



Git e Github

APOSTILA



São Paulo-2021

Créditos

Organização e Produção

Davi Domingues

Douglas de Oliveira

Fernando Esquírio Torres

Rafael Lopes dos Santos



É proibida a duplicação ou reprodução deste volume, no todo ou em parte, em quaisquer formas ou por quaisquer meios (eletrônicos, mecânico, gravação, fotocópia, distribuição pela internet ou outros), sem permissão prévia do Instituto da Oportunidade Social.

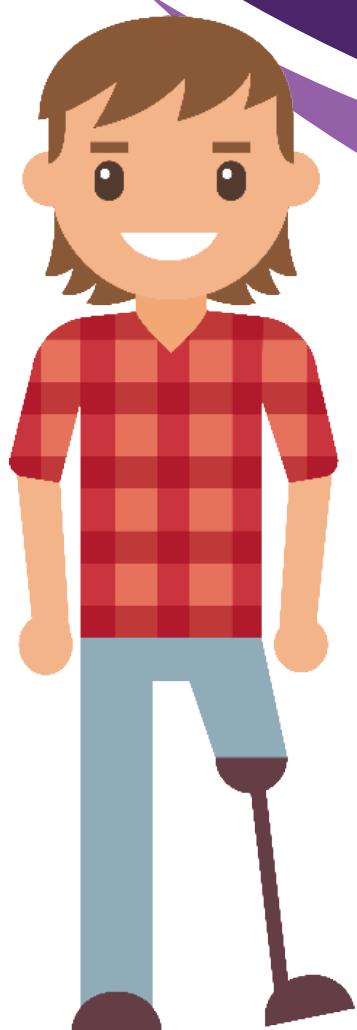
São Paulo
2021

Atenção

Em caso de dúvidas, sugestões ou reclamações. Entre em contato com a equipe de conteúdo.

educacional@ios.org.br

ESSE É SUA APOSTILA!



Ela vai te acompanhar durante todo o período do curso.

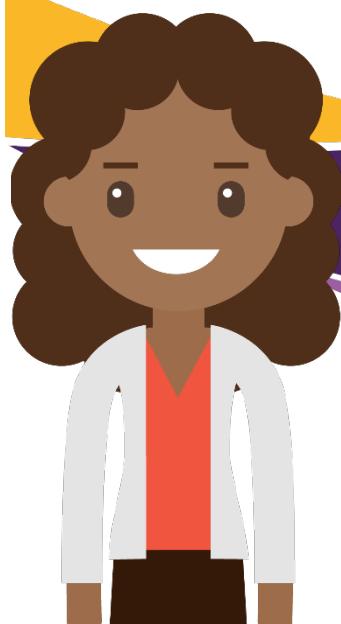
Cuide dela com carinho e responsabilidade.

Para que ela chegasse toda cheia de estilo do jeito que você está vendo, muita gente quebrou a cabeça para te entregar a melhor experiência de aprendizagem.

Ótimos estudos!

SUMÁRIO

Versionamento de código	2
Sistema de controle de versão	3
O que é o Github?	4
Criando um repositório	7
O que é o Git?	8
Funcionamento do Git	9
Comandos básicos no Git	11
Instalando o Git no seu computador	12
Instalando o Git no Linux.....	13
Instalando o Git no macOS	13
Instalando o Git no Windows.....	13
Sincronizar o VS Code com o Github.....	20
Comandos do Git	25
Aprendendo mais comandos do Git	26
Primeiros comandos no Git	27
Usar o arquivo .gitignore	38
Criando branches	43
Pull em um repositório remoto	48
Criando arquivo REAME.md	53
Clonando um diretório já existente	54
Comando pull	57



Versionamento de código

Os objetivos desta aula são:

- Conhecer a estrutura de versionadores de código;
- Compreender o funcionamento básico do Git e GitHub.

Bons estudos!

Sistema de controle de versão

O controle de versão de um projeto é importante para ajudar pessoas a **gerenciarem alterações** realizadas em algum arquivo, através do registro de todas as modificações realizadas. Para a engenharia de software, o controle de versões é uma classe de sistemas responsáveis por gerenciar mudanças em programas de computadores, documentos, páginas web ou outras coleções de informação.

Um sistema de controle de versões (**VCS, Version Control System**) é um software computacional encarregado de gerenciar diferentes versões no desenvolvimento de um documento qualquer. Esse tipo de sistema é bastante utilizado em empresas e instituições de tecnologia e no desenvolvimento de software para o controle de versões de projetos. Além disso, ele é bastante comum no desenvolvimento de softwares livres. Os **VCS** mais comuns são: **CVS, Mercurial, Git e SVN (Subversion)**; e as comerciais: **SourceSafe, TFS, PVCS** (Serena) e **ClearCase**.

Nesse material, vamos focar no **Git**, que tem tido bastante destaque por causa do **GitHub**.



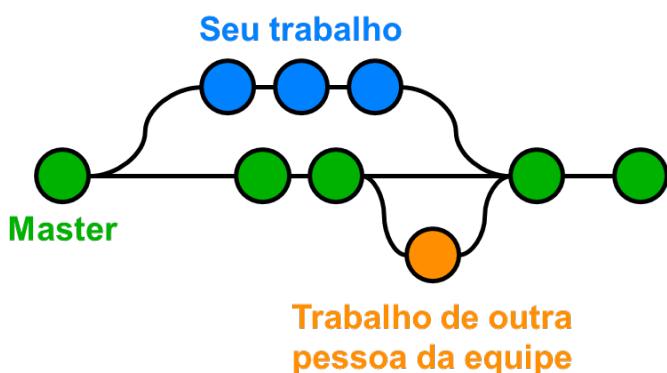
Importante!

Quer saber mais sobre alguns sistemas de controle de versões:

<https://www.up.edu.br/blogs/engenharia-da-computacao/2016/03/18/os-servicos-de-versionamento-de-codigo/>

Vamos utilizar um sistema de versionamento de código, porque ele traz diversas vantagens, tais como:

- **Controle do histórico:** esse tipo de sistema mantém um histórico de versões do documento/projeto, permitindo resgatar versões mais antigas e estáveis.
- **Trabalho em equipe:** o documento/projeto pode ser compartilhado com diversas pessoas e permite que elas trabalhem e realizem modificações nesse conjunto de informações ao mesmo tempo. Também, é possível controlar o acesso de cada usuário ou grupo de usuários.
- **Marcação e resgate de versões estáveis:** os desenvolvedores que utilizam um VCS podem marcar versões estáveis do projeto em desenvolvimento para ser possível resgatar essas versões se necessário.
- **Ramificação do projeto:** os sistemas de controle de versões permitem criar linhas diversas de desenvolvimento para que os programadores trabalhem paralelamente sem que uma linha interfira na outra ou interferir na linha principal do projeto.



Diferentes linhas de desenvolvimento do projeto.

- **Segurança:** esses softwares possuem níveis de segurança que impedem o acesso não autorizado ao projeto e, também, a invasão de agentes maliciosos.
- **Rastreabilidade:** o VCS tem a característica de se poder conhecer o local, o estado e a qualidade do arquivo.
- **Organização:** esse tipo de sistema exibe os arquivos em uma interface visual estruturada, onde podem ser vistos todos os arquivos controlados, desde a origem até o projeto por completo.
- **Confiança:** o desenvolvedor não corre o risco de perder os arquivos por eventos imprevistos aleatórios.

Resumidamente, os **sistemas de controle de versões** permitem **manter o histórico das alterações** de um mesmo arquivo ou de vários arquivos ao longo do tempo de vida do projeto e são essenciais para o desenvolvimento de softwares ou sistemas grandes e complexos.

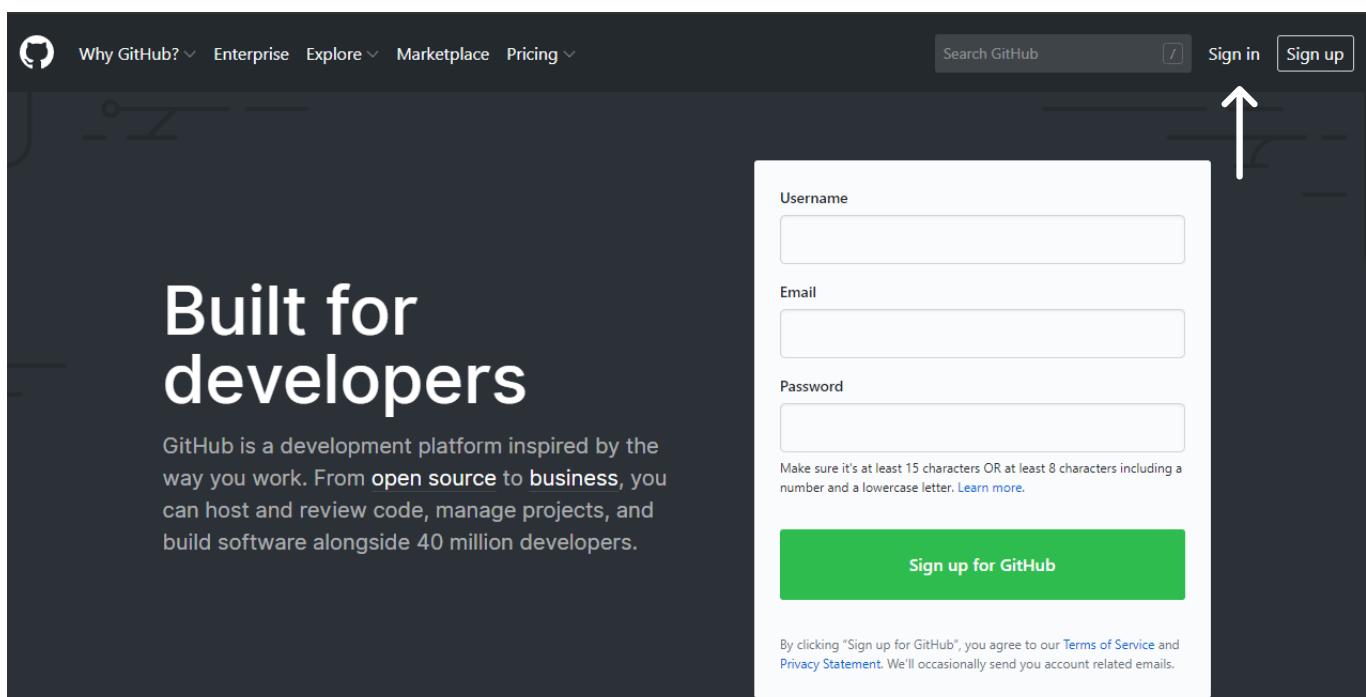
O que é o Github?

O GitHub é um **repositório** que fornece a **hospedagem** para o **desenvolvimento de software e controle de versão usando Git**. Ele oferece funcionalidades do Git como: controle distribuído de versões e um gerenciamento de código-fonte (**SCM, Source Code Management**) e, também, funcionalidades próprias como: **rastreamento de erros, controle de acesso de colaboração, requisição de recursos, gerenciamento de tarefas, wikis**, etc.

Ele foi criado por Chris Wanstrath, P. J. Hyett, Tom Preston-Werner e Scott Chacon. Juntos, eles criaram a empresa GitHub Inc., que existia desde **2007** em **São Francisco**. Desde **2012**, a **Microsoft era um grande usuário do GitHub** e em **outubro de 2018**, ela anunciou a **compra da empresa GitHub Inc por 7,5 bilhões de dólares**.

O site para você criar uma conta no GitHub é: <https://github.com/>, para criar sua conta você deve clicar em **Sign up**.





Site do GitHub.

[Join GitHub](#)

Create your account

Username *

Email address *

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more](#).

Email preferences

Send me occasional product updates, announcements, and offers.

Verify your account

Por favor, resolva esse desafio para que possamos saber que você é uma pessoa de verdade.

[Verifique](#)



Você deverá preencher os campos: Nome do Usuário (**Username**), endereço de e-mail (**Email address**) e senha (**Password**). O campo Nome do Usuário deve ser um nome inédito, que não existe no GitHub (O próprio GitHub faz essa verificação). O endereço de e-mail deve ser um e-mail válido e ativo e a **senha** deve conter pelo menos 15 caracteres ou 8 caracteres incluindo um número e letras maiúsculas. Com tudo preenchido, faça a verificação de pessoa, para ter certeza de que você não é um bot e clique no botão **Create account**.

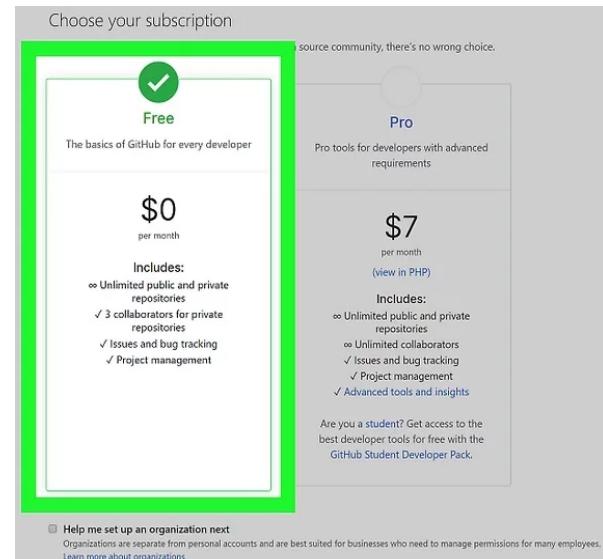
[Create account](#)

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

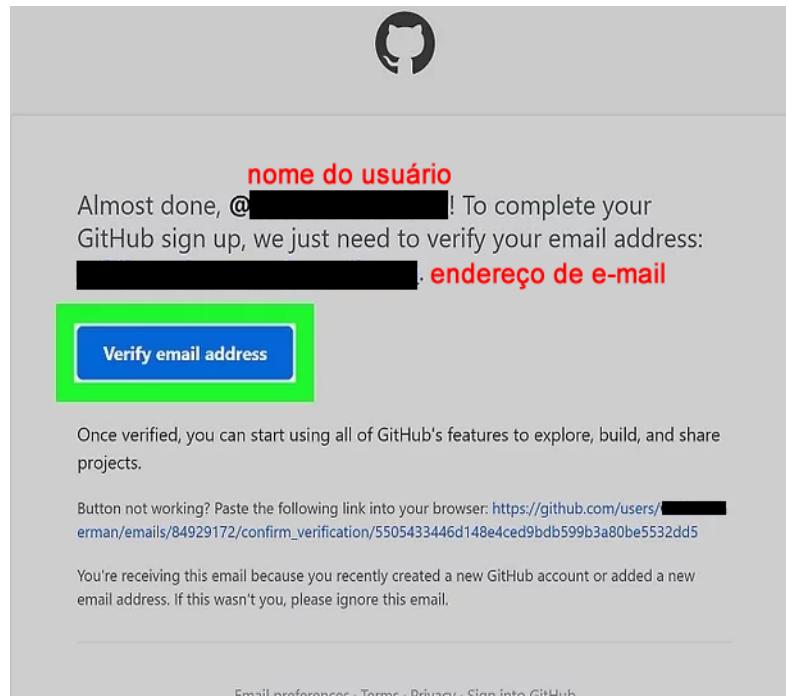
Campos para criação de conta no GitHub.

Na próxima página, O **GitHub** solicitará a você escolher o **plano de subscrição**. As opções disponíveis são:

- **Free**: repositórios públicos e privados ilimitados, até três colaboradores, fórum de problemas (issues), monitoramento de bugs e ferramentas de gerenciamento de projetos.
- **Pro**: acesso ilimitado a todos os repositórios, número ilimitado de colaboradores, fórum de problemas (issues), monitoramento de bugs e ferramentas avançadas.
- **Team**: todas as funcionalidades mencionadas anteriormente, controle de acesso de equipe e gerenciamento de usuários.
- **Enterprise**: todas as funcionalidades do plano Team, hospedagem própria ou em nuvem, prioridade na prestação de suporte técnico, login único (single sign-on) e mais.



Para um usuário inicial e não corporativo é melhor optar pela opção gratuita.



The screenshot shows a verification email from GitHub. It starts with the GitHub logo and the recipient's name, 'nome do usuário'. Below it, there is a message: 'Almost done, @[REDACTED]! To complete your GitHub sign up, we just need to verify your email address: [REDACTED]. [REDACTED] endereço de e-mail'. A blue button labeled 'Verify email address' is highlighted with a green box. The text continues: 'Once verified, you can start using all of GitHub's features to explore, build, and share projects.' At the bottom, there is a link to paste into a browser: 'https://github.com/users/[REDACTED]/erman/emails/84929172/confirm_verification/5505433446d148e4ced9bdb599b3a80be5532dd5'. A note below says: 'You're receiving this email because you recently created a new GitHub account or added a new email address. If this wasn't you, please ignore this email.' At the very bottom, there are links for 'Email preferences', 'Terms', 'Privacy', and 'Sign into GitHub'.

Verificação do E-mail.

Você receberá **um e-mail de confirmação** e assim que você realizar essas operações, você deverá selecionar as suas preferências ou pular esse passo. Então, seu cadastro estará finalizado.

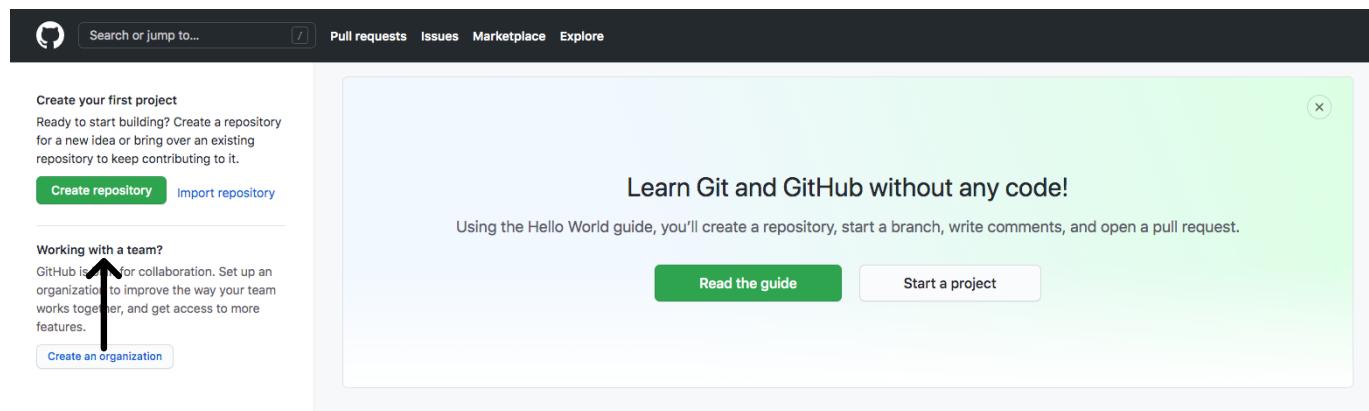
Agora, **edite seu perfil** para deixar o github com sua cara!

Não esqueça de seguir seus amigos, pois na internet relacionamento é tudo!

Criando um repositório.

O **repositório** é onde vamos **organizar e armazenar** o nosso **projeto**. Entenda cada repositório como uma **gaveta**, e nela vamos armazenar os arquivos relacionados com o projeto. Algo muito parecido quando organizamos a gaveta de **meias, camisas e calças**.

O primeiro passo é criar a gaveta, ou seja, o repositório. Para isso clique em **Create Repository**



Em seguida, vamos dar mais detalhes do repositório que será criado.

Vamos informar o **nome do repositório**, sua **descrição**, se ele é **publico** ou **privado** e marcar a opção **Add a README file** e clique em **Create repository**.

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 *

 conteudoios / MeuPrimeiroDiretorio ✓

Great repository names are short and memorable. Need inspiration? How about [laughing-system](#)?

Description (optional)
Criando meu primeiro diretório

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](#).

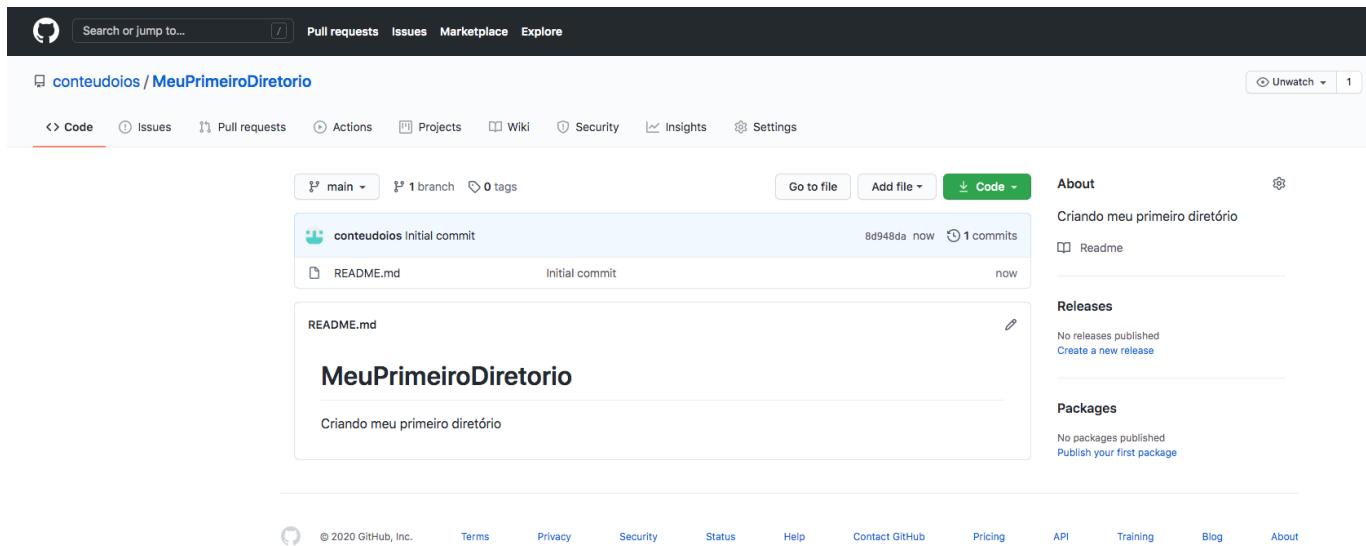
Add .gitignore
Choose which files not to track from a list of templates. [Learn more](#).

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

E nosso repositório foi criado!



Tá, legal! Mas e como vou **colocar** meus **códigos** lá no **github**? É ai que entra o **git**!

O que é o Git?

O **Git** é um sistema **open-source** de controle de versões gratuito e distribuído projetado para trabalhar com diversos tipos de projetos. Além disso, ele fornece ao desenvolvedor uma maneira muito mais inteligente e eficaz de organizar seu projeto. O **Git** foi inicialmente projetado e desenvolvido por **Linus Torvalds**, em **2005**, para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos. Alguns dos objetivos do Git são: velocidade, design simples, forte suporte para o desenvolvimento não linear (permitir muitas ramificações em paralelo), totalmente distribuído, capaz de lidar com grandes projetos eficientemente como, por exemplo, o **kernel do Linux**.



O **Git** vem **evoluindo** e **amadurecendo** desde **2005** e está se tornando um sistema para manter a qualidade desde o início do projeto. Além de o Git ser muito eficiente com grandes projetos, ele possui um sistema de ramificação de versões eficiente para o desenvolvimento não linear, **permitindo a customização de versões sem alterar o projeto principal**.

Como foi dito, o **Git** é um sistema de controle de versão distribuído e ele pode ser usado como um servidor pronto para uso. Os servidores dedicados de **Git** incluem funcionalidades extras, tais como:

- Controle de acesso;
- Revisão por pares;
- Ferramentas de visualização;
- Repositório, etc.

Repositório é o lugar onde os códigos ficam localizados e eles podem ser **públicos** ou **privados**. Entenda nesse momento: um sistema repositório é diferente de um sistema versionador.



Repositório versus. Versionador.

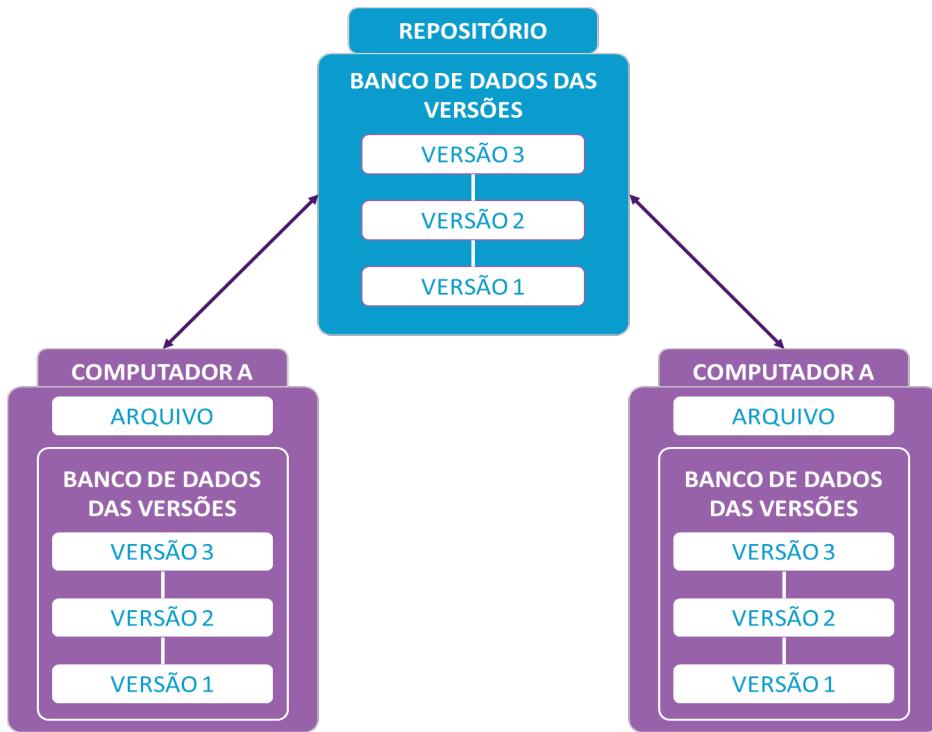
Os **serviços** de **Git** são plataformas de hospedagem de código-fonte e permitem que os desenvolvedores contribuam em projetos privados ou abertos (mais conhecidos como projetos open source). Cada um pode ter funcionalidades ou recursos específicos e diversos serviços de **Git** possuem repositórios fechados, em que seus colaboradores ou funcionários de uma determinada empresa desenvolvam seus projetos com ferramentas de gerenciamento organizacional. Os **serviços** de **Git** mais populares são **GitHub**, **GitLab**, **Bitbucket** e **SourceForce**.



Símbolos dos serviços de Git, na ordem a esquerda para a direita: GitLab, GitHub, Bitbucket e SourceForce.

Funcionamento do Git

Os servidores, como o Git Hub, são **repositórios remotos**, que podem ser acessados por diversos usuários. Esses usuários podem fazer o download de uma cópia do repositório em seu computador de trabalho, realizar alterações no código localmente e depois submeter a atualização para o repositório.



Acesso dos usuários ao repositório.

Assim o **arquivo** no **computador local**, que está **sendo atualizado**, pode assumir **três estados**: **modificado**, **staged** e **committed**.

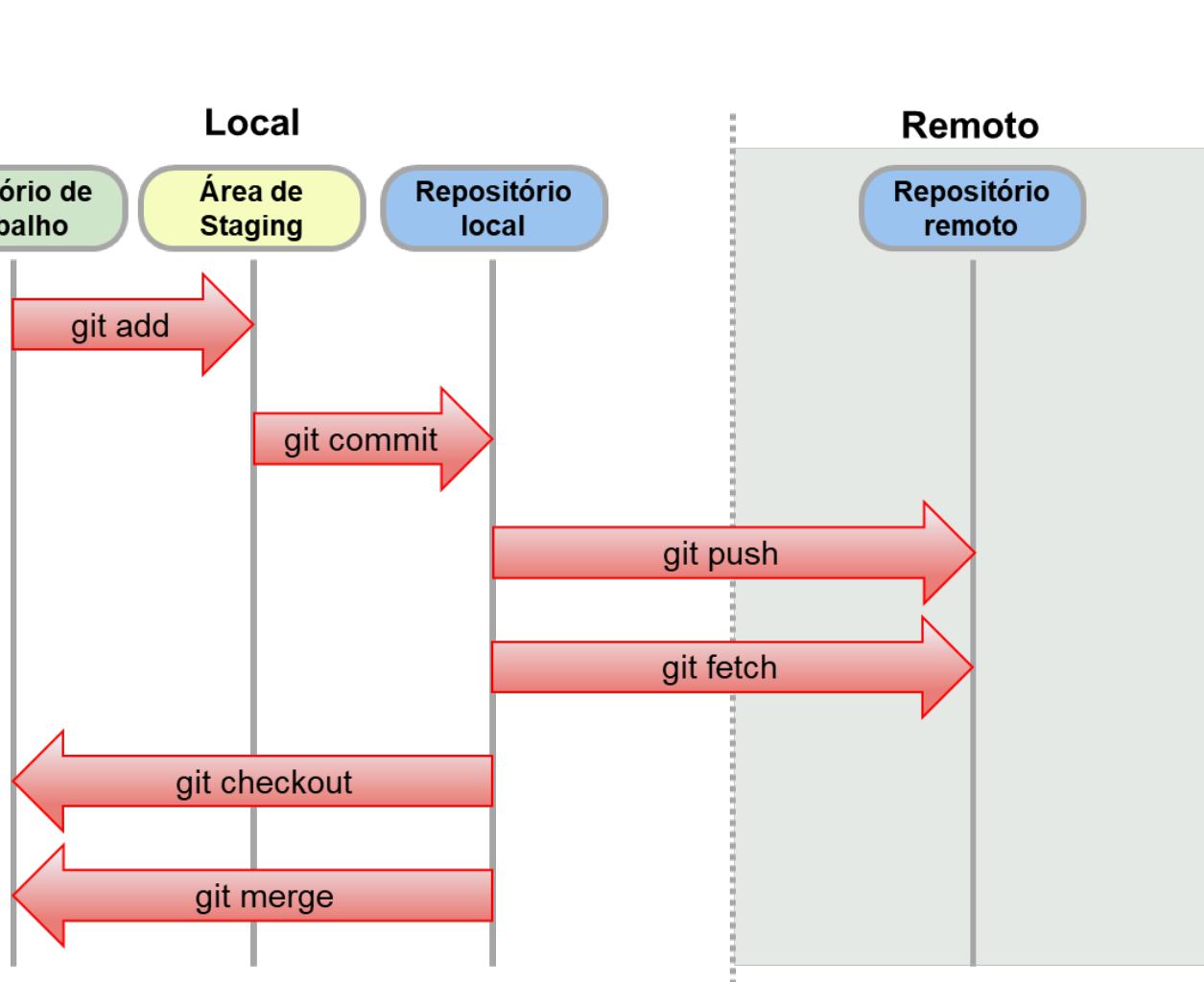
O estado **Modificado** significa que você alterou o arquivo, mas não deu o **commit** para o seu banco de dados local.

O estado **Stage** significa que você escolheu quais alterações do arquivo modificado na sua versão atual deseja enviar ao **commit**.

O estado **Committed** significa, que o arquivo modificado está **salvo no repositório local**.

Então, para garantir o desenvolvimento do projeto de uma maneira correta e segura, o Git tem uma estrutura basicamente dividida em dois ambientes:

- **Remoto**: que é composto pelo Repositório remoto.
- **Local**: que é composto pelo Repositório local, pela Área de Staging e pelo Diretório de trabalho.



No **ambiente local**, o **Diretório de trabalho** possui uma versão única do código armazenado no disco para ser usado e modificado, ou seja, ele contém o arquivo ou arquivos que você de fato está alterando. A área de **Staging** é uma área intermediária, que serve para você escolher quais alterações feitas no código serão enviadas para o seu **Repositório local**. Ela armazena as modificações que você escolheu enviar. O **Repositório local** possui o código fonte na sua estação de trabalho com o código completo. No ambiente remoto, o **Repositório remoto (Nuvem)** contém o código disponível para todas as outras pessoas. O arquivo transita de uma região para outra dentro da estrutura do projeto Git, através dos comandos disponíveis.

Importante!

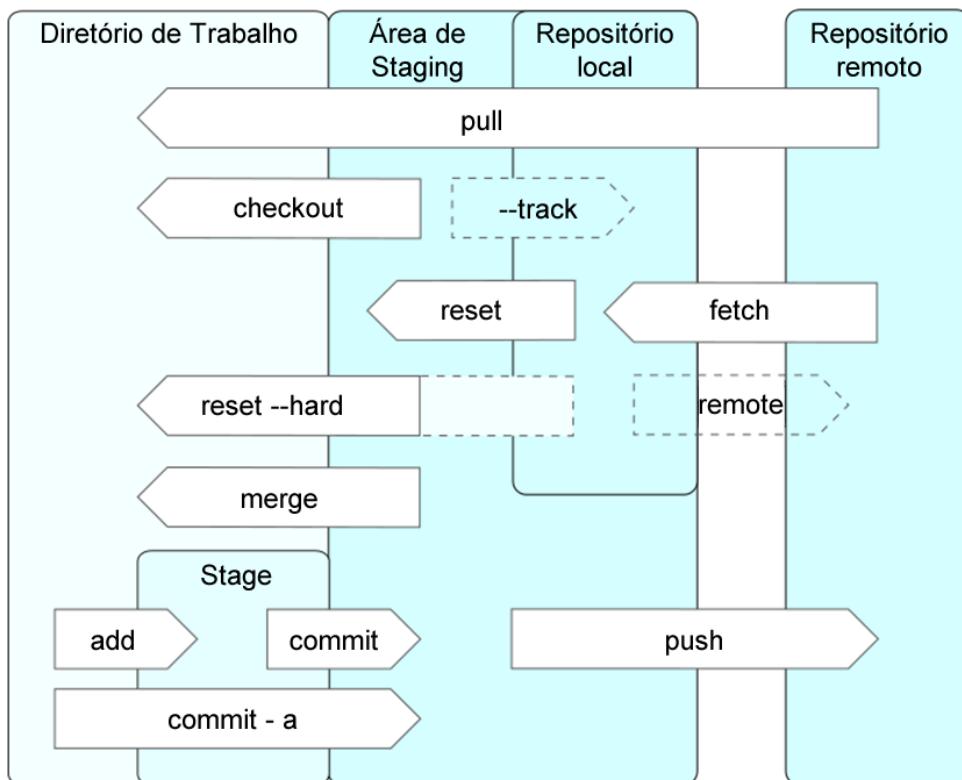


Quer saber mais sobre o sistema Git, baixe esse livro gratuito para você aprender mais: <https://git-scm.com/book/pt-br/v2>

Comandos básicos no Git

Alguns comandos básicos que você pode executar no Git:

Símbolo	Descrição
git init	Inicializa um repositório Local Git
git add <nome_do_arquivo>	Transfere as alterações do código de trabalho para a área de <i>staging</i> . Por exemplo: git add index.html
git status	Verifica o status do <i>branch</i> de trabalho
git commit	Transfere as alterações do código da área de <i>staging</i> para o repositório local.
git push	Busca o seu repositório local e envia para o repositório remoto.
git pull	Busca o seu repositório remoto e mescla (merge) com o repositório local.
git checkout	Altera entre ramificações do código.
git merge	Mescla duas ramificações.
git fetch	Busca o repositório remoto sem mesclar (merge) como o repositório local.



Instalando o Git no seu computador

Essa instalação é necessária para você **utilizar o Git para gerenciar os arquivos de um projeto em seu computador**. O Git está disponível para os sistemas operacionais **Linux, macOS e Windows**.

Instalando o Git no Linux

Para instalar o Git em distribuições Linux (**Debian/Ubuntu**), você deve digitar o comando no terminal:

```
sudo apt-get install git
```

Para instalar o Git em distribuições Linux (**Fedora**), você deve digitar o comando no terminal:

```
sudo yum install git
```

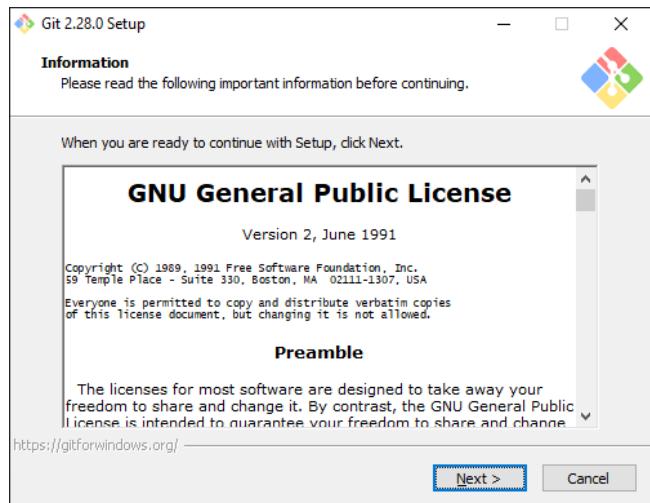
Instalando o Git no macOS

No **macOs**, você pode utilizar o **brew** para instalar o Git. Para isso, você deve usar o comando no terminal:

```
brew install git
```

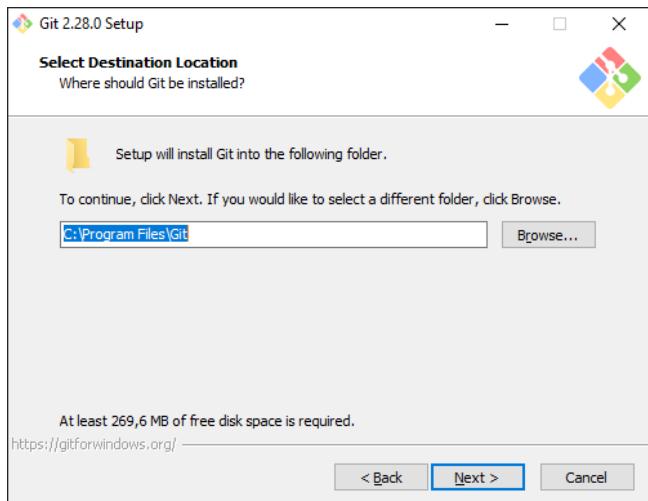
Instalando o Git no Windows

No Windows, você fazer o download do instalador no site <https://git-scm.com/download/win> e fazer a instalação como qualquer software no Windows. Para isso, siga os seguintes passos:
Execute o arquivo baixado. No Windows, aparecerá a janela **Information**. Clique no botão **Next**.



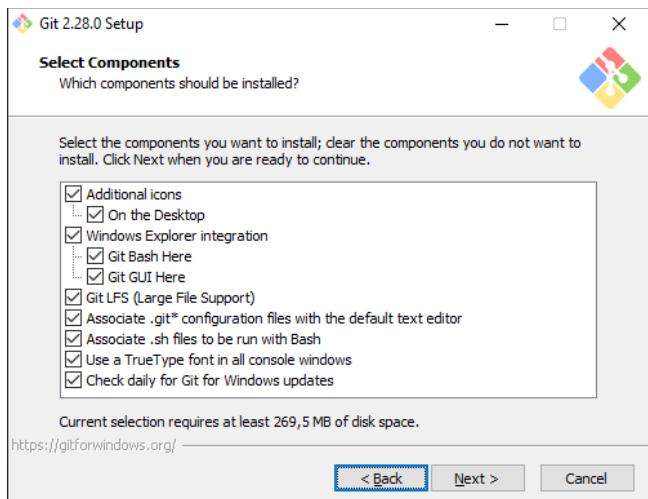
Instalação do Git no Windows

Na janela **Select Destination Location**, selecione o diretório de instalação. Clique no botão **Next**.



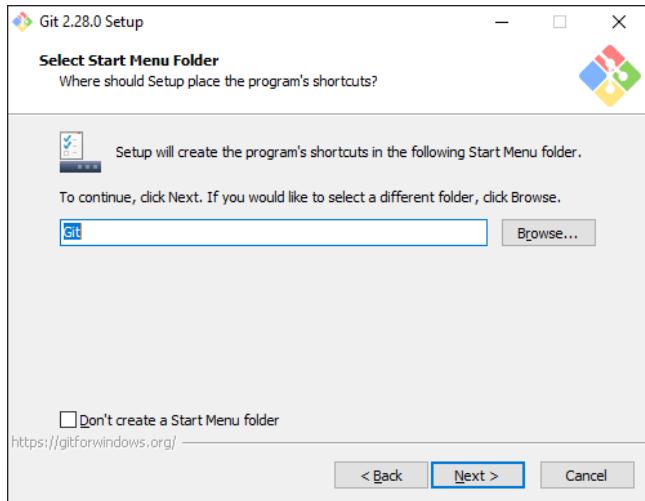
Seleção do diretório de instalação

Na janela **Select Components**, marque os componentes que deseja instalar. Clique no botão **Next**.



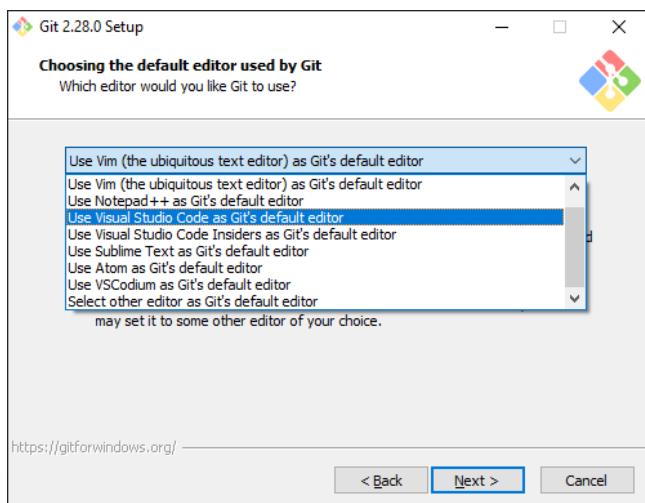
Escolha dos componentes para serem instalados

Na janela **Select Start Menu Folder**, escolha se você quer criar um atalho no menu Iniciar. Clique no botão **Next**.



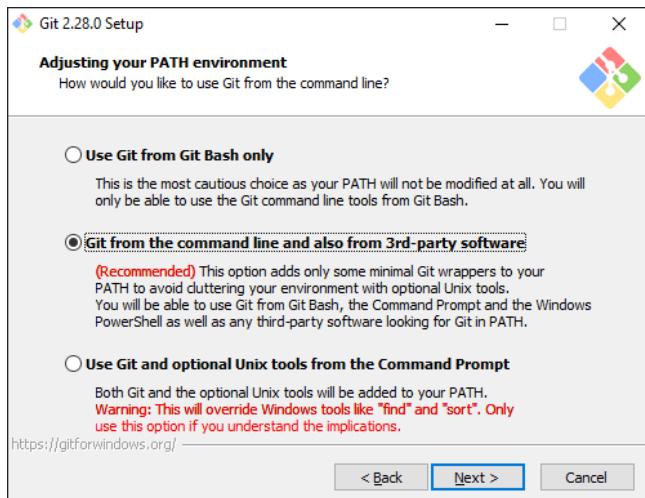
Criar atalho no Menu Iniciar.

Na janela **Choosing the default editor used by Git**, selecione o **Visual Studio Code** como seu editor padrão. Clique no botão **Next**.



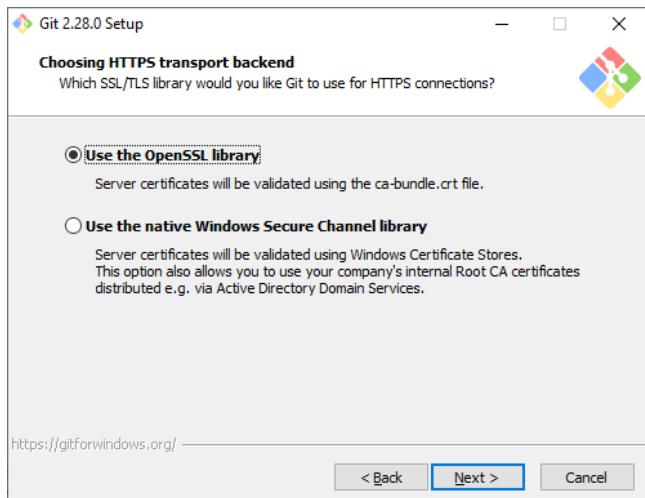
Escolha do Editor Padrão.

Na janela **Adjunsting your Path enviroment**, deixe as opções padrão selecionadas e clique no botão **Next**.



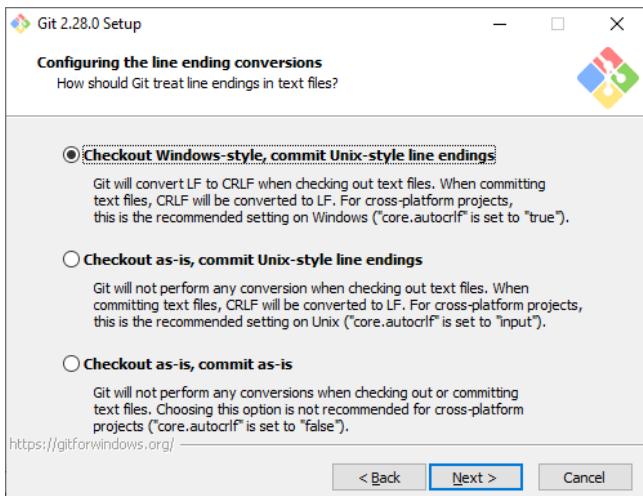
Ajuste do caminho no ambiente de desenvolvimento.

Na janela **Choosing HTTPS transport backend**, deixe as opções padrão e clique no botão **Next**



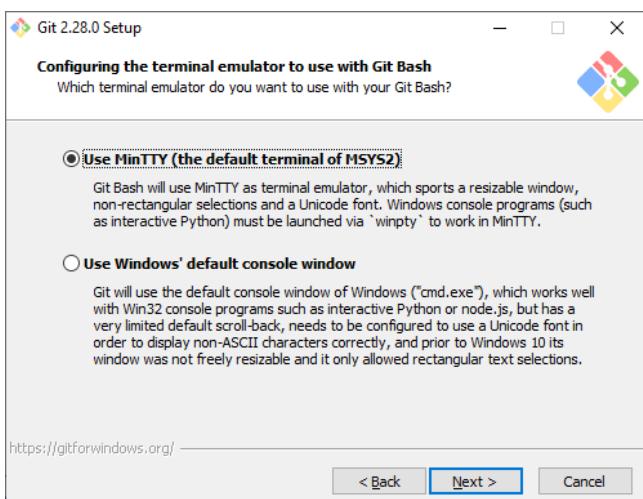
Escolha do backend para o transporte de HTTPS.

Na janela **Configuring the line ending conversions**, deixe as opções padrão selecionadas e clique no botão **Next**.



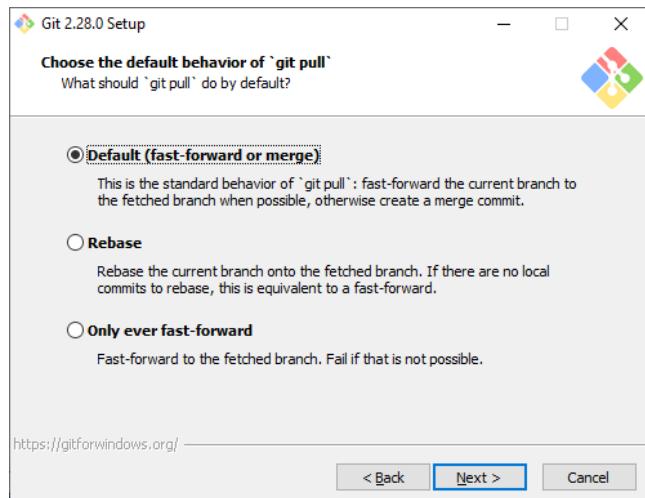
Configuring the line ending conversions.

Na janela **Configuring the terminal emulator to use with Git Bash**, deixe as opções padrão selecionadas e clique no botão **Next**.



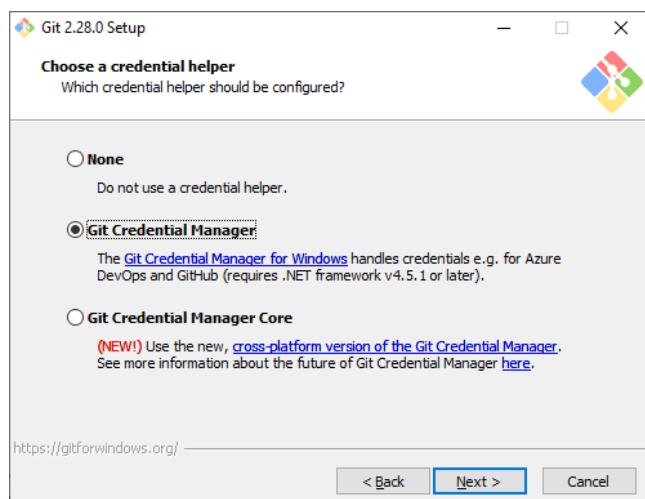
Configuring the terminal emulator to use with Git Bash.

Na janela **Choose the default behavior of 'git pull'**, deixe as opções padrão selecionadas e clique no botão **Next**.



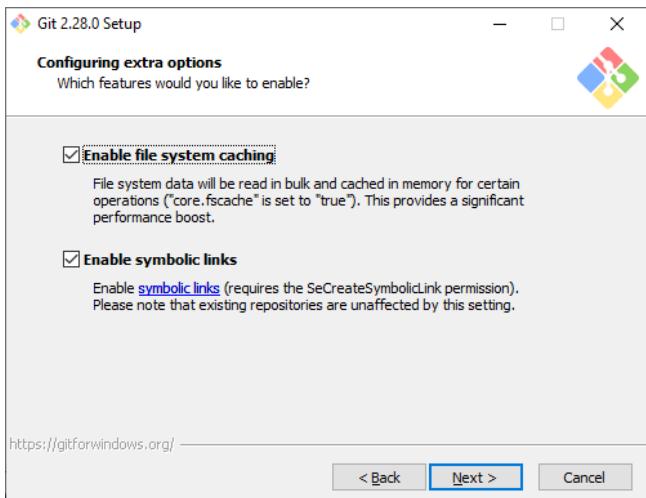
Choose the default behavior of 'git pull'.

Na janela **Choose a credential helper**, deixe as opções padrão selecionadas e clique no botão **Next**.



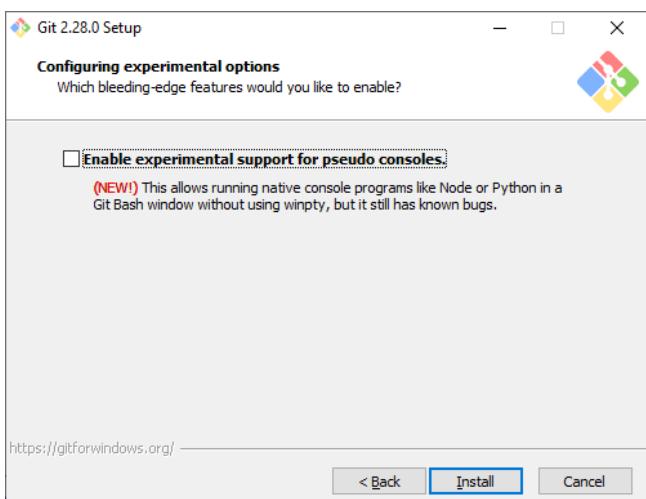
Choose a credential helper.

Na janela **Configuring extra options**, deixe as opções padrão selecionadas e clique no botão **Next**.



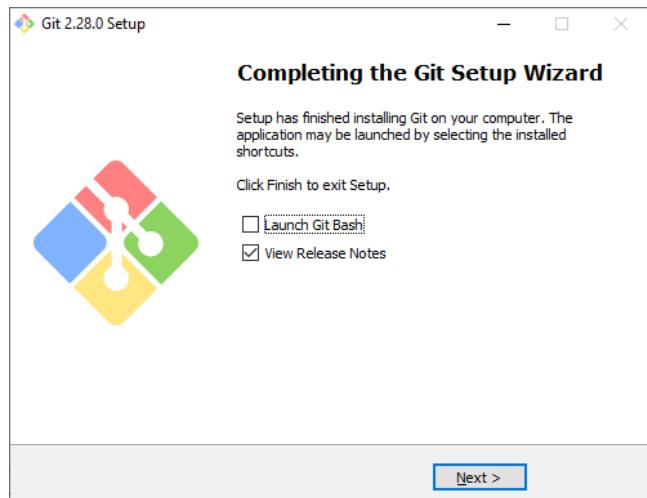
Configuring extra options.

Na janela **Configuring experimental options**, deixe a opção padrão selecionada e clique no botão **Next**.



Configuring experimental options.

Na janela **Completing the Git Setup Wizard**, deixe as opções padrão selecionadas e clique no botão **Next**.

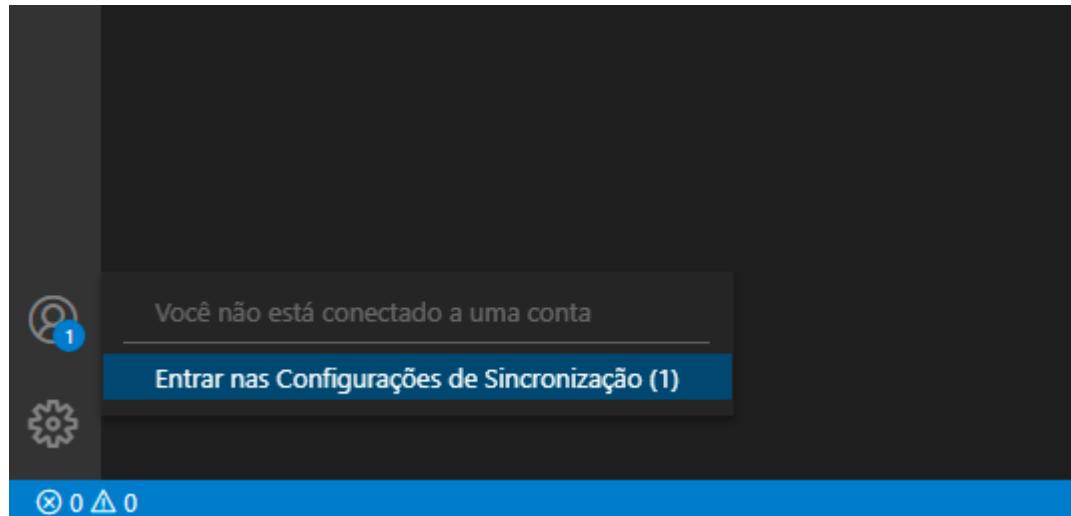


Completing the Git Setup Wizard.

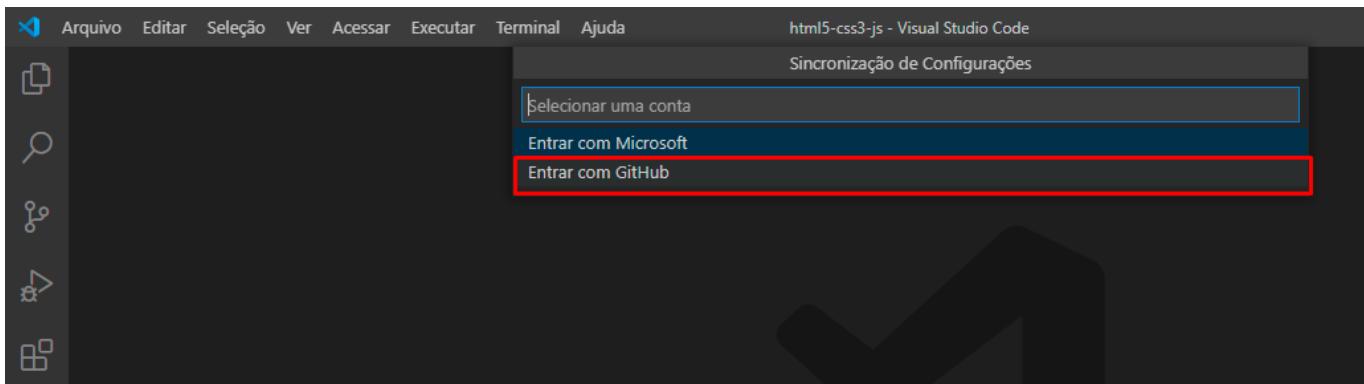
Sincronizar o VS Code com o Github.

Usar o **GitHub** com o **VS Code** permite que você compartilhe seu código-fonte e trabalhe de forma colaborativa com outras pessoas. Além disso, pode ser uma ótima alternativa para o **GitHub Desktop**, que possui apenas versão para computadores **64bits**. Para fazer a integração do **VS Code** com o **GitHub** é necessário configurar o GitHub para funcionar no VS Code.

Acesse o Menu **Contas**, que está disponível na parte inferior da barra de Menu lateral **do Visual Studio Code** e clique em **Entrar nas Configurações de Sincronização**.



A opção de **Sincronização de Contas** será aberta com as escolhas disponíveis. Você deve selecionar a opção **Entrar com GitHub**.



Você será direcionado para a página de **autorização** de acesso do **VS Code ao GitHub**. Você deve clicar no botão **Continue**.



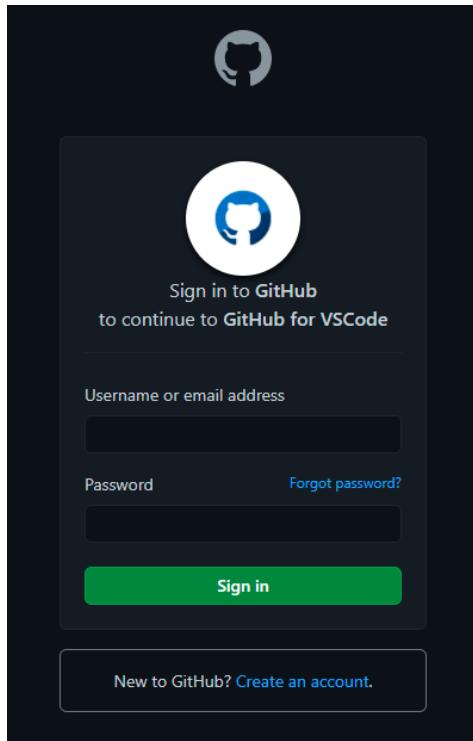
Authorize Visual Studio Code to access GitHub

If you initiated this authorization from Visual Studio Code, click 'Continue' to authorize access to GitHub

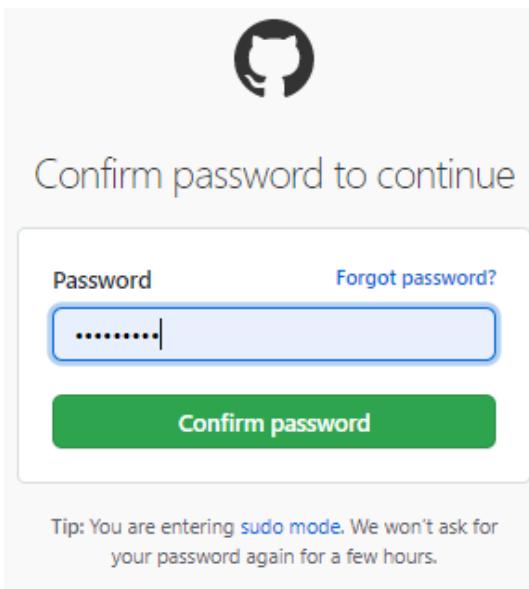
[Continue](#)

[Do not authorize](#)

Algumas vezes, o **GitHub** pode solicitar que você realize o **login** na sua **conta**.



Na página seguinte, confirme o seu **password** e clique em **Confirm password**.



Abrirá uma mensagem no navegador web para abrir o seu **VS Code** e a página será **atualizada com a confirmação da autorização**.



Success!

Authorization was successful. You will be redirected back to Visual Studio Code

Didn't work?

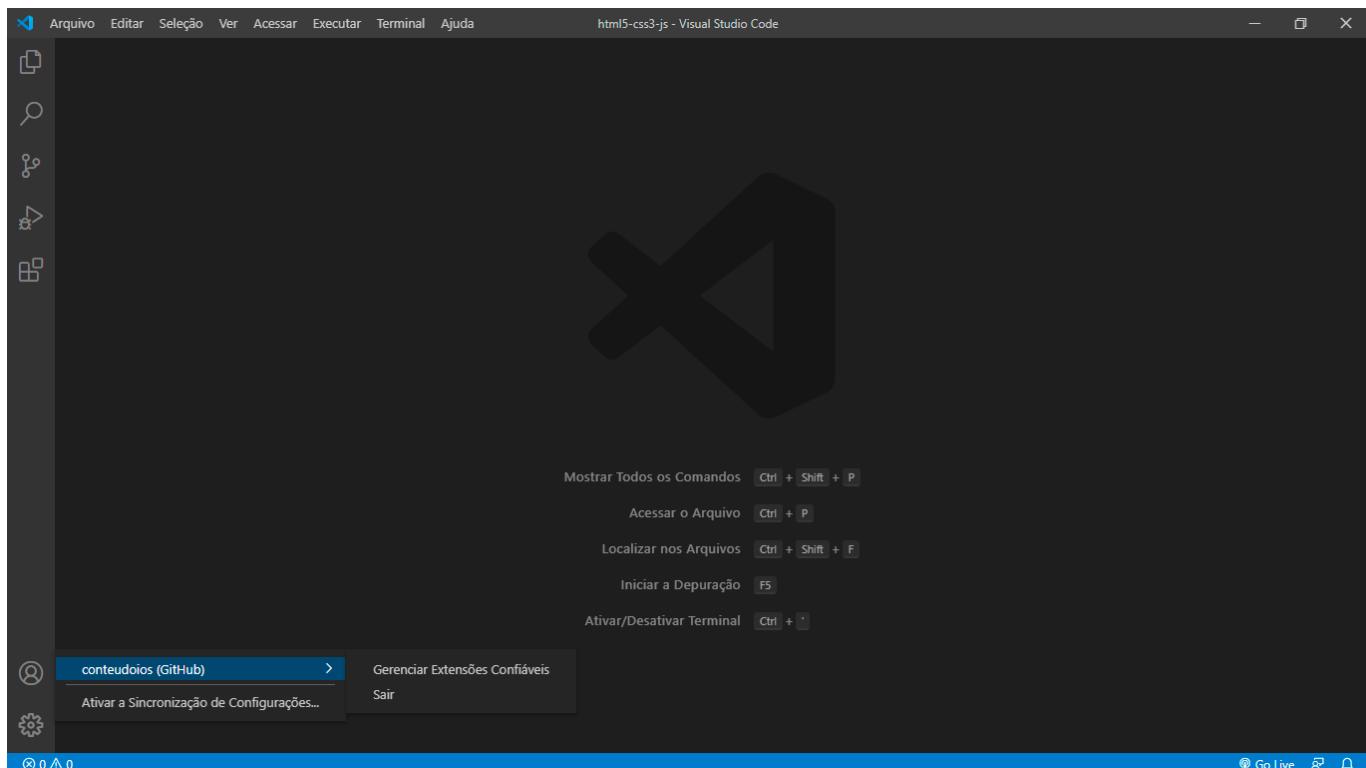
If you aren't redirected, you can add the token manually.

Your authorization token:

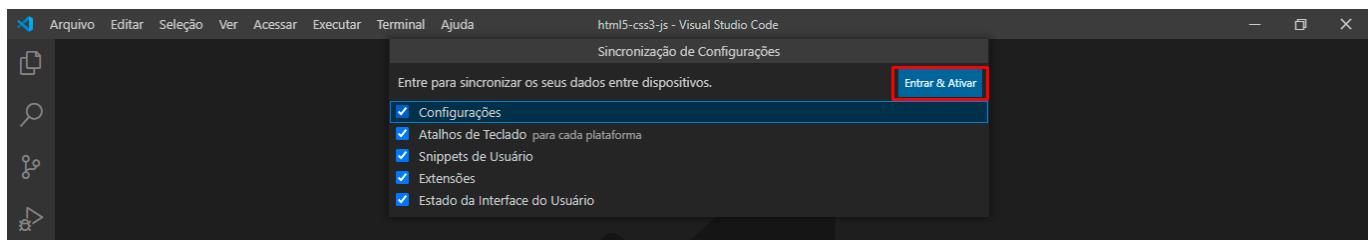
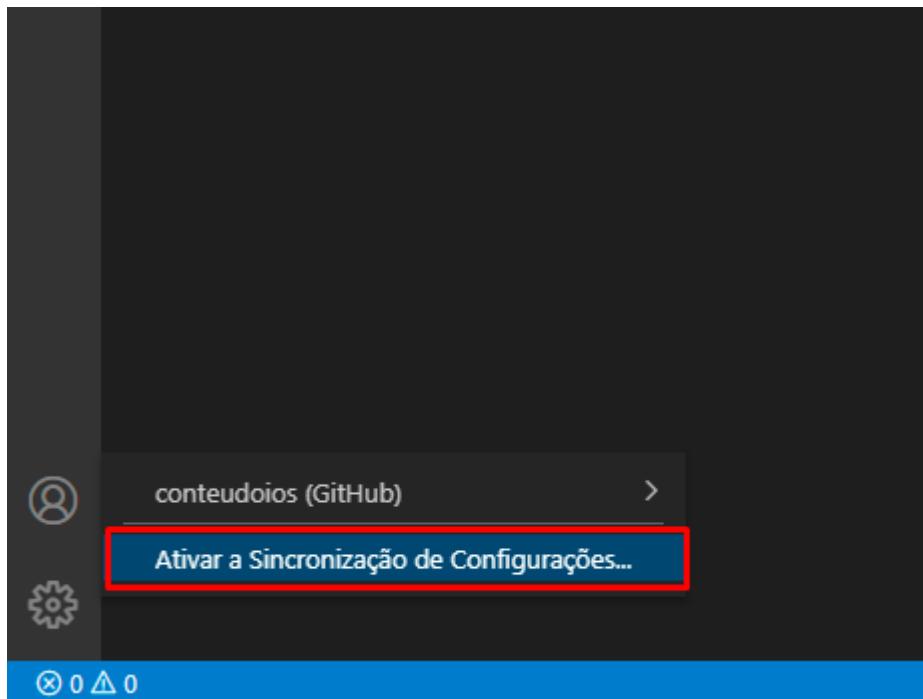
```
vscode://vscode.github-authentication/did-authenticate?  
windowId=1&code=0b143499f32a38b22c55&state=83bcb7ac-565f-4df7-a742-d896e4d897f7
```

1. Copy the token.
2. Switch back to VS code.
3. Click Signing in to github.com... in the status bar.
4. Paste the token and hit enter.

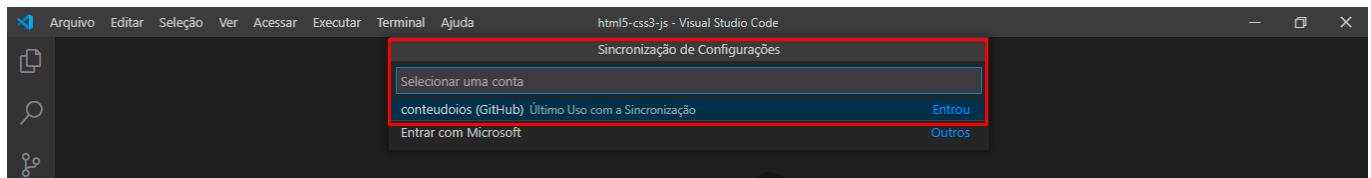
Tudo certo, agora **VS Code tem autorização para usar o seu GitHub**. Você pode conferir no Menu **Contas**, que está disponível na parte inferior da barra de Menu lateral do Visual Studio Code.



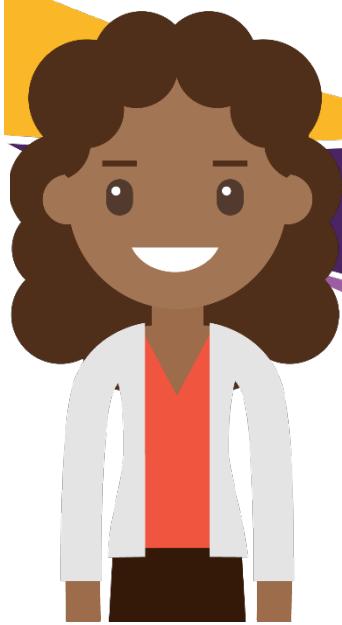
Clique em **Ativa nas Configurações de Sincronização**. Se for a primeira vez que você está configurando uma janela de pop-up com o título **Sincronização de Configuração** irá abrir (), marque todas as opções e clique em **Entrar & Ativar**.



Por fim, você deve selecionar uma conta, clique no seu Login no **GitHub**.



Caso a autorização não funcione, primeiro você deve reiniciar o **VS Code** e tentar novamente fazer autorização, repetindo os passos anteriores.



Comandos do Git

Os objetivos desta aula são:

- Aplicar o conhecimento aprendido criando e atualizando um repositório.

Bons estudos!

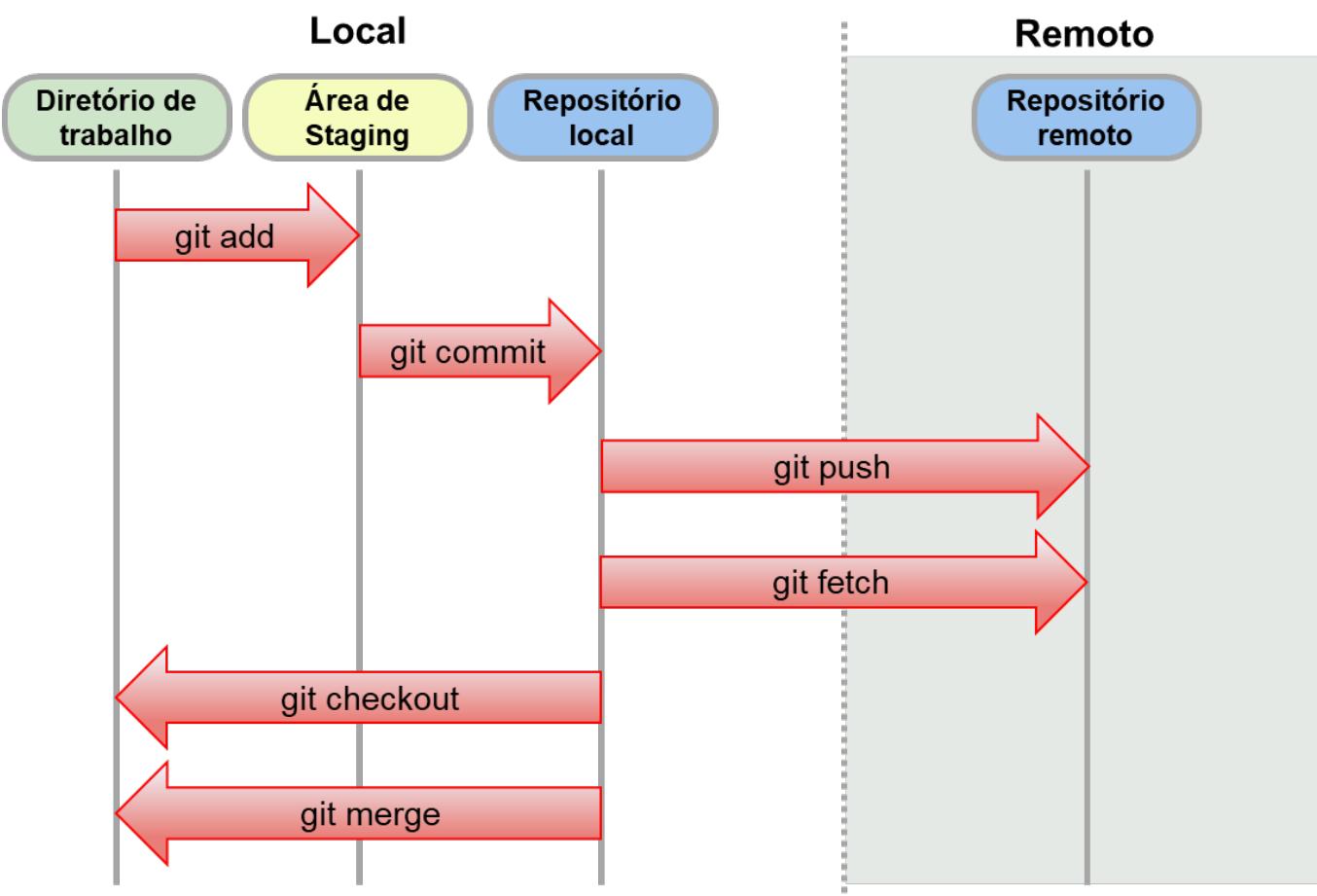
Aprendendo mais comandos do Git



Dica!

Procure digitar todos os comandos mostrados aqui para não dar nenhum erro ao executar um determinado comando. Se você apenas copiar de um arquivo word ou de pdf o comando no terminal de comando, geralmente o terminal interpreta alguns símbolos de forma diferente, mesmo que visualmente esteja muito parecido.

Outra coisa muito importante antes de começar esse tutorial, esteja com a **imagem a seguir na cabeça, para você saber o que é cada região: Diretório de trabalho, Staging Area, Repositório local e Repositório remoto**. E principalmente, lembre-se o alcance de cada comando, por exemplo o comando **git add** envia um arquivo do **Diretório de trabalho** para a **Staging area**.



Primeiros comandos no Git

Os comandos do Git são executados em um terminal de comandos, que todos sistema operacional possui:

No Linux: Linux Shell ou terminal

No macOS: Terminal

No Windows: PowerSheell, prompt de comando ou, se você fez a instalação do Git, terá a disposição do Git Bash.

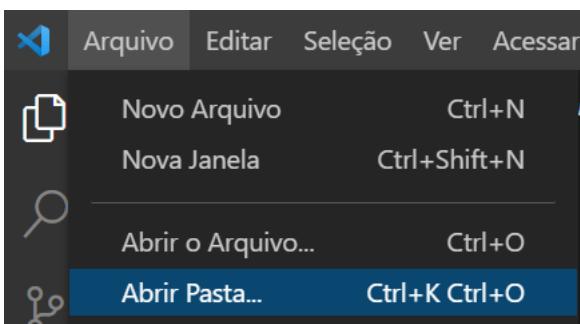
Nesse tutorial, usaremos o terminal da **IDE do VS Code**, que, no Windows, utiliza o terminal **powershell**. Os comandos mostrados nesse material é padrão para qualquer sistema operacional (SO) que você estiver usando, mas quando houver alguma diferença, como no Windows, destacaremos para você.

Vamos executar os primeiros comandos

Para você entender melhor as ações que devem ser realizadas e o que você deve esperar da execução de um comando, você deve seguir os seguintes passos:

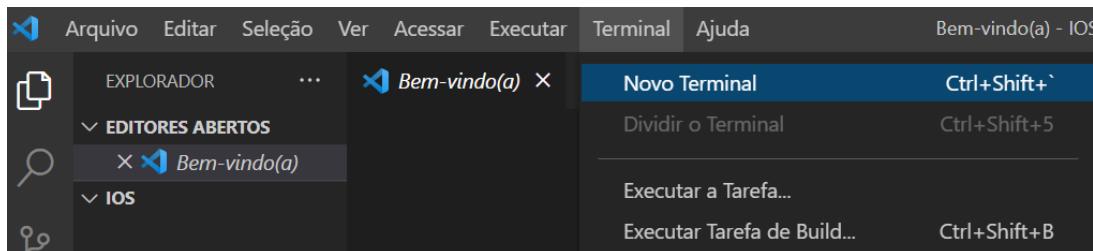
Crie um diretório chamado **IOS**.

Na IDE do VC Code, selecione o diretório **IOS** no seu computador para você trabalhar. Para isso, acesse o Menu **Arquivo** e selecione **Abrir Pasta** ou use a combinação dos atalhos **Ctrl+K** seguindo de **Ctrl+O**.

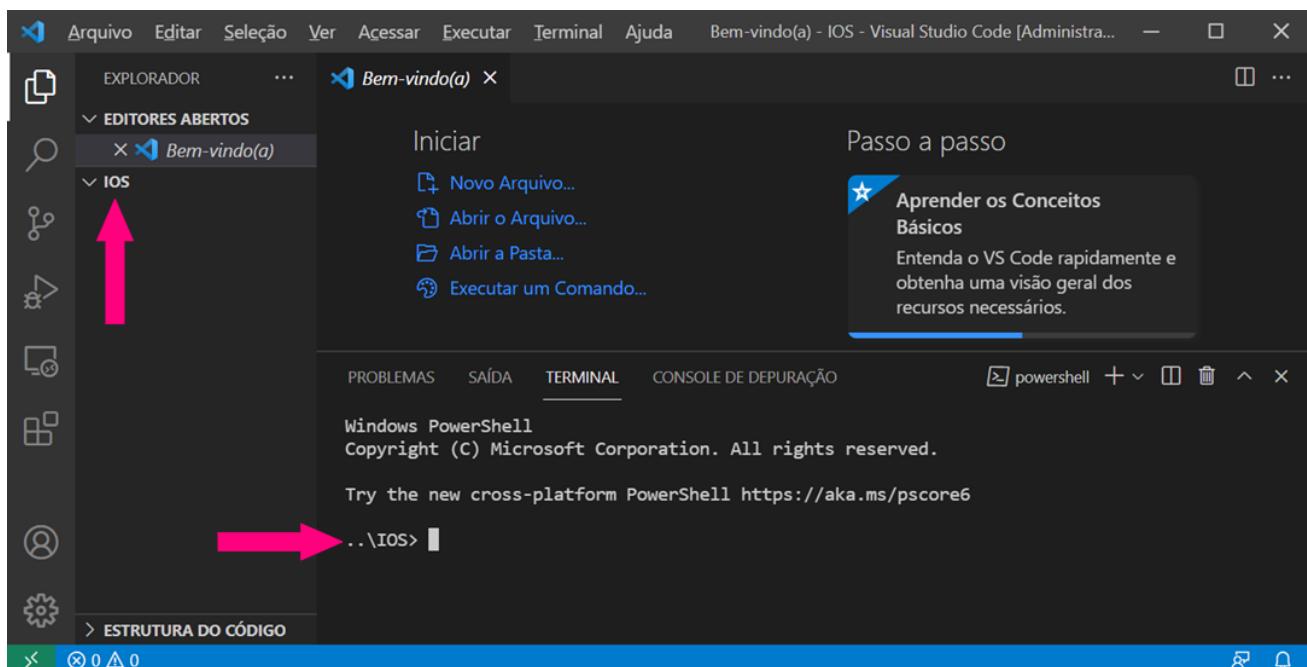


Na janela que irá abrir selecione um diretório dentro do seu computador para ser o diretório de trabalho.

Agora, você deve abrir o terminal no VS Code. Acesse o Menu **Terminal** e selecione a opção **Novo Terminal** ou use o atalho **Ctrl+Shift+`**



Se você escolheu a pasta corretamente, o terminal irá ser aberto já no seu diretório de trabalho.



O primeiro comando que você pode executar no terminal é para verificar a **versão do git instalado na sua máquina**. Então execute o comando:

```
git --version
```

Se você instalou tudo corretamente no seu computador aparecerá a mensagem com a **versão do git instalada**. No seu caso, o número da versão pode ser diferente da mostrada na imagem, mas isso não é problema. Agora se uma mensagem de erro aparecer, pode ser que você deverá fazer a instalação do git novamente.

```
..\IOS> git --version
git version 2.33.0.windows.2
..\IOS> []
```

Agora dentro desse diretório vamos criar alguns arquivos vazios para serem gerenciados. Existem várias formas para criar esses arquivos, mas como estamos trabalhando no terminal de comando, vamos fazer tudo por ele 😊.

Execute um dos dois comandos abaixo, dependendo do SO que você estiver usando, para criar arquivo **index.html** no diretório.



No Windows

```
New-Item index.html
```



No Linux, macOS ou Git Bash

```
touch index.html
```

Dica!

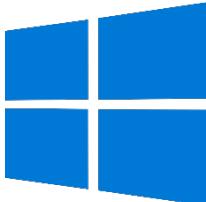


Existe uma forma de instalar o `touch` no Windows e desse modo utilizar o mesmo comando que os outros sistemas operacionais, mas é necessário ter o Node.js instalado na sua máquina. Então para não ficar um tutorial enorme, vamos citar as diferenças a medida que elas vão aparecendo. Mas se você quiser dar uma verificada de como fazer essa configuração, pode conferir em:

<https://dev.to/gautham495/use-touch-command-in-windows-10-1c8h>

Se o arquivo foi criado com sucesso uma mensagem parecida com mostrada abaixo irá aparecer no seu terminal.

No Windows



```
..\\IOS> New-Item index.html

Directory: C:\Users\Bigode\Documents\IOS

Mode          LastWriteTime    Length Name
----          -----          --  -
-a---          9/20/2021     19:03      0 index.html

..\\IOS> []
```

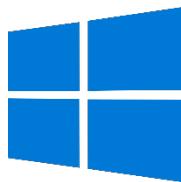
No Linux, macOS ou Git Bash



```
MINGW64:/c/Users/Bigode... ━ ━ X
Bigode@Bigode MINGW64 ~/Documents/IOS
$ touch index.html

Bigode@Bigode MINGW64 ~/Documents/IOS
$
```

Vamos criar mais um arquivo chamado `app.js`. Então, no terminal, execute o comando:



No Windows

```
New-Item app.js
```



No Linux, macOS ou Git Bash

```
touch app.js
```

Agora, você tem **dois arquivos criados no seu diretório**. Esses arquivos aparecem na aba explorador do **VS Code**.



Insira o código simples no arquivo **index.html**. Abra o arquivo no **VS Code** e insira o código mostrado abaixo.

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
    <title>Meu site</title>
</head>

<body>
    Esse é meu site.
</body>

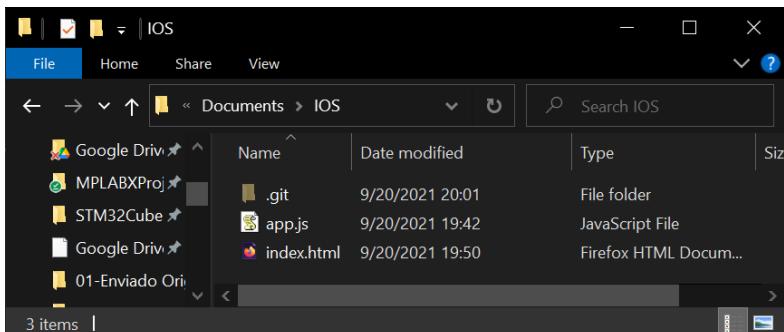
</html>
```

Com o arquivo **index.html** salvo, vamos agora **inicializar o nosso repositório git localmente** no computador. Para isso, temos que voltar ao **terminal do VS Code** e executar o comando:

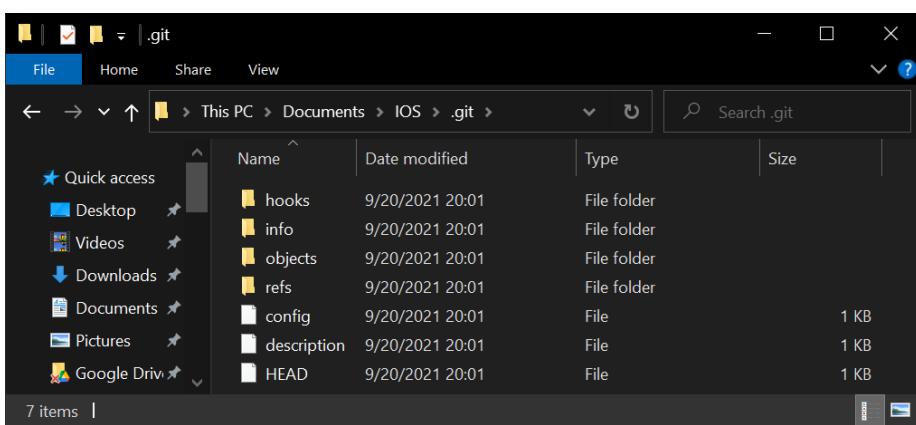
```
git init
```

```
..\IOS> git init
Initialized empty Git repository in C:/Users/Bigode/Documents/IOS/.git/
..\IOS> |
```

Ao executar esse comando, você está criando uma **pasta oculta git no seu diretório de trabalho**. Se estiver no Windows, habilite a opção “**Mostrar arquivos ocultos**”, também é interessante você desmarcar a opção “**Ocultar extensões de arquivos conhecidos**”.



Se você entrar no diretório, visualizará diversos diretórios e arquivos que são necessários para o **git gerenciar o repositório**.



Observe no canto inferior esquerdo da **IDE do VS Code**, que agora ele está indicando que você está na **master** do seu repositório local. Na verdade, esse **master no VS Code é um botão que você pode pressionar para criar ramificações ou fazer checkout no repositório**.

```
..\\IOS> New-Item app.js
Directory: C:\\Users\\Bia\\Desktop\\IOS

Mode          LastW
----          -----
-a-----      9/20/2021

..\\IOS> git init
Initialized empty Git repository in ..\\IOS\\.
..\\IOS> 
```

Importante!

 O nome **master** está sendo substituído por **main** em muitos repositórios como o **GilLab** e **GitHub**, mas durante a produção desse material não foi alterado no **Git** e nem no **VS Code**.

Você deve agora vincular o seu **nome** e **endereço de e-mail** ao repositório. Para fazer isso execute o comando:

```
git config --global user.name 'Coloque aqui o seu nome'
```

E depois o comando para configurar o e-mail:

```
git config --global user.email 'Coloque aqui o seu email'
```

PROBLEMAS	SAÍDA	TERMINAL	CONSOLE DE DEPURAÇÃO
			..\\IOS> git config --global user.name 'Fernando' ..\\IOS> git config --global user.email 'fernando@esquario.com.br' ..\\IOS> █



Importante!

Não copie e cole o código, pois pode acontecer de nessa transição algum caractere seja colado errado, principalmente os dois sinais de subtração (--). Mesmo que esteja muito parecido, digite o código manualmente, para ter certeza de que o comando está correto.

Você pode verificar se a configuração foi realizada com sucesso através do comando:

```
git config --list
```

PROBLEMAS	SAÍDA	TERMINAL	CONSOLE DE DEPURAÇÃO
			Try the new cross-platform PowerShell https://aka.ms/powershell ..\\IOS> git config --list diff.astextplain.textconv=astextplain filter.lfs.clean=git-lfs clean -- %f filter.lfs.smudge=git-lfs smudge -- %f filter.lfs.process=git-lfs filter-process filter.lfs.required=true http.sslbackend=openssl http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt core.autocrlf=true core.fscache=true core.symlinks=true pull.rebase=false credential.helper=manager-core credential=https://dev.azure.com/usehttppath=true init.defaultbranch=master filter.lfs.clean=git-lfs clean -- %f filter.lfs.smudge=git-lfs smudge -- %f filter.lfs.process=git-lfs filter-process filter.lfs.required=true user.name=Fernando user.email=fernando@esquario.com.br core.editor="C:\\Users\\Bigode\\AppData\\Local\\Programs\\Microsoft VS Code\\Code.exe" --wait core.repositoryformatversion=0 core.filemode=false core.bare=false core.logallrefupdates=true core.ignorecase=true ..\\IOS> █

Atenção: Caso você digitar esse comando e o **prompt do terminal ser substituído por dois pontos**, significado que você entrou na **configuração do git**, para **sair** digite **q** ou **wq** para sair da configuração.

O prompt padrão normal deve ter o sinal **de maior que >** ou de **\$**, dependendo do **SO** que você está usando.

Prompt normal

```
..\IOS> █ ou $ |
```

Prompt de configuração do git

```
:█
```

Os dois arquivos criados **não estão a área de staging do repositório**. Você pode conferir através do comando:

```
git status
```

```
..\IOS> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app.js
    index.html

nothing added to commit but untracked files present (use "git add" to track)
..\IOS> █
```

O **vermelho** indica que estão fora da **staging area** e os arquivos **não estão sendo rastreados**.

Esse comando de **status** é interessante também, porque ele mostra em qual **branch** estamos trabalhando, que nesse caso é na **master (main)** e, também. Se houve algo **commit**.

```
..\IOS> git status
On branch master

No commits yet
```

Então vamos **adicionar ao repositório** apenas o arquivo **index.html** através do comando:

```
git add index.html
```

```
..\IOS> git add index.html
..\IOS> █
```

O git não exibe nenhuma mensagem, mas se você verificar o **status** é possível notar a mudança:

```
..\IOS> git status
On branch master

No commits yet

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

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

..\IOS> █
```

Agora o nosso arquivo **index.html** está na **staging area** e apenas o arquivo **app.js** não está sendo rastreado.

Você pode retirar um arquivo do **staging area** através do comando:

```
git rm --cached index.html
```

```
..\IOS> git rm --cached index.html
rm 'index.html'
..\IOS> █
```

Se você verificar o **status** do git agora, **voltou como estava no início**, ou seja, **ambos os arquivos não estão sendo rastreados**:

```
..\IOS> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app.js
    index.html

nothing added to commit but untracked files present (use "git add" to track)
..\IOS> █
```

Você pode adicionar vários arquivos com a mesma extensão através do comando:

```
git add *.html
```

Se você conferir o **status** do git **todos os arquivos com a extensão .html estão na staging area**.

```
..\IOS> git add *.html
..\IOS> git status
On branch master

No commits yet

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

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

..\IOS> █
```

Vamos retirar o arquivo **index.html** da **staging area** novamente.

```
git rm --cached index.html
```

E agora vamos adicionar **todos** os arquivos **sem exceção** na **staging area** através do comando:

```
git add .
```

Se você conferir o **status** do git todos os arquivos estão na **staging area**.

```
..\IOS> git add .
..\IOS> git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file: app.js
  new file: index.html

..\IOS> █
```

Agora vamos modificar o arquivo **index.html** e colocar **uma nova linha do body**.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title>Meu site</title>
  </head>
  <body>
    Esse é meu site.
    Coloquei uma nova linha.
  </body>
</html>
```

Salve o arquivo e verifique o **status do git**. Observe que o git “**reclama**” que existe uma **versão no diretório local diferente da versão na staging area**.

```
..\IOS> git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:  app.js
  new file:  index.html
```

→ staging area


```
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
  modified:  index.html
```

→ Diretório local

..\IOS> █

Para resolver isso precisamos adicionar o arquivo na **staging area**.

```
git add .
```

```
git add index.html
```

Ao verificar o **status** logo após o comando **git add**, você verá que agora o **git não “reclama” mais.**

```
..\IOS> git add .
..\IOS> git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:  app.js
  new file:  index.html
```

..\IOS> █

Importante!



Você não precisa ficar executando o comando **git add** a cada modificação simples no arquivo que está trabalhando. O ideal fazer isso sempre que fizer mudanças significativas ou no momento que você para descansar ou quando quiser testar uma atualização para produção. Colocar em produção muitas vezes é necessário não só fazer o **add** e o **commit** como enviar para o repositório remoto.

Agora vamos executar o **commit** dos arquivos com o comando:

```
git commit 'Mensagem'
```

```
..\IOS> git commit -m 'Fazendo o commit inicial'
[master (root-commit) da44491] Fazendo o commit inicial
 2 files changed, 9 insertions(+)
 create mode 100644 app.js
 create mode 100644 index.html
..\IOS>
```

Se você tentar fazer o **commit novamente** irá aparecer uma mensagem que **não há nada para ser “commitado”**.

```
..\\IOS> git commit -m 'Fazendo o commit inicial'  
[master (root-commit) da44491] Fazendo o commit inicial  
2 files changed, 9 insertions(+)  
create mode 100644 app.js  
create mode 100644 index.html  
..\\IOS> git commit -m 'Outro commit'  
On branch master  
nothing to commit, working tree clean
```

Mensagem

Se você verificar o **status** do **git**, verá que não tem mais **nenhum arquivo para ser “commitado”** na **staging area**, pois agora eles já estão no repositório **local do git**.

```
..\\IOS> git status  
On branch master  
nothing to commit, working tree clean  
..\\IOS>
```

Vamos inserir o seguinte código no arquivo **app.js**:

```
console.log('Olá alunos!');
```

Salve o arquivo **app.js** e verifique o **status** do **git** e você verá que o arquivo **app.js** foi modificado e está **diferente do arquivo no repositório local**:

```
..\\IOS> git status  
On branch master  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
    modified:   app.js  
  
no changes added to commit (use "git add" and/or "git commit -a")  
..\\IOS>
```

Então vamos adicionar novamente o arquivo para a **staging area**.

```
..\\IOS> git add .  
..\\IOS> git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   app.js  
  
..\\IOS>
```

E em seguida fazer o **commit** para o **repositório local**.

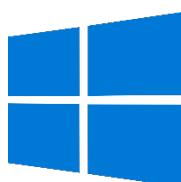
```
..\\IOS> git commit -m 'Atualização do arquivo app.js'  
[master 736eac2] Atualização do arquivo app.js  
1 file changed, 1 insertion(+)  
..\\IOS> git status  
On branch master  
nothing to commit, working tree clean  
..\\IOS>
```

Usar o arquivo .gitignore

O arquivo **.gitignore** é um arquivo de texto onde cada linha contém um **padrão/regra** para **arquivos ou diretórios que serão ignorados**, ou seja, **que não serão rastreados pelo git**. Isso é interessante, quando você estiver trabalhando com pacotes de bibliotecas de terceiros ou com frameworks e **não quer armazenar no seu git esses arquivos**, apenas a sua aplicação.

Vamos continuar “**brincando**” no terminal de comando do VS Code e aprender a usar esse arquivo. Para isso, siga os seguintes passos:

Vamos então incluir o arquivo **.gitignore** no nosso repositório. Para isso, execute o comando:



No Windows

```
New-Item .gitignore
```



No Linux, macOS ou Git Bash

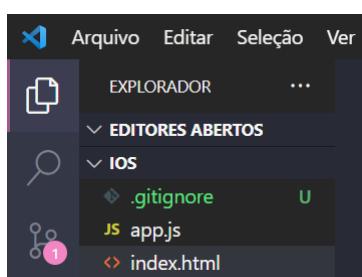
```
touch .gitignore
```



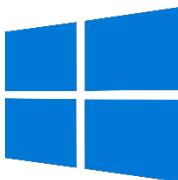
Importante!

No Windows, só é possível criar arquivos sem nome somente com a extensão através do terminal de comando.

Se você visualizar a aba **explorador** o arquivo está dentro do diretório de trabalho.



Agora, vamos **criar um arquivo que não desejamos incluir no repositório**. Esse arquivo será **to_do_list.txt**, que irá conter a lista de coisa a fazer no projeto e não precisa ter sua versão controlada pelo git.



No Windows

```
New-Item to_do_list.txt
```



No Linux, macOS ou Git Bash

```
touch to_do_list.txt
```

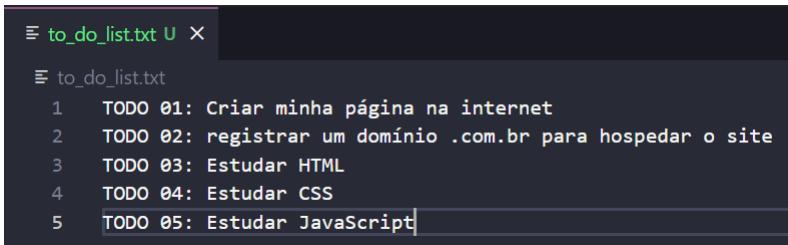
```
..\IOS> New-Item to_do_list.txt

Directory: C:\Users\Bigode\Documents\IOS

Mode          LastWriteTime      Length Name
----          <-----           ----- 
-a---        9/20/2021    22:46          0 to_do_list.txt

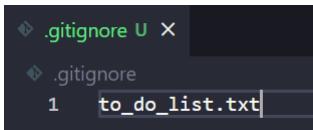
..\IOS> []
```

Vamos **abrir** o arquivo no **VS Code** e colocar qualquer conteúdo.



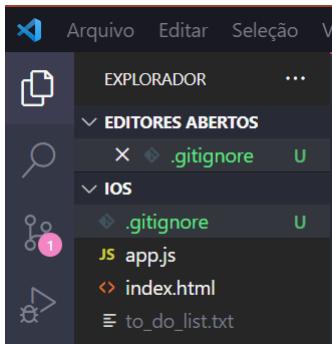
```
to_do_list.txt U X
to_do_list.txt
1 TODO 01: Criar minha página na internet
2 TODO 02: registrar um domínio .com.br para hospedar o site
3 TODO 03: Estudar HTML
4 TODO 04: Estudar CSS
5 TODO 05: Estudar JavaScript
```

Agora vamos colocar o nome do arquivo **to_do_list.txt** no dentro do arquivo **.gitignore** para indicar que esse **arquivo será ignorado pelo git**.



```
.gitignore U X
.gitignore
1 to_do_list.txt
```

Salve o arquivo **.gitignore** e veja na aba **explorador** que **o arquivo fica na cor cinza**. Indicando que ele é um arquivo **ignorado nesse repositório**.



Voltando ao terminal do **VS Code**. Vamos verificar o **status** do **git**. Observe que apenas o arquivo **.gitignore** está sendo **mostrado como não rastreado**, o arquivo **to_do_list.txt** é ignorado nesse momento.

```
..\IOS> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

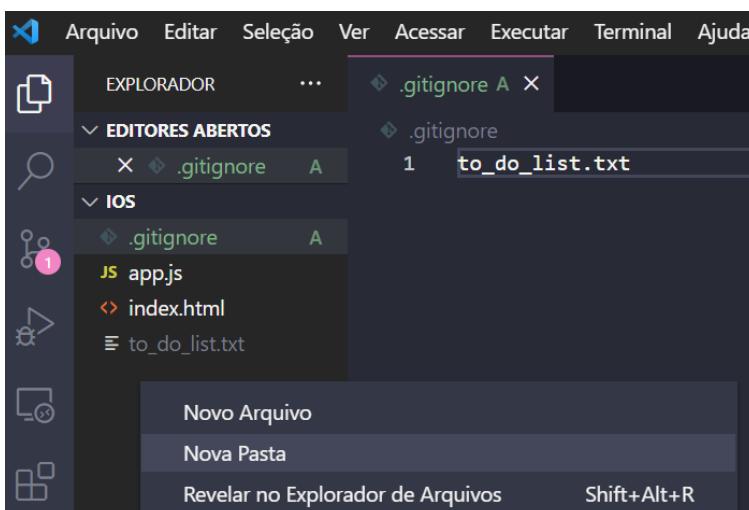
nothing added to commit but untracked files present (use "git add" to track)
..\IOS> █
```

Vamos fazer um **add** de **todos os arquivos para a staging area**. Ao verificar o **status**, apenas o arquivo **.ignore** foi **adicionado**.

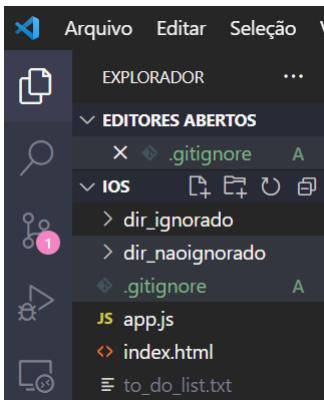
```
..\IOS> git add .
..\IOS> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .gitignore

..\IOS> █
```

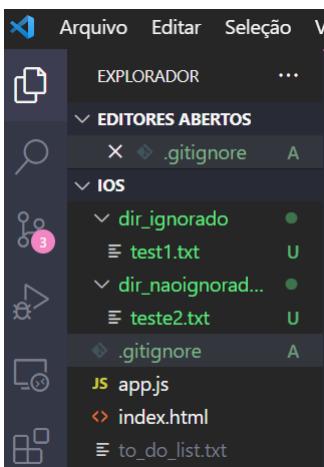
Você também pode ignorar **diretórios completos**. Vamos criar um diretório no projeto, clique com o botão direito na região do projeto, que não seja em um arquivo do mesmo e escolha nova pasta.



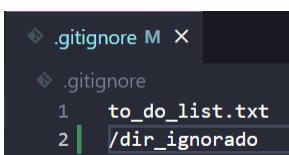
Crie **dois diretórios** como mostrado na figura abaixo:



E crie um arquivo dentro desses diretórios:



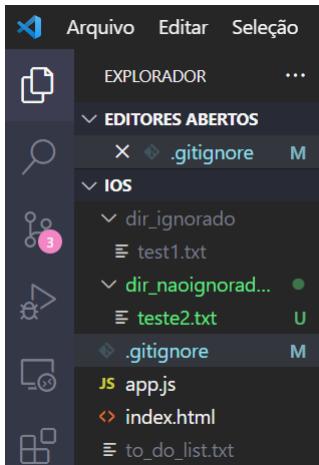
Agora, você deve inserir o **nome desse diretório no arquivo .gitignore** para ser ignorado pelo git. Observe que para diretório é necessário colocar a barra (/).



```
❶ .gitignore M X
❷ .gitignore
1   to_do_list.txt
2   /dir_ignorado
```

Salve o arquivo **.gitignore** e veja que o **diretório ignorado está na cor do diretório e o outro está normal.**

Observação: essa configuração fará o git ignorar o diretório e todos os arquivos dentro dele.



Se você verificar o **status** do git ,verá que ele “reclamará” do **arquivo .gitignore que está atualizado e diferente do que está na staging area.**

```
..\IOS> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  .gitignore

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:  .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    dir_naoignorado/
..\IOS> █
```

Vamos fazer o **add** e o **commit** desses arquivos.

```
..\IOS> git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
  add
..\IOS> git add .
..\IOS> git commit -m 'Arquivo .gitignore e novo diretorio'
[master f0eb54e] Arquivo .gitignore e novo diretorio
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 dir_naoignorado/test2/teste2.txt
..\IOS> █
```

Se você verificar o **status** do **git**, verá que **não existe nada para ser adicionado na staging area e para ser “commitado” no diretório local.**

```
..\IOS> git status
On branch master
nothing to commit, working tree clean
..\IOS> █
```

Você pode ver o **histórico de commits** do repositório com o comando:

```
git log
```

```
..\IOS> git log
commit f0eb54e7a405b7711b60dca8252af224502e46e5 (HEAD -> master)
Author: Fernando <fernando@esquирio.com.br>
Date:   Mon Sep 20 23:19:30 2021 +0200

    Arquivo .gitignore e novo diretório

commit 736eac207023ceb6cc5ed5ec6fe6cf519b1421f
Author: Fernando <fernando@esquирio.com.br>
Date:   Mon Sep 20 22:16:02 2021 +0200

    Atualização do arquivo app.js

commit da44491d7aca72f6f810c9ccc552ee9ddd50c991
Author: Fernando <fernando@esquирio.com.br>
Date:   Mon Sep 20 21:51:15 2021 +0200
....skipping...
commit f0eb54e7a405b7711b60dca8252af224502e46e5 (HEAD -> master)
Author: Fernando <fernando@esquирio.com.br>
Date:   Mon Sep 20 23:19:30 2021 +0200

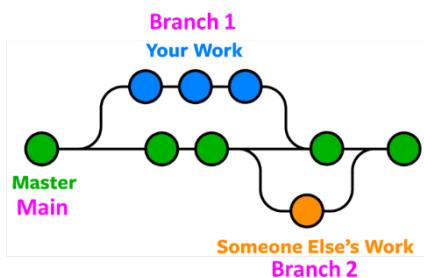
    Arquivo .gitignore e novo diretório

commit 736eac207023ceb6cc5ed5ec6fe6cf519b1421f
Author: Fernando <fernando@esquирio.com.br>
Date:   Mon Sep 20 22:16:02 2021 +0200
    Usuário e e-mail
    de quem fez o
    commit
commit da44491d7aca72f6f810c9ccc552ee9ddd50c991
Author: Fernando <fernando@esquирio.com.br>
Date:   Mon Sep 20 21:51:15 2021 +0200
    ↑
    Fazendo o commit inicial → Mensagem do commit
```

Para sair do log do git você pode **pressionar a tecla q**.

Criando branches

No contexto de versionamento de arquivos e gerenciamento de configuração de software, **branch é a duplicação de um objeto sob controle de versão, tal como um código fonte ou uma árvore de diretório**. Isso quer dizer que **branches são ramificações ou bifurcações dentro de um projeto git**. Criando uma branch de um código em desenvolvimento, você pode modificá-lo paralelamente e, desse modo, é possível que tenhamos diferentes versões baseadas em um mesmo ponto do tempo. Branches são muito úteis para que a equipe se organize, para que um desenvolvedor faça a correção do link em uma branch, enquanto outro desenvolvedor continua commitando em outra branch, sem interferirem um no processo do outro.



Você como uma pessoa desenvolvedora e **precisará fazer uma tarefa ou adicionar um novo recurso em um software em desenvolvimento**, você não vai trabalhar diretamente na master (main), deve fazer uma **branch** (parcial ou total do projeto) e trabalhar nessa **duplicação do código e, só depois de implementar e testar, você poderá fazer o merge na master (main)**.

Vamos criar uma **branch** no nosso **repositório**. Para isso você deve seguir os passos:

Você pode criar uma **branch** com o comando

```
git branch Nome_do_Branch
```

```
..\IOS> git branch mybranch
..\IOS> █
```

Executar o comando acima, **cria a branch, mas não alterar onde você está trabalhando**. Se você verificar com o comando **git status**, verá que **ainda está na master (main)**.

```
..\IOS> git status
On branch master
nothing to commit, working tree clean
..\IOS> █
```

Para **alterar para a branch criada**, você de executar o comando:

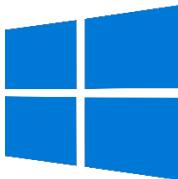
```
git checkout Nome_do_Branch
```

```
..\IOS> git checkout mybranch
Switched to branch 'mybranch'
..\IOS> █
```

Se você verificar o **status do git** verá que agora **você está na nova branch criada**.

```
..\IOS> git status → Branch atual
On branch mybranch
nothing to commit, working tree clean
..\IOS> █
```

Agora estamos na branch criada. Então vamos supor que queremos criar um recurso de login na página, então devemos criar um novo arquivo:



No Windows

```
New-Item login.html
```



No Linux, macOS ou Git Bash

```
touch login.html
```

```
..\IOS> New-Item login.html

Directory: C:\Users\Bigode\Documents\IOS

Mode                LastWriteTime     Length Name
----                <-----           ----- 
-a---    9/21/2021      9:37          0 login.html

..\IOS> █
```

Abra o arquivo **login.html** e vamos inserir o código abaixo:

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <title>Meu site</title>
</head>

<body>
  <h1>Página de login</h1>
</body>

</html>
```

Agora, abra o arquivo **index.html** e vamos atualizar o código dele:

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <title>Meu site</title>
</head>

<body>
  Esse é meu site.
</body>
```

```

Coloquei uma nova linha.
Formulário de login
</body>

</html>

```

Então, a **diferença entre essa branch e a master** é um arquivo **login.html** e o arquivo **index.html** com uma atualização.

Vamos adicionar os arquivos na **staging area** da **branch atual de trabalho (mybranch)**.

```
git add .
```

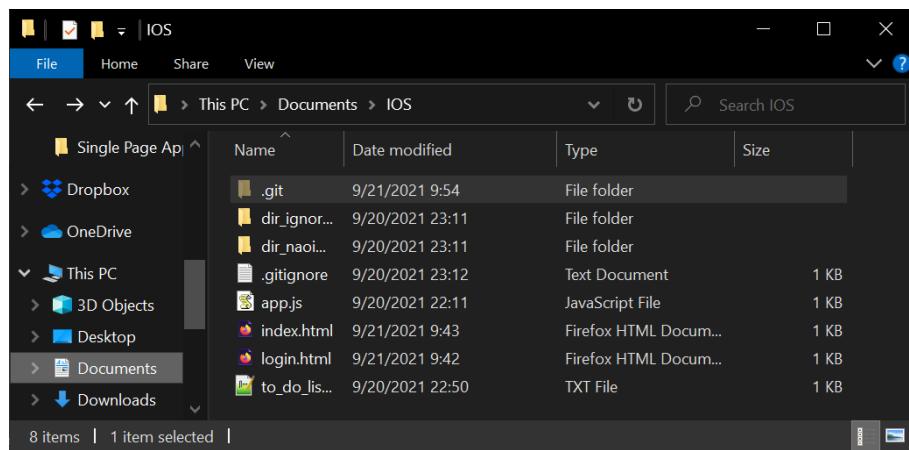
```
..\IOS> git add .
..\IOS> █
```

Vamos fazer o **commit** desses arquivos na **branch atual de trabalho (mybranch)**:

```
git commit -m 'formulario de login'
```

```
..\IOS> git commit -m 'formulario de login'
[mybranch aa53188] formulario de login
 2 files changed, 2 insertions(+)
  create mode 100644 login.html
..\IOS> █
```

Se você abrir o **diretório** no **Windows Explorer** verá todos os arquivos da **mybranch** nele.

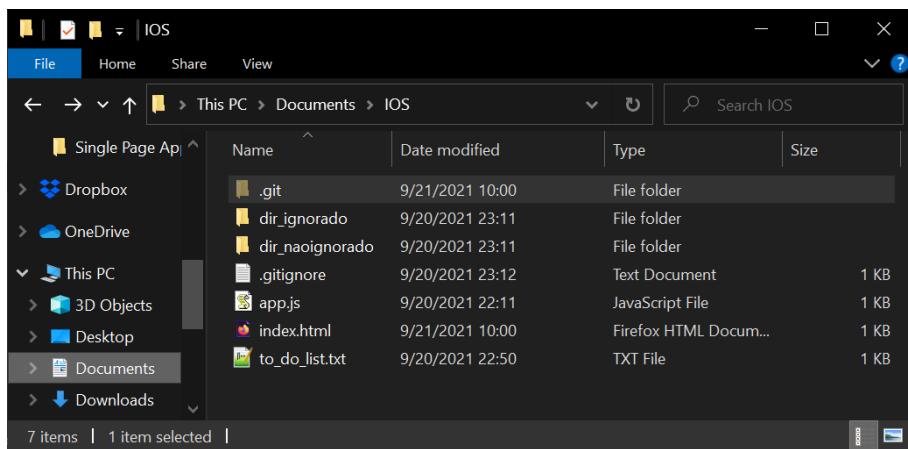


Agora, vamos voltar para a **master (main)**.

```
git checkout master
```

```
..\IOS> git checkout master
Switched to branch 'master'
..\IOS> █
```

Observe que o arquivo **login.html** não está mais aparecendo no diretório, pois ele pertence **apenas a branch mybranch** e não foi ainda feito o **merge para a master (main)**. Se você abrir o arquivo **index.html**, verá que ele **não mostra a atualização realizada anteriormente**.



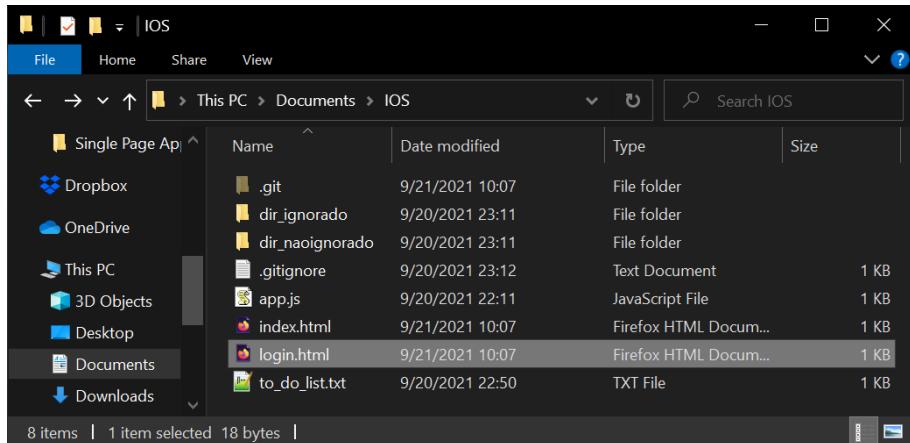
Para que ele **apareça na sua master (main)**, você deve fazer o **merge das branches (mybranch e master)**.

Para isso, **tenha certeza de que você está na branch master** e digite o comando:

```
git merge Nome_da_Branch -m 'Mensagem'
```

```
..\IOS> git merge mybranch -m 'Adicionando a mybranch a master'
Updating f0eb54e..aa53188
Fast-forward (no commit created; -m option ignored)
 index.html | 2 +-
 login.html | 1 +
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 login.html
..\IOS> █
```

Agora, você consegue **ver o arquivo login.html no diretório** e se você abrir o arquivo **index.html** vera a atualização.



Dica!

Existe muito mais assuntos sobre **branching** e **merging** no git, mas como estamos começando a aprender o assunto, deixaremos mais simples para você ir se acostumando com os conceitos. **É bem interessante você utilizar e testar branches em projetos pessoais para acostumar-se com esse tipo de tarefa.**

Pull em um repositório remoto

Até então o nosso repositório é local, ou seja, o controle de versionamento está no computador que você está trabalhando. Vamos, portanto, configurar e enviar nosso repositório local para um repositório remoto.

Vamos utilizar o GitHub como repositório remoto.

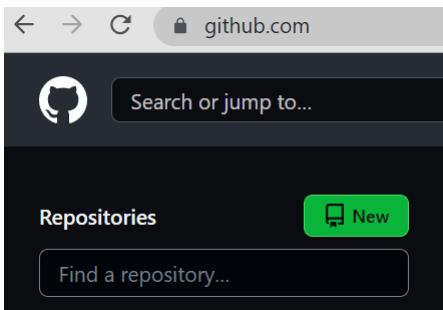


Dica!

É possível deixar seu github com uma aparência muito bacana, você pode aprender assistindo ao vídeo: <https://www.youtube.com/watch?v=TsaLQAetPLU>

Siga os passos para fazer isso:

Após fazer o login sua conta, você deve **criar um novo repositório**. Na lateral esquerda da tela, você encontra um label **Repositories** e um botão **New**. Clique nesse botão para criar seu repositório.



Preencha os campos de **Nome do Repositório** e **Descrição como você desejar**.

Você tem as opções de **Public** ou **Private**. **Public** qualquer um pode ver o seu repositório e **Private** somente quem tiver a sua permissão poderá ver. **A partir de 2019, o GitHub incluiu a possibilidade de criar repositórios privados ilimitado na conta gratuita.**

Você pode escolher **Add a README file**, que é uma opção interessante para escrever o **Markdown** do repositório. Deixa essa opção desmarcada.

Você pode escolher **Add .gitignore**, mas como nós criamos esse arquivo manualmente, deixaremos essa opção desmarcada.

Depois de preenchidos os campos e escolhidas as opções, você pode clicar no botão **Create repository**.

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 ***

 Esquiro / Teste_IOS 

Great repository names are short and memorable. Need inspiration? How about [congenial-invention](#)?

Description (optional)

Exemplo usado para o tutorial

 **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](#).

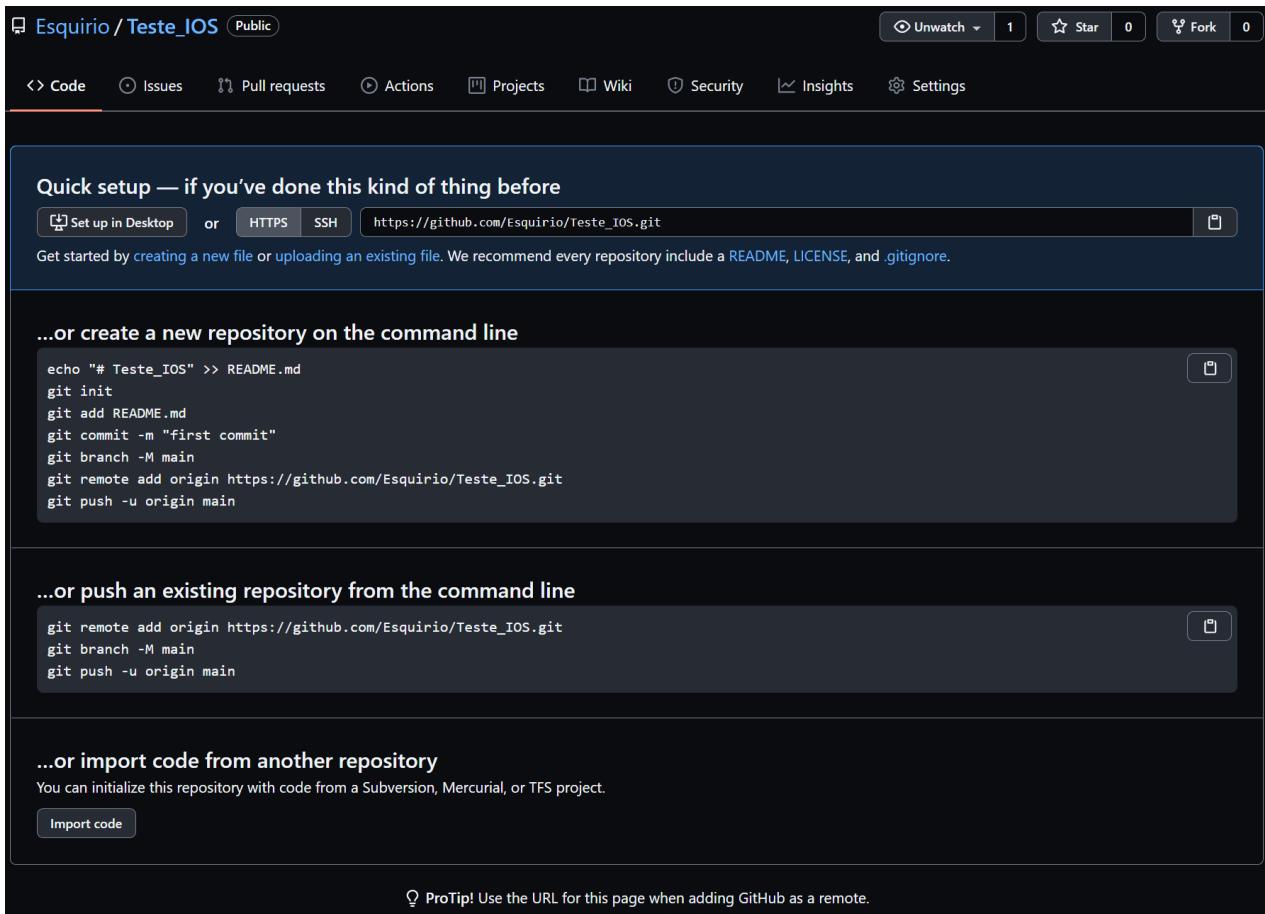
Add .gitignore
Choose which files not to track from a list of templates. [Learn more](#).

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

 **Create repository**

Agora o nosso **repositório remoto está criado**. Então, a interface do GitHub mostrará **o repositório com nada dentro dele**. Estamos interessados na mensagem **...or create a new repository on the command line**. Nessa mensagem temos os comandos necessários para vincular o repositório local ao repositório remoto:

Comando	Descrição
git init	Inicializa o repositório localmente (já fizemos no início do tutorial)
git add README.md	Adiciona um arquivo README.md, que é interessante para você escrever informações de como usar o seu repositório. Por exemplo: se você construiu uma biblioteca, pode descrever aqui um passo a passo de como utilizar a sua biblioteca.
git commit -m "first commit"	Comando para fazer o primeiro commit (também já fizemos no início do tutorial)
git branch -M main	Comando para criar a branch main (master)
git remote add origin https://github.com/Esquирio/Teste_IOS.git	Comando para associar o repositório local ao repositório remoto.
git push -u origin main	Comando para enviar os arquivos para o repositório online.



The screenshot shows a GitHub repository page for 'Esquирio / Teste_IOS'. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The repository is public, has 1 watch, 0 stars, 0 forks, and 0 issues. A 'Quick setup' section provides instructions for creating a new repository on the command line, showing the following terminal commands:

```
Set up in Desktop or HTTPS SSH https://github.com/Esquирio/Teste_IOS.git
echo "# Teste_IOS" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Esquирio/Teste_IOS.git
git push -u origin main
```

Below this, another section shows commands for pushing an existing repository from the command line:

```
git remote add origin https://github.com/Esquирio/Teste_IOS.git
git branch -M main
git push -u origin main
```

At the bottom, there's a note about importing code from another repository, a 'Import code' button, and a pro tip: 'ProTip! Use the URL for this page when adding GitHub as a remote.'



Importante!

Se na hora de você criar o seu repositório, você **marcar a opção Add README file**. O GitHub **não exibirá as informações como mostrar nessa página**. Desse modo, não vem a informação com os comandos para **criar o repositório por linha de comando ou dar um push de um repositório existente através de linha de comando**.

Precisamos vincular o nosso repositório local ao repositório remoto. Pois se executarmos o comando no terminal do VS Code:

```
git remote
```

```
..\IOS> git remote  
..\IOS> █
```

Ele mostra que **existe repositório remoto vinculado a esse repositório local**.

Então para vincular o repositório local ao repositório remoto que acabamos de criar, basta executar o comando:

```
git remote add origin <Endereço do seu repositório>
```

No nosso exemplo, o comando é:

```
git remote add origin https://github.com/Esquирio/Teste_IOS.git
```

```
..\IOS> git remote add origin https://github.com/Esquирio/Teste_IOS.git  
..\IOS> git remote  
origin  
..\IOS> █
```

Pronto, agora o **repositório local está vinculado ao repositório remoto** e podemos então fazer **push** para **enviar os arquivos ficarem armazenados online**. Para isso execute o comando:

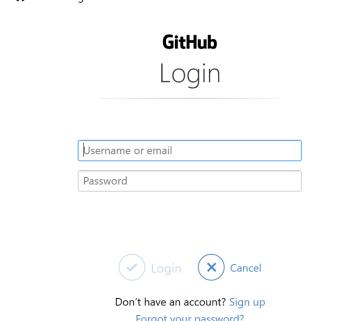
```
git push -u origin master
```

```
..\IOS> git push -u origin master  
Enumerating objects: 16, done.  
Counting objects: 100% (16/16), done.  
Delta compression using up to 12 threads  
Compressing objects: 100% (10/10), done.  
Writing objects: 100% (16/16), 1.40 KiB | 1.40 MiB/s, done.  
Total 16 (delta 2), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (2/2), done.  
To https://github.com/Esquирio/Teste_IOS.git  
 * [new branch]      master -> master  
Branch 'master' set up to track remote branch 'master' from 'origin'.  
..\IOS> █
```

Observação1: quando criamos a nossa branch principal localmente com o git, **não escolhemos o nome para ela**. Sendo assim, o **git coloca o nome padrão master**, por isso tivemos de usar **master ao invés de main como o GitHub sugeriu**. Se você quiser pode em um próximo

repositório utilizar o comando **git branch -m <nome_antigo> <nome_novo>** para renomear a **branch principal**.

Observação2: Se você não estiver logado no **GitHub** na sua **IDE** ou no **git bash**, provavelmente uma janela de login irá aparecer solicitando que você entre na sua conta.

 GitHub Login

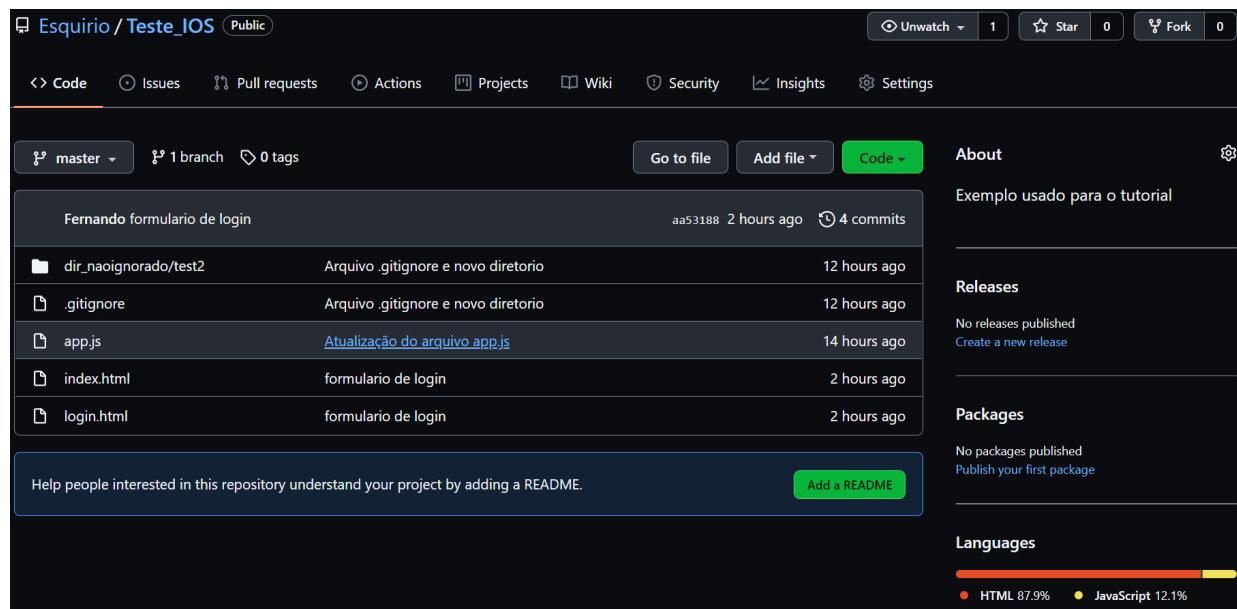
GitHub

Login

 Login  Cancel

Don't have an account? Sign up
Forgot your password?

Se você for no seu repositório remoto na página do GitHub, verá que os arquivos criados já estão armazenados online.



The screenshot shows a GitHub repository named "Esquиро / Teste_IOS". The repository is public and has 1 branch and 0 tags. It contains several files uploaded by "Fernando formulario de login" (commit ID aa53188, 2 hours ago). The files include "dir_naoignorado/test2" (Arquivo .gitignore e novo diretório), ".gitignore" (Arquivo .gitignore e novo diretório), "app.js" (Atualização do arquivo app.js), "index.html" (formulario de login), and "login.html" (formulario de login). The repository also includes sections for About, Releases, Packages, and Languages, with no activity in those areas.

Se você observar uma mensagem na parte de baixo dizendo: **Ajude pessoas interessadas nesse repositório a entender mais sobre o seu projeto adicionando um README**.

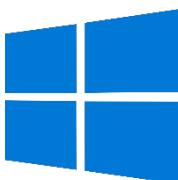
Importante!

A interface do GitHub está em constante modificação, portanto pode ser que você se depare com diferenças entre o que está apresentado nesse material de sua interface. Por isso, é aconselhável utilizar constantemente o seu GitHub para ser estar familiarizado com a interface dele.



Criando arquivo README.md

Vamos então criar o arquivo **README.md** para colocar informações sobre o nosso repositório. Voltaremos ao terminal da IDE e vamos executar o seguinte comando:



No Windows

```
New-Item README.md
```



No Linux, macOS ou Git Bash

```
touch README.md
```

```
..\IOS> New-Item README.md

Directory: C:\Users\Bigode\Documents\IOS

Mode          LastWriteTime     Length Name
----          -----          ---- 
-a---        9/21/2021      12:01          0 README.md

..\IOS> █
```

Então vamos abrir o arquivo no **VS Code** e inserir um **texto utilizando Markdown**:

```
# Meu site
```

Esse é um tutorial para mostrar como trabalhar com o **Git** e o **GitHub**

```
## Para o que serve esse repositório?
```

Esse repositório serve para aprender a usar o **Git** e o **GitHub**



Dica!

Não vamos entrar em detalhes sobre a linguagem Markdown, pois nosso tutorial irá ficar muito grande. Mas existem vários sites que explicam mais sobre o Markdown e você pode conferir a documentação oficial, que está disponível em: <https://guides.github.com/features/mastering-markdown/>

Então vamos voltar ao terminal e adicionar o nosso arquivo na **staging area** e em seguida fazer o **commit**. Isso fará com que nosso arquivo **README.md** esteja no repositório local.

```
git add .
git commit -m 'Adicionando o arquivo README.md'
```

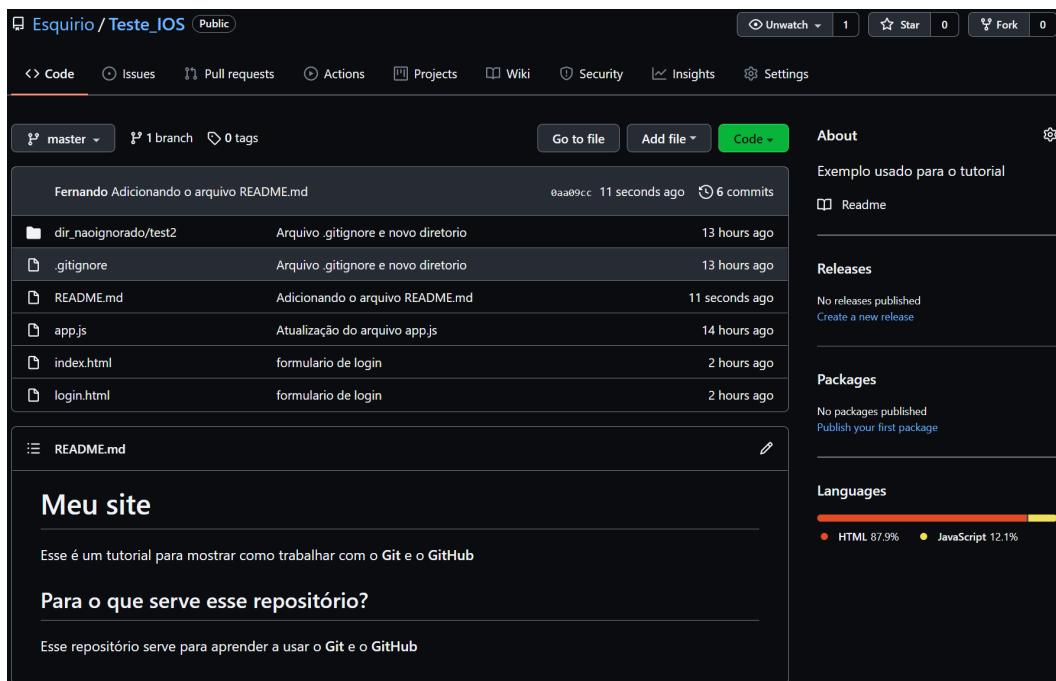
```
..\IOS> git add .
..\IOS> git commit -m 'Adicionando o arquivo README.md'
[master 9aa6fed] Adicionando o arquivo README.md
 1 file changed, 5 insertions(+)
 create mode 100644 README.md
..\IOS> █
```

Agora precisamos enviar o arquivo **READ.md** para o **repositório remoto**. Para isso execute o comando:

```
git push
```

```
..\IOS> git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 408 bytes | 408.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Esquirio/Teste_IOS.git
  aa53188..9aa6fed master -> master
..\IOS> █
```

Agora nosso **arquivo já está online** e se atualizarmos a **página do GitHub**, veremos o **texto que colocamos com a formatação definida pela linguagem Markdown** sendo exibido logo abaixo dos arquivos.



The screenshot shows a GitHub repository page for 'Esquirio / Teste_IOS'. The repository has 1 branch and 0 tags. The 'Code' tab is selected, showing a list of files: .gitignore, README.md, app.js, index.html, and login.html. The README.md file content is displayed below the code listing:

```
Fernando Adicionando o arquivo README.md
dir_naoignorado/test2 Arquivo .gitignore e novo diretório
.gitignore Arquivo .gitignore e novo diretório
README.md Adicionando o arquivo README.md
app.js Atualização do arquivo app.js
index.html formulário de login
login.html formulário de login
```

About

Exemplo usado para o tutorial

Readme

Releases

No releases published Create a new release

Packages

No packages published Publish your first package

Languages

HTML 87.9% JavaScript 12.1%

Meu site

Esse é um tutorial para mostrar como trabalhar com o Git e o GitHub

Para o que serve esse repositório?

Esse repositório serve para aprender a usar o Git e o GitHub

Clonando um diretório já existente

Se alguém tiver interessado no seu **repositório público**, e quiser contribuir, será necessário **clonar** o mesmo, ou simplesmente **fazer o download dos arquivos do repositório**.

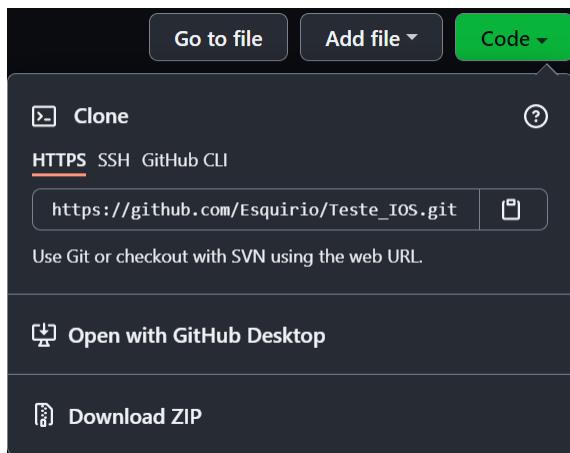
Essa parte do tutorial, você pode fazer sozinho no seu computador ou a distância com um colega de classe. Vamos ver como fazer isso, siga os seguintes passos:

Clique no **botão verde em seu repositório**, no canto superior direito, com o rótulo **Code**



Irá abrir uma série de opções para clonar o diretório:

- A primeira opção é **Clone**, que você criar um clone do repositório, então qualquer modificação que for feita poderá ser enviada para o repositório online. Você pode clonar (**Clone**) pelo endereço **HTTPS** do repositório, utilizar **SSH** ou **GitHubCLI**,
- Você pode também abrir com o GitHub Desktop (**Open with GitHub Desktop**). Nesse caso, a interface do GitHub Desktop irá perguntar se você criar um clone ou apenas baixar os arquivos.
- Você apenas fazer download do arquivo compactado (**Download ZIP**). Nesse caso, o repositório baixado não tem nenhum vínculo com o repositório original.



Vamos clonar o diretório. Para isso você deve copiar o endereço **HTTPS** e usá-lo para **clonar em outro diretório que não seja o mesmo que estamos trabalhando**.



Importante!

O endereço do seu repositório será diferente do mostrado no exemplo.

Pelo terminal do **VS Code no Windows**, você pode executar os comandos a seguir para sair do diretório de trabalho atual, criar um novo diretório, entrar nesse diretório e então verificar que ele está vazio. Execute um comando por vez.

Comando	Descrição
<code>cd ..</code>	Volta um nível de diretório
<code>mkdir Novo_IOS</code>	Cria um novo diretório, apenas para SO Windows
<code>cd \Novo_IOS\</code>	Entra nesse novo diretório
<code>ls</code>	Lista os arquivos desse diretório

```
..\IOS> cd ..
..\Documents> mkdir Novo_IOS

Directory: C:\Users\Bigode\Documents

Mode          LastWriteTime      Length Name
----          -----          ---- 
d-----        9/21/2021       12:31   Novo_IOS

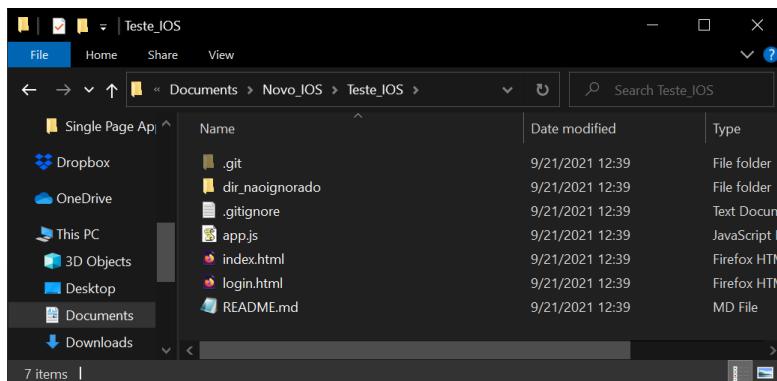
..\Documents> cd .\Novo_IOS\
..\Novo_IOS> ls
..\Novo_IOS>
```

Agora estamos no novo diretório (**Novo_IOS**), podemos **clonar o repositório já existente**. Para isso, você tem que ter **copiado o endereço HTTPS do seu repositório** e executar o comando:

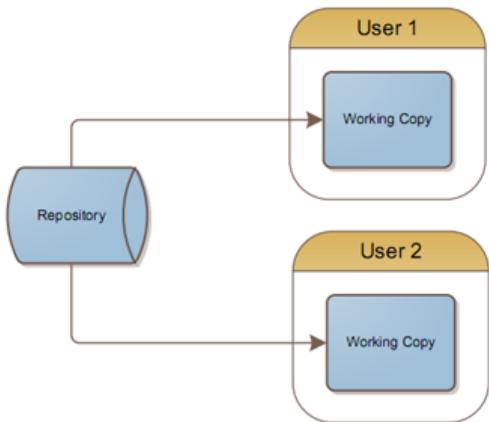
```
git clone <Coloque aqui o endereço do repositório>
```

```
..\Novo_IOS> git clone https://github.com/Esquirio/Teste_IOS.git
Cloning into 'Teste_IOS'...
remote: Enumerating objects: 22, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 22 (delta 5), reused 21 (delta 4), pack-reused 0
Receiving objects: 100% (22/22), done.
Resolving deltas: 100% (5/5), done.
..\Novo_IOS>
```

No seu novo diretório, é possível ver os arquivos do repositório clonado.



Observe que, **dentro do diretório foi criada automaticamente uma pasta com o nome do diretório clonado** e esse será nosso novo repositório local. **Agora temos dois repositórios locais**.

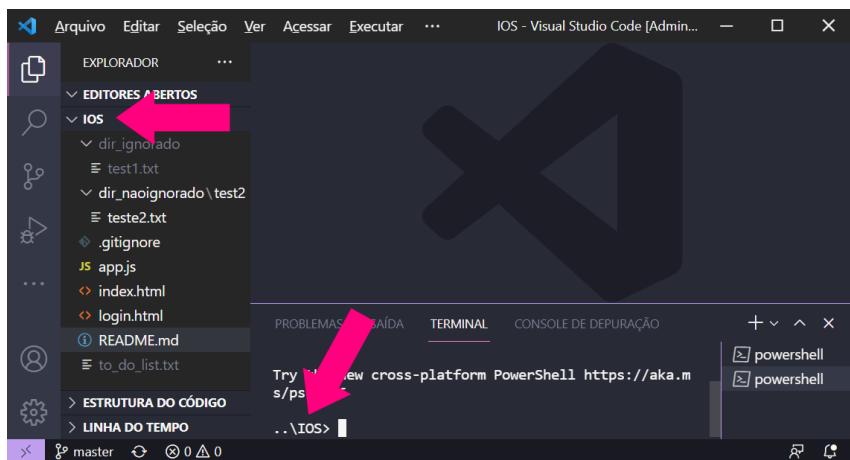


Qualquer mudança realizada aqui agora poderá ser enviada para o repositório original. Esse é uma boa forma de fazer trabalhos colaborativos.

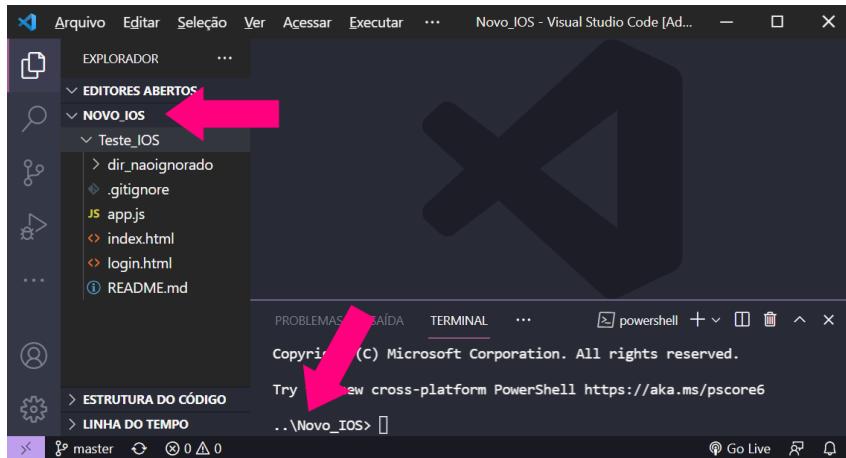
Comando pull

Essa parte do tutorial, você pode fazer sozinho no seu computador, se já estava fazendo assim antes, ou a distância com um colega de classe. Vamos fazer um teste do comando **pull**. Para isso, siga os passos:

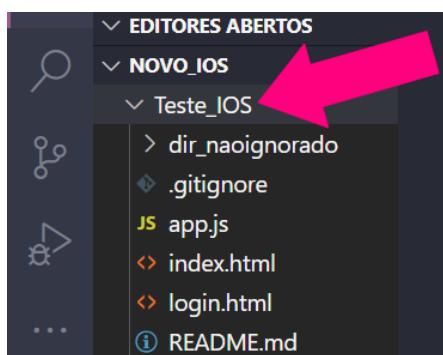
Na IDE do VS Code com a pasta **IOS** aberta, feche o terminal e abra novamente. Isso irá garantir que o terminal estará aberto na pasta correta. Vamos chamar essa instância do programa VS Code de **IDE_IOS**.



Agora vamos abrir outra instância do **VS Code**, basta clicar no ícone do **VS Code novamente**. No menu Arquivo, selecione a opção **Abrir Pasta** e selecione o novo diretório criado, no exemplo é **Novo_IOS**. Abra o terminal de comando. Iremos chamar essa instância do VS Code de **IDE_NOVO_IOS**.



Observe que na pasta **NOVO_IOS** tem um diretório **Teste_IOS**, que é nosso repositório local clonado do repositório remoto.



Agora temos **duas instâncias de IDE abertas no seu computador ou uma com você e outra com seu colega de classe.**



Na **IDE_NOVO_IOS**, que temos o **clone do repositório**, vamos abrir o arquivo **index.html** e fazer uma pequena alteração e, depois, salvar o arquivo.

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
    <title>Meu site</title>
</head>

<body>
    Esse é meu site.
    Coloquei uma nova linha.
    Formulário de login
    Alteração feita na IDE_NOVO_IOS
</body>

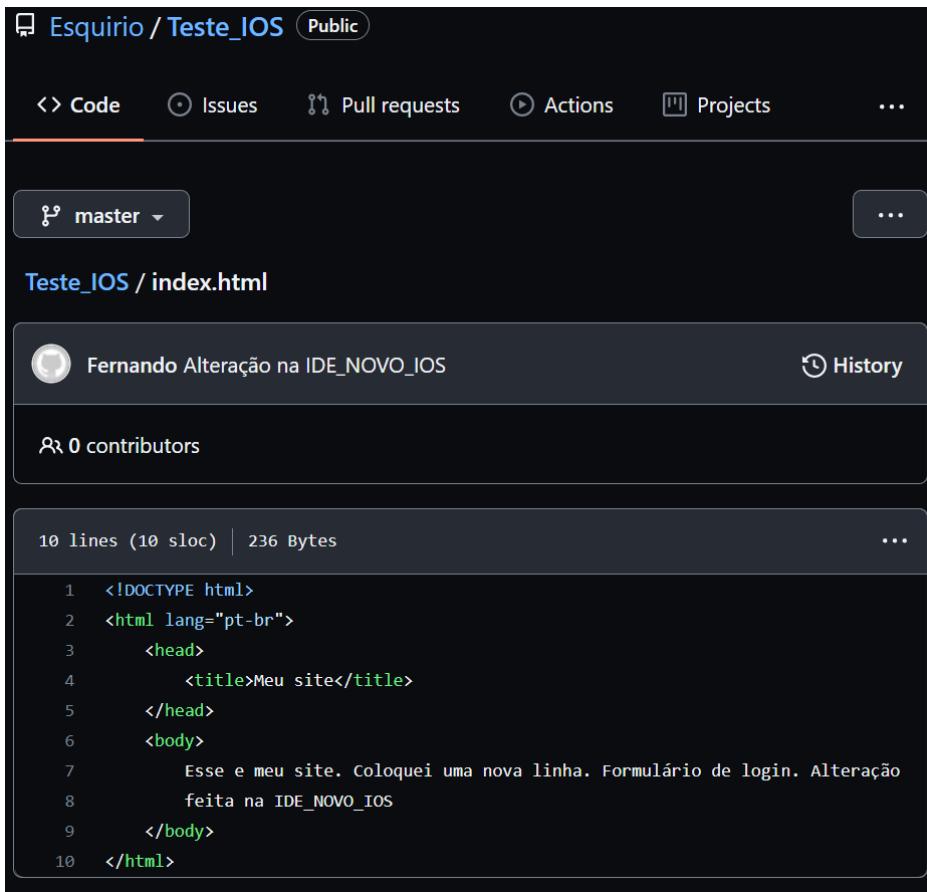
</html>
```

Ainda na **IDE_NOVO_IOS**, vamos adicionar o **arquivo modificado na staging area, fazer o commit para o repositório local Novo_IOS** e depois **enviar para o repositório remoto**. Execute um comando por vez.

Comando	Descrição
cd Teste_IOS	Entra no diretório do repositório local (se você já estiver dentro desse diretório no terminal não precisa escutar esse comando)
git add .	Adiciona os arquivos com modificações na <i>staging area</i> .
git commit -m 'Alteração na IDE_NOVO_IOS'	Faz o commit com as alterações para o repositório local.
git push	Envia para o repositório remoto

```
..\Novo_IOS> cd Teste_IOS
..\Teste_IOS> git add .
..\Teste_IOS> git commit -m 'Alteração na IDE_NOVO_IOS'
[master fa2d4f4] Alteração na IDE_NOVO_IOS
 1 file changed, 2 insertions(+), 1 deletion(-)
..\Teste_IOS> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 368 bytes | 368.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects
.
To https://github.com/Esquирio/Teste_IOS.git
 0aa09cc..fa2d4f4  master -> master
..\Teste_IOS> █
```

Se você olhar no **repositório remoto** do seu **GitHub**, verá que o arquivo **index.html** está **alterado e atualizado**.



Teste_IOS / index.html

Fernando Alteração na IDE_NOVO_IOS

0 contributors

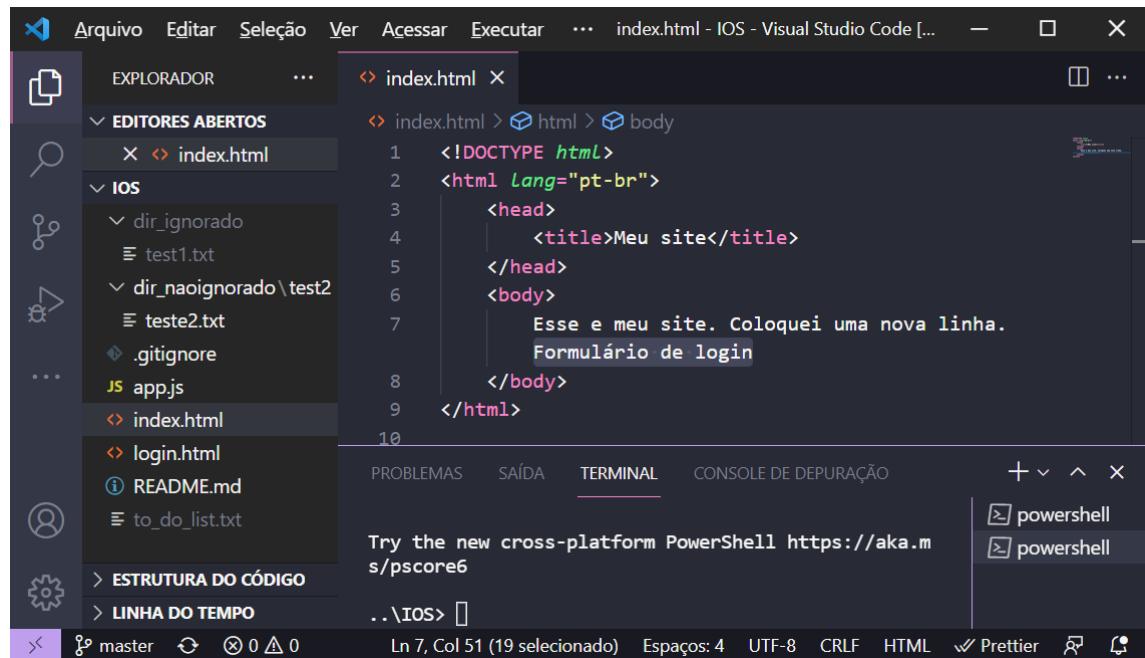
10 lines (10 sloc) | 236 Bytes

```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3      <head>
4          <title>Meu site</title>
5      </head>
6      <body>
7          Esse é meu site. Coloquei uma nova linha. Formulário de login. Alteração
8          feita na IDE_NOVO_IOS
9      </body>
10     </html>

```

Mas se você abrir o arquivo **index.html** na **IDE_IOS**, ele não vai estar atualizado.



Arquivo Editar Seleção Ver Acessar Executar ... index.html - IOS - Visual Studio Code [...]

EXPLORADOR ... < index.html X

EDITORES ABERTOS < index.html X

IOS < index.html > html > body

```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3      <head>
4          <title>Meu site</title>
5      </head>
6      <body>
7          Esse é meu site. Coloquei uma nova linha.
8          Formulário de login
9      </body>
10     </html>

```

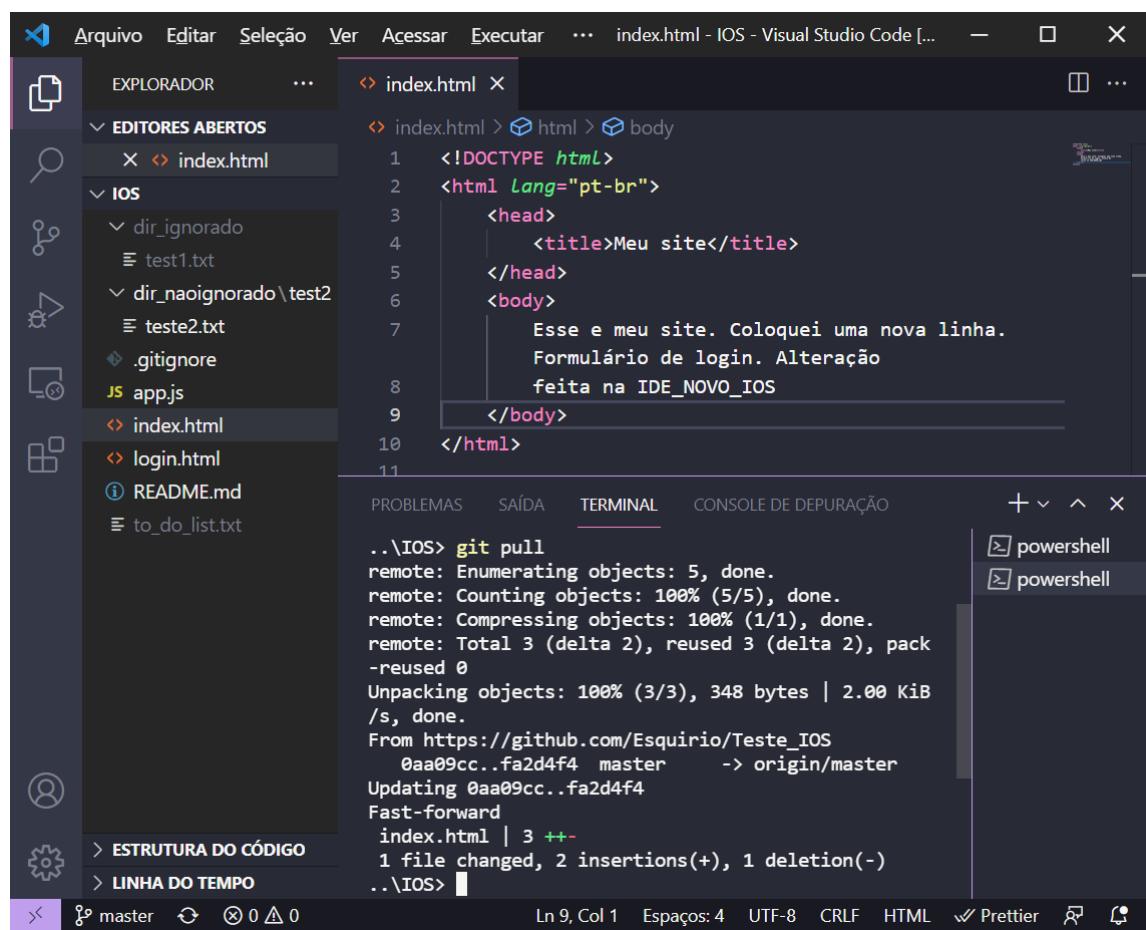
PROBLEMAS SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO

Try the new cross-platform PowerShell <https://aka.ms/powershell>

... \IOS < master > 0 △ 0

Sendo assim, antes da **pessoa programadora que está com o repositório desatualizado começar a desenvolver**, ele deve dar um **git pull**, para **buscar a versão mais recente dos arquivos** e, então, começar a trabalhar.

```
git pull
```



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files like index.html, login.html, app.js, README.md, and to_do_list.txt.
- Editor (Top Center):** Displays the content of index.html, which includes HTML code and a note: "Esse é meu site. Coloquei uma nova linha. Formulário de login. Alteração feita na IDE_NOVO_IOS".
- Terminal (Bottom):** Shows the output of the git pull command:

```
..\IOS> git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), 348 bytes | 2.00 KiB /s, done.
From https://github.com/Esquirio/Teste_IOS
  0aa09cc..fa2d4f4 master      -> origin/master
Updating 0aa09cc..fa2d4f4
Fast-forward
  index.html | 3 ++
    1 file changed, 2 insertions(+), 1 deletion(-)
..\IOS>
```
- Status Bar (Bottom):** Shows the current branch as master, commit count (0), and other status indicators.