Material Didático - Git/GitHub

SUMÁRIO

- **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.

Cronograma das aulas do curso de Git/GitHub

Aula 01 - 1 hora de duração:

- Introdução aos sistemas de controle de versão. **(10 minutos)**
- Introdução ao git. **(5 minutos)**
- Introdução ao github. **(5 minutos)**
- Explicando os comandos (commit, push, fetch, pull, branch) **(10 minutos)**
- Criando uma conta no github. **(5 minutos)**
- Criando um repositório. **(10 minutos)**
- Realizando os comandos no repositório do aluno (commit, push, fetch e pull). **(10 minutos)**

Aula 02 - 1 hora de duração:

- Clonando o repositório do curso. **(10 minutos)**
- Exercício 01 - Pedra, papel ou tesoura. **(15 minutos)**
- Alterando e revertendo um commit. **(15 minutos)**

Aula 03 - 1 hora de duração:

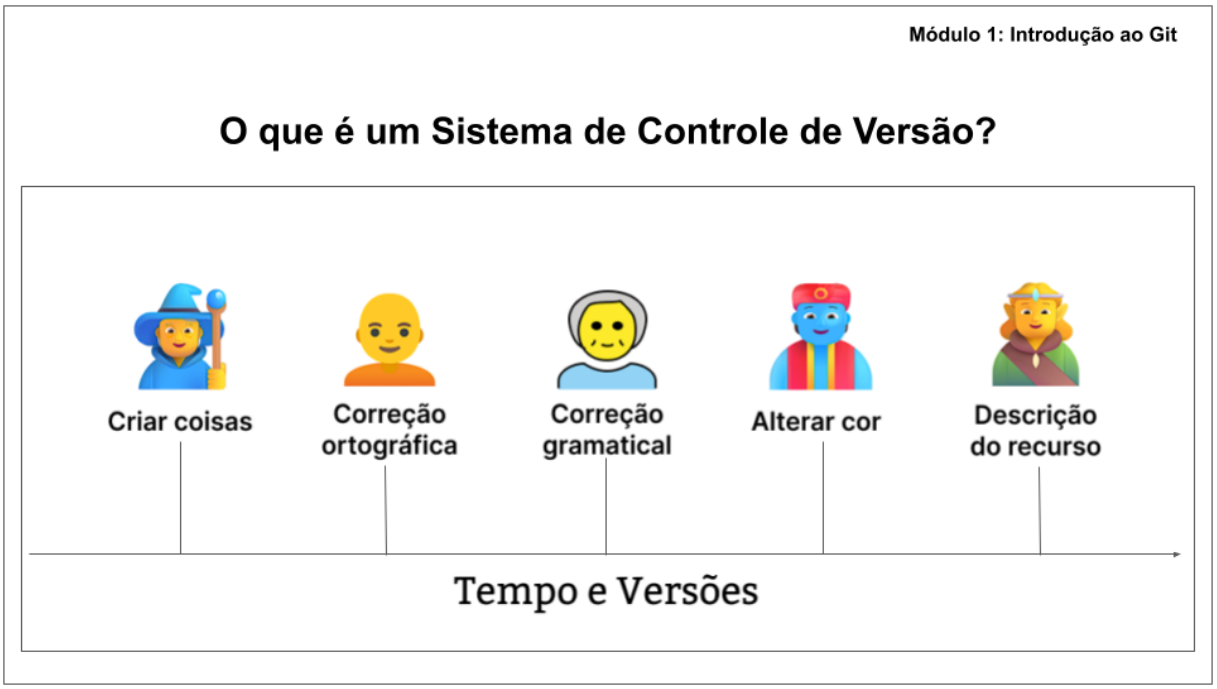
- Criando uma branch. **(5 minutos)**
- Realizando o merge. **(10 minutos)**
- O que é um pull request. **(5 minutos)**
- Abrir um pull request. **(10 minutos)**
- Aprovando um pull request. **(5 minutos)**
- Tipos de merge. **(10 minutos)**
- Realizando o merge. **(10 minutos)**
- Deletando uma branch. **(5 minutos)**

Aula 04 - 1 hora de duração:

- Exercício 02: Desembaralhe frases. **(15 minutos)**
- Exercício 03: Conte uma história. **(15 minutos)**
- O que é uma tag? **(5 minutos)**
- Criando uma tag. **(10 minutos)**
- O que é um release? **(5 minutos)**
- Criando um release. **(10 minutos)**

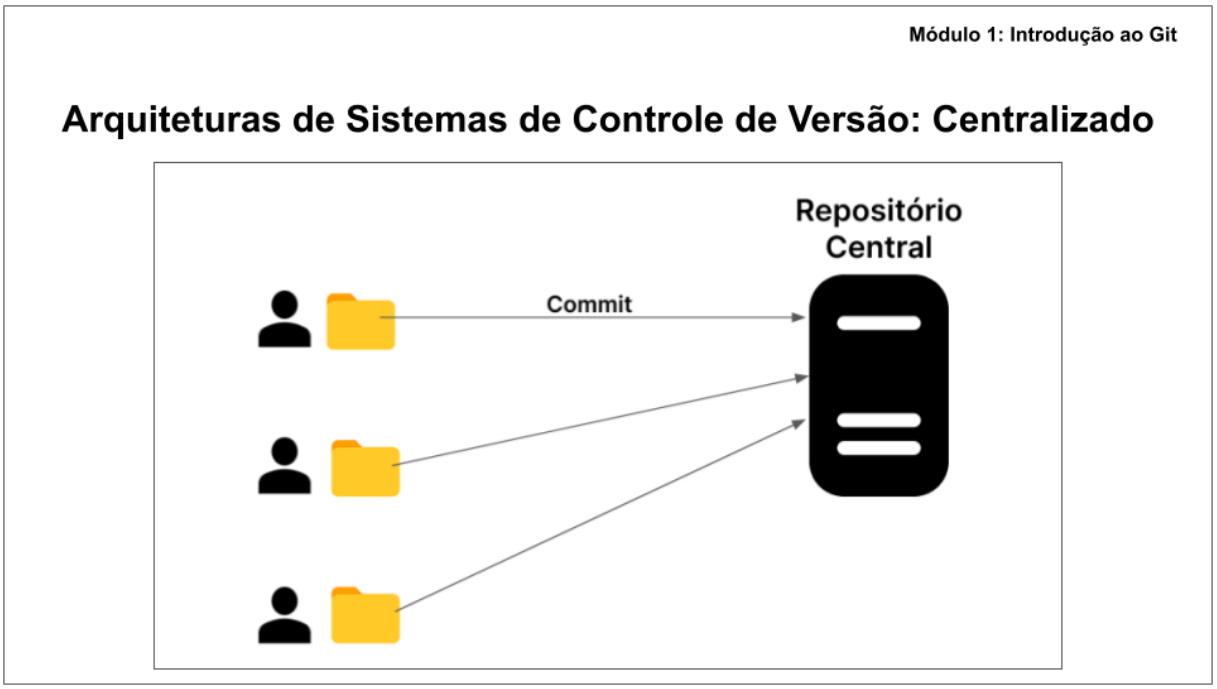
Aula 05 - 1 hora de duração:

- O que é issue? **(10 minutos)**
- Designando um colaborador para uma issue. **(5 minutos)**
- Criando uma issue. **(10 minutos)**
- Fechando uma issue.**(10 minutos)**
- Gitignore. **(20 minutos)**

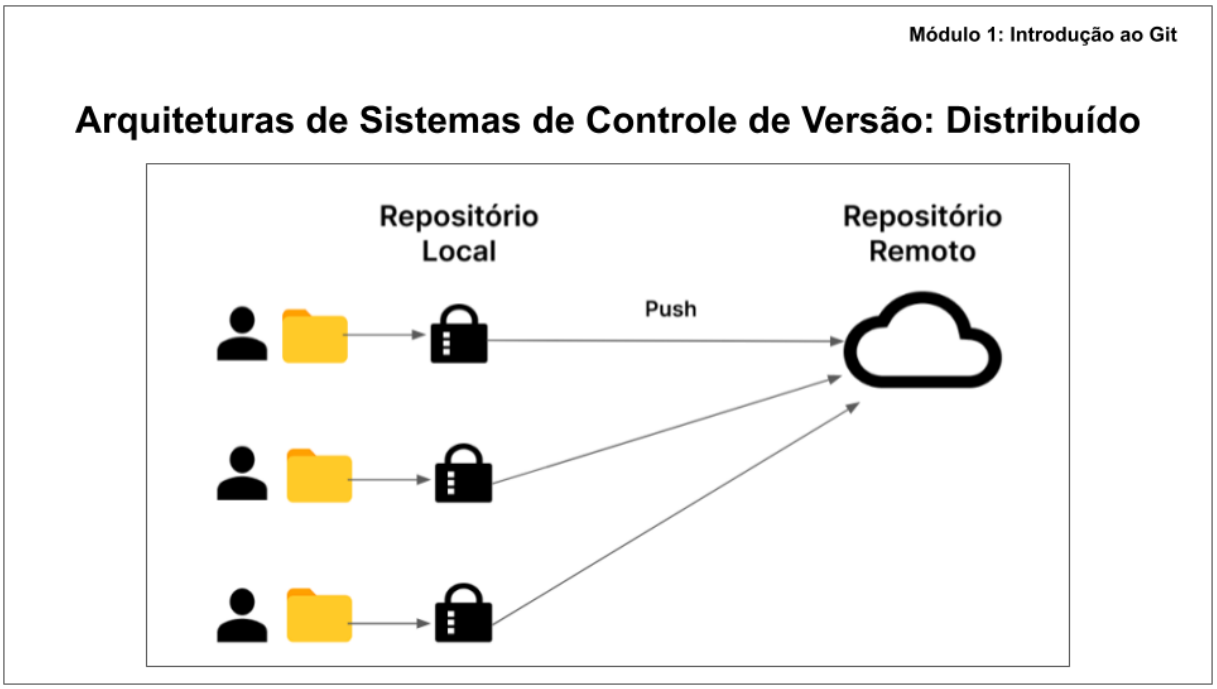


Primeiramente é importante saber o que é um sistema de controle de versão, basicamente, os controladores de versão de código estão relacionados à capacidade de controlar e administrar qualquer tipo de alteração realizada em um arquivo ou conjunto de arquivos. Esse controle de versão permite você saber 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.



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.



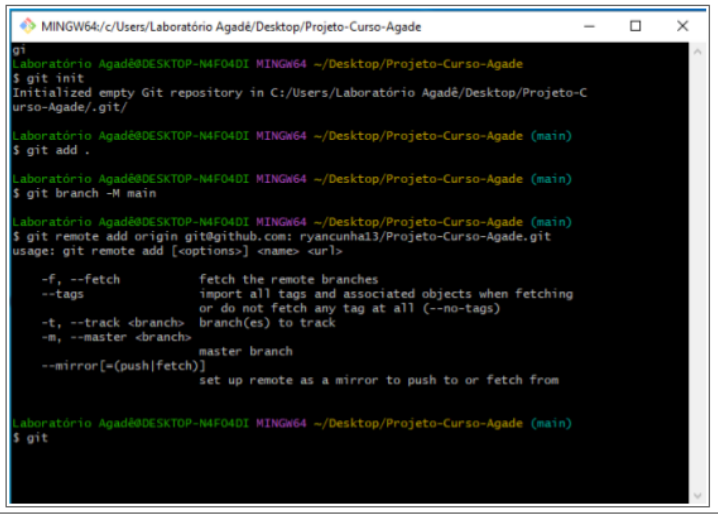
Surgiu então, os sistemas de controle de versão distribuídos, que diferente do centralizado, cada pessoa tem o seu repositório local, ou seja, o repositório em sua máquina, e após feitas as alterações, elas são enviadas para um repositório remoto, permitindo assim uma melhor segurança e controle de versão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

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

O que é o 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.

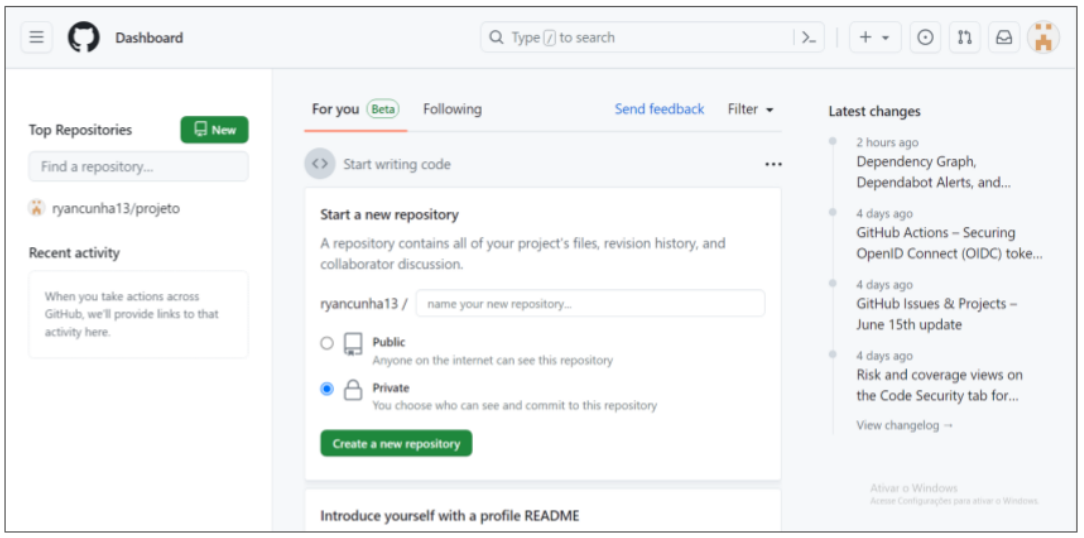
Vantagens

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 ter acesso à 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.

Desvantagens

A principal desvantagem do git é a sua complexidade com relação às possibilidades dos códigos e comandos, dificultando a aprendizagem de um iniciante.

O que é o GitHub?



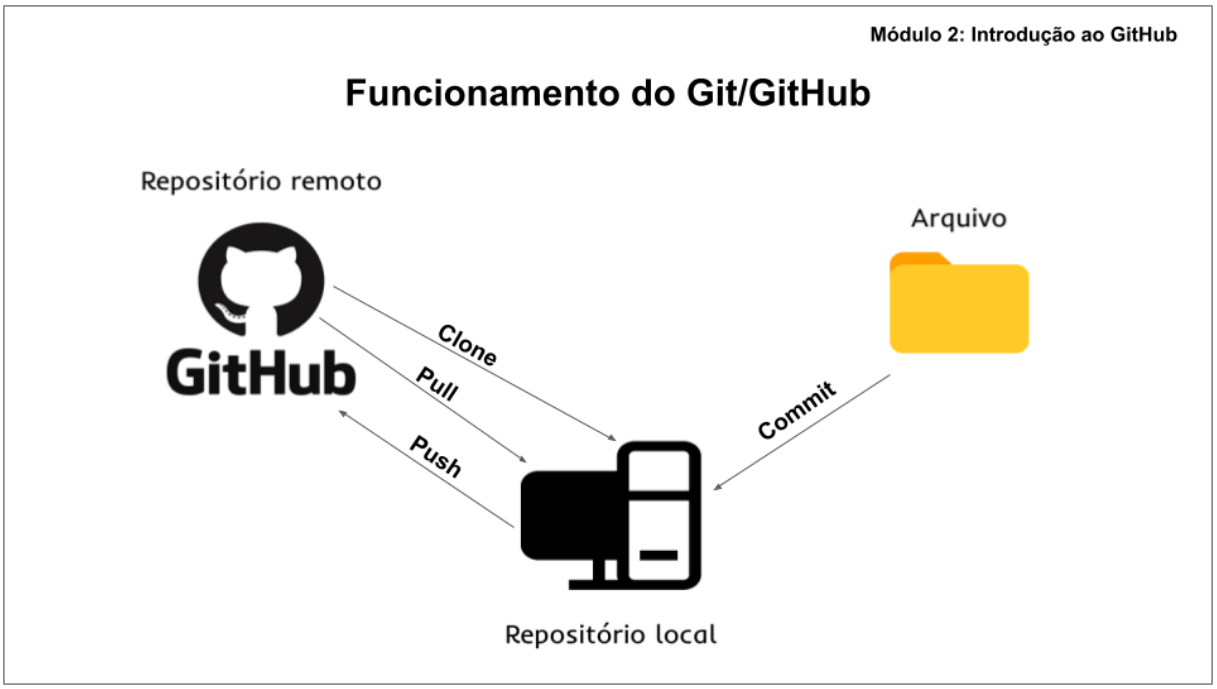
O GitHub é uma ferramenta gratuita de hospedagem de repositórios Git, ou seja, é o site onde ficará armazenado todos os códigos e arquivos enviados previamente utilizando os comandos do Git, permitindo os usuários terem uma ideia visual daquilo que foi produzido, seja em termos de código, alterações, adições ou remoções. Ele é 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 ter acesso à todos os recursos disponíveis.

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.



- **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

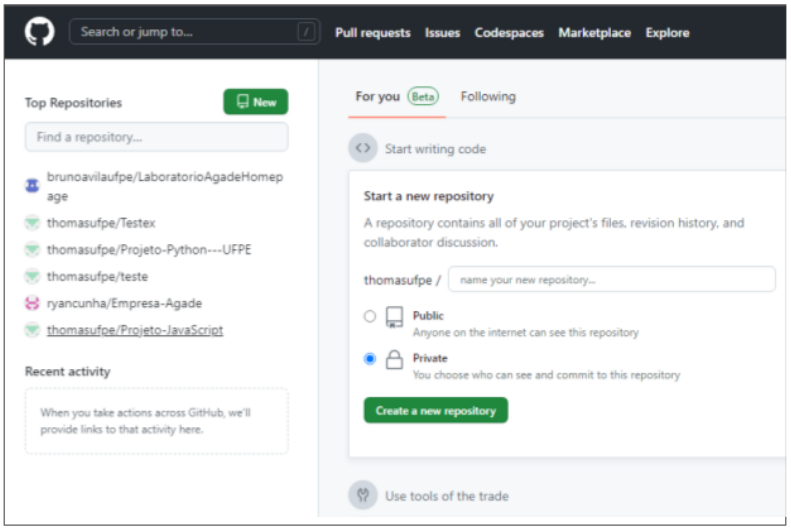
ao ambiente local que há novas atualizações, porém o fetch apenas avisa, a partir do fetch o usuário decide fazer o pull.

- **Pull**: o pull serve para importar as últimas alterações feitas no ambiente remoto, para o seu ambiente local.

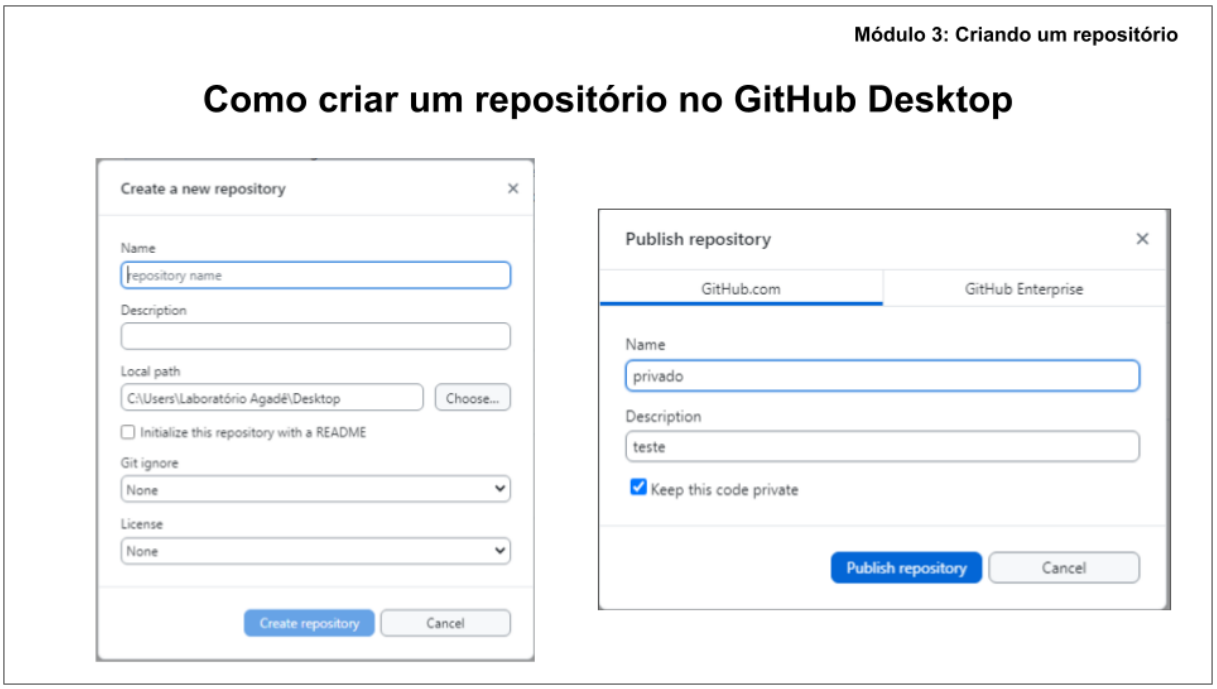
- **Branch:** as branches funcionam como uma nova linha temporal alternativa dentro do desenvolvimento de um código. Quando é criado um novo repositório dentro do GitHub, esse repositório automaticamente gera uma branch padrão, chamada de main ou master. A partir dessa branch main, geralmente são criadas outras branchs 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 branchs: 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.

Como criar um repositório no site do GitHub



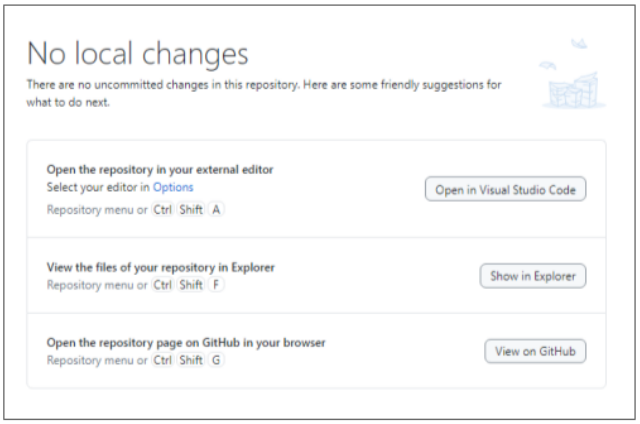
A criação do repositório é bem simples. Pode ser feita tanto no GitHub quanto no GitHub Desktop. No site do GitHub, após a conta ter sido criada, basta ir na home page, e clicar em “New”, no lado superior esquerdo. Depois, aparecerá uma caixa a qual será necessário atribuir um nome para o repositório,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



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:

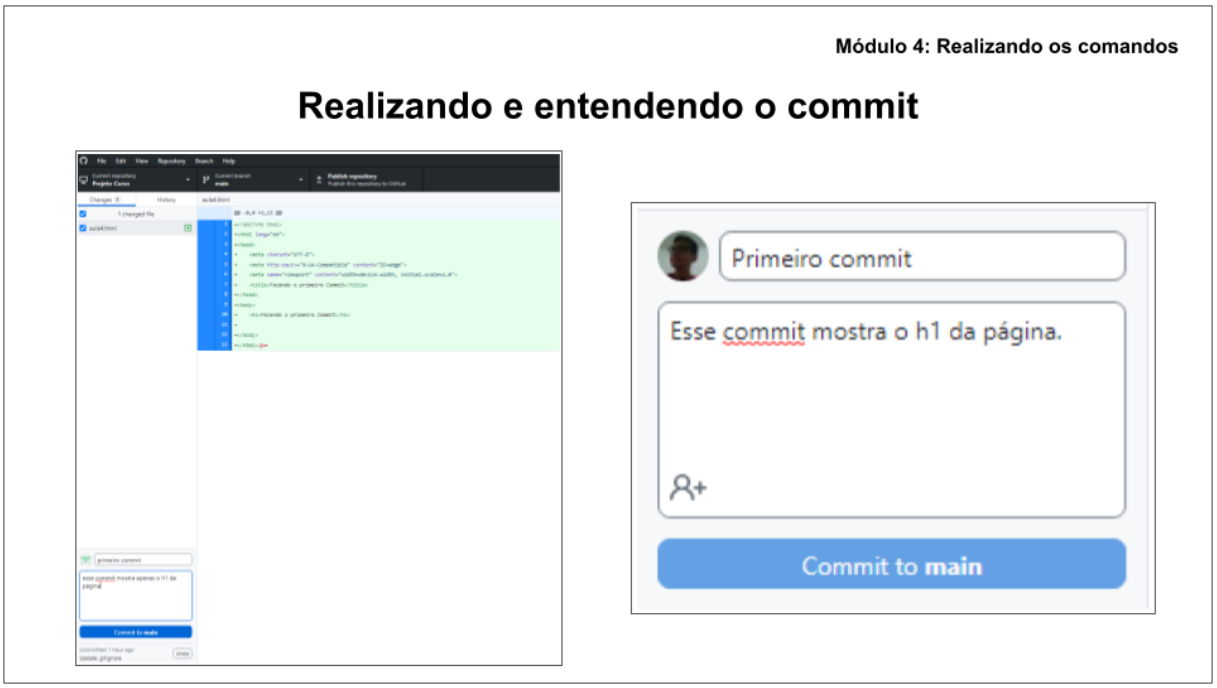
- Dê um nome ao seu repositório.
- Coloque uma descrição do repositório.
- (Opcional) Escolha um caminho para salvar o repositório.
- Marque a opção “Initialize this repository with a README”.
- (Opcional) Crie uma pasta Gitignore.
- (Opcional) Selecione uma licença para o repositório.

Realizando e entendendo o commit



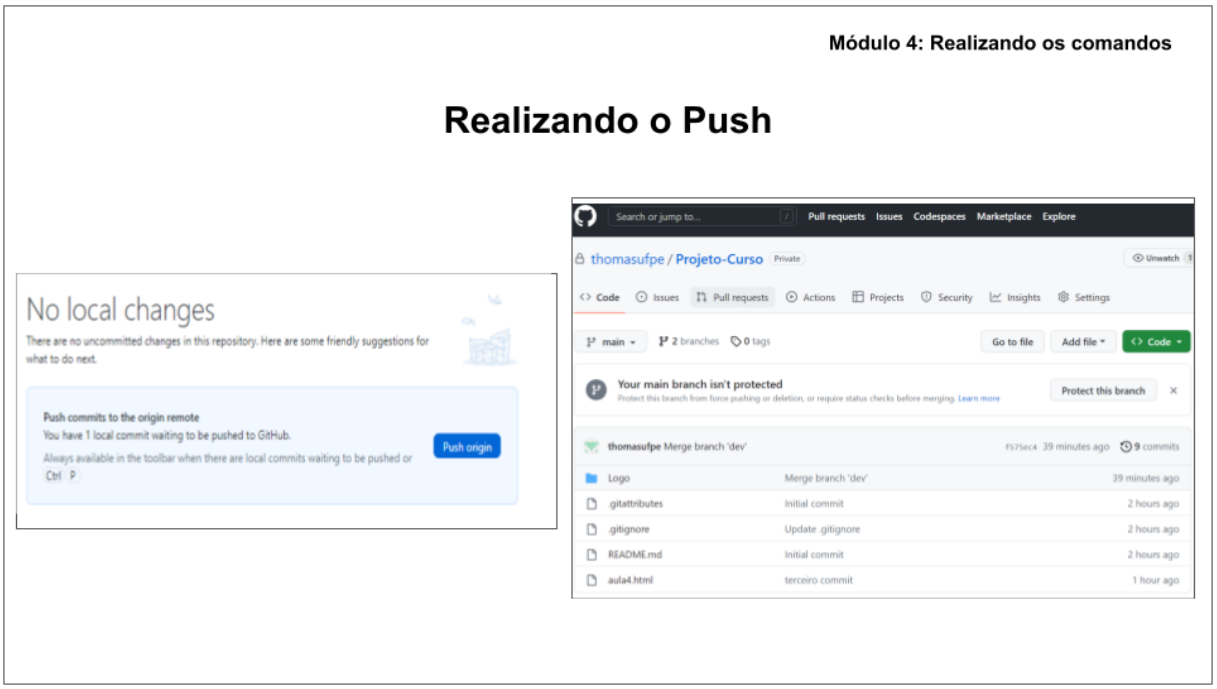
Para realizar o primeiro commit, existem duas formas: por meio do command-line interface (Interface de Linha de Comando) do Git, ou utilizando o Vs Code. No nosso curso, utilizaremos o VS Code juntamente com o Github Desktop. Então, dentro da interface inicial do GitHub Desktop, localize o “Open in Visual 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.



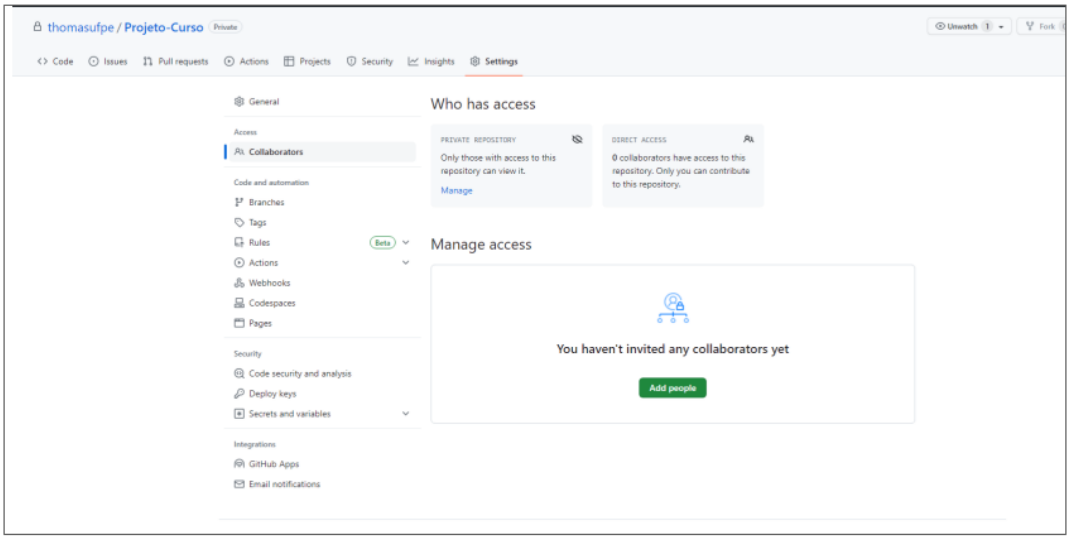
Depois de criar o primeiro código no VS Code, salve-o (Ctrl + S). Agora, quando abrir novamente o GitHub Desktop, irá abrir a interface com: o código modificado, o histórico do que foi feito, e uma caixa para atribuir o nome e a descrição do commit.

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).



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.

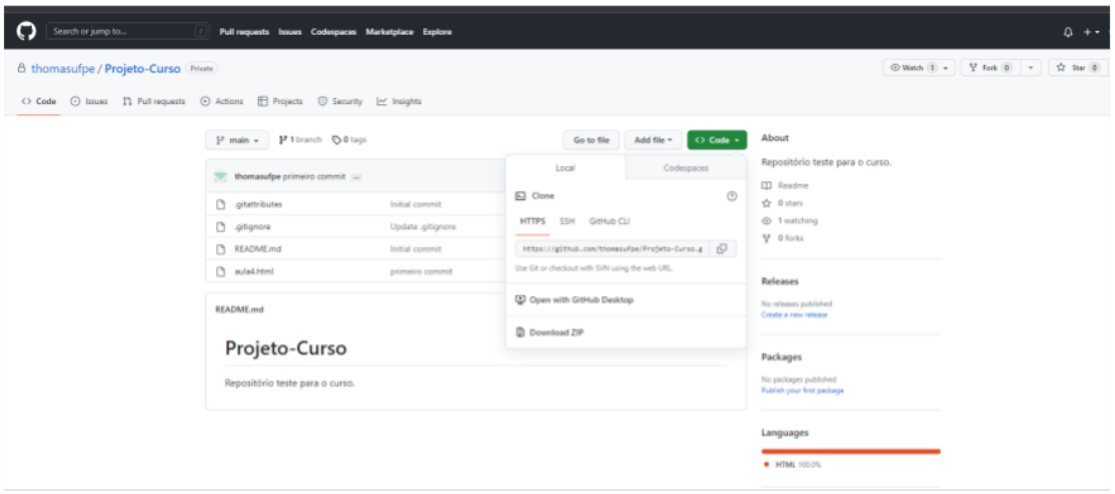
Adicionando um novo membro



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.

Módulo 4: Realizando os comandos

Copiando o link de clonagem do projeto no GitHub

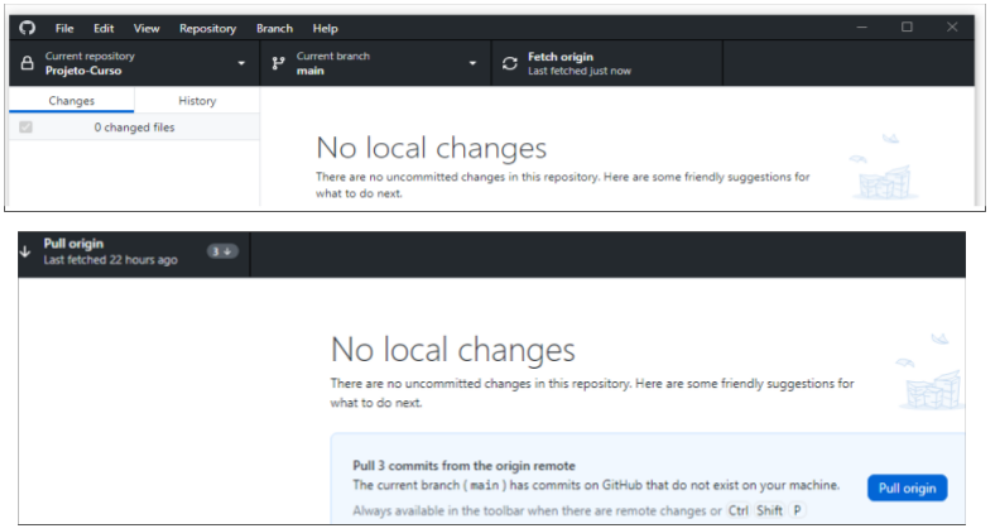


Depois que um novo membro foi adicionado, ele precisa clonar o repositório remoto, ou seja, fazer com que ele fique na sua máquina local. Para isso, basta ele ir no repositório remoto, e clicar no ícone verde “code”. Dentro dele, ele verá um link HTTPS. Basta copiar o link.

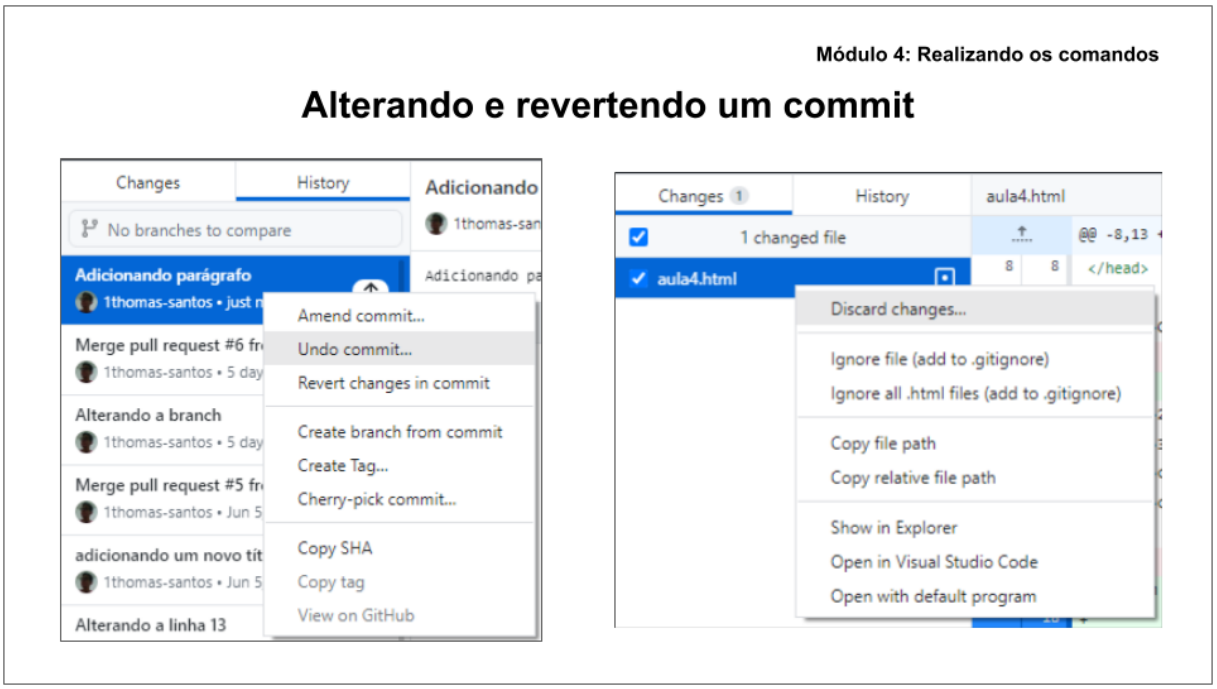
Em seguida, iremos no GitHub Desktop, na aba “File” e em “Clone repository”. Feito isso, dentro do “Clone Repository”, colamos o link previamente copiado na primeira caixa, onde tem “URL or username/repository”.

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”.

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.



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”.

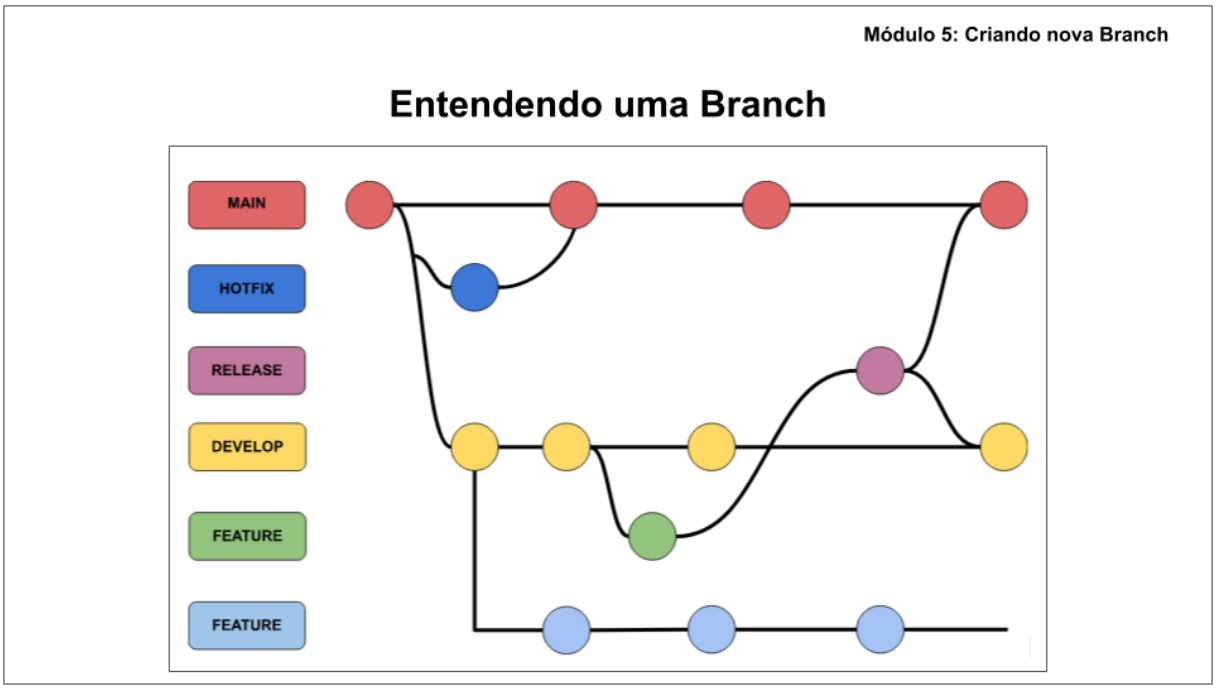
O “Amend commit” é utilizado para modificar o último commit realizado, ou seja, você pode excluir ou alterar algo no commit mais recente, porém, mantendo-o no histórico do projeto.

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.

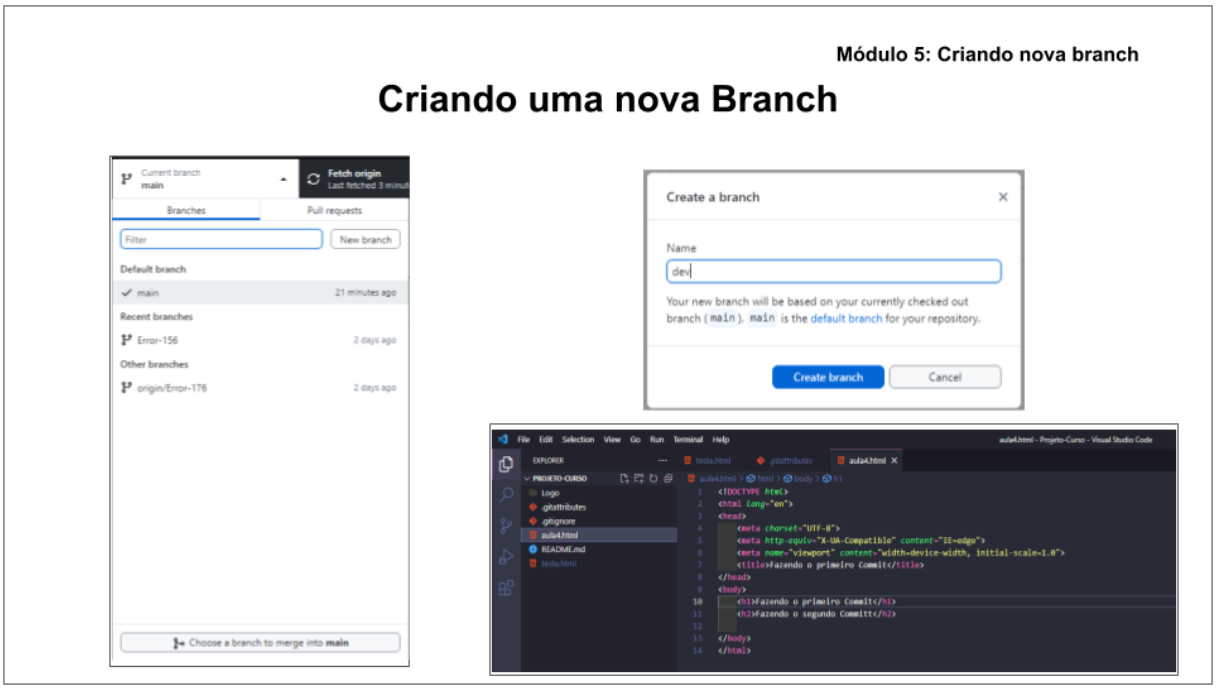
Sendo assim, como o objetivo do Git é justamente registrar todas as mudanças do projeto, o “Revert commit” geralmente é o mais utilizado quando o intuito é desfazer algum commit.

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”.



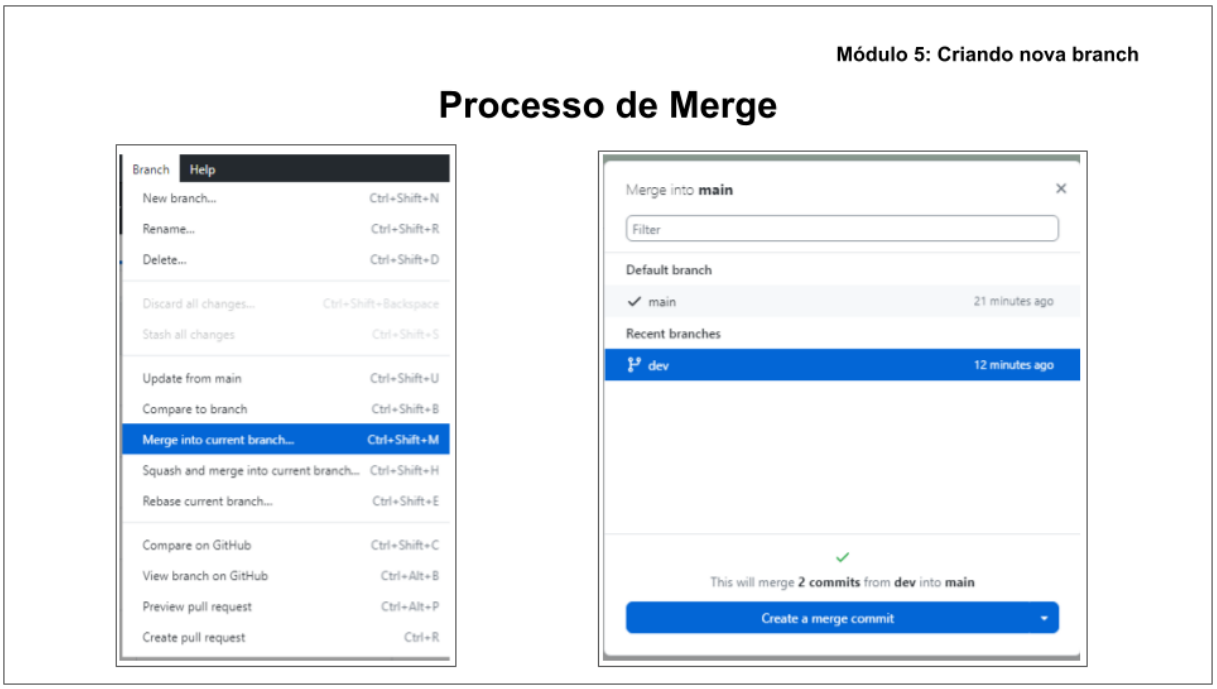
Como citado anteriormente, as branchs servem para realizar alterações no projeto sem alterar diretamente a main branch, pois, caso algo dê errado, não afetará o projeto principal.

As branchs 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”.



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.

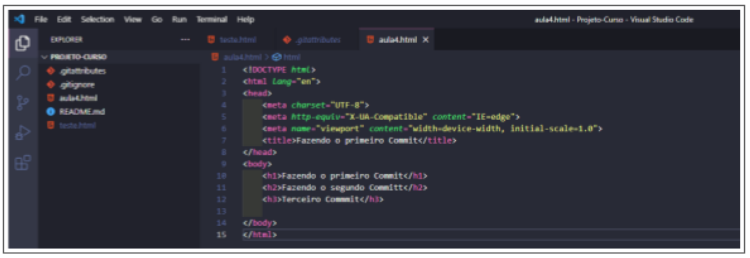
Caso queira criar uma branch a partir de outra branch que não seja a main, basta selecionar a branch desejada e repetir o mesmo processo, ou seja, clicar em “New branch” e irá aparecer uma tela perguntando a partir de qual branch você deseja criar outra branch.



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”.

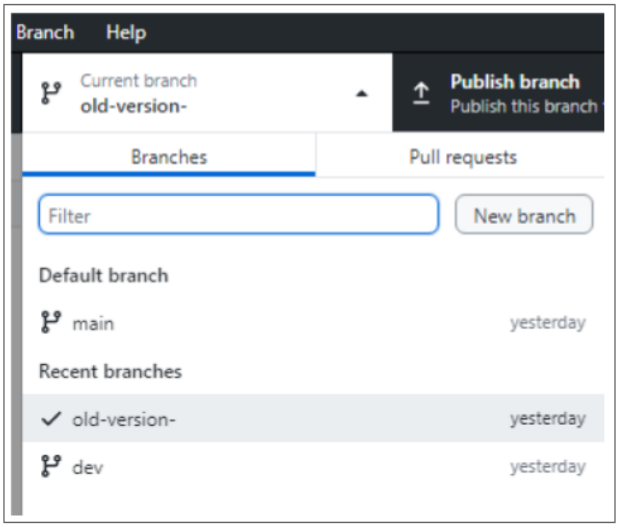
Módulo 5: Criando nova branch

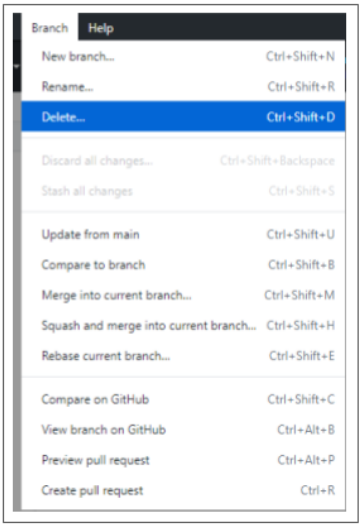
Processo de Merge



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.

Deletando uma Branch

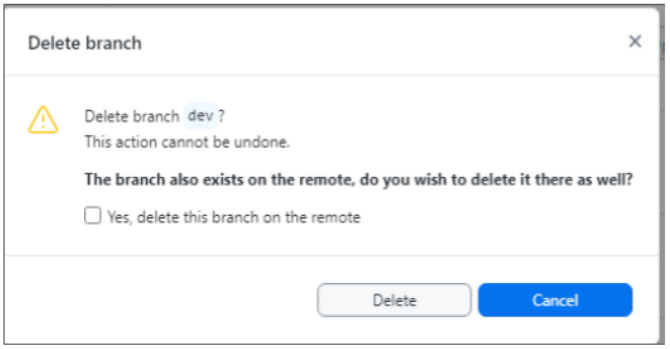


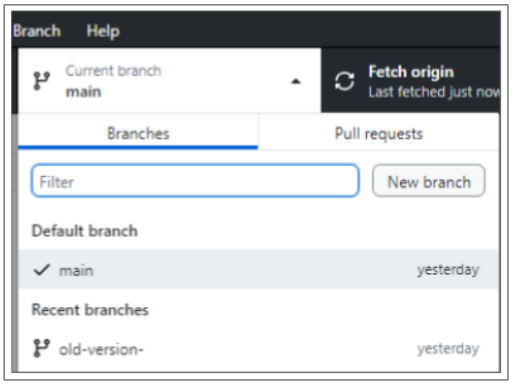


Para deletar uma branch, primeiro, selecione no GitHub Desktop a branch que deseja excluir, em seguida vá na aba “Branch”, e por fim, clique na opção “Delete”.

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

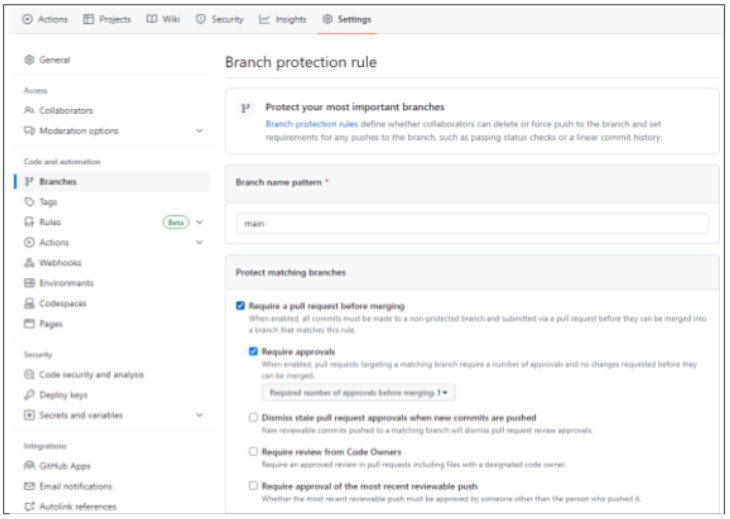
Deletando uma Branch





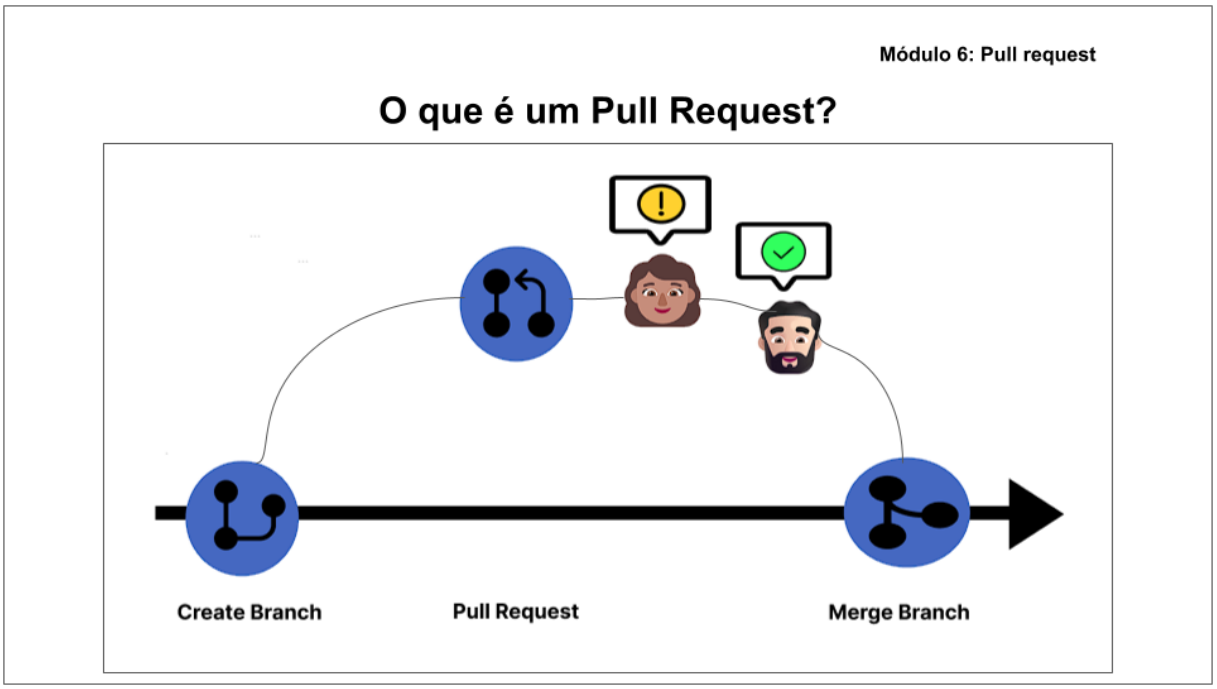
Neste aviso, é sinalizado que caso delete a branch, não será possível recuperá-la. Além disso, há a opção de deletar apenas no local e deixá-la no remoto ou deletá-la em ambos. Por fim, pode-se observar que a branch “dev”, não existe mais.

Protegendo uma Branch



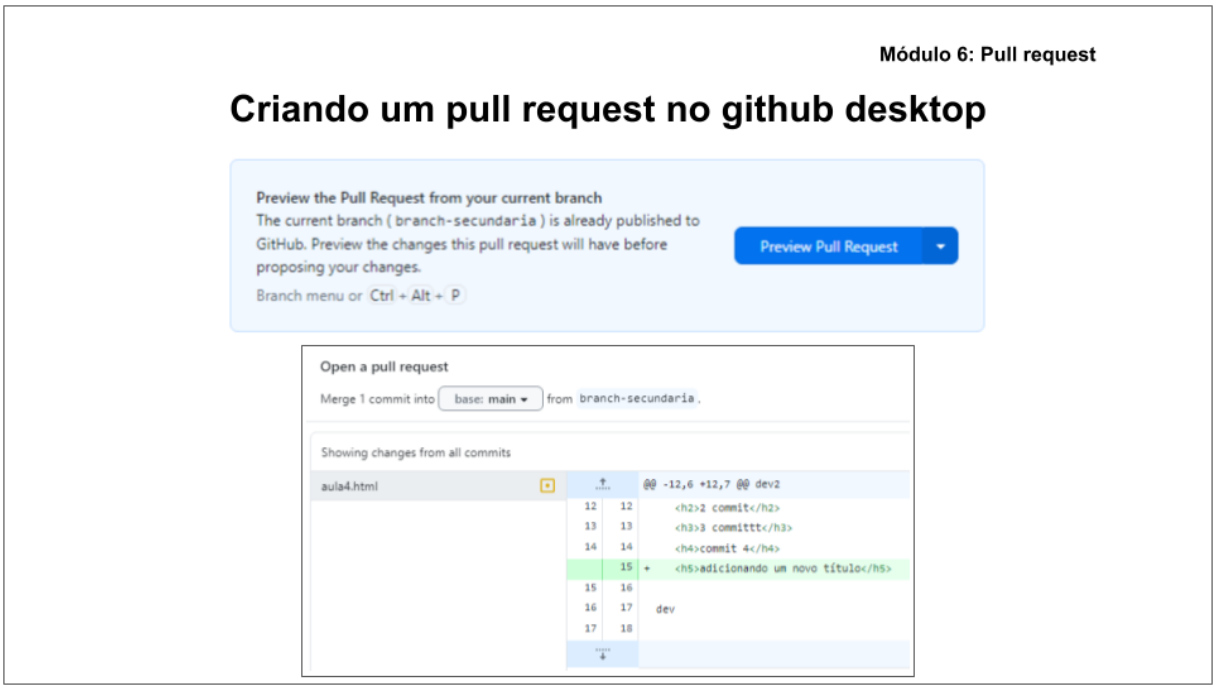
Para proteger uma branch, ou seja, exigir uma aprovação por meio do pull request para poder realizar o merge para a main branch, basta ir no repositório remoto em “Settings” e ir na opção “Branches”, e então clicar em “Add branch protection rule” e em seguida selecionar a opção “Require a pull request before merging” e “Require approvals”, para finalmente aplicar essa regra, é necessário definir em “Branch name pattern” o nome da branch que deseja proteger, por exemplo, é recomendado proteger a main branch, então basta digitar “main” em “Branch name pattern” e por fim, clicar em “Create” lá no final da página, após isso a sua main branch estará protegida.

Com isso, antes de realizar um merge será necessário uma aprovação da pessoa responsável por revisar as mudanças no repositório, e então, com a aprovação o merge será permitido.



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.

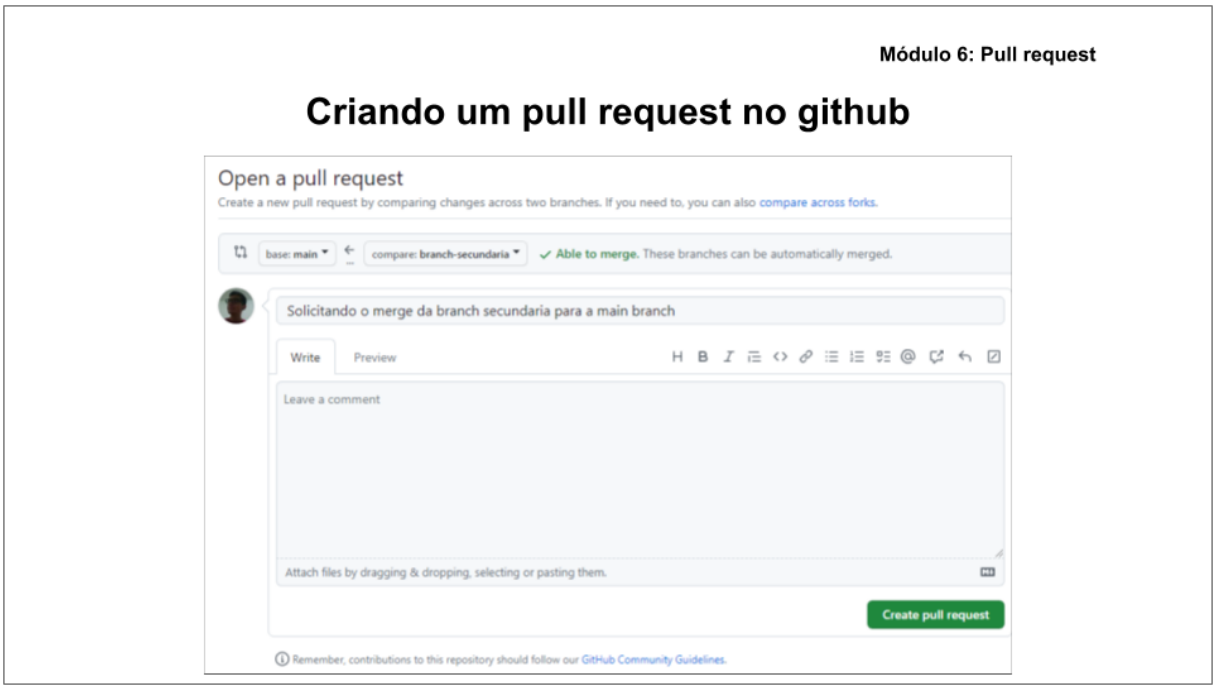
Criando um pull request no github desktop



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

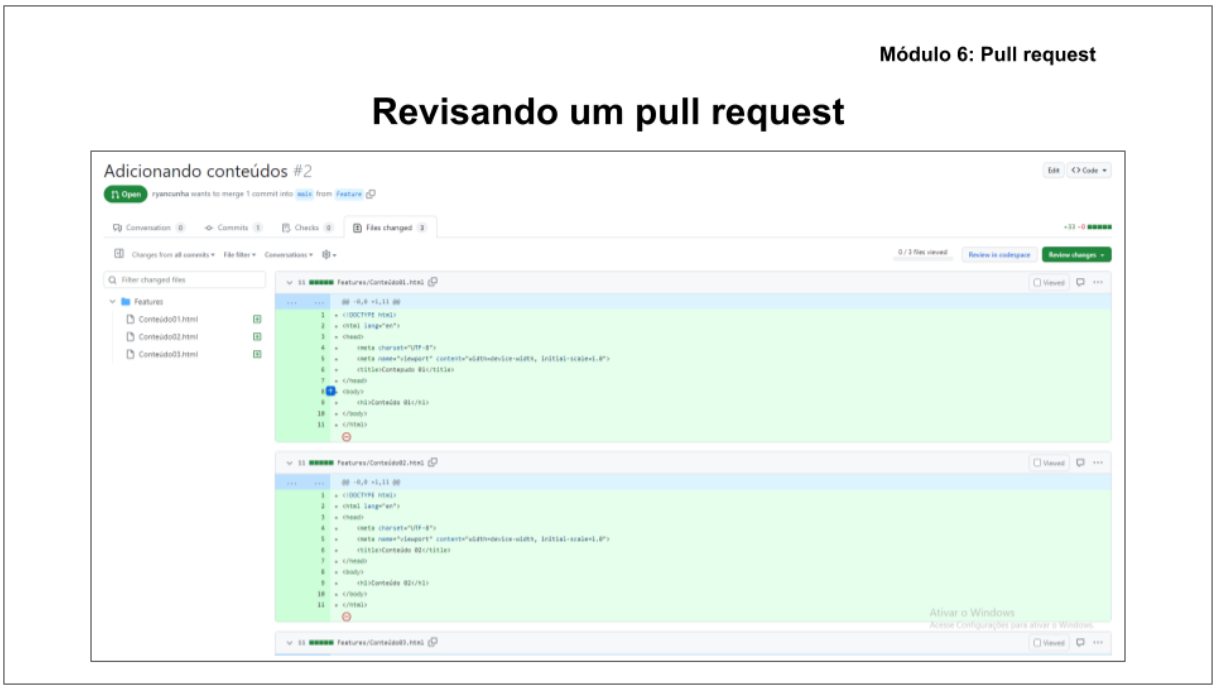
Então, após criar uma nova branch a partir da main branch, realize o push da nova branch e faça as alterações desejadas na mesma, após isso, faça o commit e o push das alterações, em seguida, no GitHub Desktop clique na opção “Preview Pull Request”.

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.



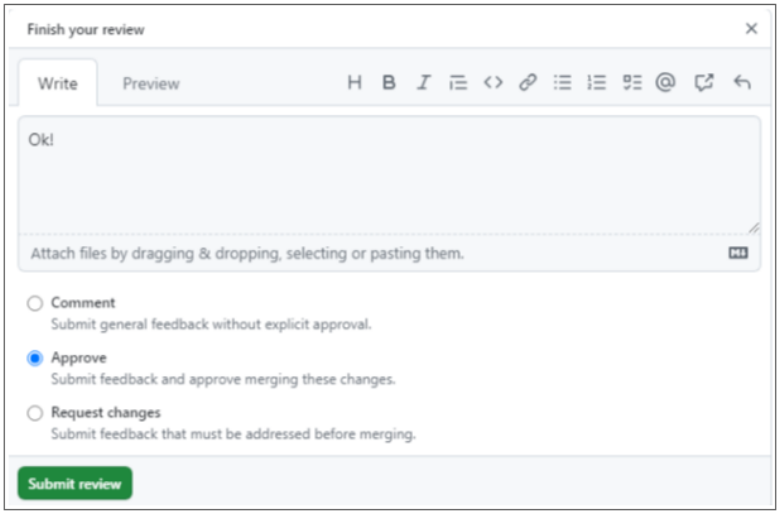
O GitHub Desktop irá lhe redirecionar para o GitHub, e lá você poderá adicionar mais informações sobre as mudanças realizadas, depois de tudo pronto, clique em “Create pull request” no GitHub.

Para realizar o merge de um pull request é necessário uma revisão aprovando o pull request, ou seja, uma revisão da pessoa responsável e com permissão para aprovar ou não um pull request. Assim, para revisar um pull request basta clicar em “Add your review”.



Depois de clicar em “Add your review” irá aparecer uma tela com as alterações daquele pull request, o responsável pelo review deve analisar as mudanças para determinar se irá aprová-las ou não. Após a análise das alterações, será feito finalmente o review, basta clicar em “Review changes”.

Aprovando um pull request



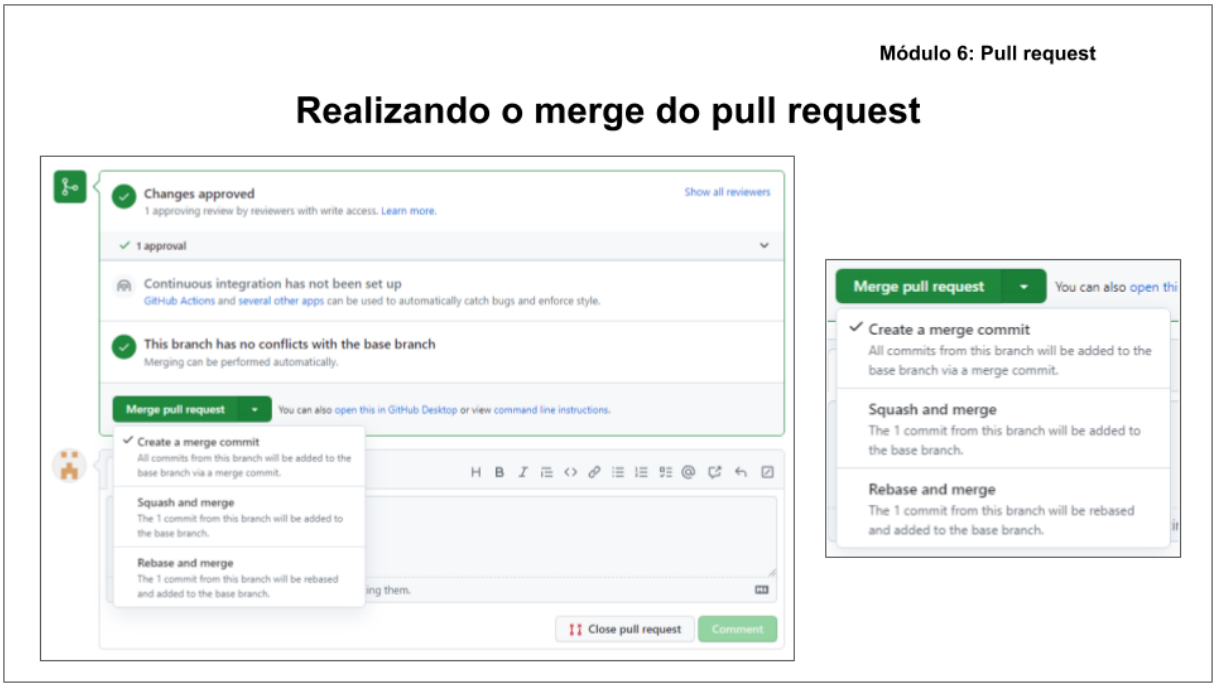
Após clicar em “Review changes” aparecerá uma tela para adicionar um comentário para o review, e as seguintes opções: comment, approve e request changes.

Comment, approve e request changes

- O “comment” é utilizado apenas para comentar e não para aprovar o pull request;

- O “approve” é utilizado para aprovar o pull request, ou seja, permite o merge do pull request;

- O “request changes” , como o nome já diz, serve para solicitar mudanças no pull request, ou seja, o revisor irá adicionar um comentário informando que o código precisa ser alterado e o que precisa alterar.



Após aprovado o pull request, o merge poderá ser realizado, no entanto, existem 3 formas de merge, que são: merge pull request, squash and merge e rebase and merge.

Merge pull request

O merge pull request irá mesclar todas as mudanças de uma determinada branch para a branch main, no entanto, em casos em que há mais de 1 commit para serem mesclados à branch-main, todos esses commits serão registrados individualmente no histórico do projeto.

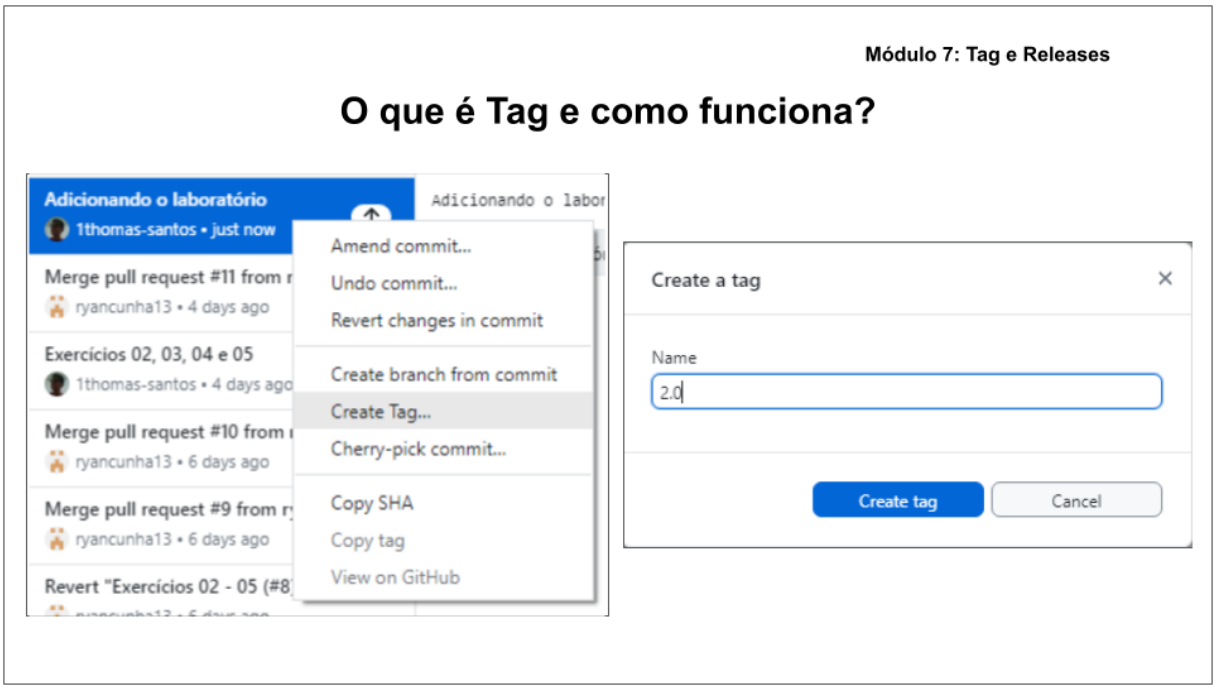
Squash and merge

O squash and merge assim como o merge pull request irá mesclar todas as mudanças de uma determinada branch para a branch main, no entanto, nos casos

em que há mais de 1 commit para serem adicionados à branch main, todos esses commits serão combinado em apenas 1 commit e mesclados à branch branch-main.

Rebase and merge

O rebase and merge assim como o merge pull request irá mesclar todas as mudanças de uma determinada branch para a branch main, no entanto, a principal diferença do rebase and merge é que essa opção reescreve o histórico da branch, criando um histórico linear do projeto, mantendo assim, o histórico mais limpo e organizado.



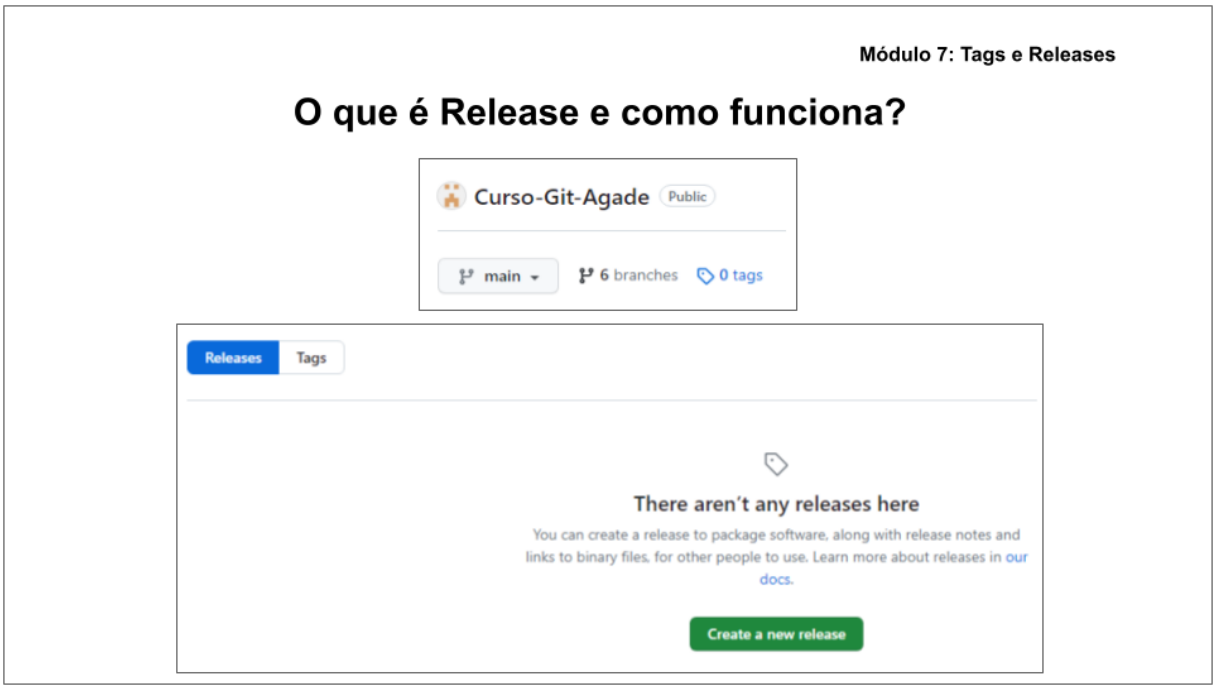
As tags servem para você destacar um commit e, a partir disso, criar versões do projeto.

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

Observações importantes:

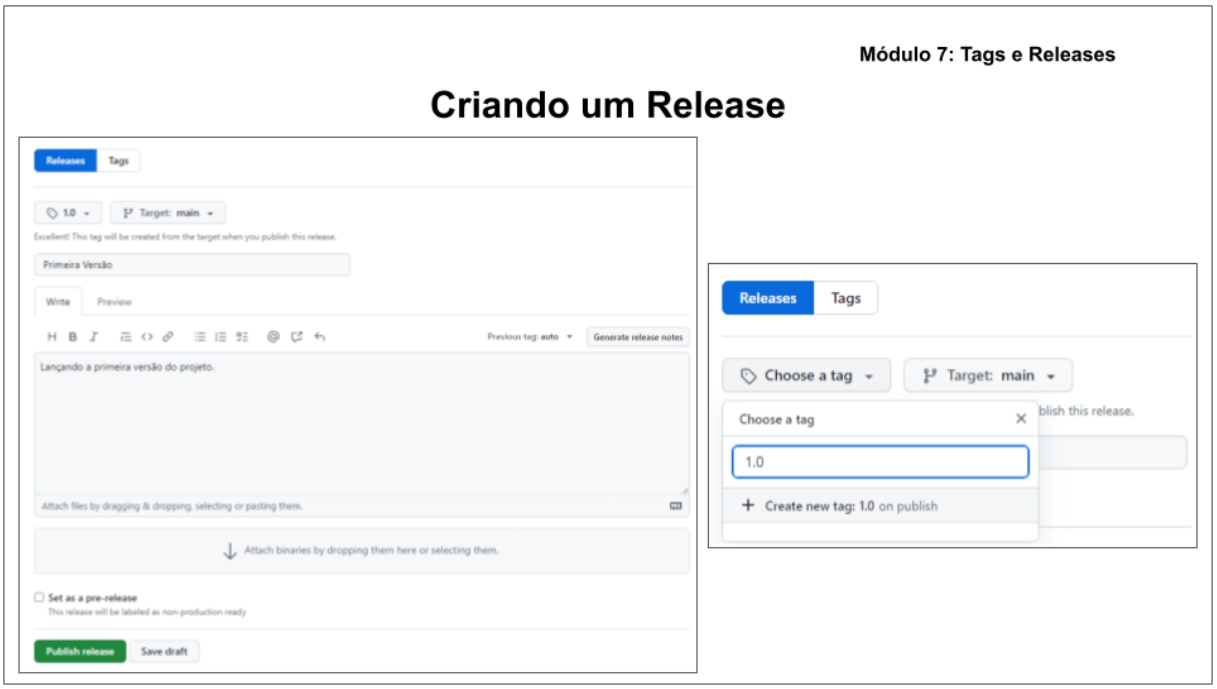
- Para deletar uma tag basta ir no commit que está com a tag e realizar o mesmo processo de criar, mas em vez de clicar em “Create Tag” clique em “Delete Tag”.

- Apenas é possível excluir completamente uma tag de um commit em um repositório local se a mesma for deletada antes de ser realizado o push.



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

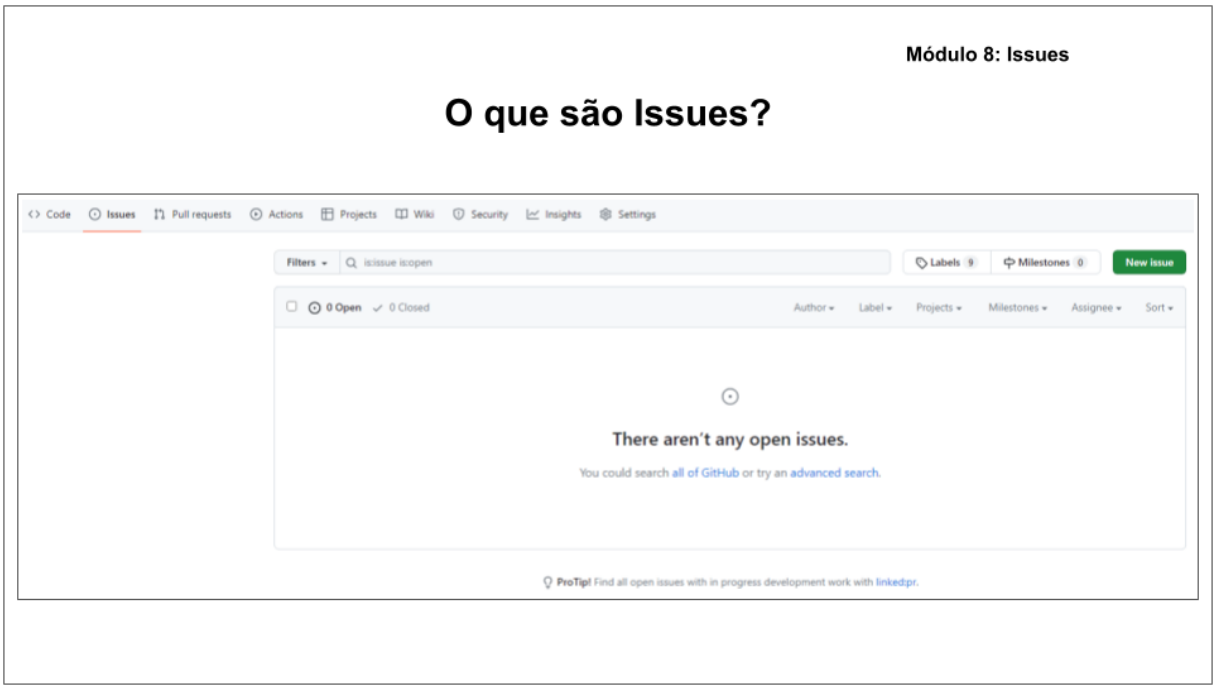
Para criar um release é necessário ir no repositório no site do GitHub e ao lado das branchs, clicar em “tags”. Após isso, você seleciona a opção “Releases” e clica em “Create a new release”.



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

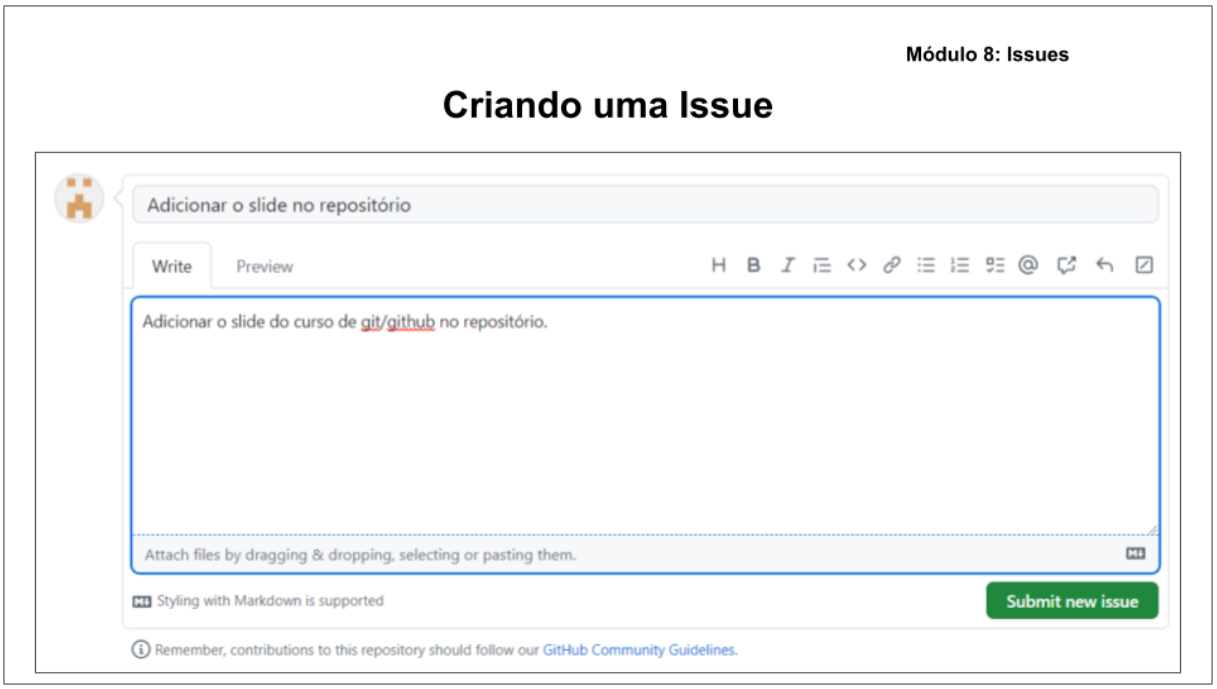
E principalmente, selecionar a tag desejada, ou, como citado anteriormente você pode criar uma tag diretamente no release, basta escolher o número da tag (versão) e clicar em “Create new tag”. Com todas as informações do release presentes e a tag escolhida, basta clicar em “Publish release” e ele será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

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.

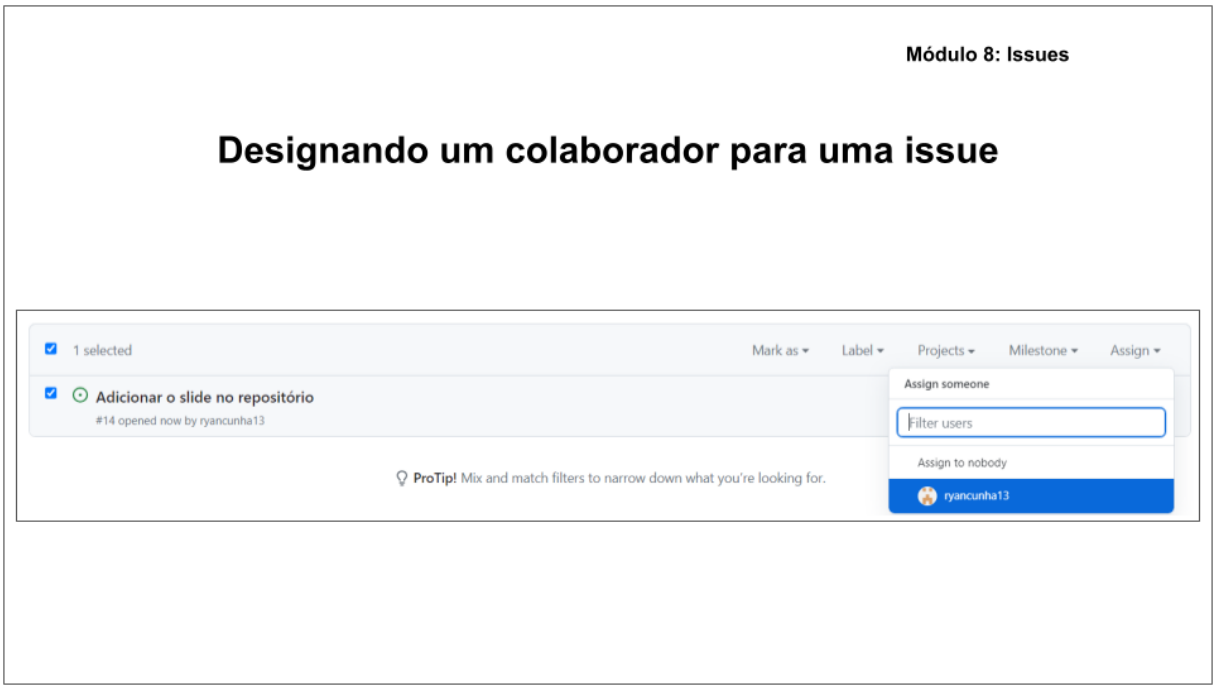


As issues servem para auxiliar na construção de um determinado projeto ou receber feedbacks dos usuários sobre o projeto. Isso é possível, pois, as issues permitem aos usuários escreverem comentários com imagens, links, emojis, entre outros, que tornam o feedback muito mais construtivo.

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

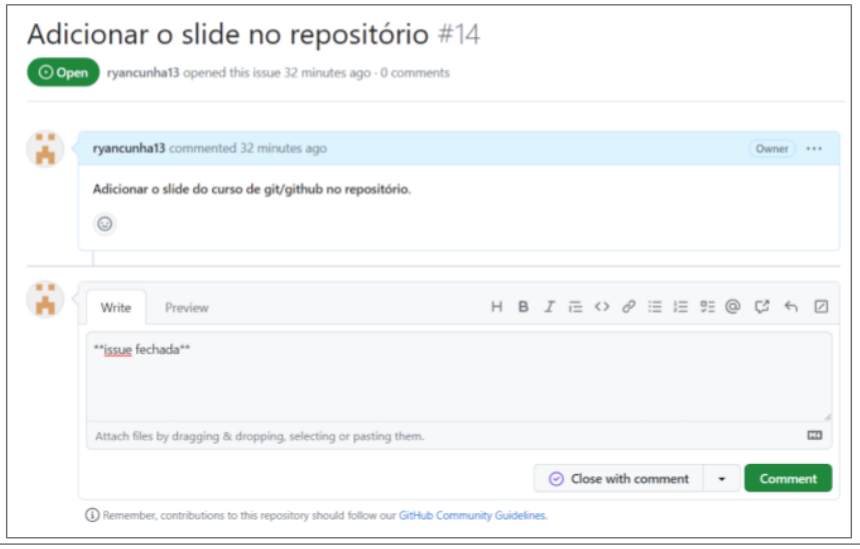


Após isso, basta colocar as informações da issue, como o título da issue e um comentário sobre a mesma. Em seguida, clique em “Submit new issue” e a issue será criada.

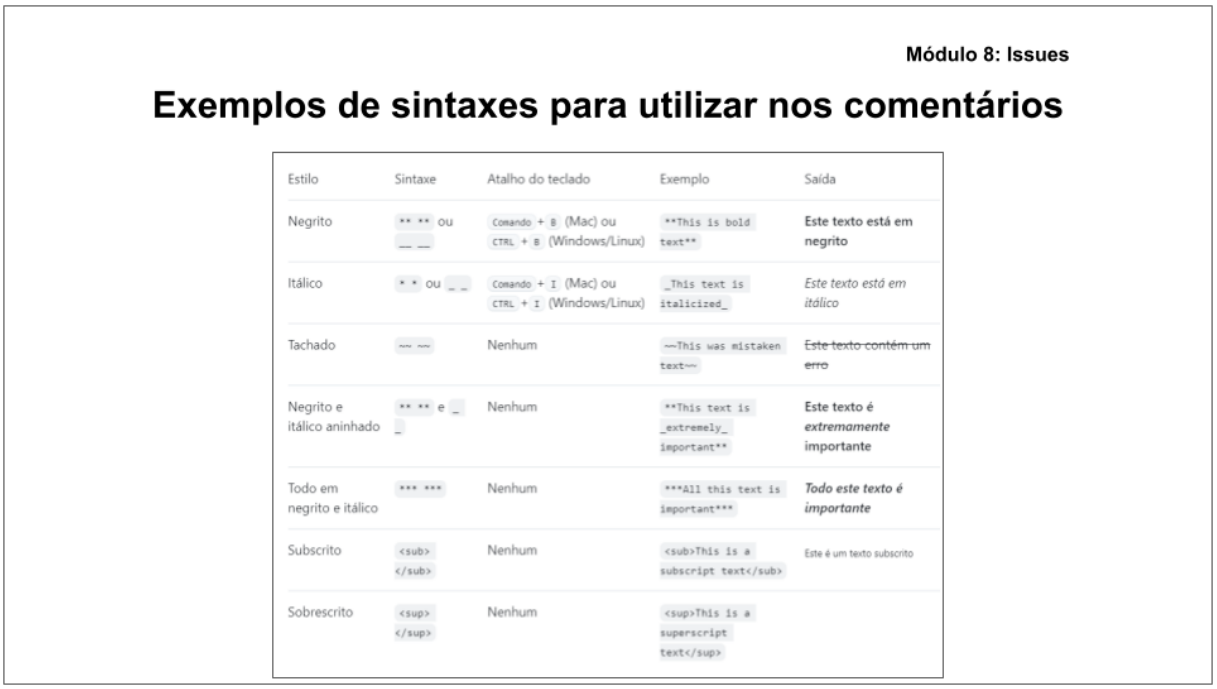


Também é possível designar um colaborador do projeto para resolver uma determinada issue. Para isso, após criada a issue, basta ir na aba issues novamente e lá, selecionar a issue desejada e clicar em “Assign” e então selecionar o colaborador que ficará encarregado por aquela issue.

Fechando uma issue



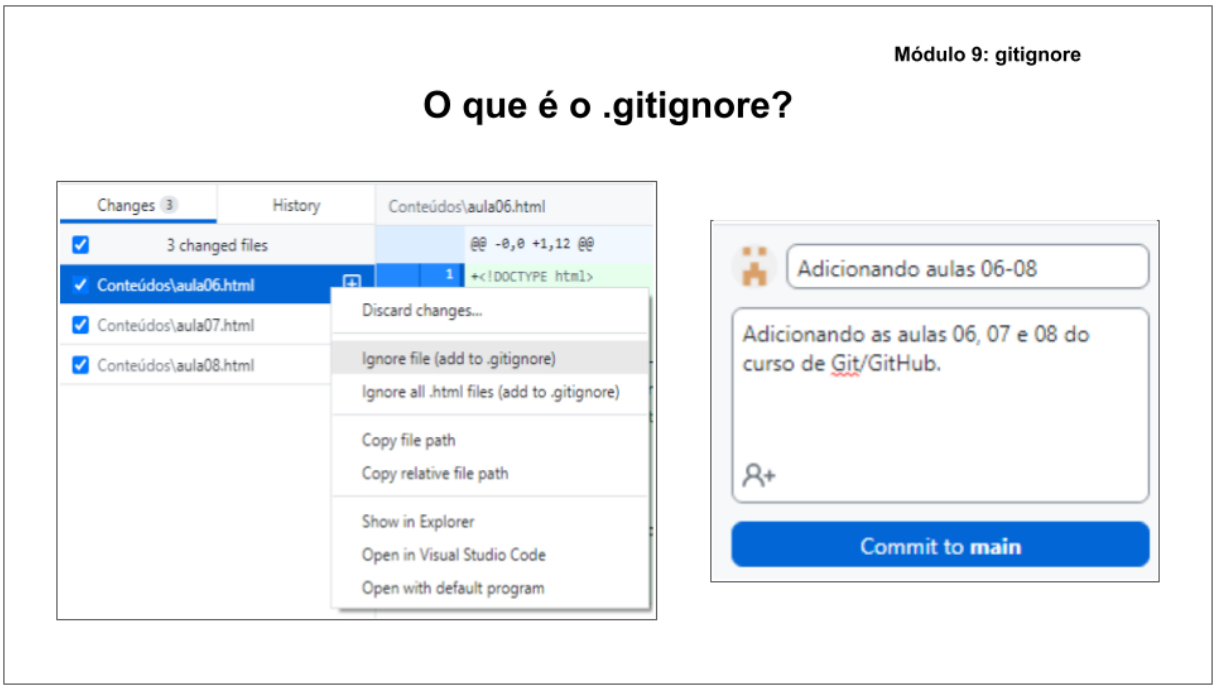
Após resolvida uma issue, é necessário fechá-la, para isso basta abrir a issue no GitHub e clicar em “Close issue”, ou, caso adicione um comentário no momento de fechar, clique em “Close with comment”.



Exemplos de sintaxes para utilizar nos comentários

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

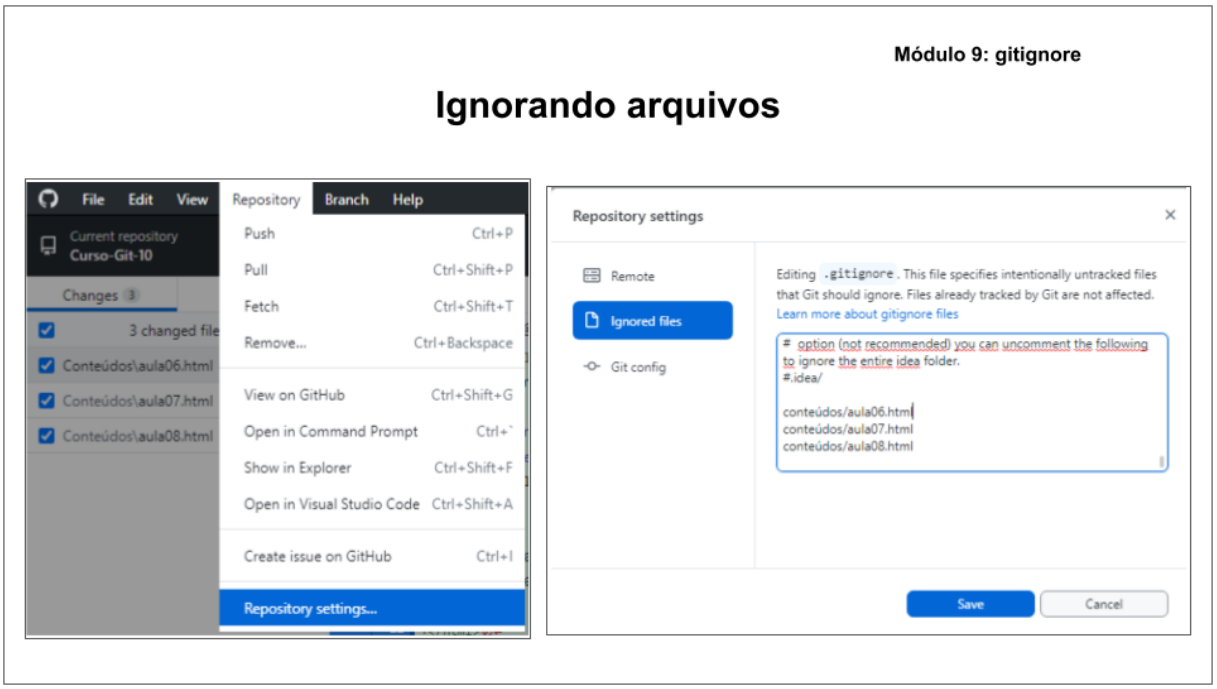
Na imagem acima temos alguns exemplos de sintaxes que podem ser utilizados nos comentários para destacá-los e torná-los mais organizados.



O gitignore como o nome já diz, é um comando utilizado para ignorar determinado arquivo ou conjunto de arquivos de um repositório, impedindo o commit desses arquivos do repositório local para o repositório remoto, ou seja, impedindo que vá para o GitHub.

Ele é utilizado geralmente quando existe algum arquivo no repositório local que não é relevante para o repositório remoto ou que não é do interesse do projeto que ele seja enviado para o GitHub.

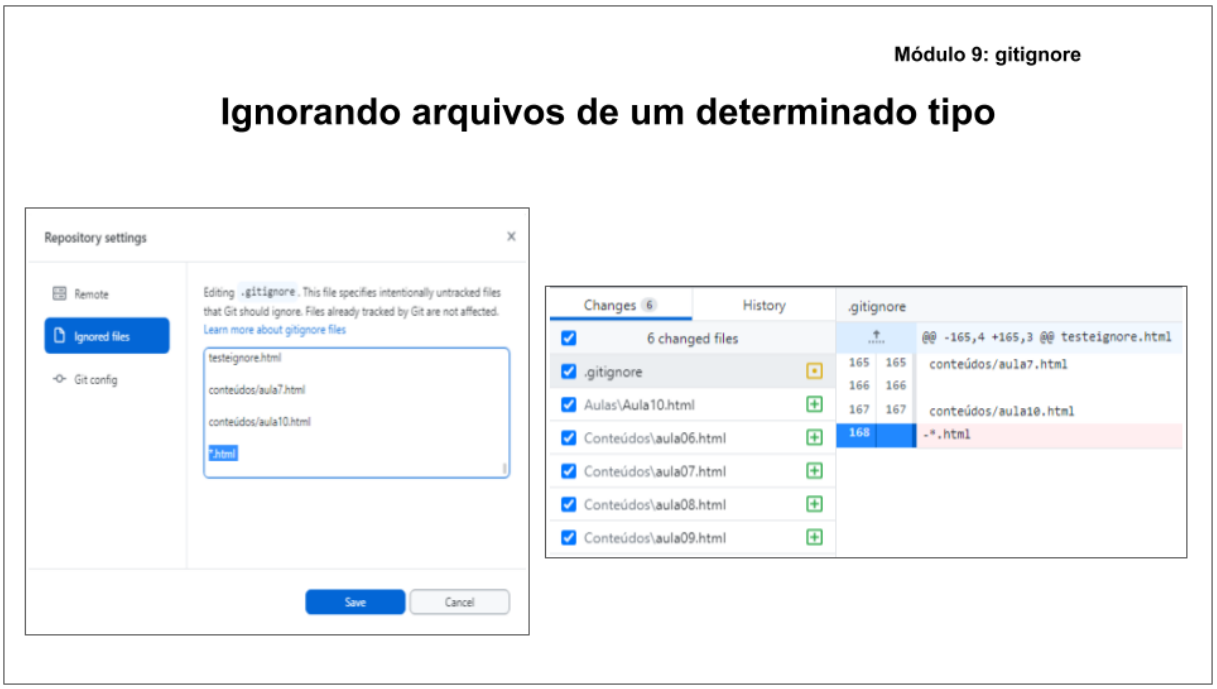
Existem duas formas de ignorar determinado arquivo, a primeira, no momento antes de realizar o commit, em “Changes”, basta clicar com o botão direito no arquivo desejado e selecionar a opção “Ignore file (add to .gitignore), em seguida, basta realizar o “Commit” dessa alteração e o arquivo será adicionado ao .gitignore e nãos erá enviado para o repositório remoto.



A segunda forma de ignorar determinado arquivo é, no momento antes de realizar o commit, vá na opção “Repository” no GitHub Desktop, clique em “Repository settings” e selecione a opção “Ignored files”, no final dessa caixa de texto você digita o “Relative Path” do arquivo (próximo slide ensina como copiar o “Relative Path” de um arquivo).

Copiando o “Relative Path” de um arquivo

Para copiar o “Relative Path” do arquivo basta ir no repositório local no Visual Studio Code e clicar com o botão direito do mouse sobre o arquivo desejado e clicar em “Copy Relative Path”, e então cole-o no “Ignored files” no GitHub Desktop.



Caso queira ignorar não apenas um arquivo específico mas um determinado tipo de arquivo, como por exemplo, ignorar todos os arquivos html do repositório, você pode no momento antes de realizar o commit de um arquivo html, você clica com o botão direito no commit e seleciona a opção “Ignore all .html files (add to .gitignore)” ou você pode fazer do outro jeito que foi mostrado anteriormente, “Repository settings”, “Ignored files” e lá você digita “*.html”, e então, após essas mudanças todos os arquivos .html que seriam commitados no repositório, serão ignorados. Caso não queira mais ignorar os arquivos .html, basta ir no “Repositor settings”, Ignored files” e excluir o comando “*.html”, assim, os próximos arquivos html não serão mais ignorados.

Observações importantes:

Os arquivos só podem ser ignorados antes de realizar o commit, ou seja, antes de enviar o arquivo criado para o repositório.

Seguindo a linha de raciocínio citada anteriormente, apenas os arquivos que ainda não foram commitados que podem ser ignorados, ou seja, caso o seu repositório já tenha vários arquivos .html que foram commitados e, a partir de determinado

momento você deseja ignorar todos os arquivos .html, os arquivos .html que já foram commitados não serão ignorados, apenas os arquivos que ainda não foram commitados ou os que serão criados futuramente que serão ignorados.

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 branchs "Error” e Criar a branch “fix”

Material complementar

https://git-scm.com/about

https://git-scm.com/doc

https://docs.github.com/pt

https://docs.github.com/pt/get-started/quickstart/hello-world