

VERSIONAMENTO GIT, ATRASSO & INTERVALO



DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

- **Revisão** (Funções & Eventos)
- **Versionamento**
 - Git
 - GitHub & GitHub Desktop
- **Intervalos de tempo**
 - setTimeout
 - setInterval

- **Funções**

- `function somarUm(n) { return n + 1; }`
- `somarUm(6); /* retorna 7 */`

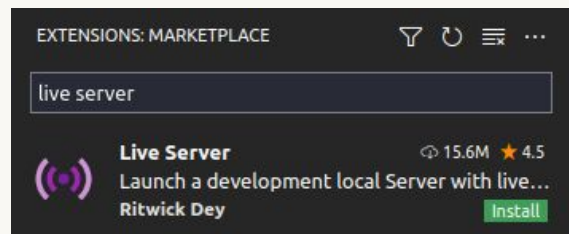
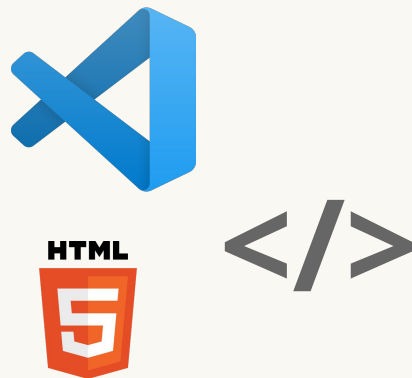
- **Eventos**

- `onclick="funcao(event)" /* EVITAR UTILIZAR */`
- `elemento.onclick = funcao;`
- `elemento.addEventListener('click', funcao);`
- `elemento.removeEventListener('click', funcao);`

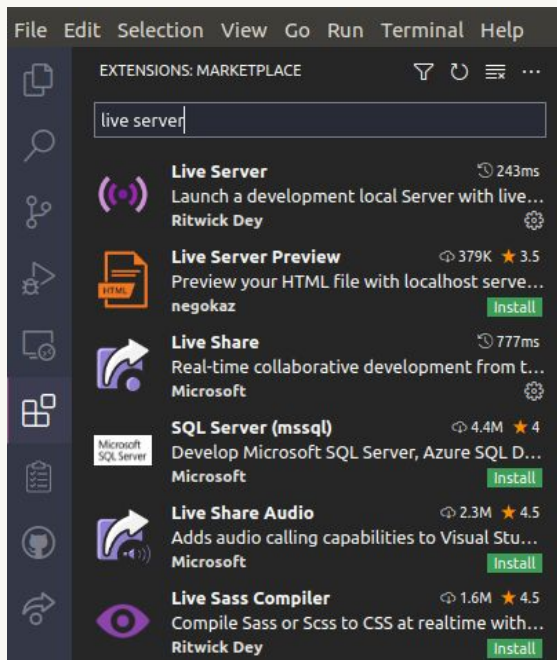
PARA A MÃO NA MASSA

- Instalar **VS Code**
(ou outro editor que se sentir mais confortável)
<https://code.visualstudio.com>
- Sugestão: Instalar extensão **Live Server** no **VS Code**
- Criar um arquivo **index.html** no seu editor

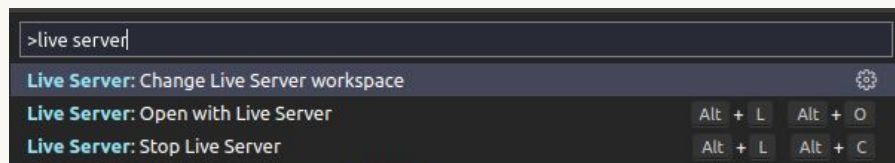
Code Sandbox | <https://codesandbox.io>
PlayCode | <https://playcode.io/new>
CodePen | <https://codepen.io/pen>
JSFiddle | <https://jsfiddle.net>



PARA A MÃO NA MASSA

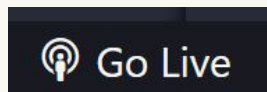


- **Ctrl + Shift + P**
Live Server: Open with Live Server



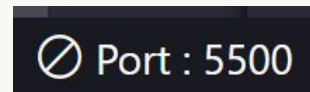
Start

- **Alt + L**
- **Alt + O**



Stop

- **Alt + L**
- **Alt + C**





- **O que é?**
- **Para que serve Git?**
- **Que problemas veio resolver?**

- Git é um sistema de controle de versão de arquivos.
- Com ele, podemos manter um histórico de todas as alterações que foram realizadas em nossos arquivos de código fonte.
- Em equipe, podemos visualizar todas as alterações realizadas por todos os membros.

- Cada nova funcionalidade (ou correção) que você adicionada na sua aplicação, você salva nesse sistema de controle.
- Muitas vezes, uma nova funcionalidade/correção significa vários arquivos alterados de uma só vez.
- Ao final do desenvolvimento de cada nova funcionalidade ou correção, salvamos o estado dos arquivos naquele momento.

VERSIONAMENTO

- Cada "**salvar**" desses, fica registrado em um "**pacotinho de alterações**". É o que chamamos de "**commit**".
- Quando você quiser voltar o estado da sua aplicação para um ponto específico do passado, ou apenas visualizar como era esse código em algum momento passado, você faz isso facilmente.
- Seja para comparar uma versão do código que funcionava com outra que não está mais funcionando, resolvendo um bug.
- Ou apenas descobrir o que foi alterado de uma versão para outra, por diversas outras razões.

VERSIONAMENTO

- Exemplos de commits:

The screenshot displays a Git commit history interface. At the top, there is a branch selector showing 'main'. Below it, a filter for 'Commits on Dec 2, 2020' is applied. The commit history is listed as follows:

Commit Message	Author	Time	Verification	File Icon	Commit Hash	Diff Icon
ajustes de posicionamento, remoção de comentários e inserção de place...	joaovmoliveira	committed 2 minutes ago			d6db790	<>
ex. lista mercado - primeira versao: adiciona na lista, salva e carre...	joaovmoliveira	committed 3 hours ago			fea524c	<>
Update README.md	joaovmoliveira	committed 4 hours ago	Verified		02a6f95	<>
Initial commit	joaovmoliveira	committed 4 hours ago	Verified		900bf74	<>

VERSIONAMENTO

```
▼ 9 ■■■■ lista_mercado/estilo.css [icon]
```

```
... @@ -0,0 +1,9 @@
```

```
1 + h1 {
2 +   text-align: center;
3 + }
4 +
5 + .container {
6 +   text-align: center;
7 +   margin: 10px;
8 + }
9 +
```

- Exemplo de um commit mostrando as alterações entre a versão anterior (vermelho) e a versão nova (verde).

VERSIONAMENTO

9 lista_mercado/index.html	
@@ -12,15 +12,16 @@	
12	<h1>Lista de Mercado</h1>
13	</header>
14	<section>
15	- <div>
16	<label for="txtItem">Item a adicionar:
	</label>
17	- <input type="text" id="txtItem">
18	<button id="btnAdd">Adicionar</button>
19	</div>
20	- <div>
21	<select name="listaMercado" id="sellista">
22	</select>
23	</div>
24	</section>
25	</body>
26	- </html>
12	<h1>Lista de Mercado</h1>
13	</header>
14	<section>
15	+ <div class="container">
16	<label for="txtItem">Item a adicionar:
	</label>
17	+
18	+ <input type="text" id="txtItem"
	placeholder="Digite um item...">
19	<button id="btnAdd">Adicionar</button>
20	</div>
21	+ <div class="container">
22	<select name="listaMercado" id="sellista">
23	</select>
24	</div>
25	</section>
26	</body>
27	+ </html>

VERSIONAMENTO

6 lista_mercado/script.js

@@ -20,8 +20,8 @@ function salvarNoLocalStorage(item) {

20 }

21

22 function criarElementoLista(itemText) {

23 - var optItem = document.createElement('option'); //

<option></option>

24 - optItem.textContent = itemText; //

<option>itemMercado</option>

25 sellista.appendChild(optItem);

26 }

27

@@ -43,4 +43,4 @@ function verificaTecla(event) {

43

44 btnAdd.addEventListener('click', adicionarNaLista);

45

46 - txtItem.addEventListener('keyup', verificaTecla);

20 }

21

22 function criarElementoLista(itemText) {

23 + var optItem = document.createElement('option');

24 + optItem.textContent = itemText;

25 sellista.appendChild(optItem);

26 }

27

43

44 btnAdd.addEventListener('click', adicionarNaLista);

45

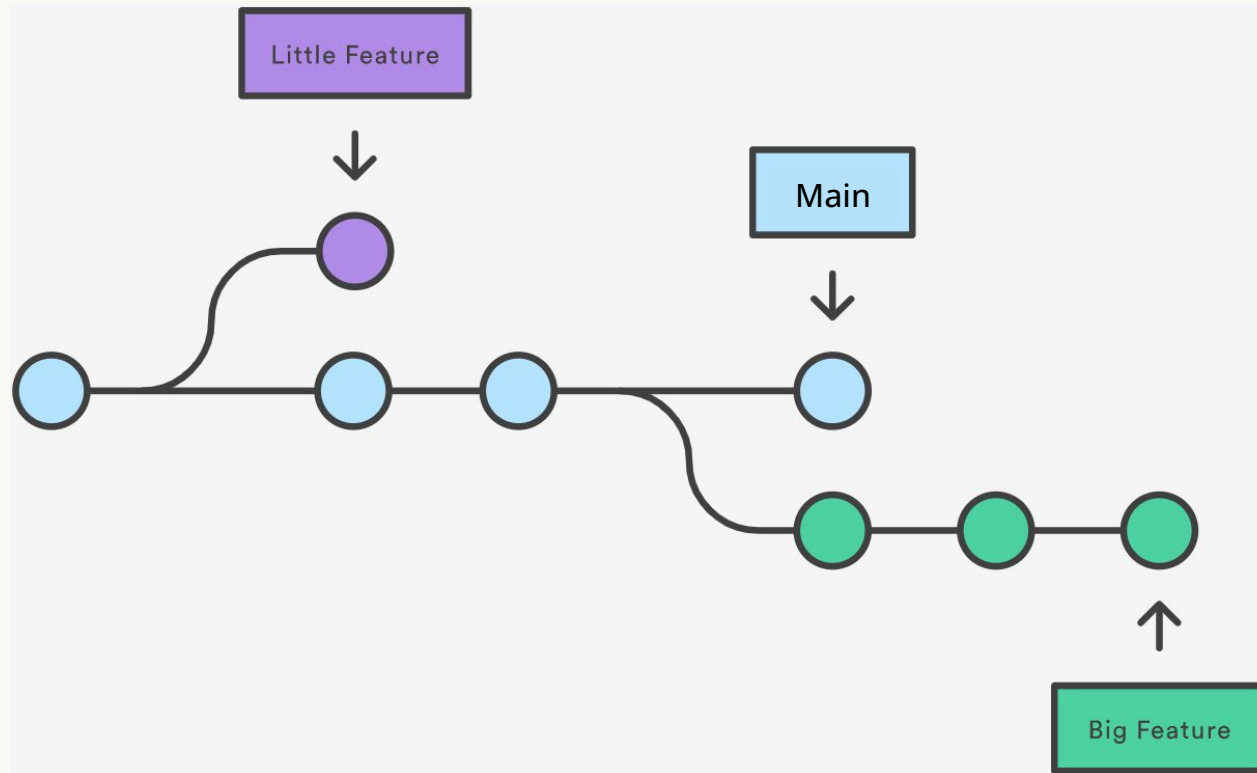
46 + txtItem.addEventListener('keyup', verificaTecla);

- Todos os nossos arquivos, assim como seus históricos, ficam salvos num **repositório**.
- Todos os colaboradores do projeto têm acesso ao repositório completo, e possuem uma cópia (clone) do repositório inteiro na sua própria máquina.
- Porém o Git só mostra uma versão por vez, o restante fica “escondido”.
- Todos os colaboradores do código tem acesso a todos os arquivos.
- Todos os colaboradores do código tem acesso ao histórico de alterações (os commits) de todos os outros.

- Antes do Git, já existiam outros sistemas de controle de versão de arquivos, como CVS e SVN.
- **Git** foi criado por **Linus Torvalds**, o criador do **Linux**.
- Torvalds criou esse sistema para corrigir alguns problemas que ele enxergava nos sistemas já existentes.
- Atualmente, é o sistema mais comum para controlar e compartilhar código fonte distribuído.

- **Branch:** dividimos nosso trabalho em "**branches**", que são "ramificações" do nosso código.
- Quando vários colaboradores estão implementando alterações ao mesmo tempo, se todos trabalharem no mesmo conjunto de arquivos, um pode atrapalhar o outro.
- Utilizando branches, podemos criar "cópias" independentes do código, e cada um trabalha na sua branch, na sua cópia do código original.
- Depois que o trabalho estiver finalizado naquela sua cópia, você pode submeter sua cópia para mesclar com o código original, o que chamamos de "**merge**".

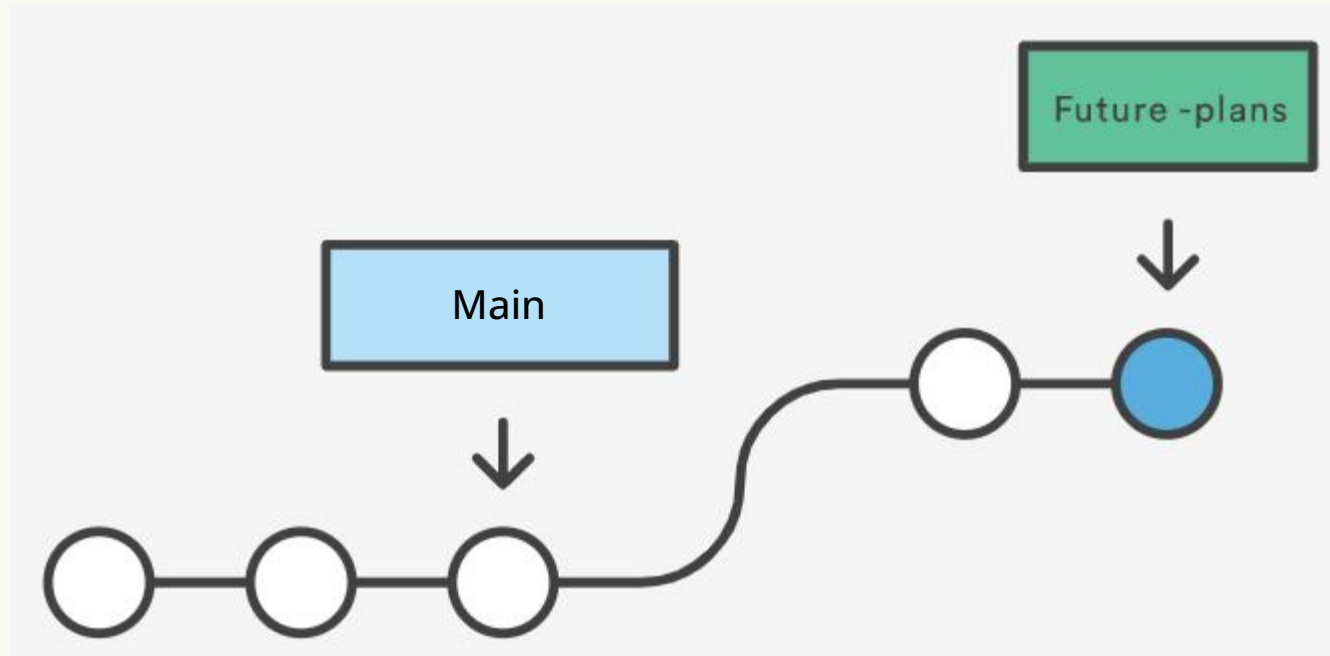
VERSIONAMENTO | Branch



Fonte: [Git Branch - Atlassian Bitbucket](#)

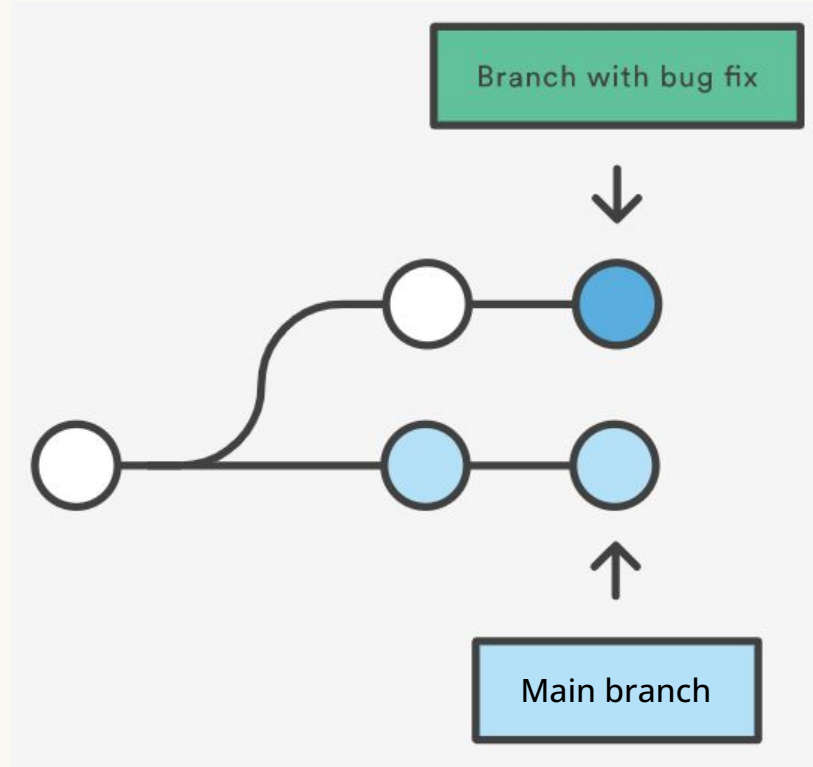
- **Merge:** é a ação de "mesclar" o conteúdo de uma branch com o conteúdo da outra.
- A sacada que o Linus teve quando criou o Git foi de que toda branch leva consigo a informação de qual era o último commit da branch original no momento da criação dessa nova branch.
- Ou seja: ao fazer um merge o Git pode comparar o conteúdo das duas branches a partir desse commit específico, em vez de comparar os arquivos inteiros.
- Isso diminui drasticamente a quantidade de conflitos gerados por merge. E quando acontece um conflito, é muito mais fácil resolvê-lo.

VERSIONAMENTO | Merge



Fonte: [Tutorial Git - Atlassian Bitbucket](#)

VERSIONAMENTO | Merge



Fonte: [Tutorial Git - Atlassian Bitbucket](#)

- **Pull Request:** é uma solicitação de que a branch principal “puxe” a sua branch alternativa.
- Basicamente, é uma solicitação de merge que você envia da sua branch para a principal.
- Um mecanismo de segurança, para que os outros colaboradores revisem suas alterações antes de mesclá-las com a branch principal.

- GitHub é um servidor que implementa o sistema Git.
- Acessado através da URL <https://github.com>
- Permite criar um repositório remoto, e a partir desse repositório, todos os colaboradores têm acesso ao código fonte.
- Existem outros, como: GitLab, Bitbucket.
- GitHub foi comprado pela Microsoft recentemente, e surpreendentemente, a empresa é a maior colaboradora em projetos open-source na plataforma.

- **Tarefa:** Quem ainda não tiver uma conta no GitHub, crie uma.
- Acesse <https://github.com> e crie sua conta.
Pode ser com o e-mail do SENAI.
Se já tiver conta, podes acrescentar o e-mail do SENAI.
- Vamos praticar os conceitos vistos nesta aula.
- Caso se interesse em utilizar Git por interface gráfica, baixe e instale o **GitHub Desktop**, é a ferramenta que utilizaremos:
<https://desktop.github.com>

- **remote** (remoto)
 - Instância principal do projeto que não está em nossa máquina
- **local**
 - Instância do projeto que está em nossa máquina
- **branch** (ramo)
 - Ramificação nomeada do projeto a partir de um commit
- **commit** (compromisso/entrega)
 - Pacote de mudanças a serem registradas/integradas

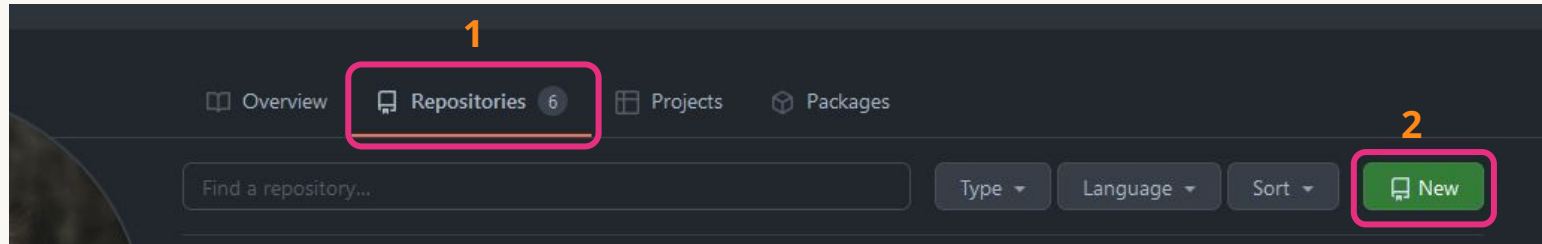
- **push** (empurrar)
 - Ação de envio das alterações locais para o remoto
- **pull** (puxar)
 - Ação de "baixar" alterações do remoto para o local
- **fetch** (buscar)
 - Ação de buscar branches/tags/dados atualizadas do projeto
- **merge** (fundir)
 - Ação de mesclar o conteúdo de uma branch com outra

- O GitHub possui um cliente com interface gráfica, para que a gente possa utilizar o Git sem precisar da linha de comando.
- Faça o download do "**GitHub Desktop**" através da URL: <https://desktop.github.com>
- Agora vamos abrir o GitHub Desktop e fazer login com a nossa conta GitHub.

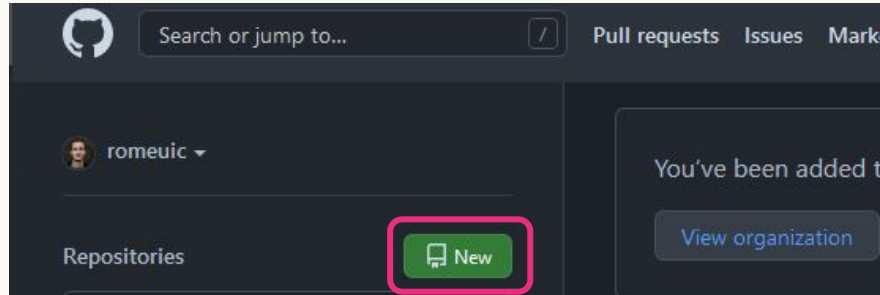
- Vamos repetir o fluxo que vimos pelo **GitHub Desktop**.
- Existem várias formas de criar um repositório novo, mas vamos pelo mais fácil: pelo site do GitHub.
 - **Lembre** de marcar para criar junto do repo um arquivo **README.md**

- <https://github.com/new>

ou





ou



Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner * **Repository name ***

 romeuic / exemplo-aula 

Great repository names are short and memorable.

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.


Initialize this repository with:


Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

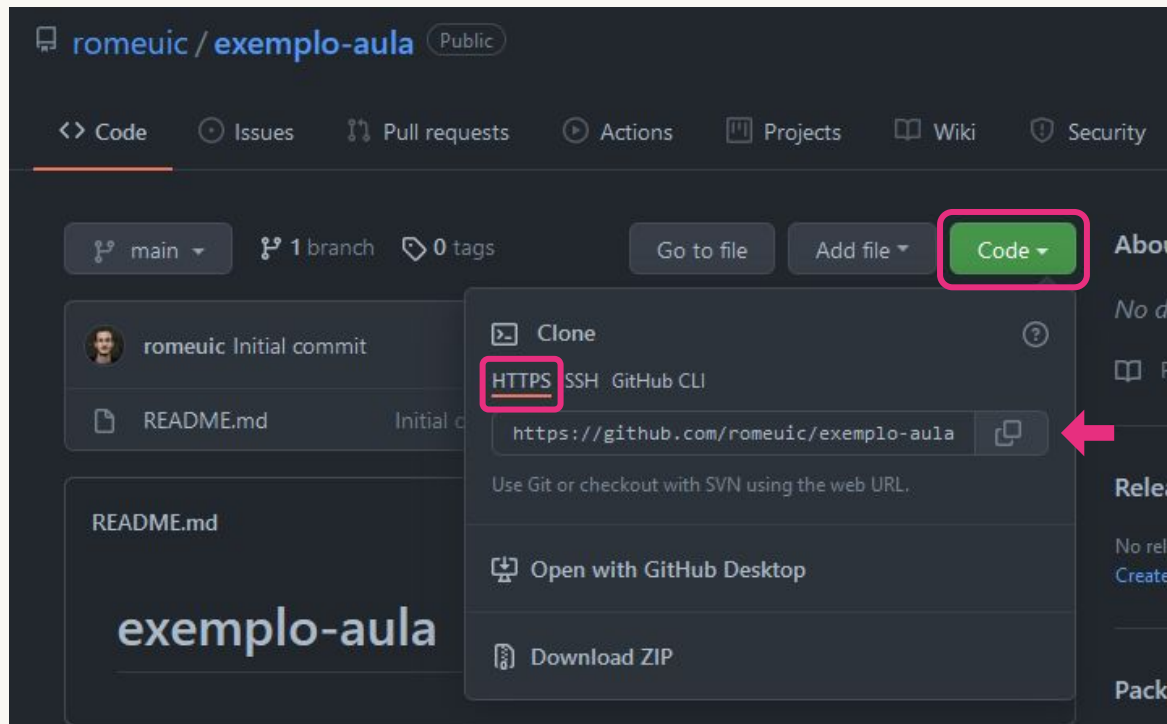
☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

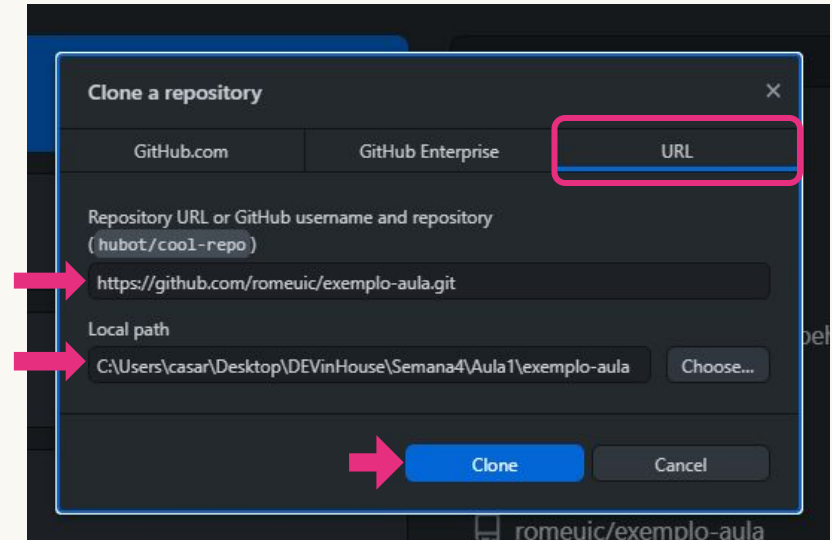
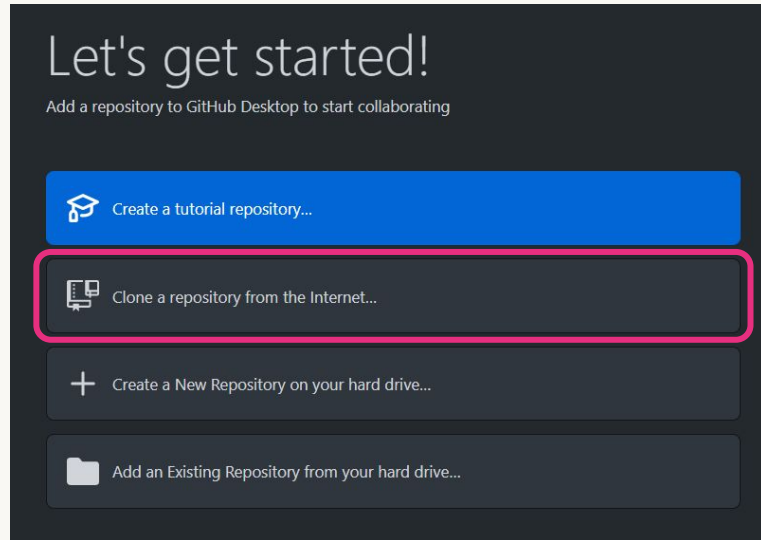
[Create repository](#) 

Escolha se deseja deixar seu repositório visível para qualquer um (**public**), ou apenas para quem você convidar (**private**)

GITHUB | Clone

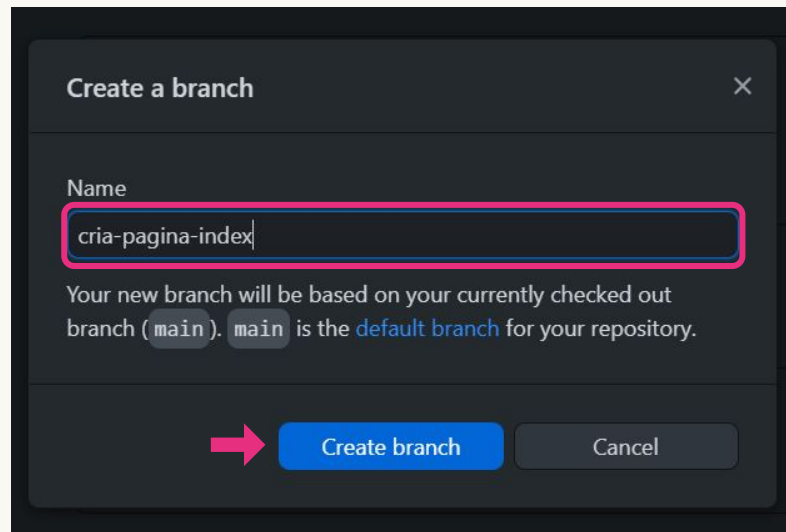
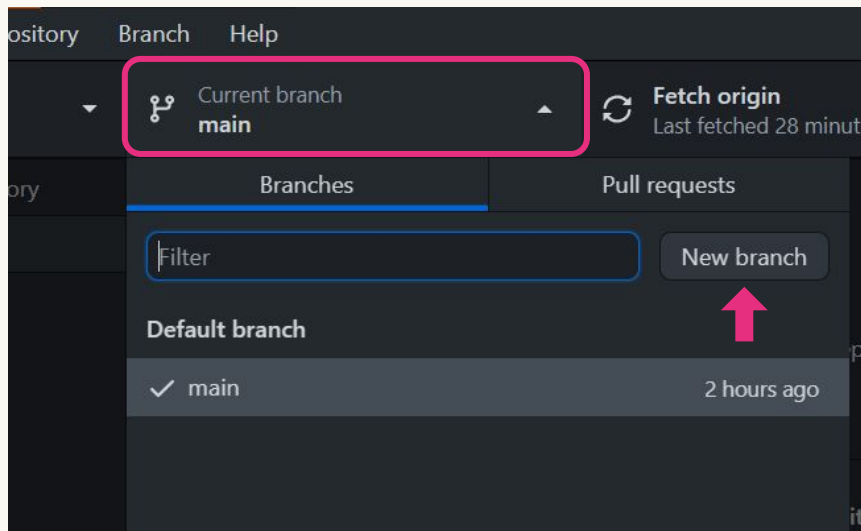


GITHUB DESKTOP | Clone

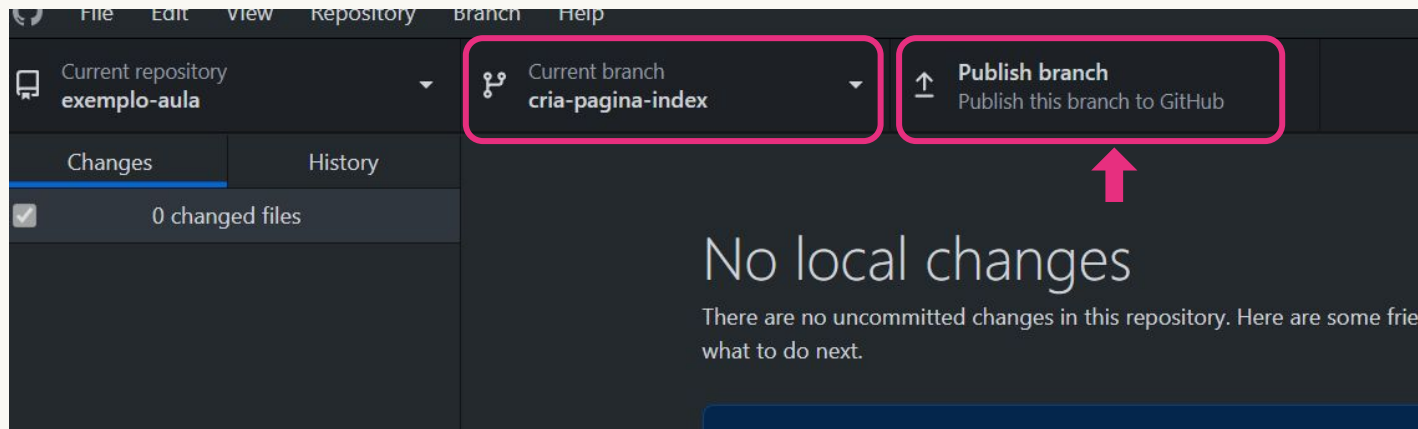


GITHUB DESKTOP | New Branch

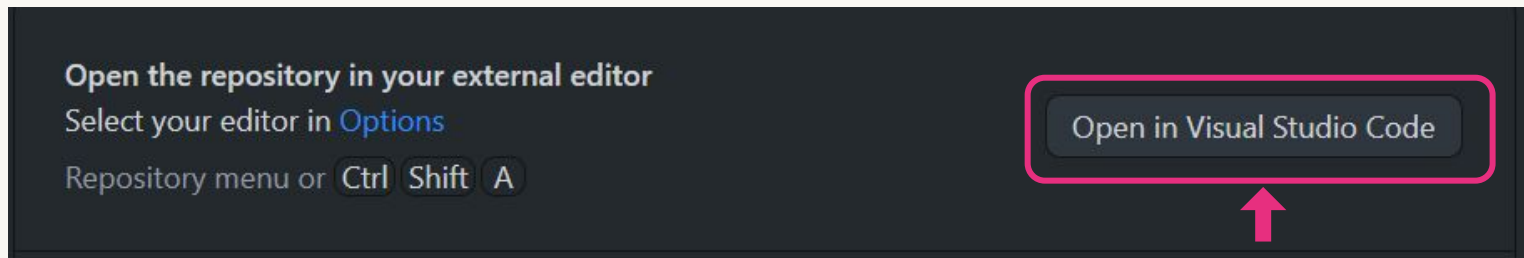
- Vamos criar uma nova **branch**, para inserir um arquivo novo no projeto



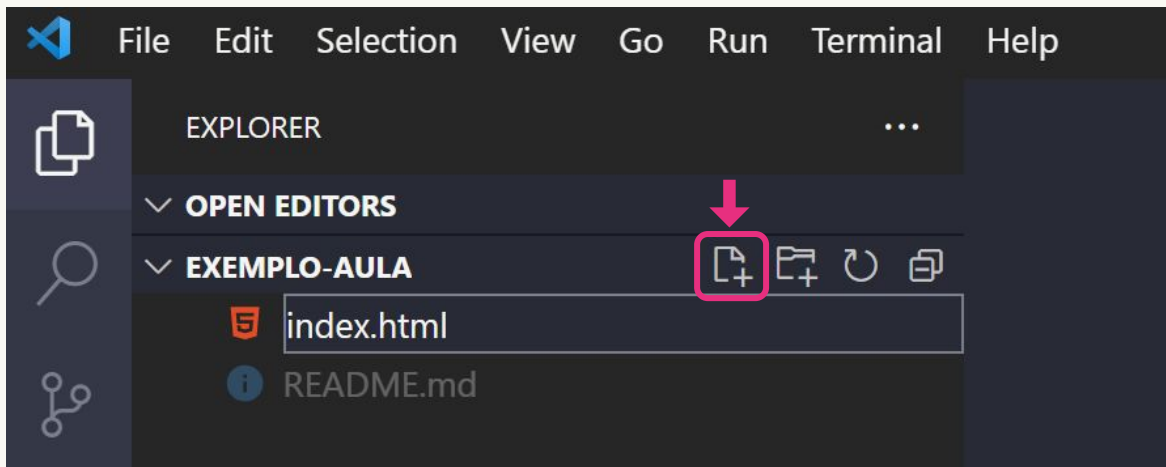
- Nossa **branch** atual está apenas no nosso computador.
- Precisamos informar ao repo remoto (lá no servidor do GitHub) a criação dessa branch. Ou seja, vamos publicá-la.



- Agora vamos fazer nosso trabalho na **branch** atual.
- Vamos abrir o nosso editor de código.
- O próprio GitHub Desktop já nos oferece essa opção.
- Mas sempre podemos abrir independentemente.

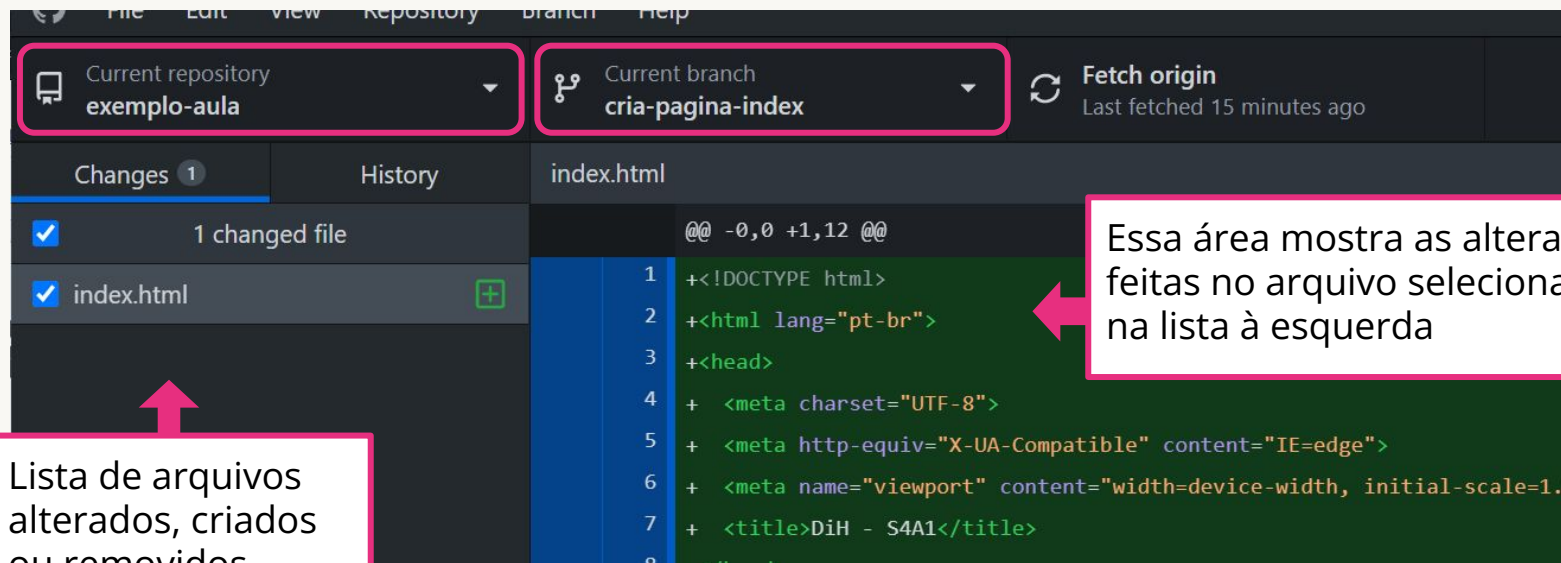


- No nosso editor favorito, podemos criar o arquivo **index.html** e escrever o que precisamos para completar nossa tarefa



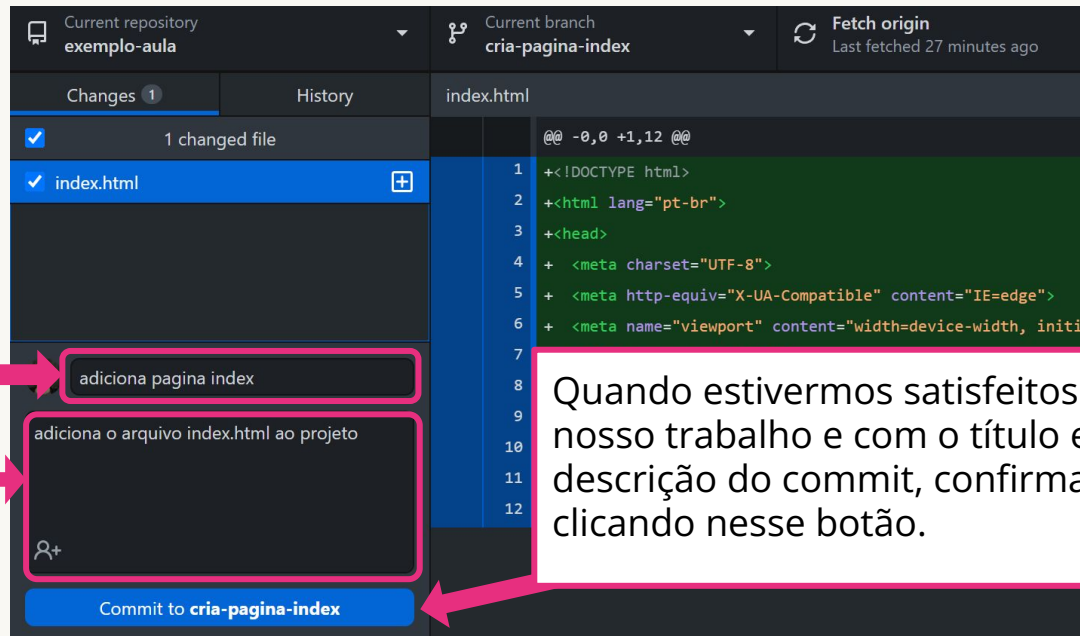
GITHUB DESKTOP | Alterações

- A medida que vamos adicionando arquivos e escrevendo nesses arquivos, essas alterações vão aparecendo no GitHub Desktop.



GITHUB DESKTOP | Commit

- Depois de escrevermos nosso código e finalizarmos a tarefa, salvamos nosso arquivo e voltamos no GitHub Desktop para criar o nosso **commit**.



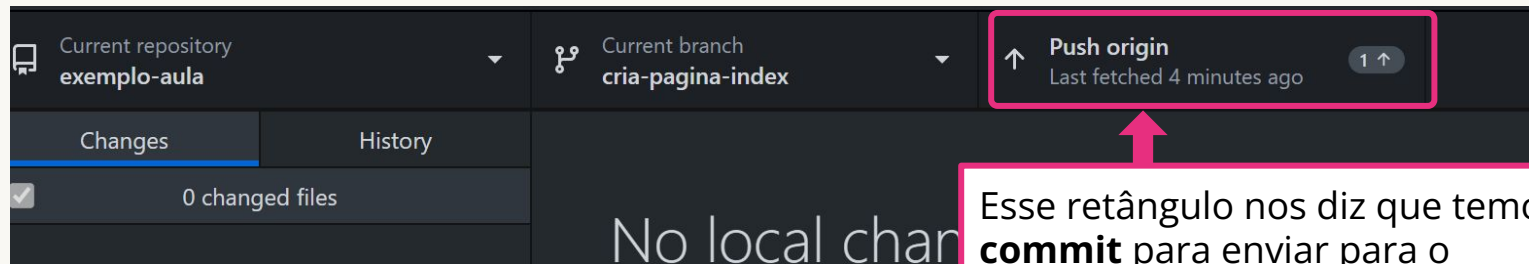
Escrevemos um título para o nosso commit (**obrigatório**)

Podemos escrever uma descrição mais detalhada sobre o que foi feito (**opcional**)

Quando estivermos satisfeitos com nosso trabalho e com o título e descrição do commit, confirmamos clicando nesse botão.

GITHUB DESKTOP | Push

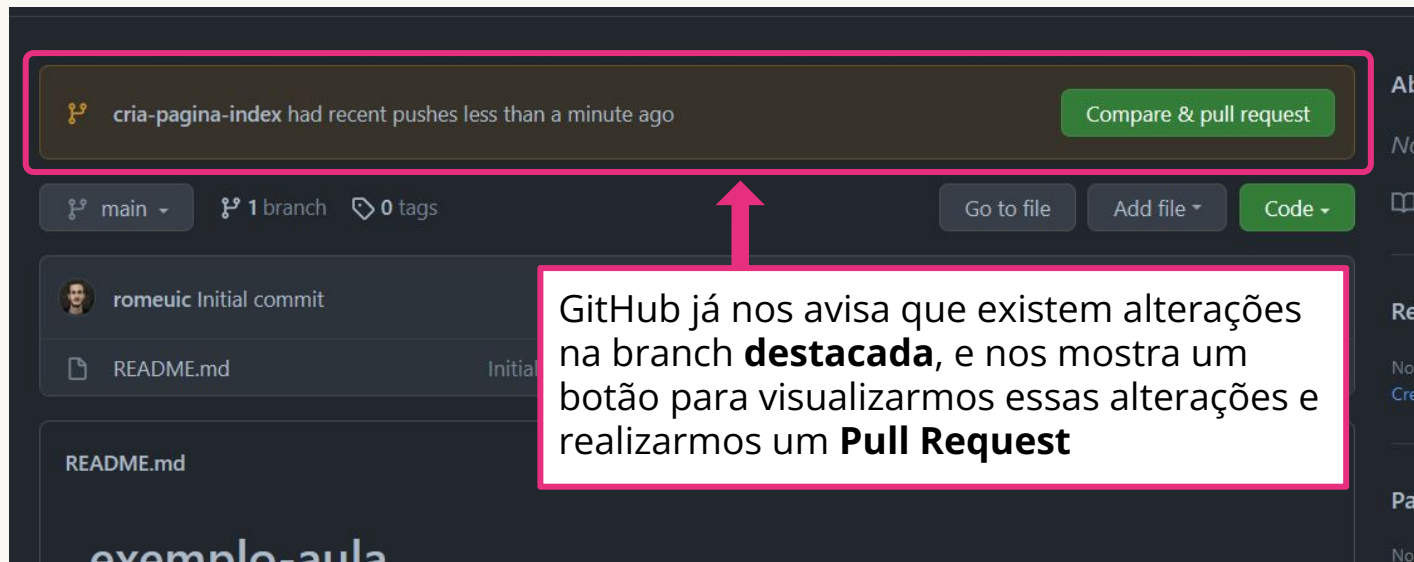
- Fizemos nosso commit, ou seja, salvamos nosso trabalho em um "pacote de alterações".
- Mas ele está salvo apenas no nosso computador.
- Precisamos enviar esse commit, ou esse "pacote de alterações", para o servidor remoto (nesse caso, GitHub).



Esse retângulo nos diz que temos 1 **commit** para enviar para o repositório "**origin**", ou seja, o repositório no servidor do GitHub.

GITHUB | Pull Request

- Depois de enviar nossas alterações para o repositório remoto, vamos ver se está tudo lá no GitHub.



GITHUB | Pull Request

- Ao clicar naquele botão, podemos visualizar as alterações realizadas e escolhermos abrir um PR ou não.

The screenshot shows the GitHub 'Open a pull request' page. The interface includes a header with the title 'Open a pull request' and a subtitle 'Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).' Below this, there are two dropdown menus: 'base: main' and 'compare: cria-pagina-index'. A green checkmark and the text 'Able to merge. These branches can be automatically merged.' are displayed next to the 'compare' dropdown. The main content area shows a commit message 'adiciona pagina index' and a description 'adiciona o arquivo index.html ao projeto'. On the right side, there are sections for 'Reviewers', 'Assignees', 'Labels', and 'Projects', each with a settings gear icon. At the bottom, there is a green button labeled 'Create pull request'.

branch destino (base)

branch origem do Pull Request

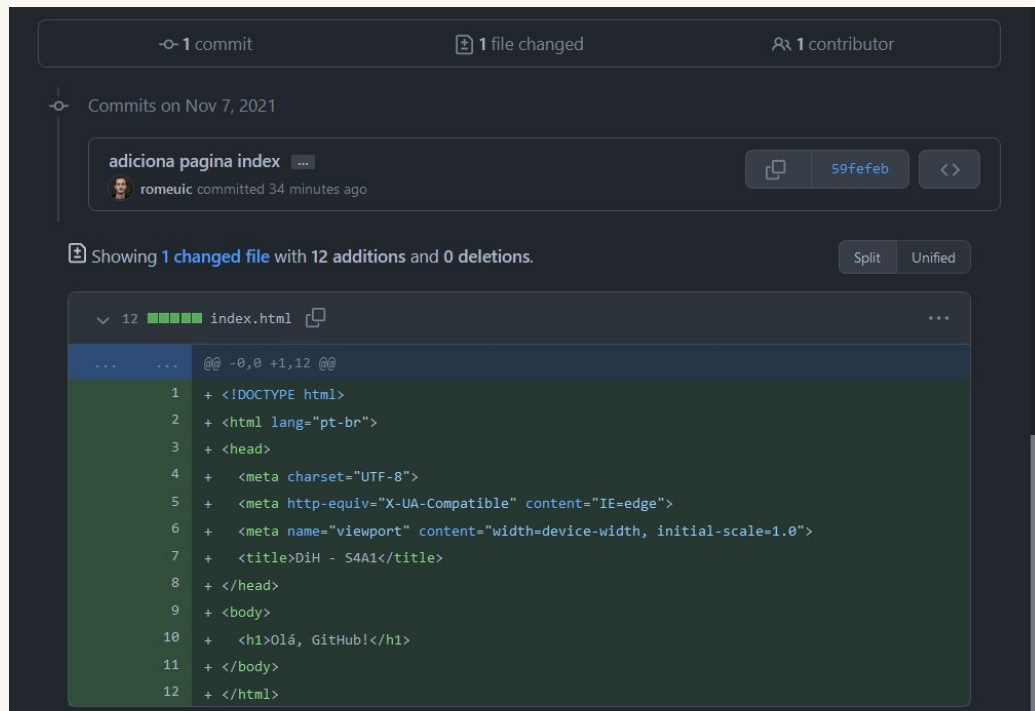
informações relevantes de responsáveis e rótulos do **Pull Request**

role para baixo para ver as alterações

Create pull request

GITHUB | Pull Request

- Podemos ver o que foi alterado nos **commits** dessa **branch**.



1 commit 1 file changed 1 contributor

Commits on Nov 7, 2021

adiciona pagina index ...
romeuic committed 34 minutes ago

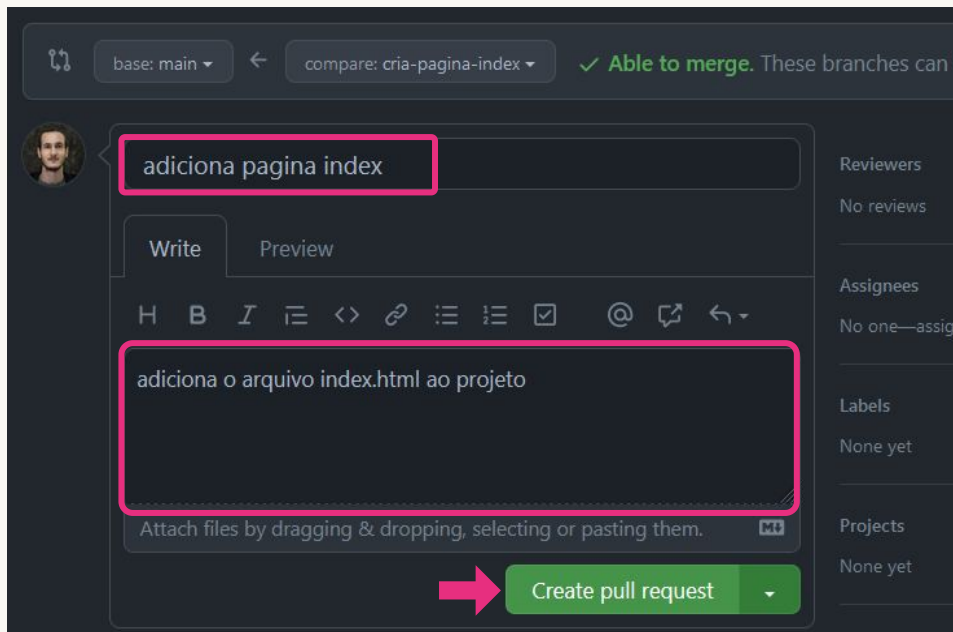
Showing 1 changed file with 12 additions and 0 deletions. Split Unified

12 index.html

```
@@ -0,0 +1,12 @@
1 + <!DOCTYPE html>
2 + <html lang="pt-br">
3 + <head>
4 +   <meta charset="UTF-8">
5 +   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6 +   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 +   <title>DiH - S4A1</title>
8 + </head>
9 + <body>
10 +   <h1>Olá, GitHub!</h1>
11 + </body>
12 + </html>
```

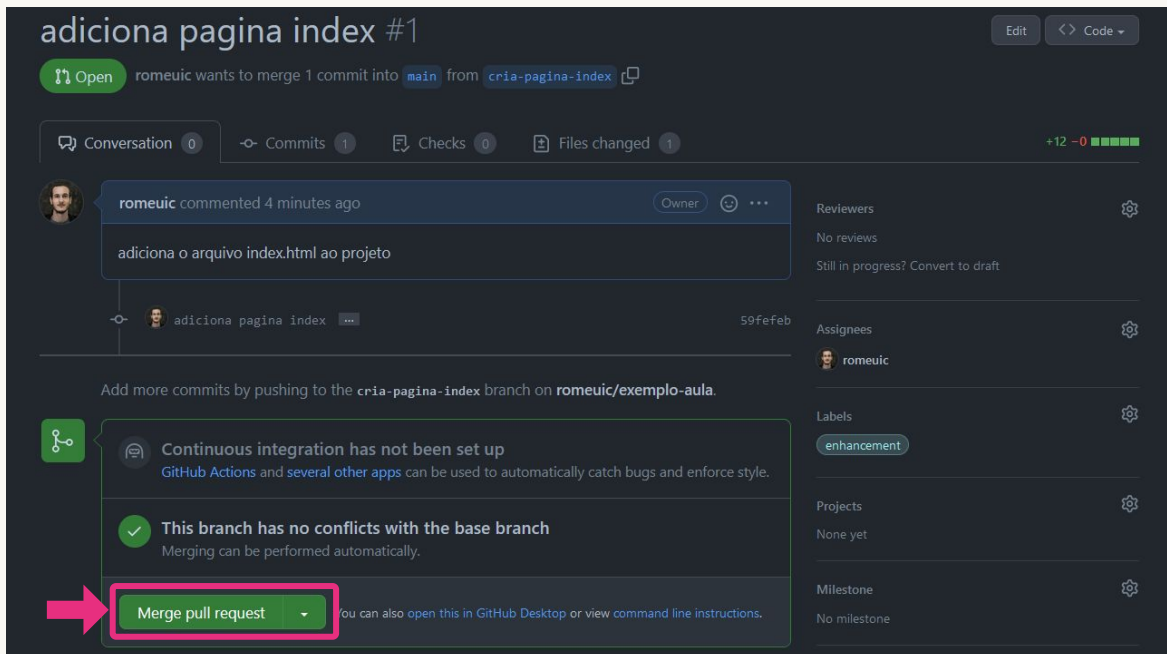
GITHUB | Pull Request

- Após ver as alterações, se estiver tudo OK, podemos alterar o **título** e a **descrição** do nosso **Pull Request**, e então criá-lo.



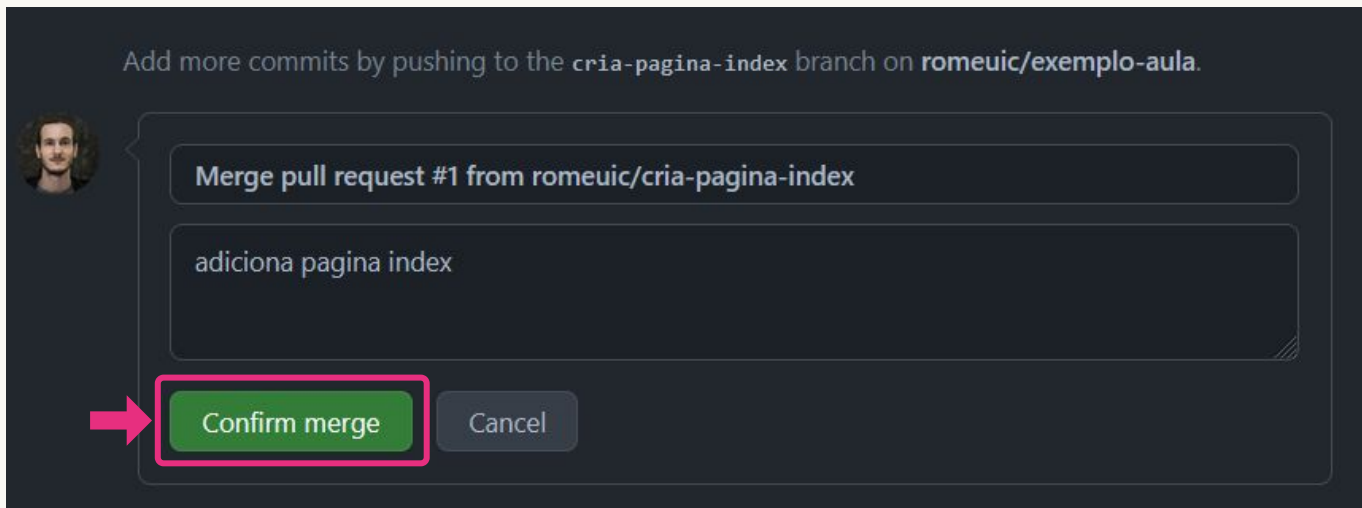
GITHUB | Merge Pull Request

- Com o PR criado, outros(as) colaboradores(as) do projeto podem revisá-lo e fazer comentários, aceitar ou recusar o **merge**.

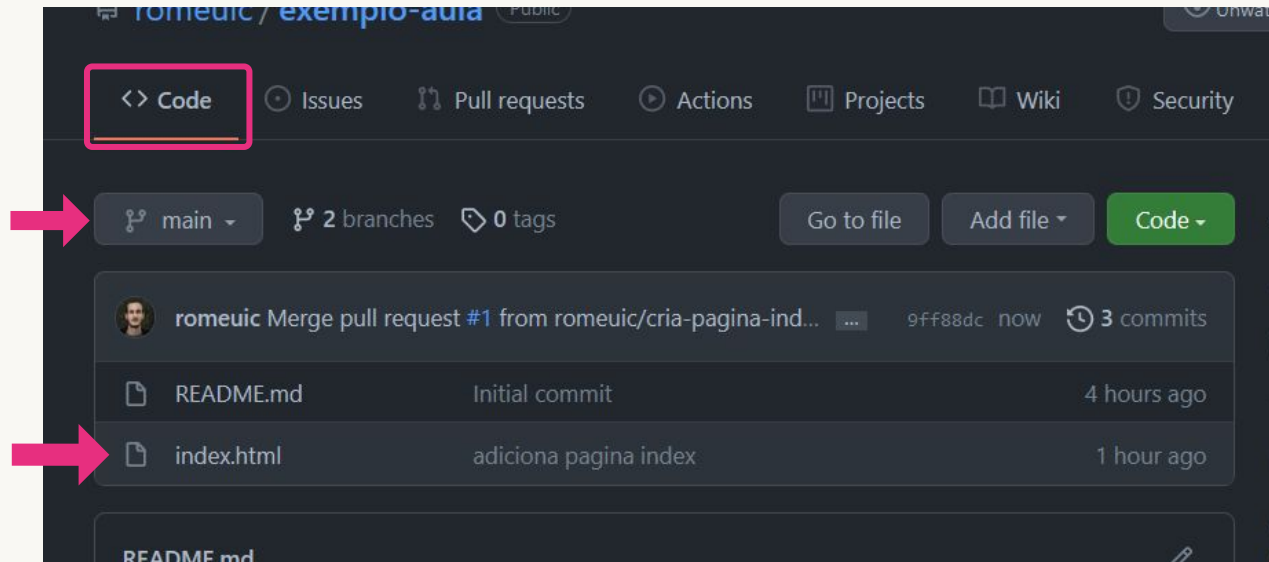


GITHUB | Merge Pull Request

- Após clicar para fazer o **merge**, o GitHub nos pede uma confirmação e podemos alterar a mensagem do **commit** de **merge**.

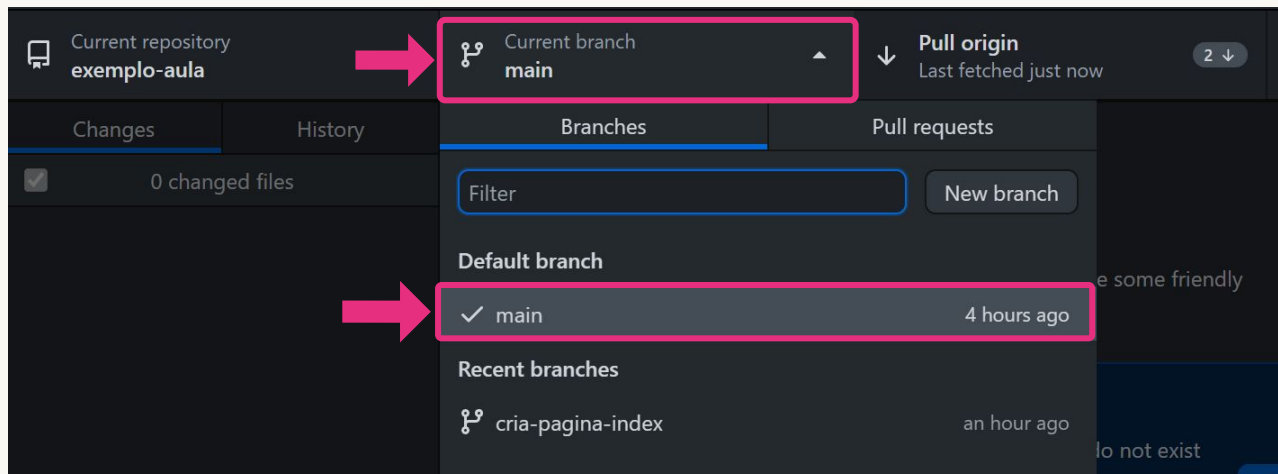


- Retornamos à aba do conteúdo do nosso repositório e vemos que os arquivos que criamos na outra **branch**, agora estão presentes na **branch** principal.

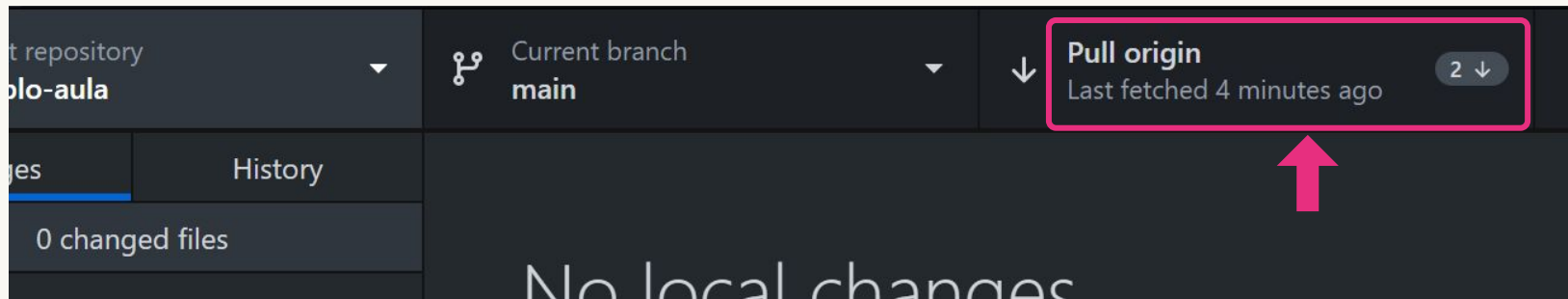


GITHUB DESKTOP | Pull

- Voltando ao GitHub Desktop, vamos atualizar nosso repositório **local**, o que está no nosso computador.
- Primeiro mudamos para a **branch "main"**, pois é onde nosso código está agora.

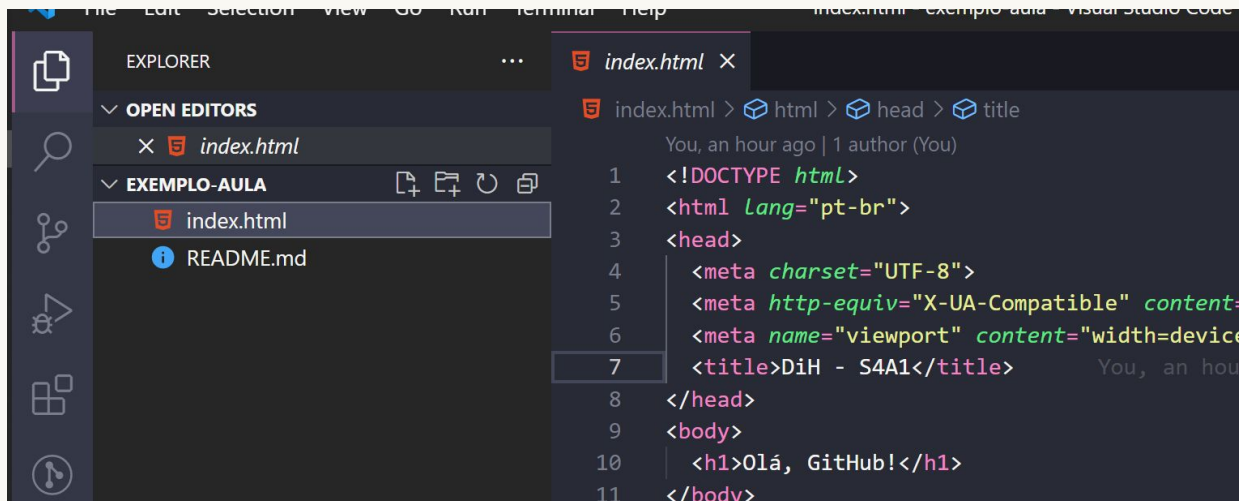


- Então clicamos no botão "**Pull origin**" para baixar a versão mais atualizada do que está no repositório **remoto** (github.com) para o nosso computador.



VS CODE

- Estamos olhando para a **branch "main"**, e o nosso computador está atualizado com a versão mais recente do repositório.
- Quando conferimos os arquivos do nosso projeto, vemos que está tudo atualizado na **branch "main"**.

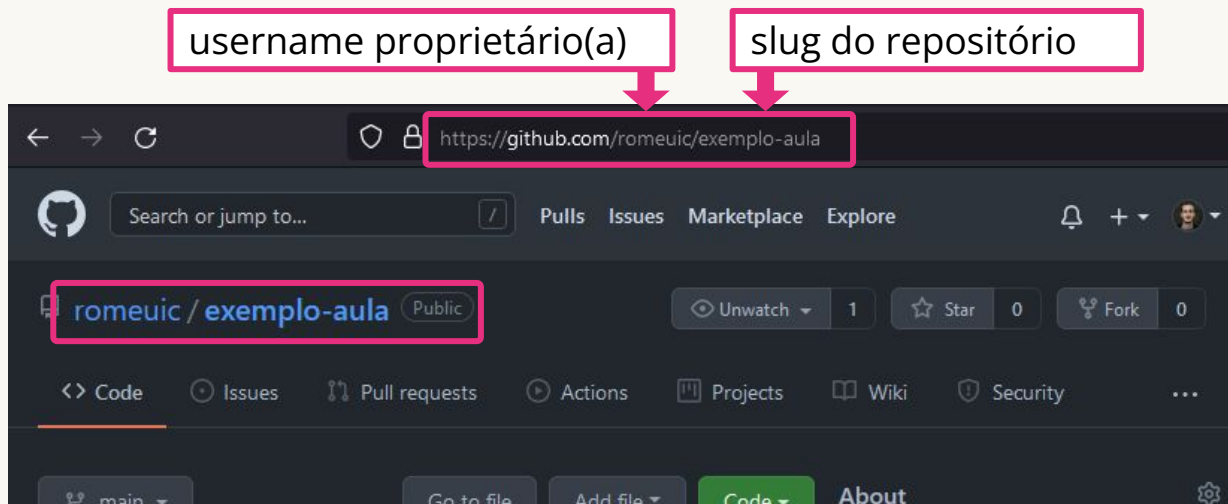


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, showing the 'EXEMPLO-AULA' folder with files 'index.html' and 'README.md'. The 'index.html' file is selected. The main editor area displays the content of 'index.html', which is an HTML document. The breadcrumb navigation at the top of the editor shows 'index.html > html > head > title'. The code in the editor is as follows:

```
1 <!DOCTYPE html>
2 <html Lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content=
6   <meta name="viewport" content="width=device
7   <title>DiH - S4A1</title>
8 </head>
9 <body>
10   <h1>Olá, GitHub!</h1>
11 </body>
```

GITHUB | Repositório

- E se você quiser pegar o link desse repositório e enviar no AVA para avaliação?
- Vamos supor que esse repositório é onde está o código do Projeto 1 do curso.



ATRASSO & INTERVALO



- **Timeout** (atraso):

Serve para definir um tempo de atraso na execução de uma função

```
setTimeout(funcao, 1000); //tempo em milissegundos (ms)
```

- **Interval** (intervalo):

Definir um tempo de intervalo entre execuções recorrentes da função

```
setInterval(funcao, 2000); //a cada 2 segundos (2000ms)
```

- **setTimeout**(function, number):

Executa uma função, uma vez, depois de um determinado tempo.

Pode executar uma função que se encontra no próprio arquivo Javascript ou uma referência de uma função definida em outro lugar.

O número representa o intervalo de tempo em milissegundos (1000 milissegundos equivalem a 1 segundo), para esperar antes de executar o código.

Pode ser cancelado utilizando a função **clearTimeout**.

- Exemplo de setTimeout:

```
function olaAtrasado() {  
  console.log('Olá atrasado!');  
}  
  
setTimeout(olaAtrasado, 2000);  
// atraso de 2 segundos (2000ms)
```

- Exemplo de clearTimeout:

```
// guardando a referência do timeout em uma variável
var t01a = setTimeout(olaAtrasado, 2000);

// cancelando o "agendamento" da execução
clearTimeout(t01a);
```

- **setInterval(function, number):**

Executa chamadas de funções ou trechos de código específicos repetidamente, de forma recorrente, com um intervalo fixo entre cada chamada (definido em milissegundos).

Para cancelar a execução desta função, basta utilizar a chamada **clearInterval**.

- Exemplo de setInterval:

```
function olaOutraVez() {  
  console.log('Olá outra vez!');  
}  
  
setInterval(olaOutraVez, 2000);  
// intervalo de 2 segundos (2000ms)
```

- Exemplo de clearInterval:

```
// guardando a referência do interval em uma variável  
var i01a = setInterval(olaOutraVez, 2000);  
  
// cancelando o "agendamento" dos intervalos  
clearInterval(i01a);
```

MATERIAL COMPLEMENTAR



What is version control? | <https://git-scm.com/video/what-is-version-control>

Vídeo: "O que são Git e GitHub?" (7min) | https://youtu.be/P4BNi_yPehc

Entendendo GIT | <https://youtu.be/6Czd1Yetaac>

Curso JavaScript #46 - setTimeout e setInterval | <https://youtu.be/tXnY9-gVN1E>

Curso JavaScript #47 - clearTimeout e clearInterval | <https://youtu.be/KV1ph8CYWi4>

MATERIAL COMPLEMENTAR



Tutorial de Git pelo GitHub (inglês) | <https://guides.github.com/activities/hello-world>

Tutorial de Git pelo Bitbucket (pt-br) | atlassian.com/br/git/tutorials/learn-git-with-bitbucket-cloud

Git Handbook | <https://guides.github.com/introduction/git-handbook>

Documentação Git | <https://git-scm.com/doc>

Referências Git | <https://git-scm.com/docs>

Hello World | GitHub | <https://guides.github.com/activities/hello-world>

setTimeout() | <https://developer.mozilla.org/en-US/docs/Web/API/setTimeout>

setInterval() | <https://developer.mozilla.org/en-US/docs/Web/API/setInterval>



DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!



<LAB365>