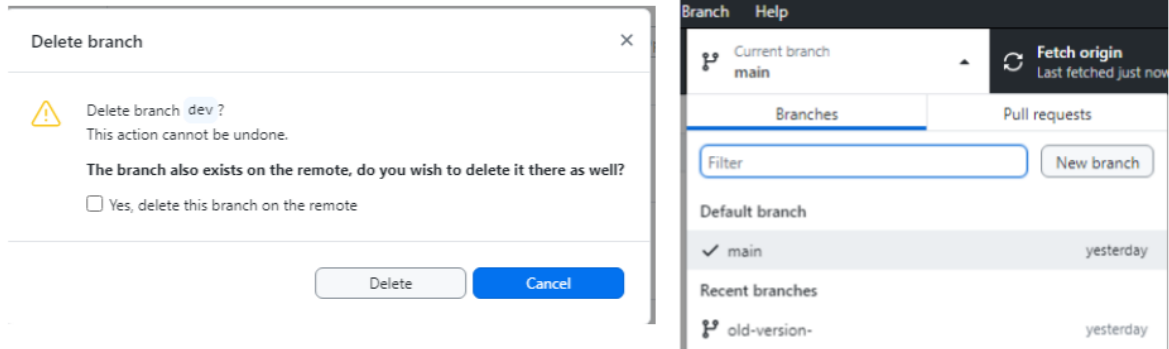
### 5.5 Deletando uma Branch



Para deletar uma branch, primeiro, selecione no GitHub Desktop a branch que deseja excluir, em seguida vá na aba “Branch”, e por fim, clique na opção “Delete”.

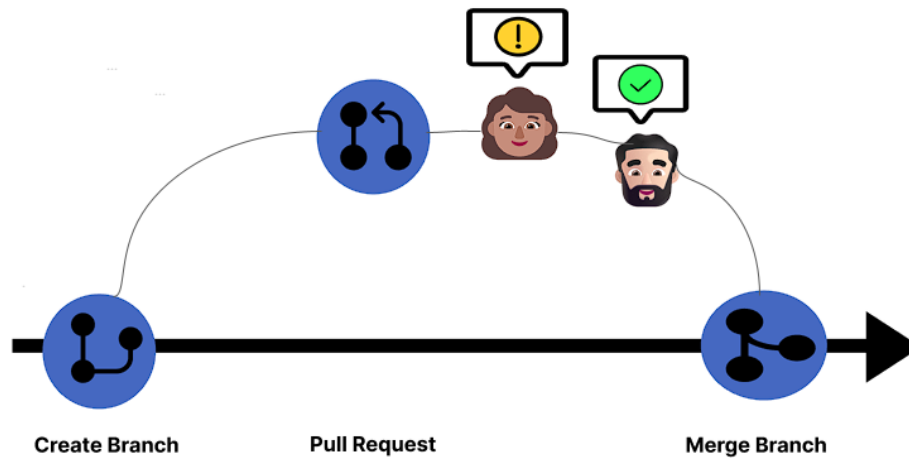
Vale ressaltar que, o GitHub Desktop por padrão não permite deletar a branch main, justamente, por ela ser a raiz principal do projeto.

### 5.6 Deletando uma Branch



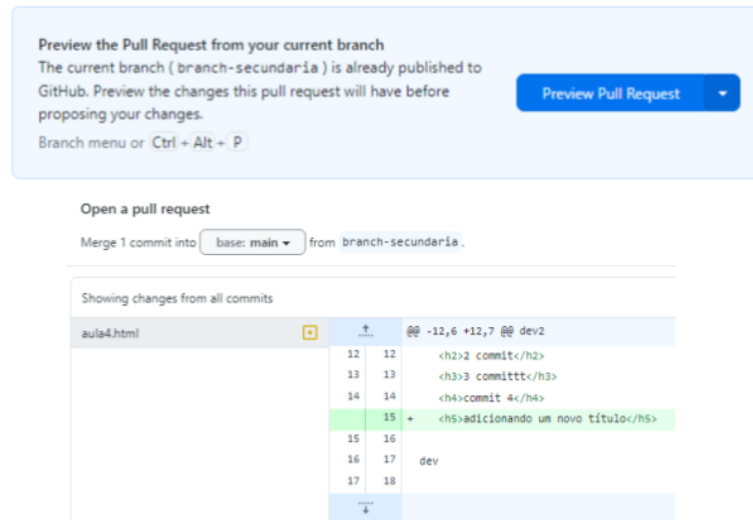
Neste aviso, é sinalizado que caso delete a branch, não será possível recuperá-la. Além disso, há a opção de deletar apenas no local e deixá-la no remoto ou deletá-la em ambos. Por fim, pode-se observar que a branch “dev”, não existe mais.

### 6.1 O que é um Pull Request?



O Pull Request é um processo em que um desenvolvedor solicita, após a alteração de um código, o merge de sua branch na branch principal. Ele indica que o desenvolvedor em questão finalizou a sua tarefa, e quer integrar sua “task” (tarefa) na branch principal para a continuação do fluxo do desenvolvimento. Posteriormente, essa solicitação é revisada, ou seja, alguém do projeto é responsável por conferir se sua alteração está de acordo com o projeto, a fim de aprovar ou não. Geralmente, os Pull Request são feitos quando algum desenvolvedor cria uma “feature” ou correção de bug, por exemplo.

## 6.2 Criando um pull request no github desktop



Para abrir uma solicitação de pull request é necessário ter outra branch além da main e, essas branches precisam estar em linhas temporais diferentes, por exemplo, se a branch secundária estiver mais atualizada que a main.

Então, após criar uma nova branch a partir da main branch, realize o push da nova branch e faça as alterações desejadas na mesma, após isso, faça o commit e o push das alterações, em seguida, no GitHub Desktop clique na opção “Preview Pull Request”.


Após clicar em “Preview Pull Request” você poderá ver as alterações feitas e compará-las com a main branch. Então, para abrir uma solicitação de pull request no GitHub, basta clicar em “Create pull request” no github desktop.

### 6.3 Criando um pull request no github

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: branch-secundaria ✓ Able to merge. These branches can be automatically merged.

 Solicitando o merge da branch secundaria para a main branch

Write Preview H B I ≡ <> 🔗 ⌨ ⌨ ⌨ ⌨ ⌨ ⌨ ⌨ ⌨ ⌨ ⌨

Leave a comment

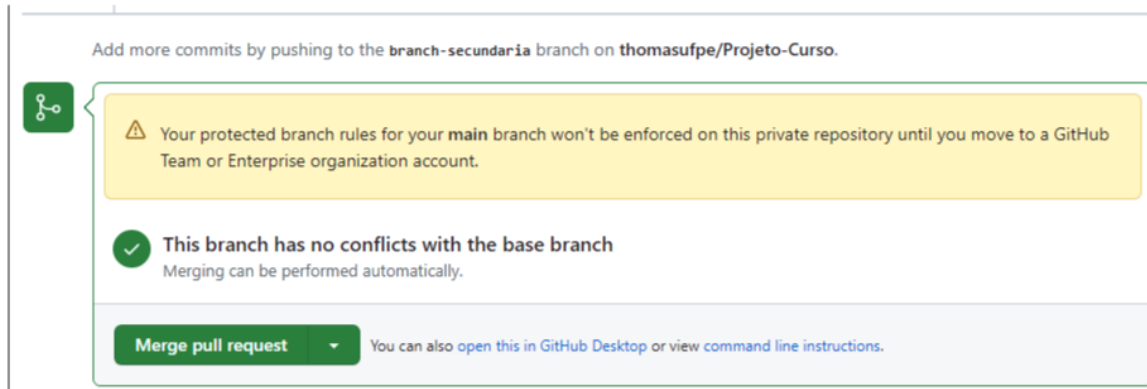
Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

O GitHub Desktop irá lhe redirecionar para o GitHub, e lá você poderá adicionar mais informações sobre as mudanças realizadas, depois de tudo pronto, clique em “Create pull request” no GitHub.

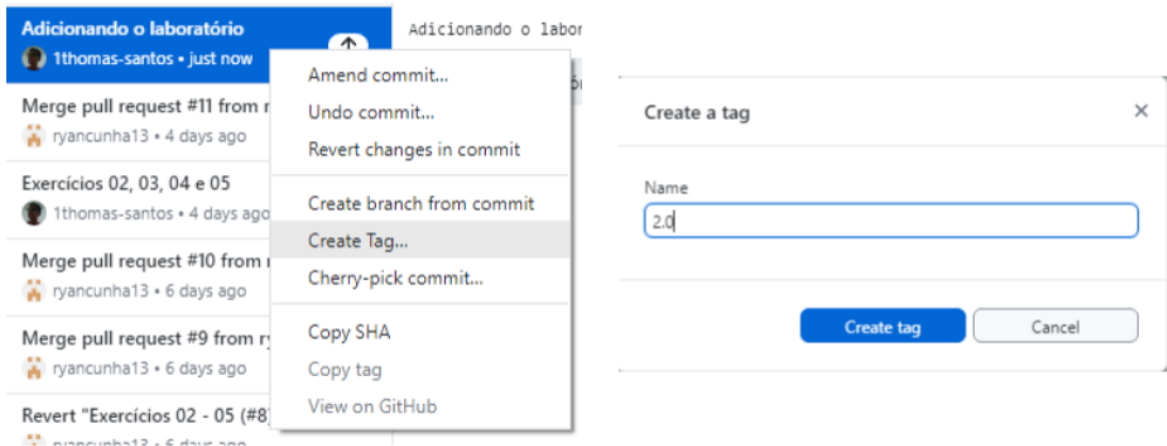
## 6.4 Realizando o merge



Agora que o pull request foi criado, a pessoa responsável pelo projeto irá analisar as alterações e, caso estejam de acordo com o projeto, basta realizar o merge clicando em "Merge pull request", ou seja, as alterações serão adicionadas diretamente na main branch.



### 7.1 O que é Tag e como funciona?



As tags servem para você destacar um commit e, a partir disso, criar versões do projeto.

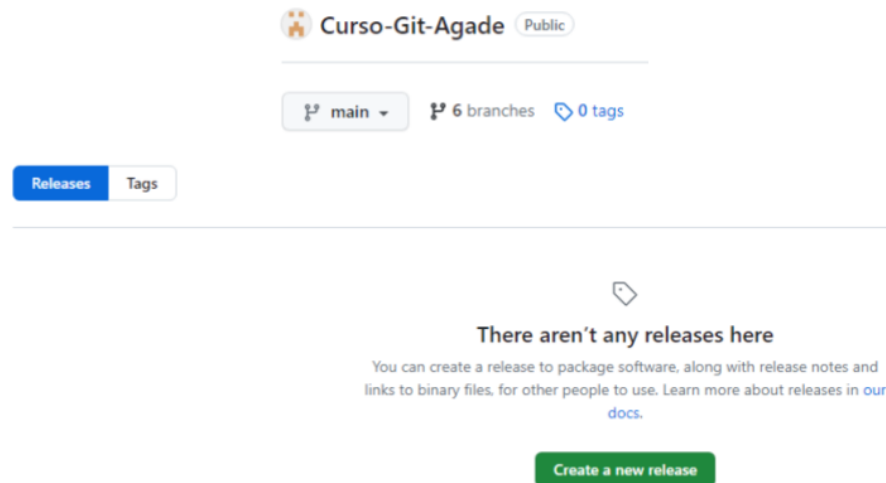
Para criar uma tag existem duas formas, a primeira seria ir no histórico do repositório no GitHub Desktop, escolher o commit no qual você deseja a partir dele criar uma versão, clicar com o botão direito e ir em “Create Tag”, feito isso, basta determinar o número da tag (versão).

Algumas observações importantes:

- Para deletar uma tag basta ir no commit que está com a tag e realizar o mesmo processo de criar, mas em vez de clicar em “Create Tag” clique em “Delete Tag”.
- Apenas é possível excluir completamente uma tag de um commit do repositório se a mesma for deletada antes de ser realizado o push.

A segunda forma e a mais recomendada de criar uma tag, é criar diretamente no momento do release, como veremos no slide a seguir.

## 7.2 O que é Release e como funciona?



Os releases são criados a partir das tags, ou seja, é o lançamento de determinada versão (tag) do projeto.

Para criar um release é necessário ir no repositório no site do GitHub e ao lado das branches, clicar em “tags”. Após isso, você seleciona a opção “Releases” e clica em “Create a new release”.

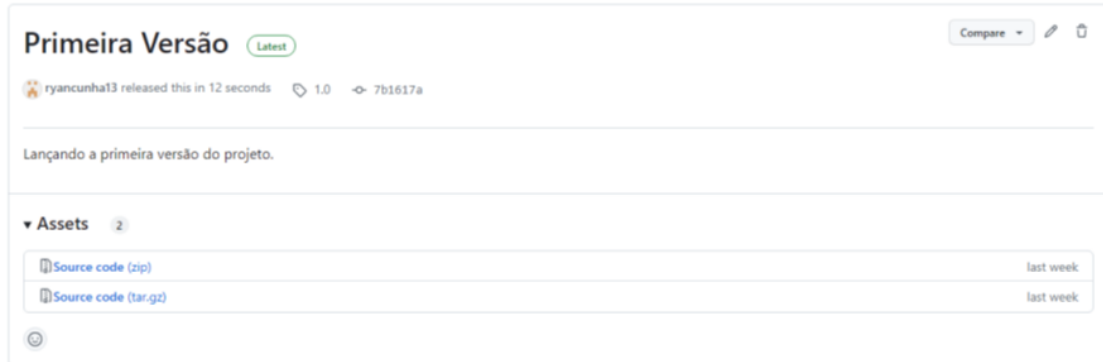
### 7.3 Criando um Release

The screenshot shows the GitHub Releases page with the 'Releases' tab selected. At the top, there's a 'Tags' dropdown menu. Below it, a 'Choose a tag' dropdown is open, showing '1.0' as the selected option. To the right of the 'Choose a tag' dropdown is a 'Target: main' dropdown. Below the 'Choose a tag' dropdown is a 'Previous tag: auto' dropdown and a 'Generate release notes' button. The main text area is titled 'Write' and contains the placeholder text 'Lançando a primeira versão do projeto.' Below the text area is a 'Publish release' button. At the bottom, there's a 'Set as a pre-release' checkbox and a 'Save draft' button.

Após isso, irá aparecer uma tela na qual você deve colocar informações sobre aquele release, como o título e a descrição.

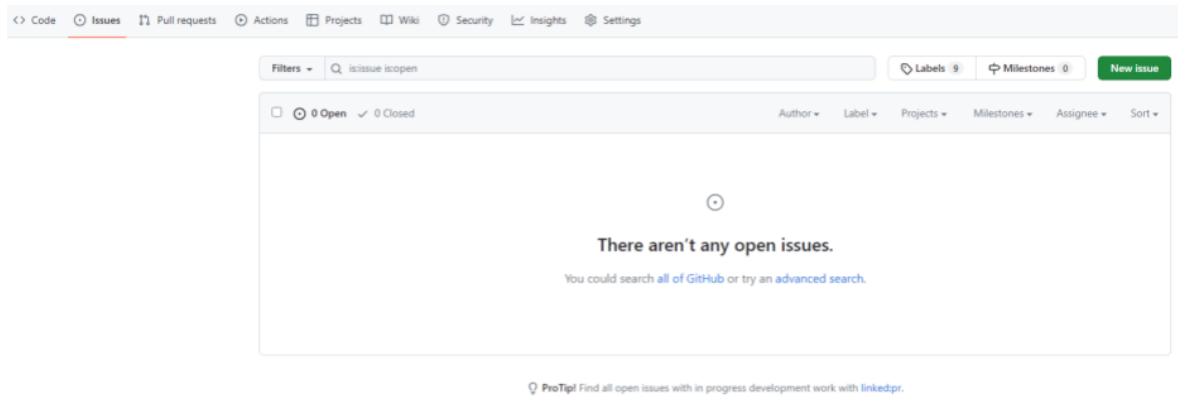
E principalmente, selecionar a tag desejada, ou, como citado anteriormente você pode criar uma tag diretamente no release, basta escolher o número da tag (versão) e clicar em “Create new tag”. Com todas as informações do release presentes e a tag escolhida, basta clicar em “Publish release” e ele será lançado.

## 7.4 Release criado



Após criado o release, é possível baixar os arquivos do release do projeto, comparar aquele release com outra tag (versão) do projeto e, caso desejar, é possível também excluir o release.

## 8.1 O que são Issues?



As issues servem para auxiliar na construção de um determinado projeto ou receber feedbacks dos usuários sobre o projeto. Isso é possível, pois, as issues permitem aos usuários escreverem comentários com imagens, links, emojis, entre outros, que tornam o feedback muito mais construtivo.

Para criar uma issue basta ir no github e clicar na aba “Issues” do repositório, em seguida clique na opção verde “New issue”.

## 8.2 Criando uma Issue

git/github no repositório.' The form includes a 'Submit new issue' button and a note about GitHub Community Guidelines." data-bbox="160 167 891 394"/>

Adicionar o slide no repositório

Write Preview

H B I

Adicionar o slide do curso de [git/github](#) no repositório.

Attach files by dragging & dropping, selecting or pasting them.

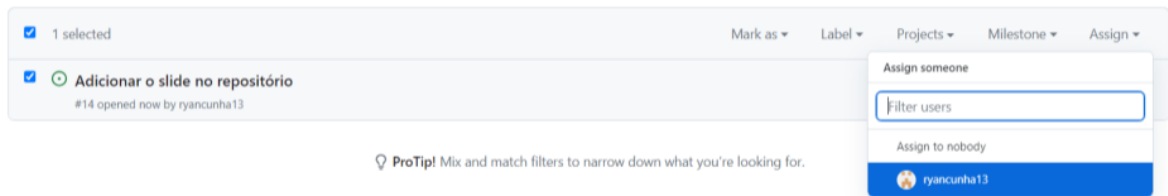
Styling with Markdown is supported

Submit new issue

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Após isso, basta colocar as informações da issue, como o título da issue e um comentário sobre a mesma. Em seguida, clique em “Submit new issue” e a issue será criada.

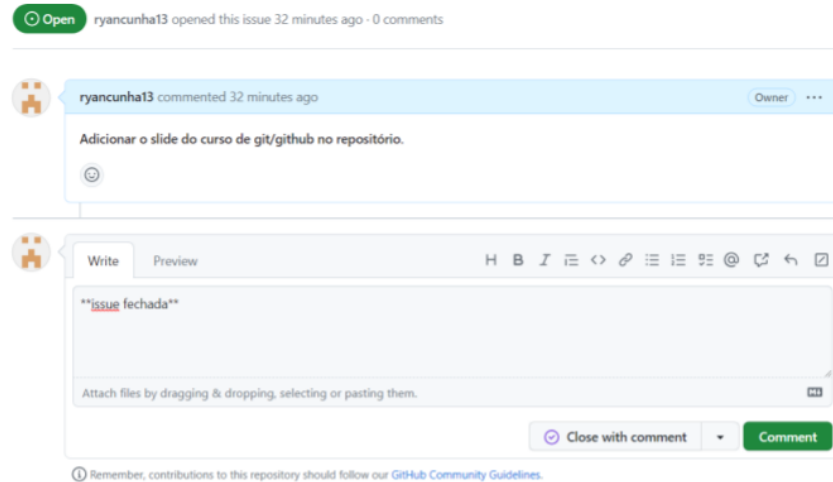
### 8.3 Designando um colaborador para uma issue



Também é possível designar um colaborador do projeto para resolver uma determinada issue. Para isso, após criada a issue, basta ir na aba issues novamente e lá, selecionar a issue desejada e clicar em “Assign” e então selecionar o colaborador que ficará encarregado por aquela issue.

## 8.4 Fechando uma issue

### Adicionar o slide no repositório #14



Após resolvida uma issue, é necessário fechá-la, para isso basta abrir a issue no GitHub e clicar em “Close issue”, ou, caso adicione um comentário no momento de fechar, clique em “Close with comment”.



### 8.5 Exemplos de sintaxes para utilizar nos comentários

Estilo	Sintaxe	Atalho do teclado	Exemplo	Saída
Negrito	<b>** **</b> OU <b><code><u>bold</u></code></b>	Comando + B (Mac) ou CTRL + B (Windows/Linux)	<b>**This is bold text**</b>	Este texto está em negrito
Itálico	<b>* *</b> OU <b><code><u>italic</u></code></b>	Comando + I (Mac) ou CTRL + I (Windows/Linux)	<b><i>_This text is italicized_</i></b>	<i>Este texto está em itálico</i>
Tachado	<b>~ ~ ~</b>	Nenhum	<b>~This was mistaken text~</b>	Este texto contém um erro
Negrito e itálico aninhado	<b><i>** **</i></b> e <b><code><u>bold_italic</u></code></b>	Nenhum	<b><i>**This text is _extremely_ important**</i></b>	Este texto é <i>extremamente</i> importante
Todo em negrito e itálico	<b><i>*** **</i></b>	Nenhum	<b><i>***All this text is important***</i></b>	<i>Todo este texto é importante</i>
Subscrito	<b><code>&lt;sub&gt;</code></b> <b><code>&lt;/sub&gt;</code></b>	Nenhum	<b><code>&lt;sub&gt;</code>This is a subscript text<code>&lt;/sub&gt;</code></b>	Este é um texto subscrito
Sobrescrito	<b><code>&lt;sup&gt;</code></b> <b><code>&lt;/sup&gt;</code></b>	Nenhum	<b><code>&lt;sup&gt;</code>This is a superscript text<code>&lt;/sup&gt;</code></b>	

Na imagem acima temos alguns exemplos de sintaxes que podem ser utilizados nos comentários para destacá-los e torná-los mais organizados.

## LABORATÓRIO

### Laboratório 1: Utilizando o git/github

#### Exercício 1: Criando um repositório local no github desktop.

1. Instale o git e o Github Desktop na sua máquina.
2. Crie uma conta no github.
3. Crie um repositório local no Github Desktop.
4. Vincule a sua conta do github com o Github Desktop.
5. Publique esse repositório local no seu github para transformá-lo em um repositório remoto.

### Laboratório 2: Aprendendo git/github

#### Exercício 2: Praticando os comandos no github desktop

1. Agora com o Github Desktop aberto, clique em “Open in Visual Studio Code” para criar o primeiro código.
2. Realize o primeiro commit utilizando o Github Desktop.
3. Confira se o seu commit foi postado remotamente.

#### Exercício 3: Clonando um repositório e criando uma nova branch

1. Clone o seguinte repositório:  
<https://github.com/ryancunha13/Curso-Git-01.git>
2. Crie uma nova branch nesse repositório.
3. Nessa nova branch, crie um novo código HTML chamado exercício 02.html
4. Faça push do branch para o repositório no GitHub.

5. Abra um pull request no repositório original, solicitando que suas alterações sejam incorporadas ao projeto principal.
6. Aguarde a revisão do pull request pelos mantenedores do projeto.
7. Se houver comentários ou solicitações de alteração, faça as alterações necessárias em seu branch local e faça push novamente.
8. Após a aprovação, o pull request será mesclado ao repositório principal.

#### **Exercício 4: Criando uma tag e um release**

1. No site do GitHub, abra o repositório “Curso-Git-01”.
2. Crie uma release com um título e uma descrição.
3. Crie uma tag para o release.
4. Publique o release.

#### **Exercício 5: Resolvendo issues**

1. Resolva as issues: Deletar as branches "Error" e Criar a branch "fix"