Produtor: Gustavo Nascimento

Conteúdo:

# **Objetivo**

Caro leitor, desejo oferecer a você não apenas um aprendizado de como utilizar uma das ferramentas de maior destaque no mercado de trabalho.

Desejo que você possa analisar e tomar suas próprias decisões para utilização da ferramenta, mostrarei a você que qualquer ferramenta que esteja disponível para uso, ela pode agregar total valor para sua utilização seja de forma privada ou pública. Até mesmo de forma distinta com um objetivo diferente de projeto.

Git e GitHub são conceitos e ferramentas utilizadas para versionamento de arquivos, mas oque ela teve como maior enfoque e valor, é de que essa ferramenta consegue trazer a todos desenvolvedores uma maneira de centralizar seus projetos, sendo assim, podendo efetuar versionamento de projetos contendo versões. Conseguindo mostrar ser uma ferramenta flexível para qualquer necessidade. Não apenas como uma ferramenta de versionamento GitHub, consegue trazer consigo diversas funções como montar um Kanban para tarefas que devam ser desenvolvidas, seja de projeto ou até mesmo um servidor de jogos.

Para melhor fixação, está ferramenta pode gerar diversos valores pense o seguinte, você tem um projeto que está construindo com seus colegas de faculdade ou até mesmo de trabalho, vocês precisam construir um sistema de cabeleireiro simples para ser entregue, então desta forma, vocês dividem tarefas e funções que precisam ser desenvolvidos, com base nisso cada pessoa pode ter um ambiente de trabalho em sua máquina, quando o desenvolvimento for concluído de cada tarefa levantada, o desenvolvedor poderá publicar seu desenvolvimento em um local aonde todos possam ter acesso e receber essas alterações assim consistindo em um projeto centralizado e atualizado.  
Mas podemos também utilizar a ferramenta para subir arquivos comuns, como Word, PDF, ou até mesmo um servidor de games, exatamente isso, imagine que você tem um servidor na sua máquina de algum jogo, mas nem sempre seu servidor pode ficar online, sendo assim, você pode subir um mapa deste servidor no GitHub, para que outra pessoa possa abrir um servidor em outra máquina, assim mantendo o mapa do jogo com os mesmos progresso.

Sumário

[**Objetivo** 2](#_Toc49174115)

[**GitHub & Git** 4](#_Toc49174116)

[Git: 4](#_Toc49174117)

[GitHub: 5](#_Toc49174118)

[Git & GitHub: 6](#_Toc49174119)

[**Criação de conta e configuração** 7](#_Toc49174120)

[**GitHub e suas funções** 12](#_Toc49174121)

[**Capítulo 1** 14](#_Toc49174122)

[Primeiro passo 14](#_Toc49174123)

[Segundo passo 17](#_Toc49174124)

[**Capítulo 2** 18](#_Toc49174125)

[Code 18](#_Toc49174126)

[Issues 19](#_Toc49174127)

[Pull requests 20](#_Toc49174128)

[Actions 21](#_Toc49174129)

[Projects 22](#_Toc49174130)

[Wiki 23](#_Toc49174131)

[Security 24](#_Toc49174132)

[Insights 25](#_Toc49174133)

[Settings 25](#_Toc49174134)

[**Capítulo 3** 27](#_Toc49174135)

[Code 27](#_Toc49174136)

[Iniciando seu repositório 28](#_Toc49174137)

[Git Bash 29](#_Toc49174138)

[Observações do capítulo: 35](#_Toc49174139)

[**Capítulo 4** 36](#_Toc49174140)

[Criando Branch 36](#_Toc49174141)

[Atualizando Branch 38](#_Toc49174142)

[**Capítulo 5** 42](#_Toc49174143)

# **GitHub & Git**

Para iniciarmos nosso treinamento na ferramenta Git Hub primeiramente precisamos saber brevemente a diferença entre Git e GitHub, assim conhecendo sua história e principalmente porque está é uma das ferramentas com um valor tão grande no mercado de trabalho e fora do mercado de trabalho também.

## Git:

Como muitas coisas boas na vida, Git começou com um pouco de destruição criativa e controvérsia ardente.

O kernel Linux é um projeto de software de código aberto de escopo bastante amplo. Durante a maior parte da vida útil da manutenção do kernel do Linux (1991–2002), as alterações no software foram passadas como patches e arquivos arquivados. Em 2002, o projeto do kernel Linux começou a usar um DVCS proprietário chamado BitKeeper.

Em 2005, o relacionamento entre a comunidade que desenvolveu o kernel Linux e a empresa comercial que desenvolveu o BitKeeper foi interrompido, e o status de gratuito da ferramenta foi revogado. Isso levou a comunidade de desenvolvimento do Linux (e em particular Linus Torvalds, o criador do Linux) a desenvolver sua própria ferramenta com base em algumas das lições que aprenderam ao usar o BitKeeper. Alguns dos objetivos do novo sistema eram os seguintes:

* Rapidez
* Design simples
* Suporte forte para desenvolvimento não linear (milhares de ramificações paralelas)
* Totalmente distribuído
* Capaz de lidar com grandes projetos como o kernel Linux de forma eficiente (velocidade e tamanho dos dados)

Desde seu nascimento em 2005, o Git evoluiu e amadureceu para ser fácil de usar e ainda assim manter essas qualidades iniciais. É incrivelmente rápido, muito eficiente com grandes projetos e possui um sistema incrível de ramificação para desenvolvimento não linear (consulte [Ramificação Git](https://git-scm.com/book/en/v2/ch00/ch03-git-branching) ).

## GitHub:

O GitHub foi lançado no ano de 2008 pelos desenvolvedores da Logical Awesome e é um sistema [Web Hosting](https://pt.wikipedia.org/wiki/Hospedagem_de_sites) compartilhado.  
O GitHub nasceu com o principal objetivo de abrigar projetos que são versionados via Git.   
O GitHub foi escrito em Ruby on Rails, o que é bem interessante pelo fato de na época a linguagem/framework possuir pouco tempo de vida e estabilidade. A medida que a popularidade foi aumentando, percebeu-se que seria necessário a implementação de funcionalidades que deixassem o GitHub com cara de rede social. Foram adicionadas listas de discussões, gráficos referentes a contribuições em projetos e até mesmo a opção de seguir algum usuário, algo muito parecido com o que o Twitter emprega. Atualmente o GitHub é tão reconhecido na comunidade que projetos de grande importância estão desfrutam dos seus serviços, como por exemplo o Linux e o Ruby on Rails.

## Git & GitHub:

A partir deste momento iremos dar início ao curso de Git, nosso objeto é de ensiná-lo a como usar o Git e GitHub da melhor forma possível agregando valor a ferramenta. Pense sempre que uma ferramenta bem desenvolvida consiste em duas palavras em questão.

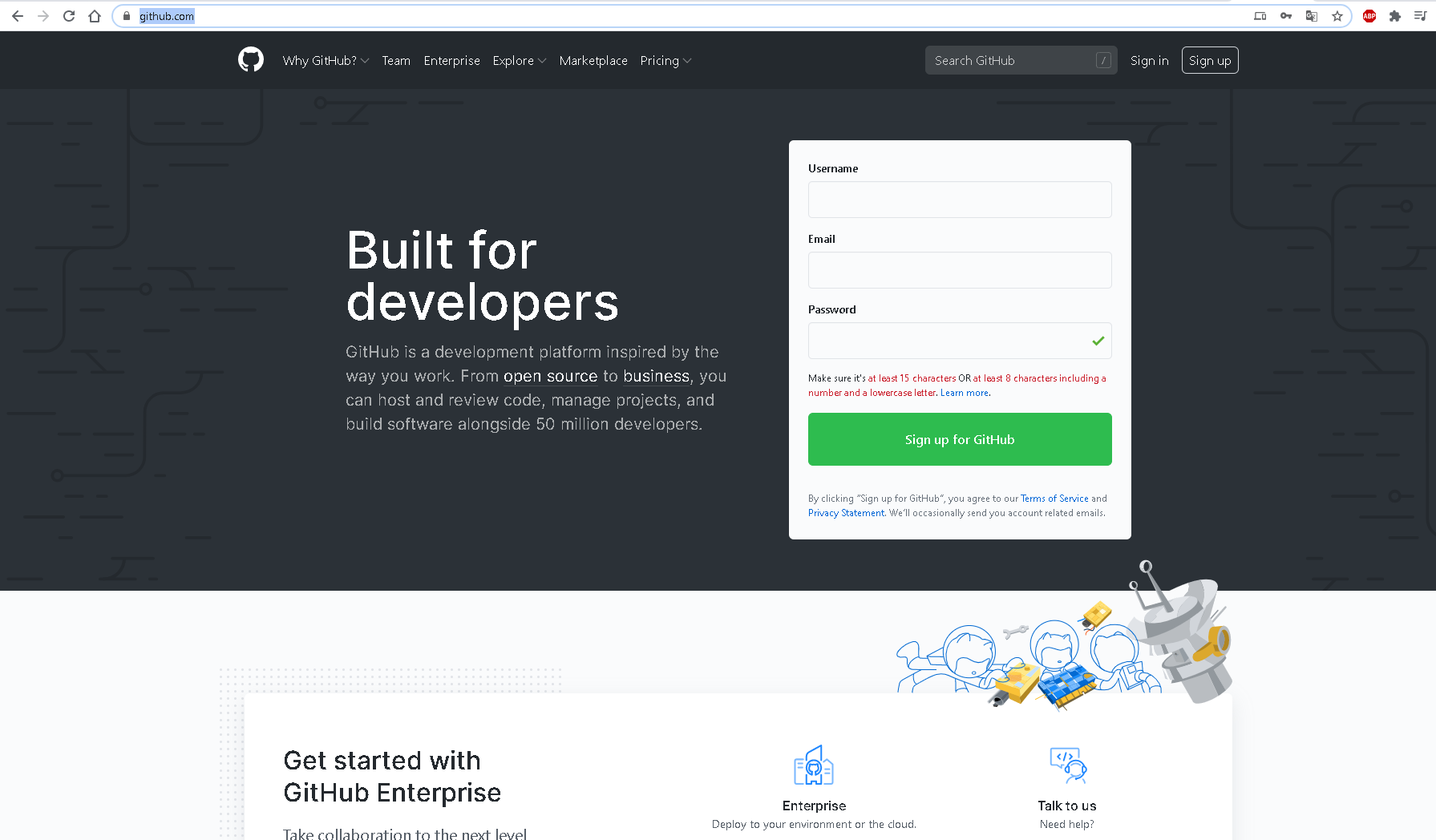
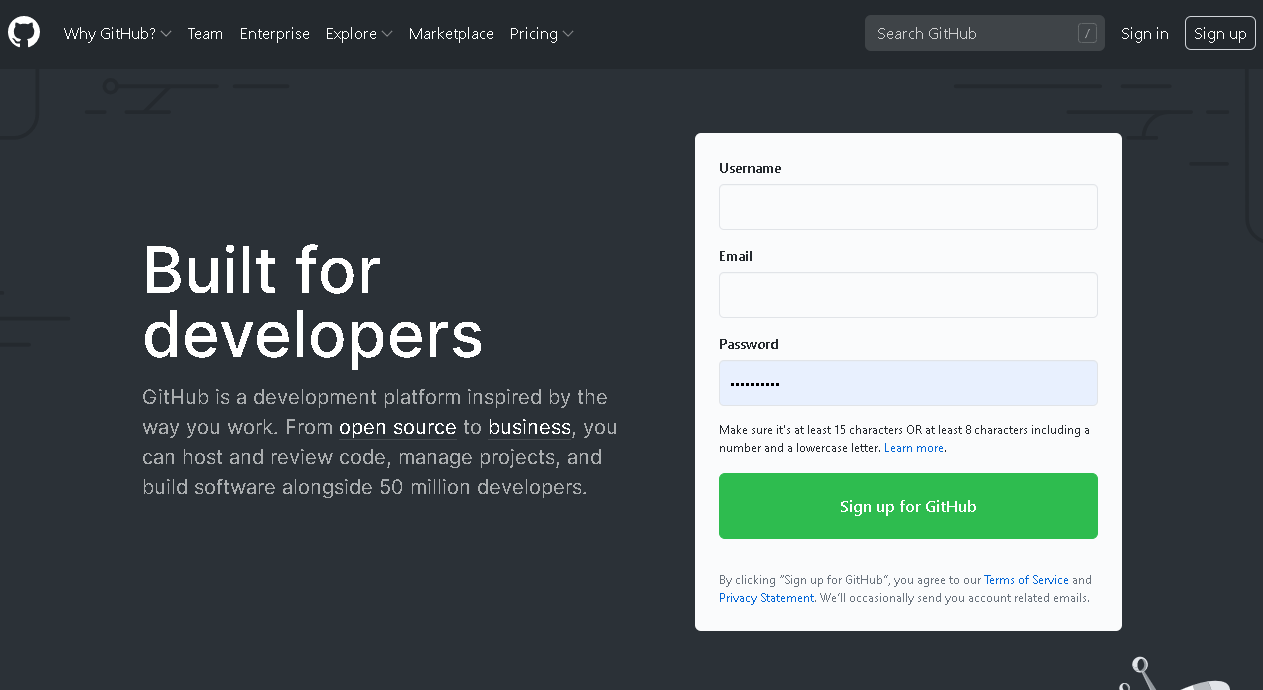
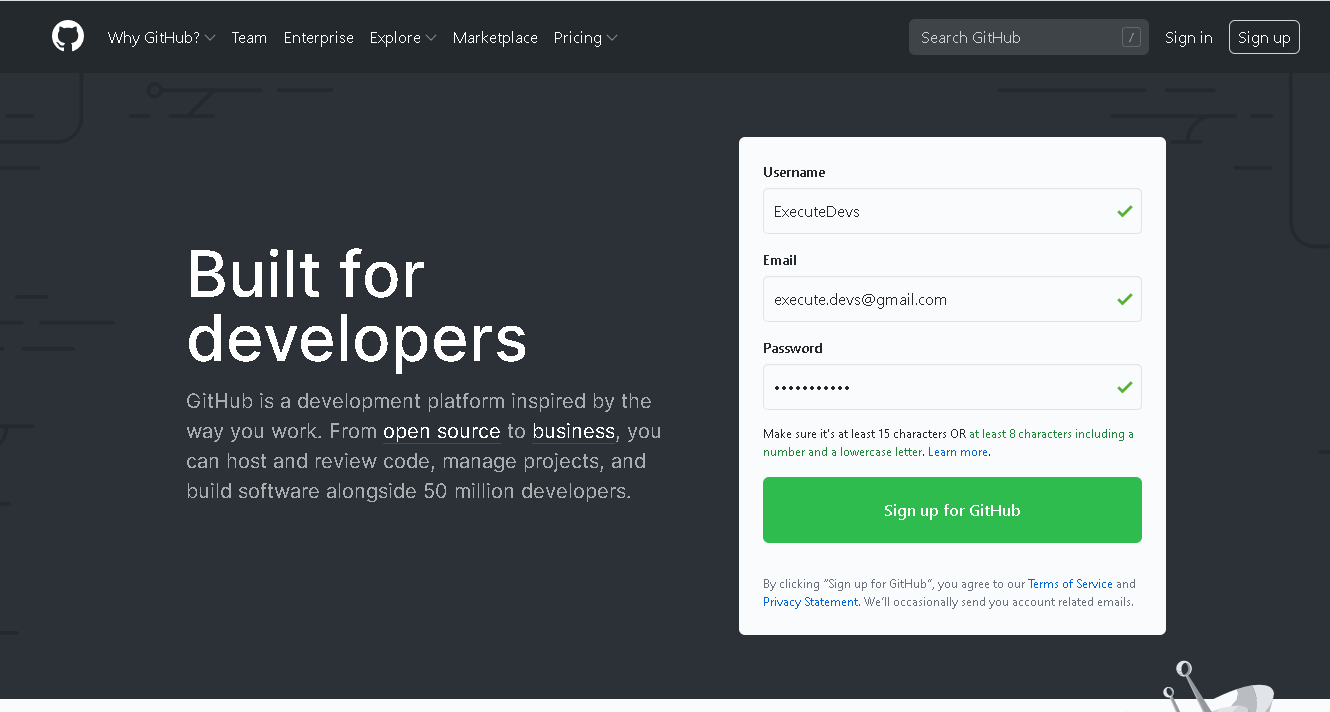
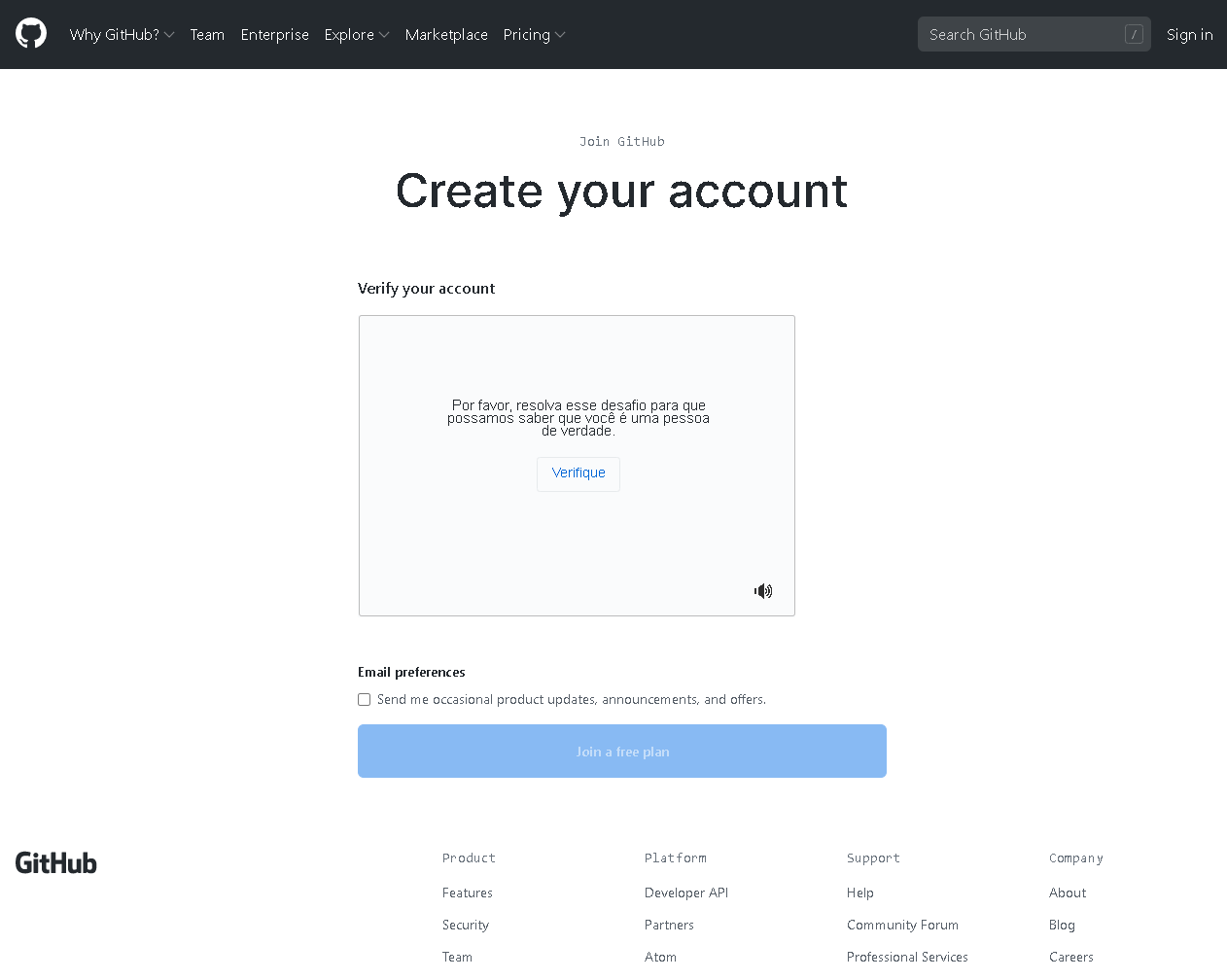
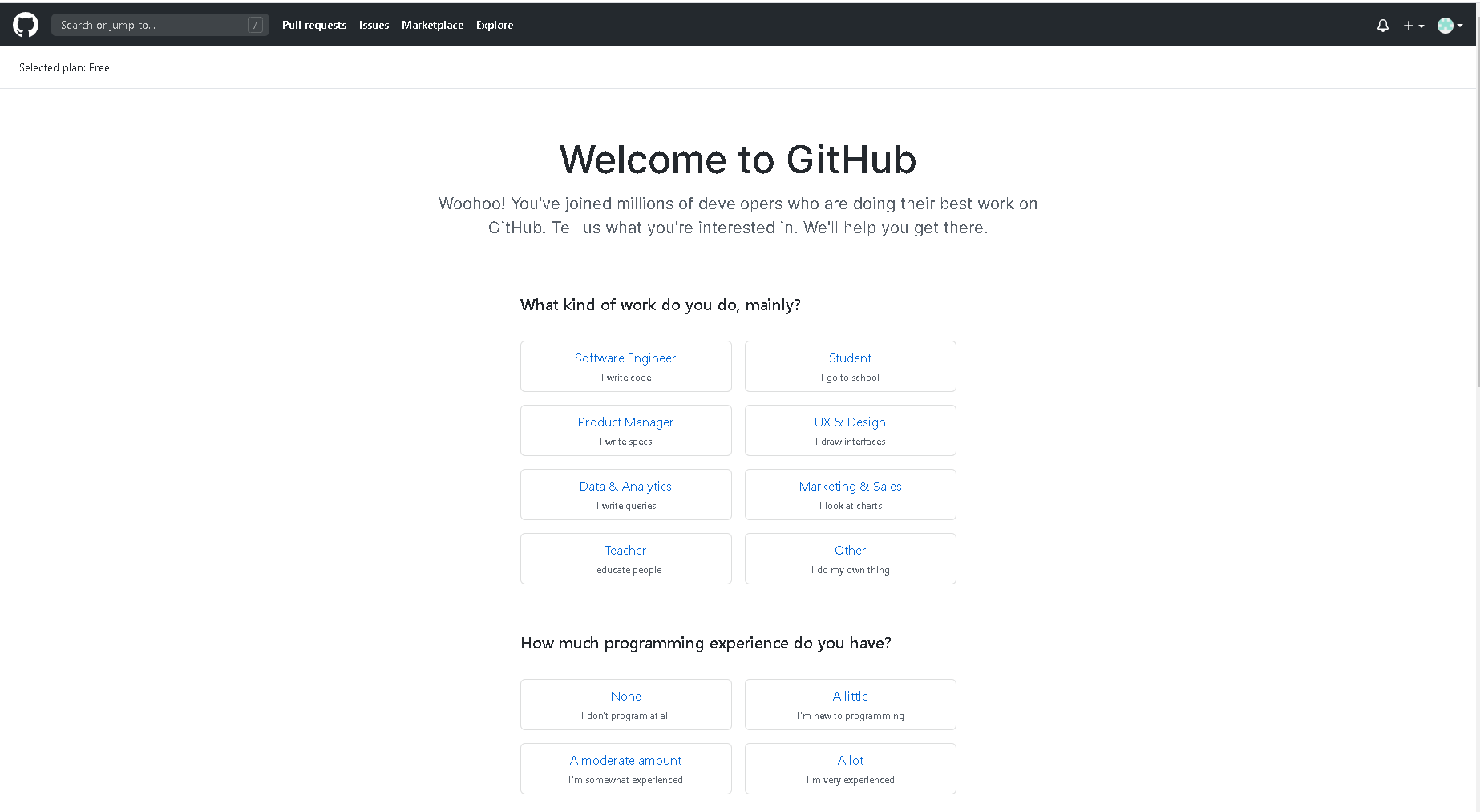
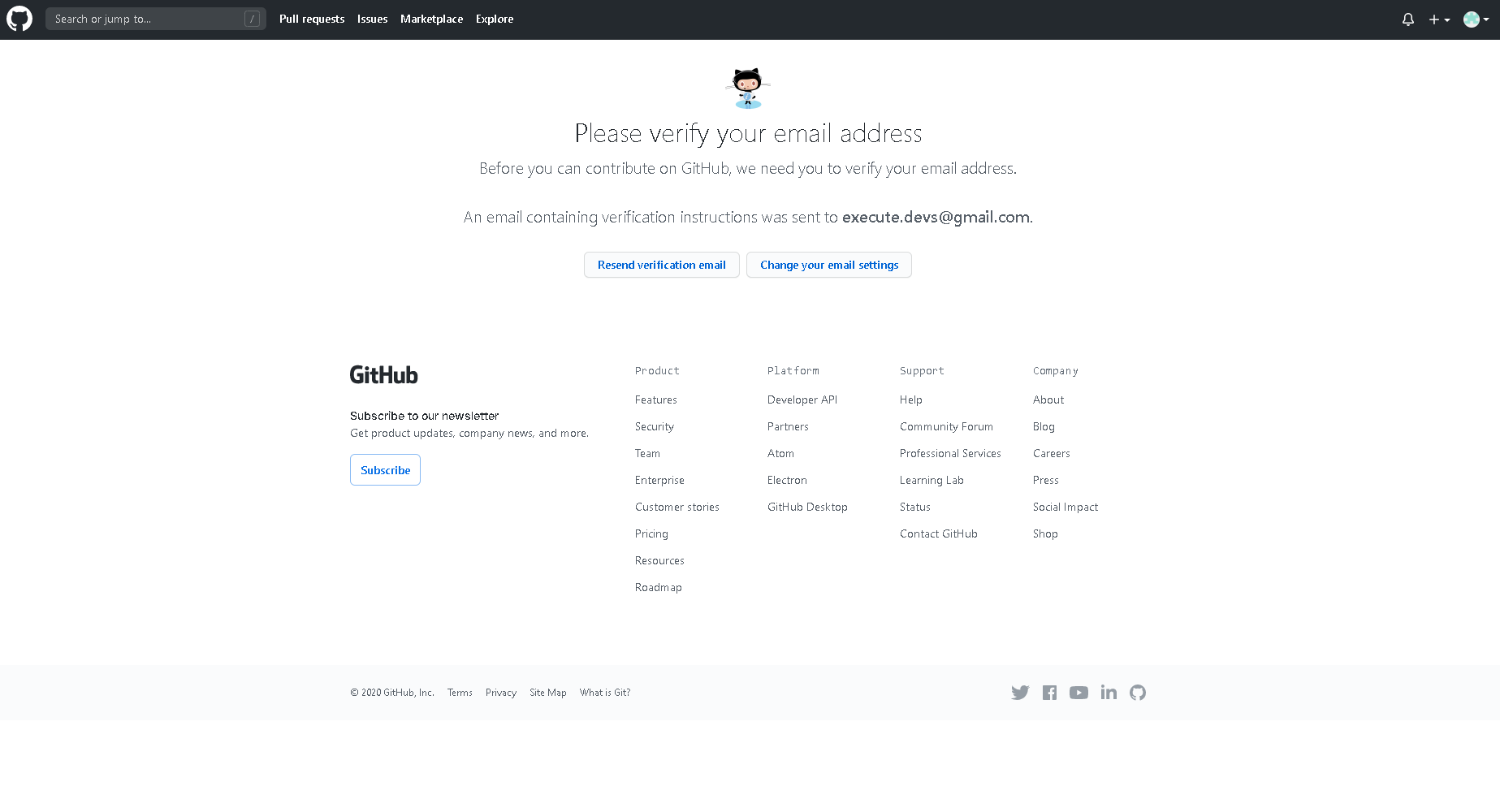
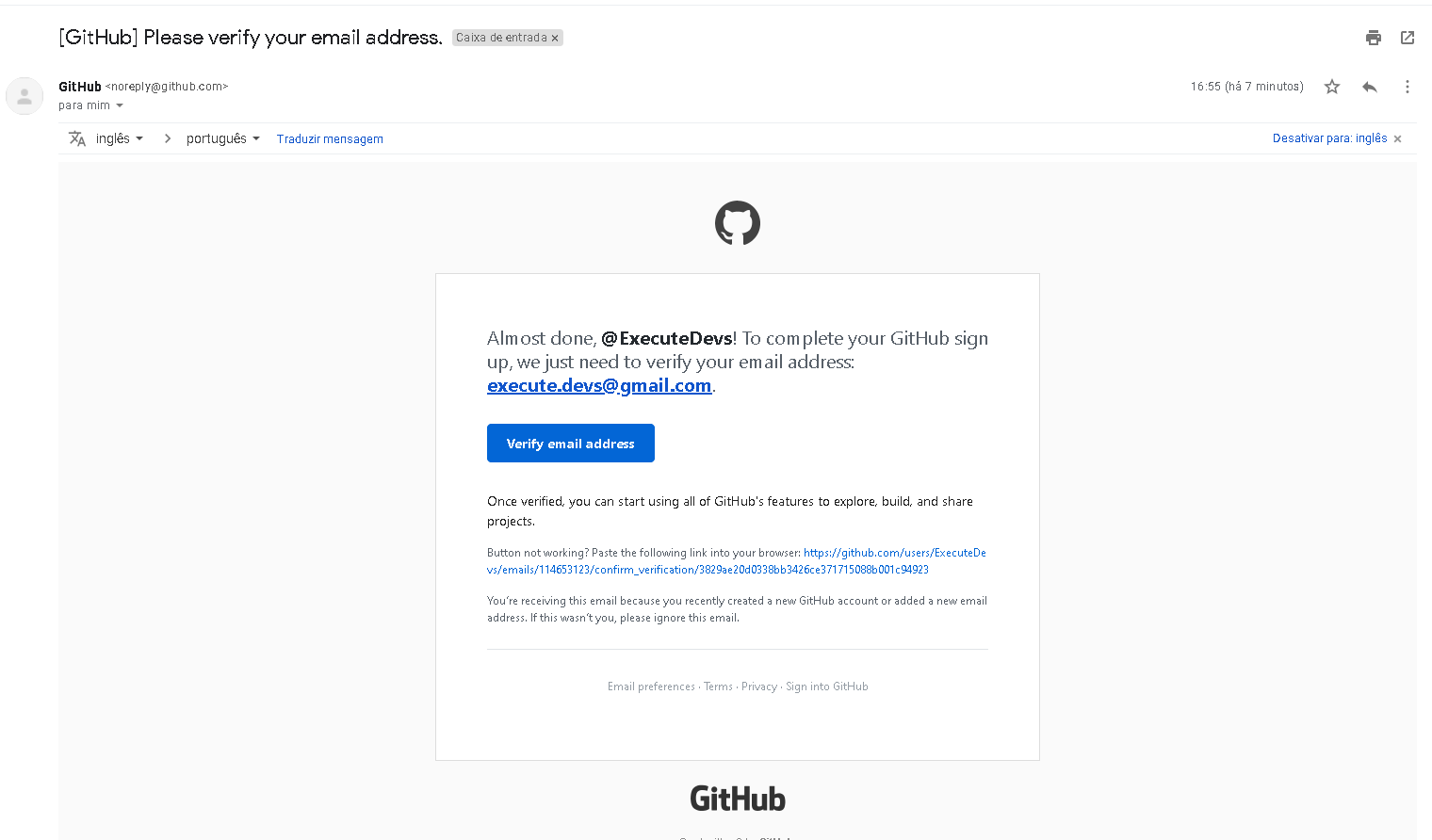
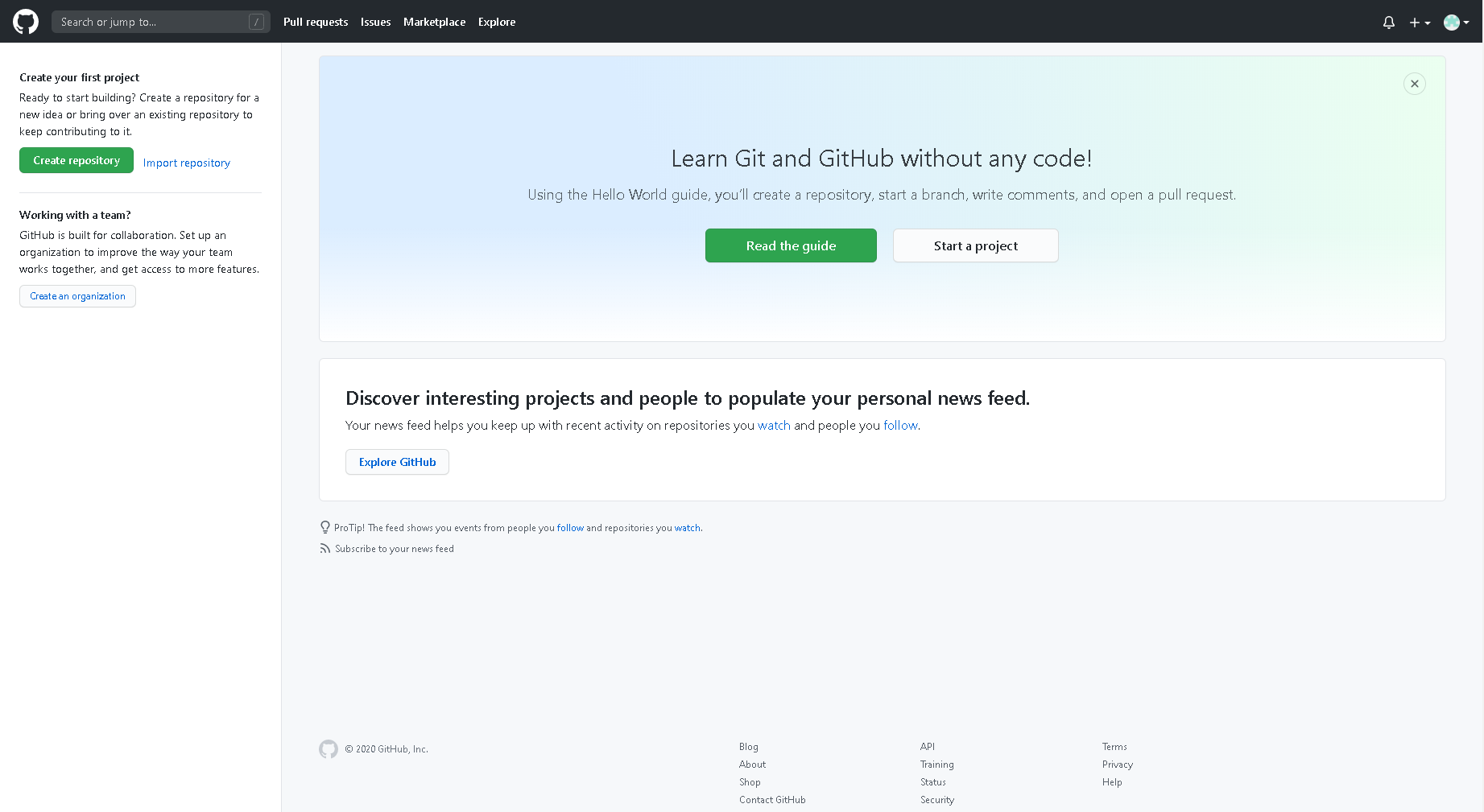
* Custo
* Valor

Quando nos referimos a custo, toda ferramenta tem o seu dinheiro que precisa ser injetado/aplicado para ser desenvolvido seja um projeto. Ou Até mesmo para usufruir de uma ferramenta existente.  
Mas quando falamos de valor estamos referindo-se oque aquela ferramenta irá agregar no seu dia a dia, em um trabalho, projeto ou até mesmo para usá-lo de forma casual (repositório para guarda Documentação/Servidor de game).

Gostaria de deixar claro que podemos usar ferramentas para nosso dia a dia e de forma inimagináveis então ao concluir este curso espero que consiga aproveitar essa ferramenta e usá-la da melhor forma possível.... Conte-me depois como foi sua experiência.

# **Criação de conta e configuração**

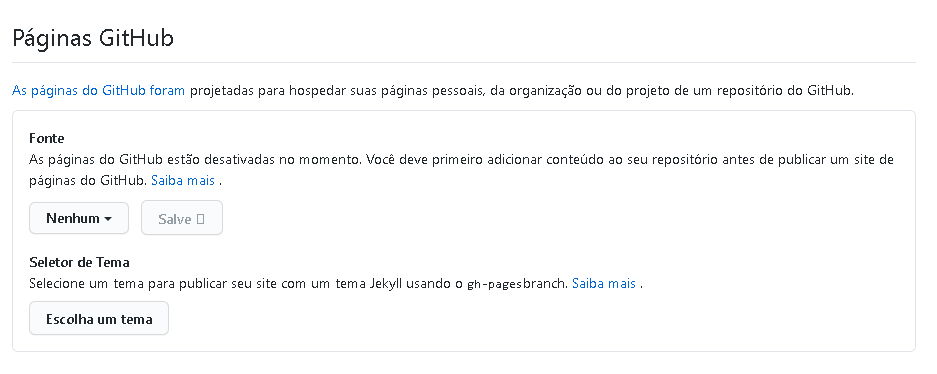
Para poder utilizar a ferramenta primeiramente vamos precisar criar uma conta, siga os passos a passos e construa sua conta com configuração para sua necessidade.

1. Acesse o site: <https://github.com/>  
   
2. Ao entrar na página irá se deparar com um site em inglês nela poderá clicar no botão Sign Up (Localizado ao lado direito superior na tela) ou bem no início da página poderá criar sua conta.  
   
3. Preencha as informações necessárias para o cadastro do login.  
    Username: desta forma as pessoas podem enviar convites para você pelo seu nome cadastrado  
    E-mail: Recomenda-se um e-mail profissional até mesmo para poder compartilhar seu projeto via e-mail para empresas que necessitarem ver seu portifólio.  
    Password: Crie uma senha para seu cadastro sempre que for configurar ou até mesmo acessar o GitHub estará protegido.
4. Confirmação de cadastro:  
   
5. Após clicar em Sign up for GitHub ele tentará validar se você não é um robô com algum método de validação:  
   
6. Últimos passos o GitHub irá questioná-lo de como você irá utilizar a ferramenta e para qual propósito, responda as questões e escreva o que melhor lhe a atende.  
   
7. Por último o GitHub irá precisar que você confirme em seu e-mail a conta para utilizar da melhor forma, cheque o e-mail cadastrado.  
     
   
8. Agora você começara a fazer sua primeira configuração para utilização do GitHub em paralelo vamos apresentar suas telas.  
   

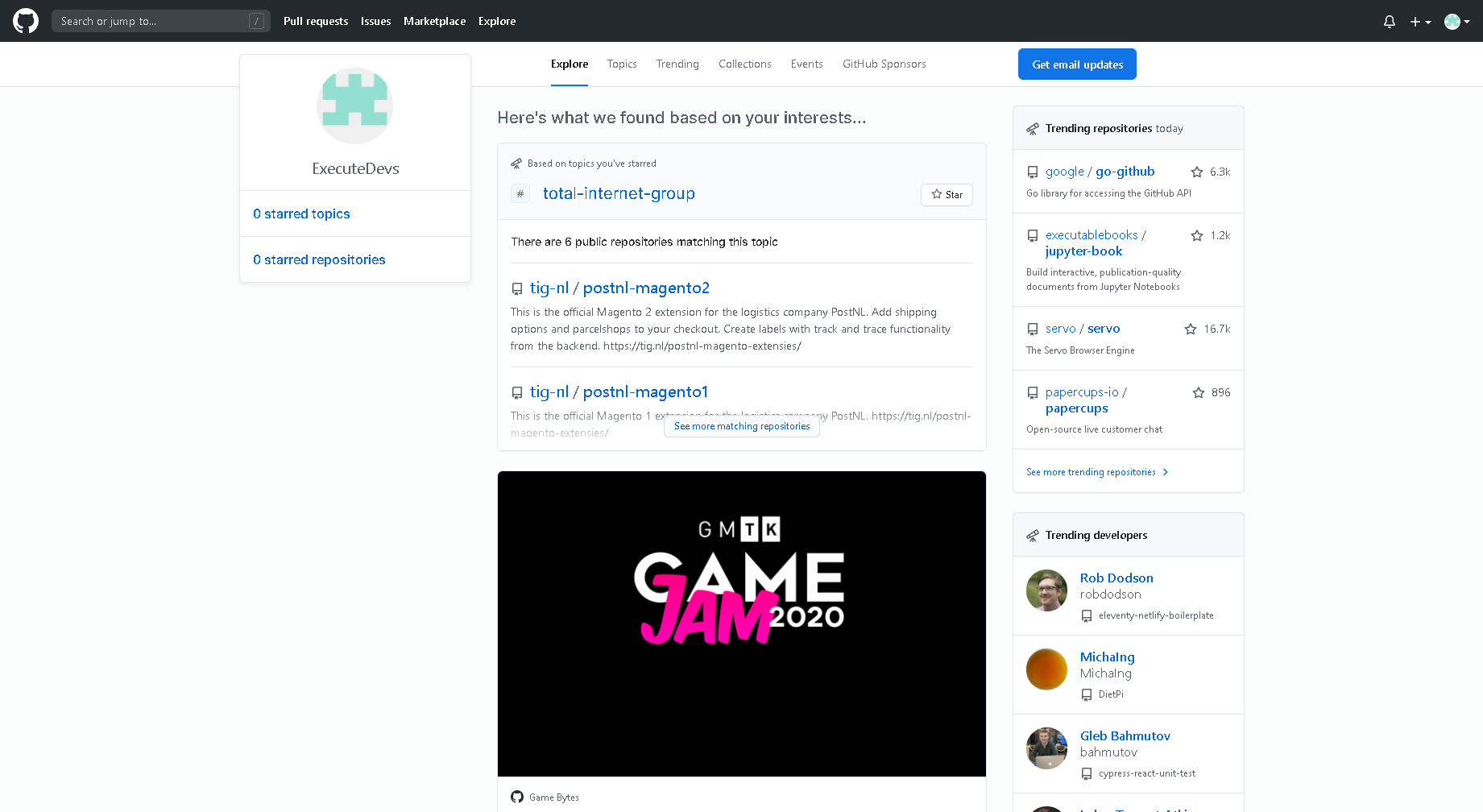
# **GitHub e suas funções**

Quando utilizamos o GitHub na maioria dos casos as pessoas pensam em apenas fazer controle de versionamento de arquivo... Ou seja estamos falando de deixar projetos disponíveis e atualizados para serem manipulados em outras máquinas ou servidores.

Mas quando falamos em versionamento de arquivos não podemos apenas nos apegar que está ferramenta tem apenas este propósito, podemos hospedar páginas HTML estáticas, contendo CSS, também JS, pode-se dizer que podemos fazer um portifólio para que você possa usar de forma gratuita e disponível para outras pessoas acessarem e descobrirem o seu grau de conhecimento a respeito de Web.



Temos também uma rede social muito poderosa podendo ser utilizada por todos, para compartilhar pensamentos, discussões, pedir ajuda ou até mesmo publicar um projeto publico para que outras pessoas possam ajudar a construção do seu site/aplicativo.

 GitHub tem um conceito muito simples de arquitetura, imagine uma árvore contento diversos galhos e nestes galhos cada um tem sua quantidade de folhas, todos galhos são ligados ao tronco da árvore, logo se este tronco de árvore for alterado (danificado) todos os galhos conectados a este tronco vão sofrer juntamente, ou até mesmo o processo inverso, se um galho muito grande for retirado, o tronco central da árvore irá sentir essa mudança.

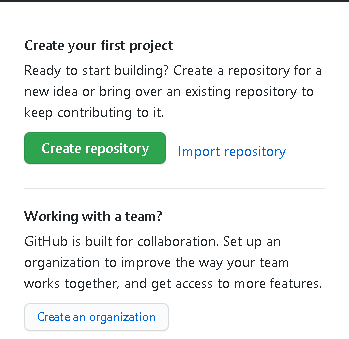
GitHub tem o mesmo conceito, imagine um projeto sendo desenvolvido ou até mesmo um servidor, você precisa de uma equipe para desenvolver, então você como líder cria um repositório, neste repositório você irá conter um ambiente chamado de **Master** (tronco central da árvore), desta forma o projeto que consiste na **Master** será seu projeto padrão. Em seguida vem a etapa de desenvolver novas funções a este projeto, cada desenvolvedor irá precisa criar uma nova função, está função não poderá ser desenvolvida na **Master**, porque desta forma todos desenvolvedor acabaram por ventura modificando o código aonde outro desenvolvedor está mexendo, a solução para este caso é criar galhos (**Branch**) para que cada desenvolvedor tenha o seu ambiente de desenvolvimento, sem modificar algo aonde seu parceiro também esteja mexendo, assim que cada desenvolvedor terminar de efetuar suas alterações eles enviam seus códigos para a Master. Por final a Master sempre irá conter o projeto atualizado e versionado.

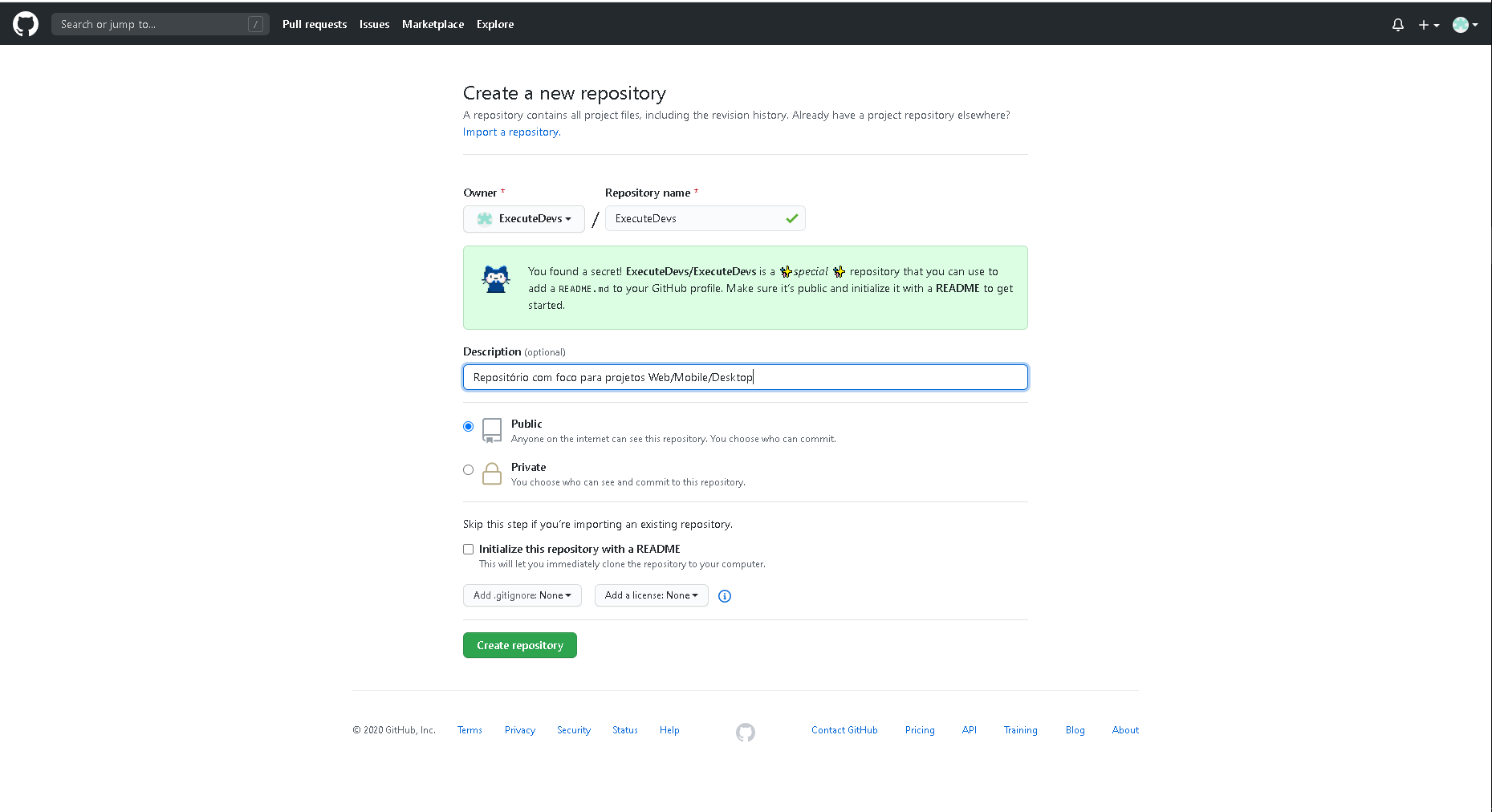
# **Capítulo 1**

Criando seu primeiro Repositório

## Primeiro passo

O primeiro passo a ser tomado para iniciarmos o versionamento de arquivos é criando seu primeiro repositório no site do GitHub: <https://github.com/>  
Ao entrar no site pela primeira vez, para criar um repositório você irá reparar no canto esquerdo superior o GitHub questionando se você está pronto para criar seu primeiro repositório. Basta apenas clicar no botão verde Create Repository.



Na etapa a seguir você irá configurar seu repositório após a imagem irei descrever cada função e etapa. 

Abaixo irei descrever cada opção:

* Owner  
  Proprietário deste repostório no caso será propriedade sua este repositório.
* Repository name  
  Neste espaço você irá colocar o nome do seu repostitório que achar mais adequeado (ExecuteDevs ou ServidorJogo).
* Description  
  Você irá colocar uma descrição para você e que outras pessoas saibam qual a finalidade deste repositório.
* Public / Private  
  Neste local você irá configurar seu repositório para ele ser publico ou privado.  
  Caso seu objetivo seja ter este repositório para controle pessoal, projetos pessoais ... é recomendado que seja criado como privado assim apenas você ou pessoas que convidar poderam manusear estes arquivos.  
  Mas se o seu intuíto seja construir uma aplicação por exemplo uma Calculadora para ajudar diversas pessoas, seja descobrindo como criar uma calculadora ou até mesmo para usarem o aplicativo recomenda-se deixar este repositório publico, mas lembrando que se optar em deixar este repositório publico qualquer pessoas pode-rá manipular os arquivos que você colocou.
* Initialize this repository with a README  
  Quando marcada essa opção seus projetos vão conter um “TXT” aonde você poderá editar para colocar regras e informações que achar necessário.
* GitGnore  
  O GitGnore é uma configuração para projetos, aonde ele irá ignorar arquivos que você não deseja subir ou não seja necessário, por exemplo vamos dizer que você está construir uma aplicação Web, está aplicação Web contém dependências para você programar, então neste caso você precisa baixar as dependências. Vamos dizer que você tenha 1 GB de dependência subir tudo isso para seu diretório não é viável, porque sempre que você ou outra pessoa for pegar a versão mais atual vai precisar baixar este 1 GB. Qual solução neste caso?  
  Criamos um arquivo que vai dizer quais são suas dependências e ele irá baixar na sua máquina local, então o gitGnore estará configurado para subir o arquivo de dependência e não os downloads efetuados em sua máquina.
* Add License  
  Caso necessário um projeto e necessitar de licenças basta apenas procurar a licença que condiz com a linguagem que você está desenvolvendo seu projeto.

Por último basta apenas clicar no botão verde de Create Repository

## Segundo passo

Também para efetuarmos todo tipo de versionamento recomenda-se que baixe e instale o GitHub para desktop.

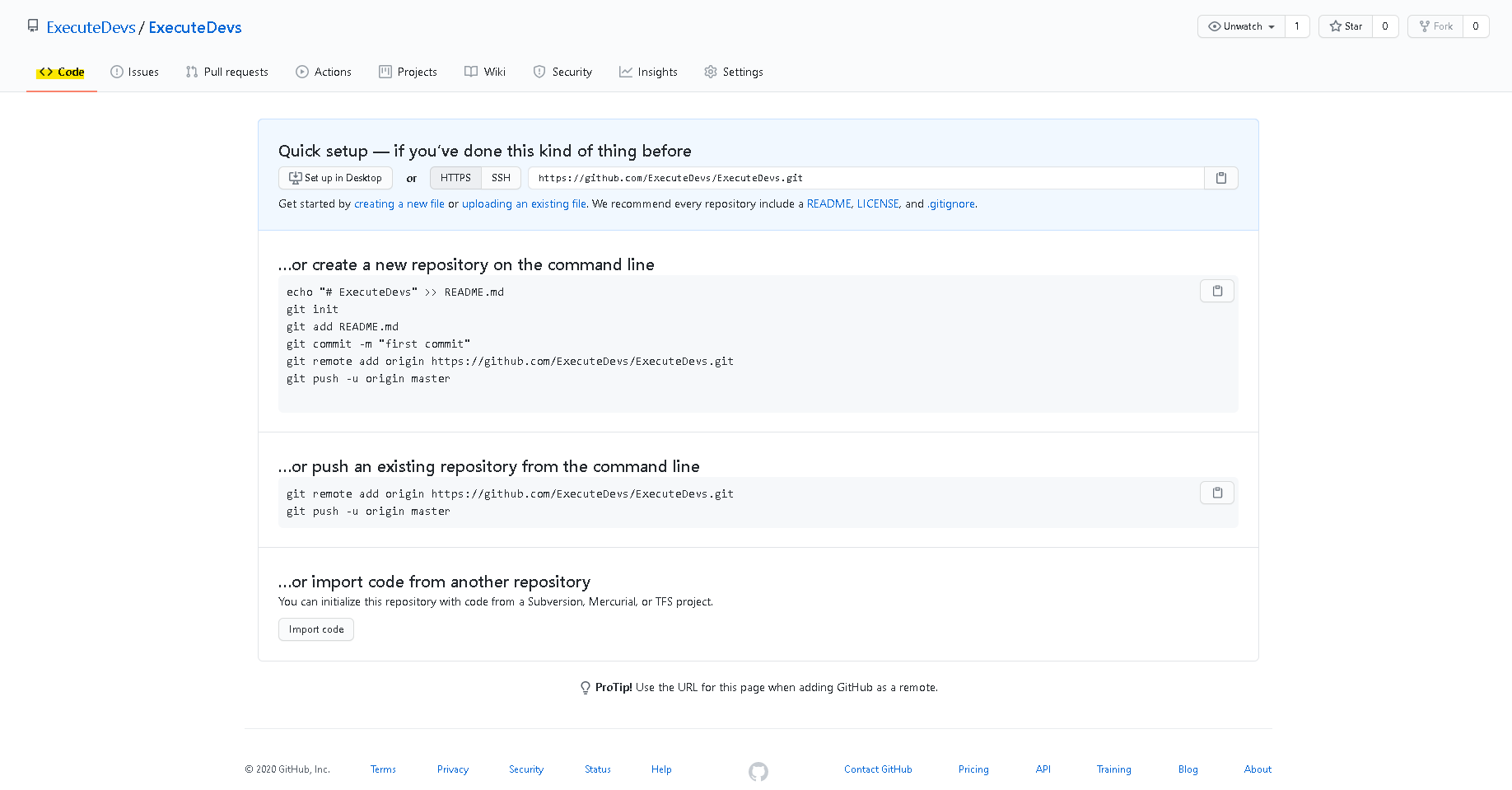
Link: <https://desktop.github.com/>

# **Capítulo 2**

Configurando seu Repositório

Neste capítulo vamos configurar seu repositório para que você possa entender oque pode construir ou até mesmo alterar e excluir.

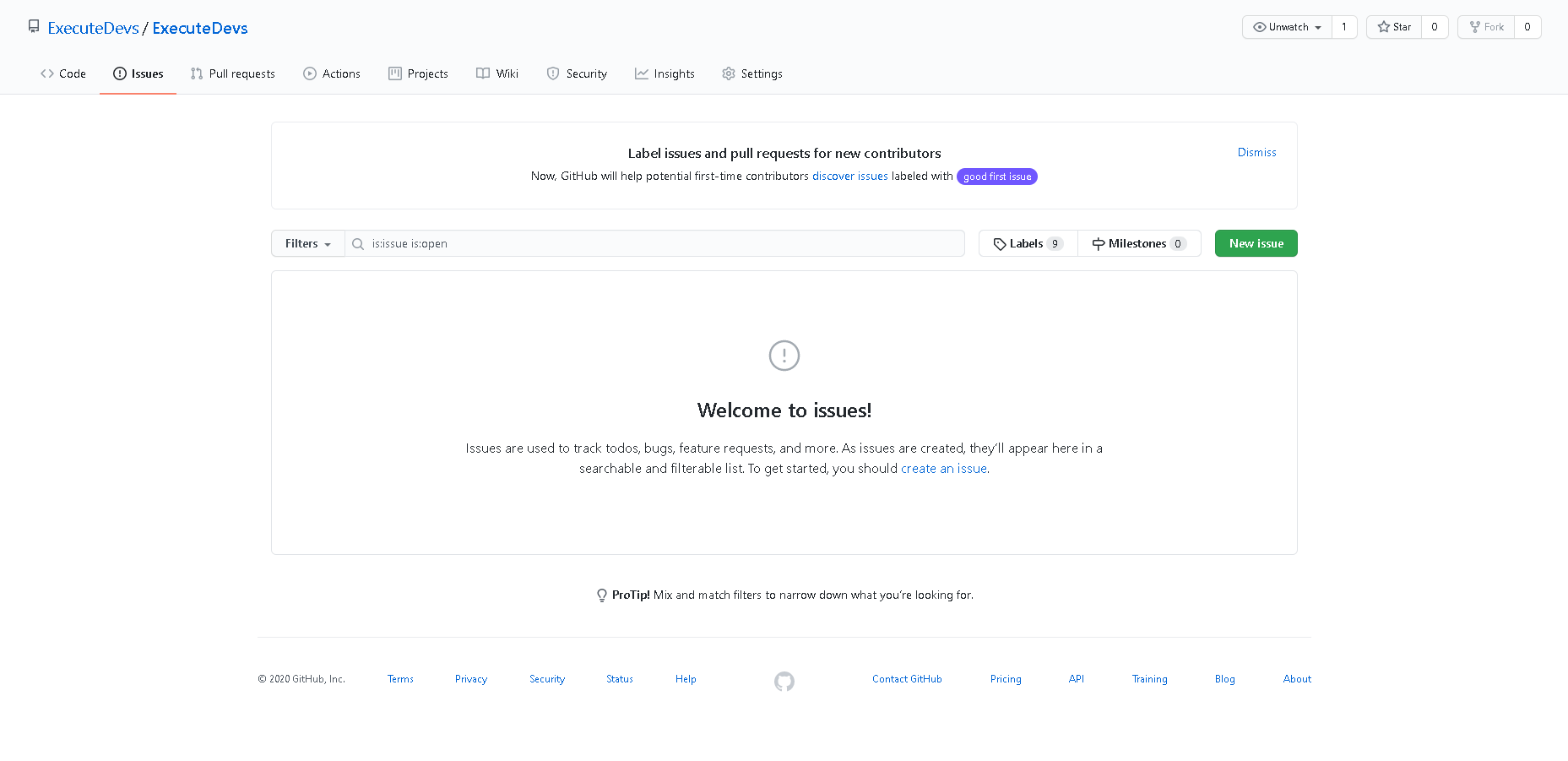
## Code



Nesta primeira aba em Code estará concentrado seu projeto/código/arquivos desta forma podendo monitorar e gerenciar seu repositório.

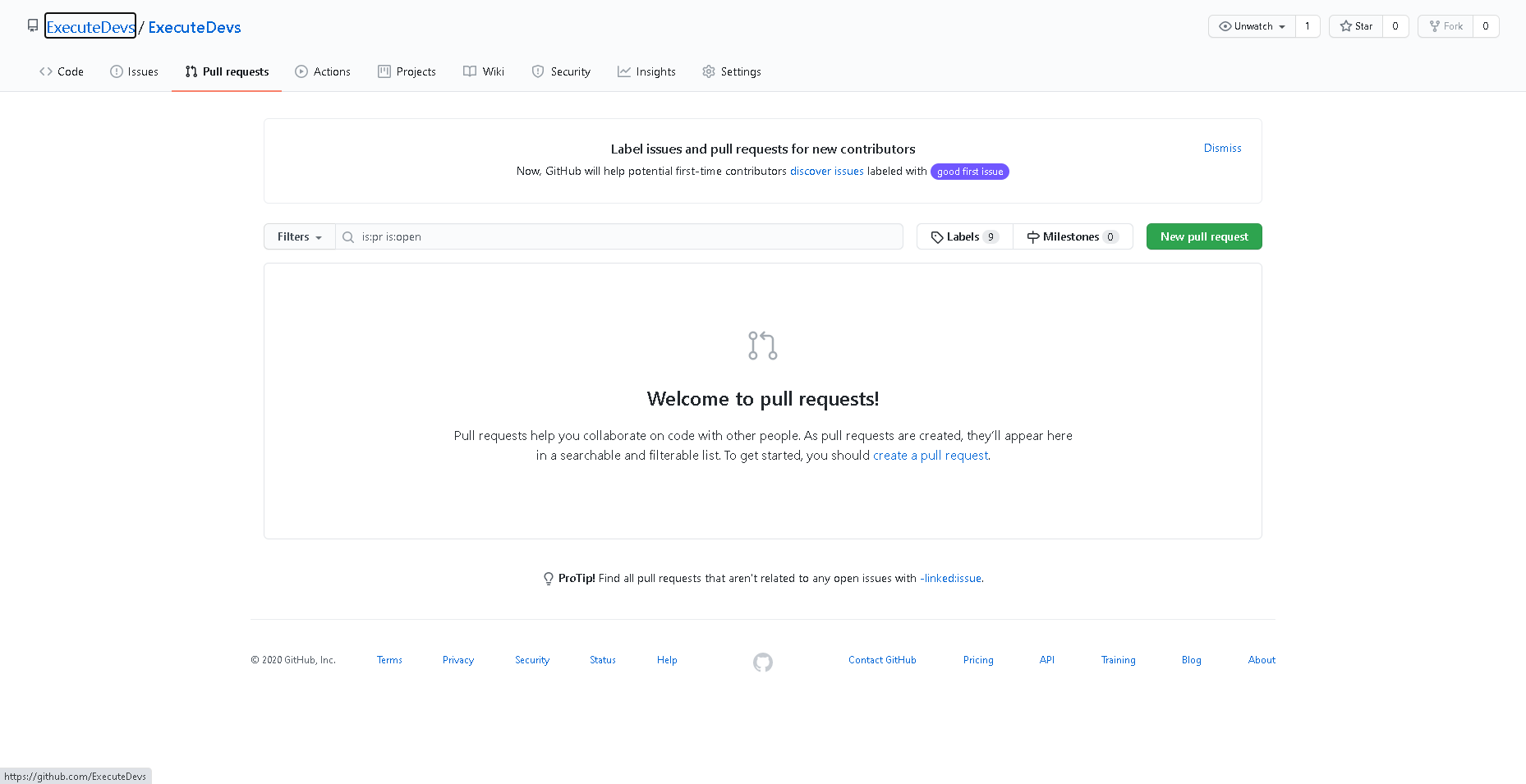
Neste primeiro passo seu repositório estará vazio e explicando para você como pode começar a carregar arquivos em seu repositório, faremos isso na prática, mas apenas no Capítulo 3.

## Issues



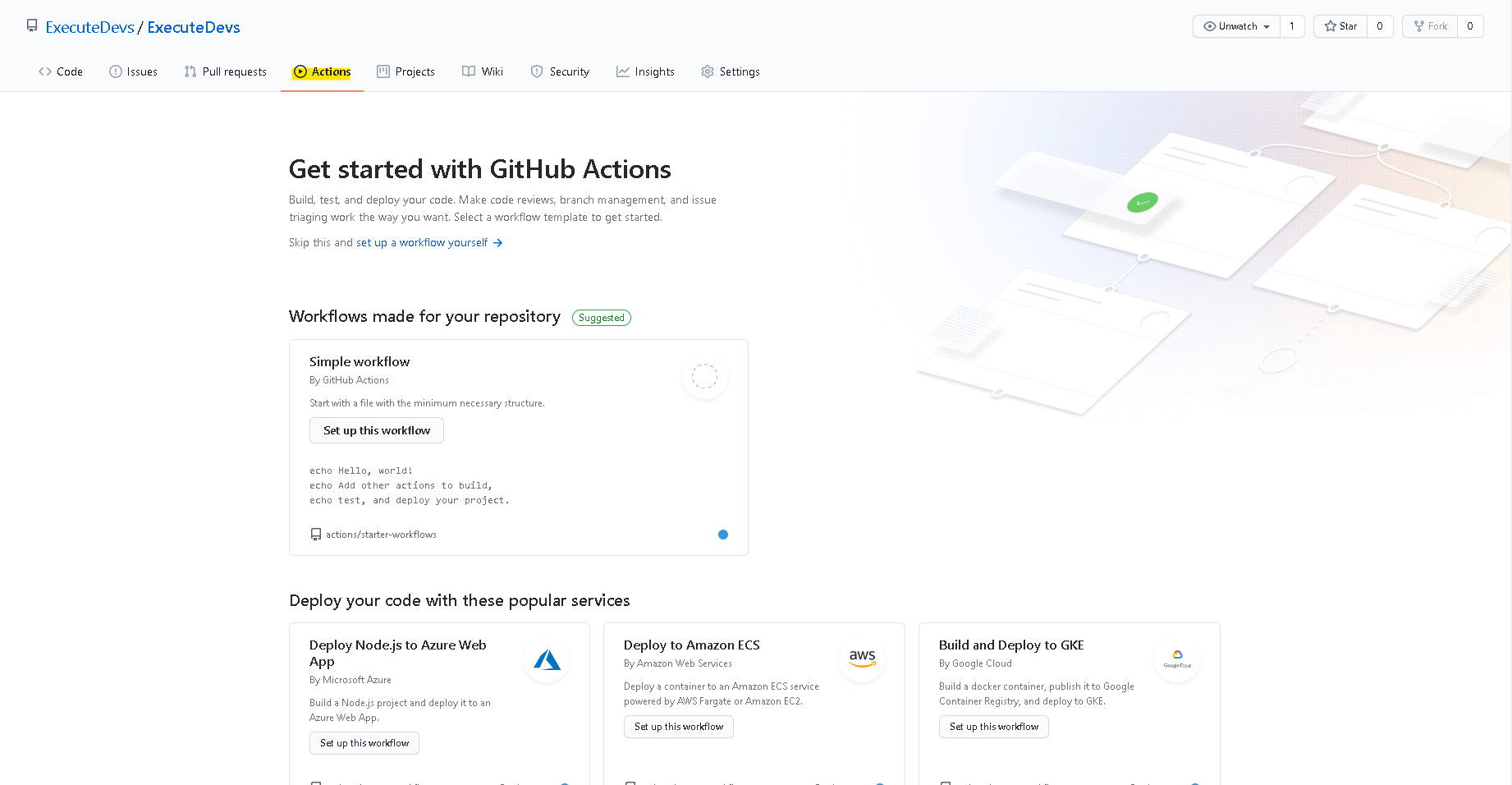
Nesta aba de **Issues** você consegue monitorar oque está ocorrendo em seu projeto. Sendo assim imagine que você tem um projeto Web aonde você e diversos desenvolvedores estão trabalhando, alguém acabou subindo uma versão errada, ou até mesmo antes do tempo você consegue monitorar todo o ciclo.  
Juntamente nesta aba você consegue fazer demarcações **Milestones**, onde desta forma você pode colocar ponteiros indicando um exemplo quando aquele projeto foi publicado .

## Pull requests



Trabalhando com **Pull Request**, como mencionado acima o GitHub trabalha com um método parecido com arvore, desta forma imagine que você como o dono do repositório está trabalhando com diversos desenvolvedores, cada desenvolver irá conter uma **Branch** em sua máquina, desta forma sempre que o desenvolvedor terminar de codificar e precisar mantar para a raiz da arvore o código novo e atualizado, ele vai passa pela sua autorização aonde você irá verificar o que foi modificado, quando e por quem, ciente destas alterações você pode aceitar ou rejeitar esse **Push**(envio) realizado pelo desenvolvedor.

## Actions



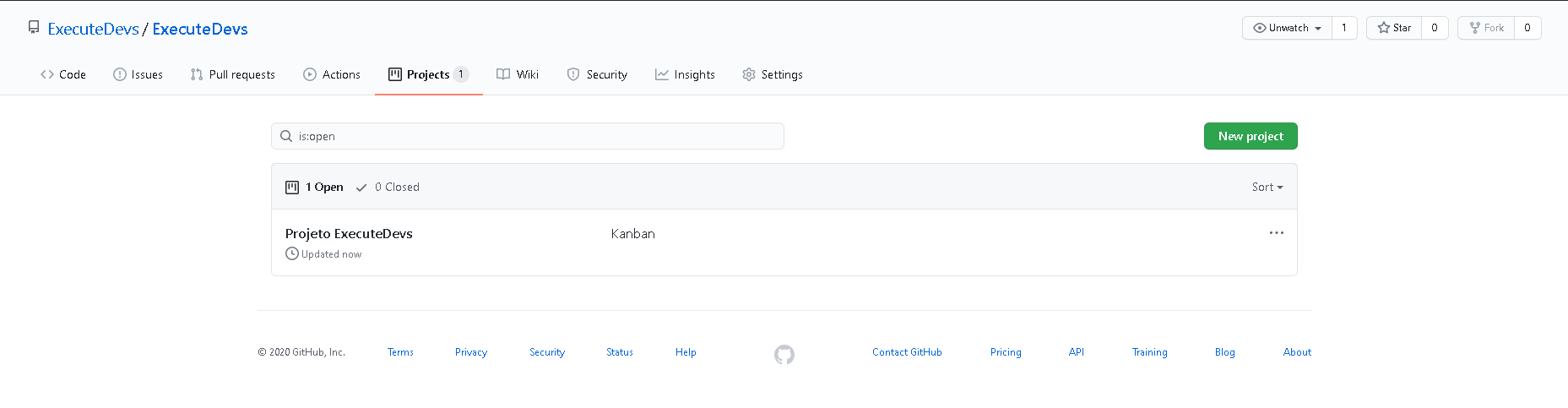
Trabalhando um pouco mais voltado com projetos na aba **Actions** conseguimos utilizar diversos serviços, dentre eles estão:

* Publicação de projeto
* Testes em aplicações e projetos
* Automações

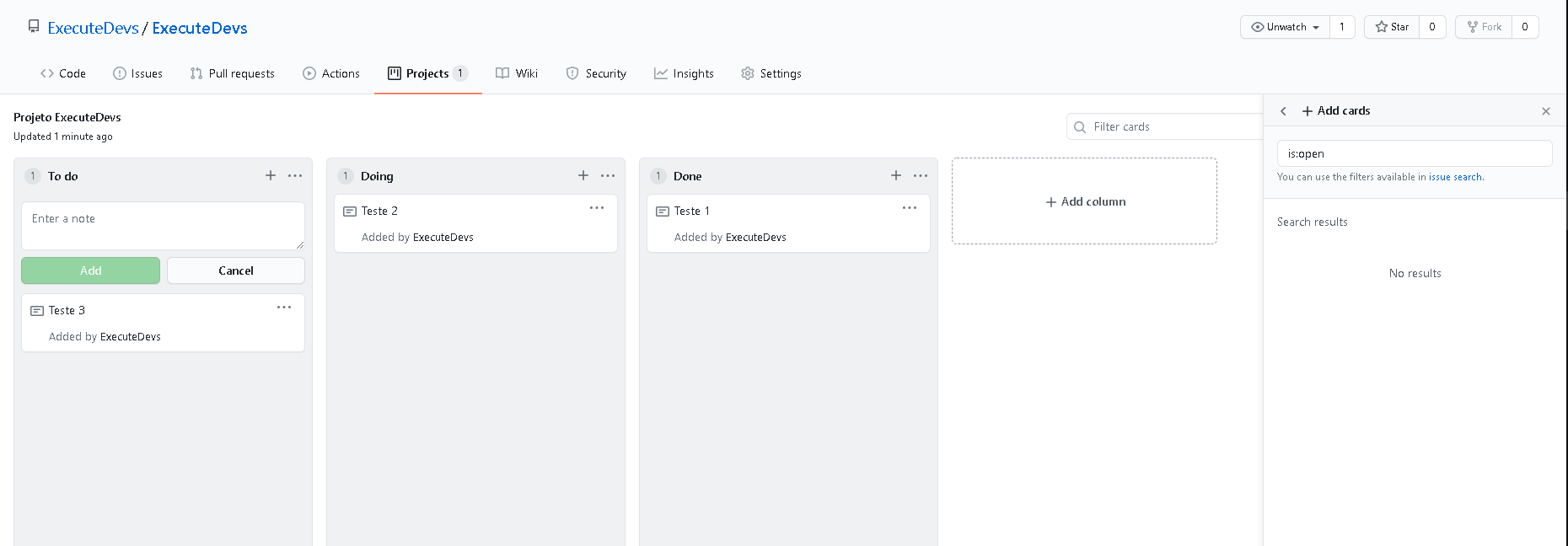
Pense da seguinte forma, você tem uma aplicação em JSP ou .Net aonde está efetuando o versionamento dela no seu repositório, diremos que você precise efetuar a publicação deste projeto agora na **AWS**, dentro de **Actions** você consegue efetuar o deploy desta aplicação na **AWS** diretamente do seu repositório.

Pesando além você pode criar um ciclo de automação aonde ao colocar o projeto mais atual no seu repositório ele irá testar seu projeto e publicar automaticamente na **AWS**.

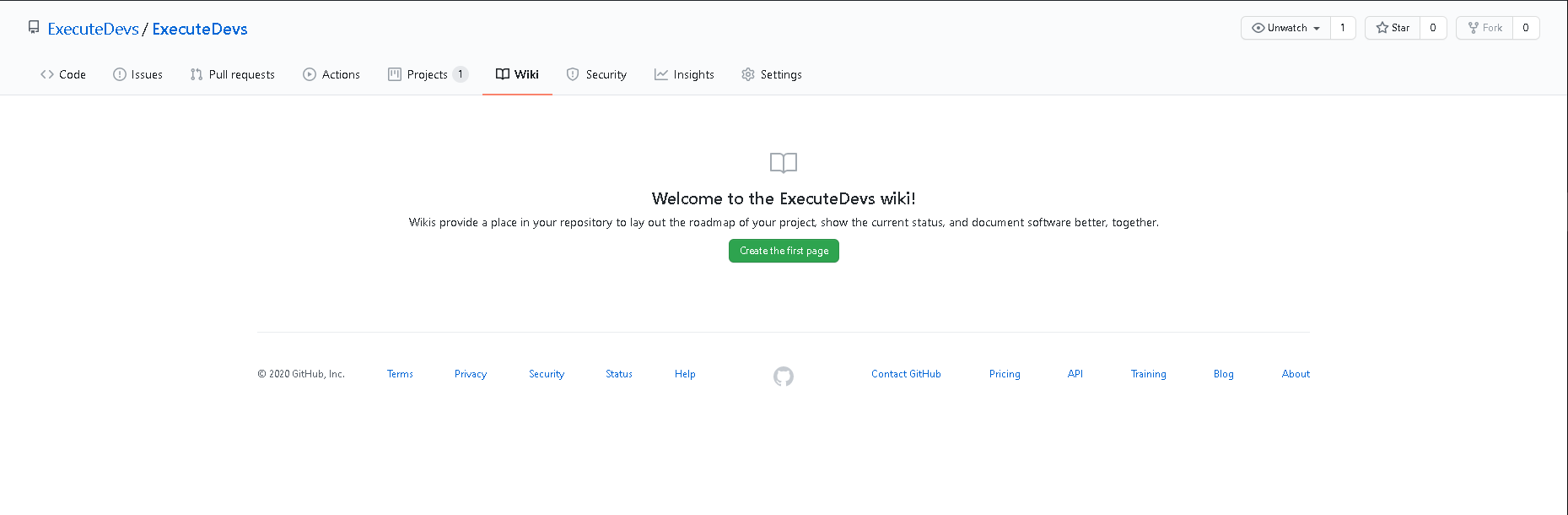
## Projects



Entrando mais afundo em projetos, podemos fazer um gerenciamento de tarefas a desenvolver dentro do próprio GitHub. Podemos montar um Kanban dentro do GitHub onde podemos centralizar tudo em uma única ferramenta, sem necessidade de manter duas ou mais ferramentas para auxílio do desenvolvimento do projeto e até mesmo gerenciamento e divisão de tarefas.

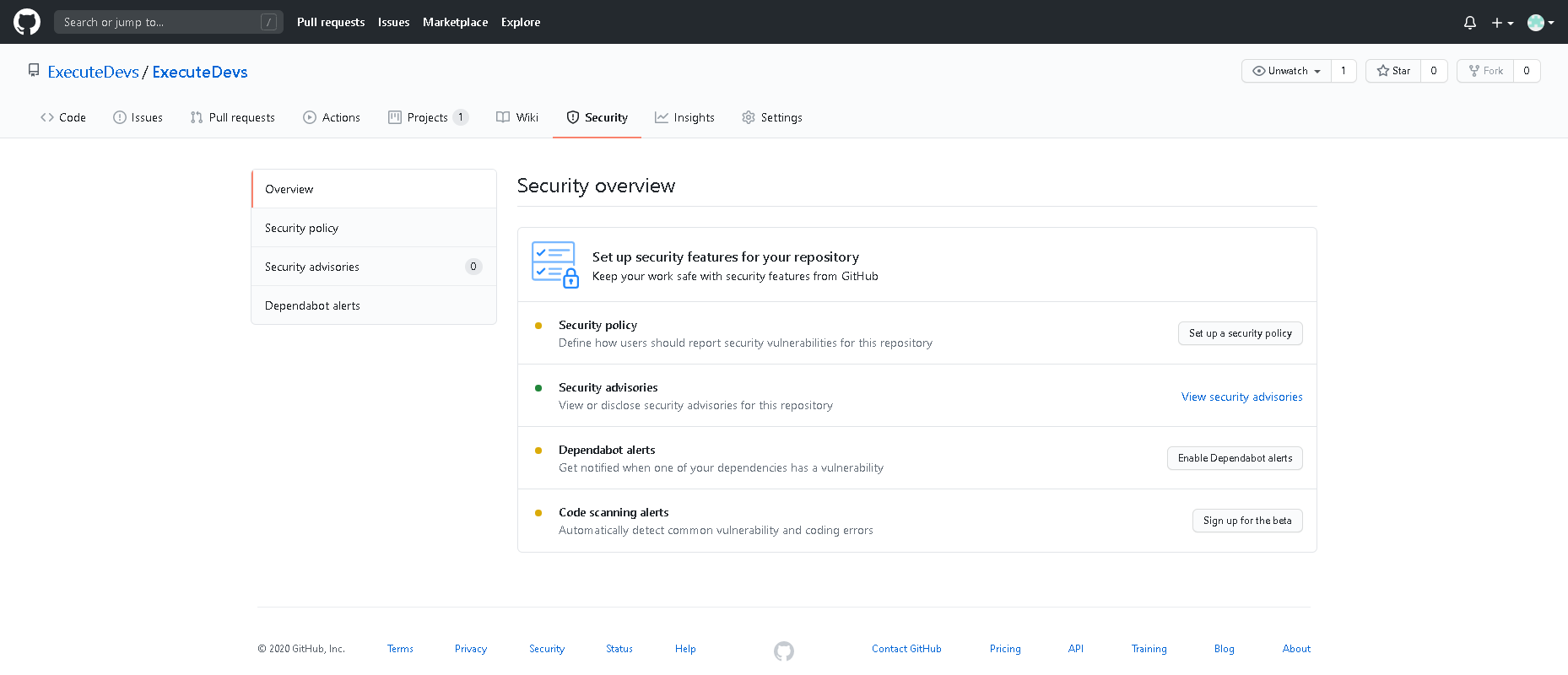


## Wiki



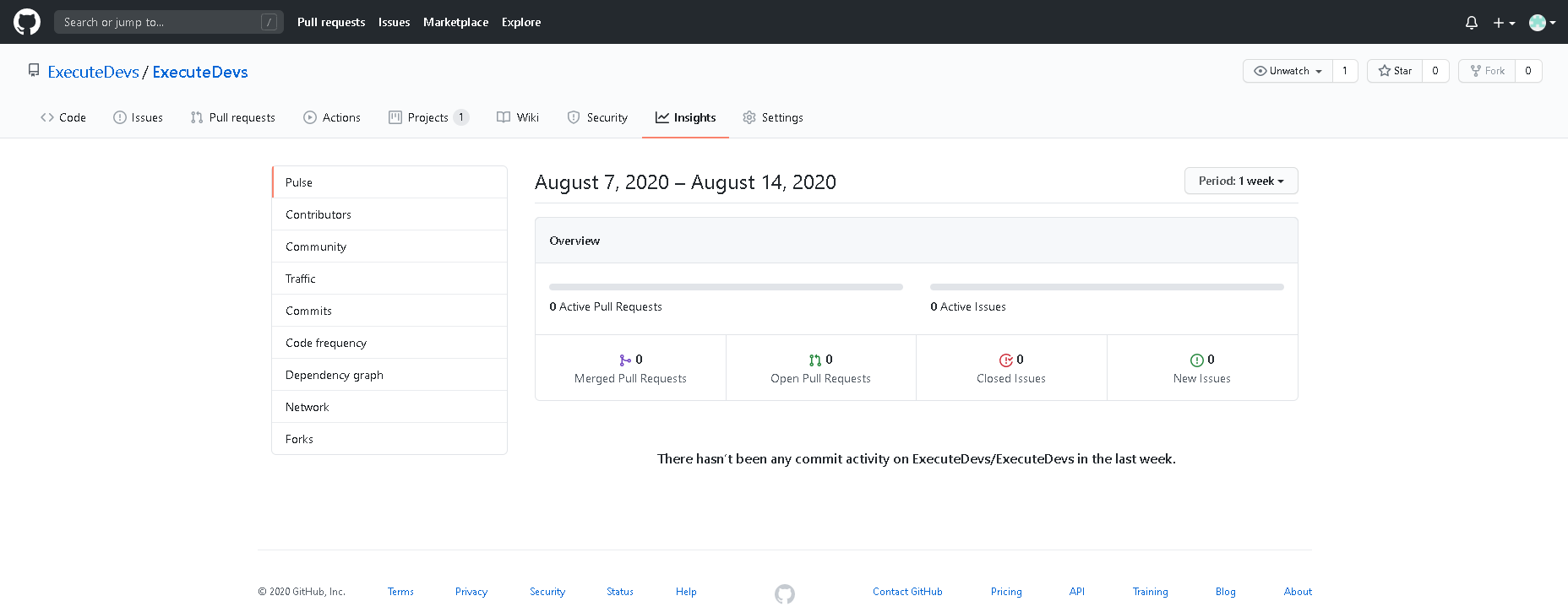
Pensando mais adiante e explorando temos a aba Wiki, sua função pode ser definida de uma forma bem prática de direta, pense que todo projeto necessita de uma documentação sendo assim, aonde mais podemos atribuir uma documentação de forma específica e centralizada? No próprio repositório do projeto.

## Security



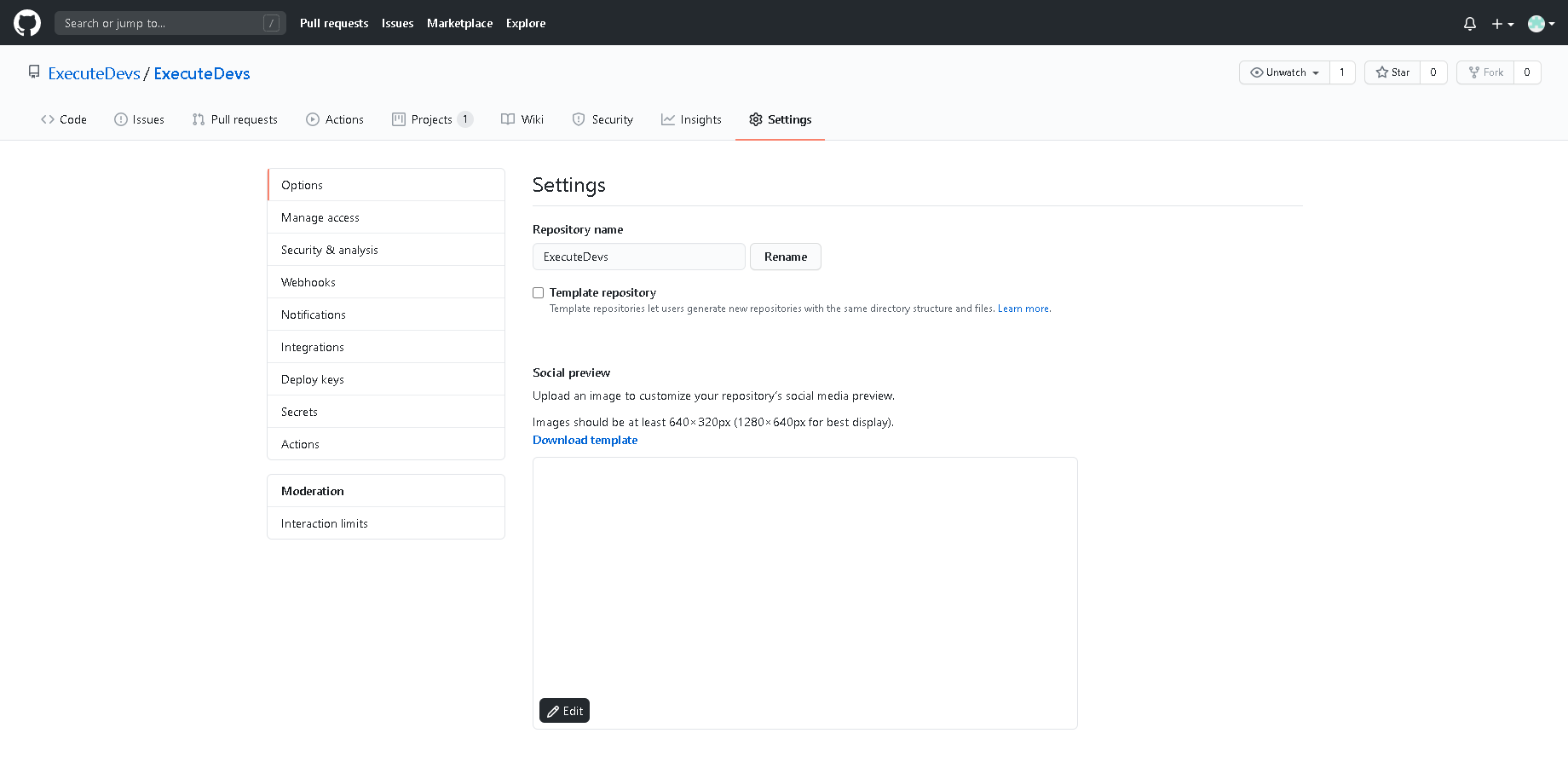
Um pouco mais para o lado e nos deparamos com a aba de Security, neste momento imagine que você tem um projeto público, com uma finalidade de fornecer e também receber ajuda de outros desenvolvedores para construção de um projeto, desta forma diversas pessoas podem acessar e manipular seu repositório, mas agora pense bem isso poderia ser perigoso porque alguém pode apagar seu projeto e tentar subir uma pasta vazia para seu repositório, sendo assim desta forma Security é uma aba aonde você pode organizar politicas de segurança e visualizar até mesmo falhas e visualizar se a segurança do seu projeto foi violada.

## Insights



Podemos ter diversos acompanhamentos pela plataforma e uma delas é monitoramento via gráficos de como está sendo utilizado o repositório, ou seja, podemos ver os colaboradores do repositório e sua frequência, podemos monitorar qual o volume que está subindo de dados e o fluxo dentre outras opções.

## Settings



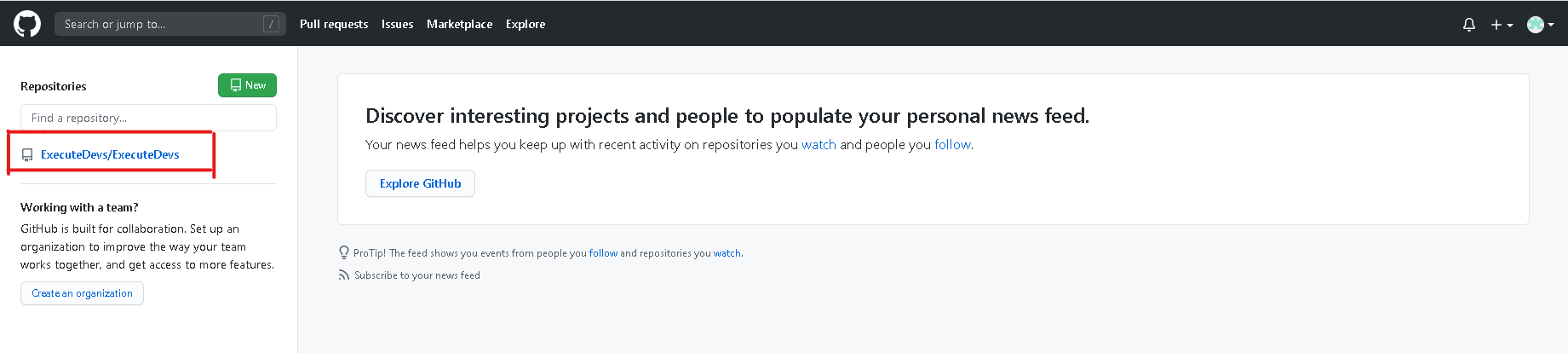
Por ultimo mas não menos importante encontramos na ultima aba Settings tudo em relação a um repositório ou seja, podemos configurar ele de todas formas, alterando de privado para publico e publico para privado, podemos convidar pessoas para trabalhar e ajuda em nosso repositório, alterar nomes, criar chaves de acesso dentre outros coisas que você poderá visualizar nos próximos capítulos.

# **Capítulo 3**

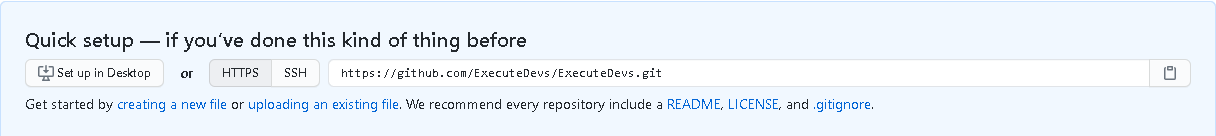
Efetuando seu primeiro versionamento

Neste capítulo vamos iniciar seu primeiro versionamento de arquivos

## Code



Agora para iniciarmos a terceira parte do curso, você deverá selecionar um projeto que deseja fazer o versionamento em sua máquina. Após decidir qual projeto deseja versionar selecione clicando no nome do projeto.



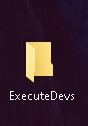
Em seguida procure na aba Code, a imagem acima onde ele referencia o link para iniciar o repositório do seu projeto em sua máquina.

Podemos acessar o repositório de duas formas:

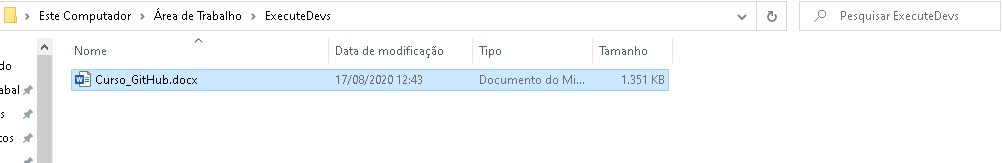
* SSH  
  Acessamos o repositório via Shel (Linha de comando)
* HTTPS  
  Acessamos o repositório via URL (Como se acessaríamos uma página Web)

Logo após localizar essa informação copie o link HTTPS do seu repositório, você pode selecionar a URL e pressionar Ctrl + C ou clicando no ícone ao fim da linha.

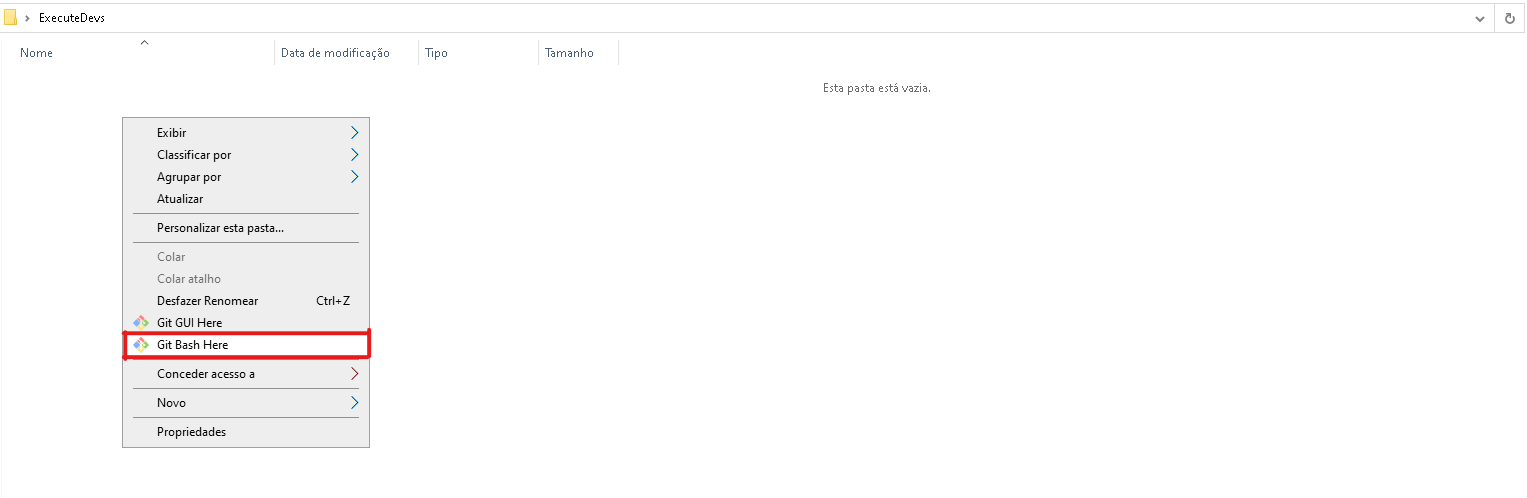
## Iniciando seu repositório



Para iniciarmos seu repositório de maneira adequada, primeiro crie uma pasta na sua área de trabalho com o nome que achar adequado.



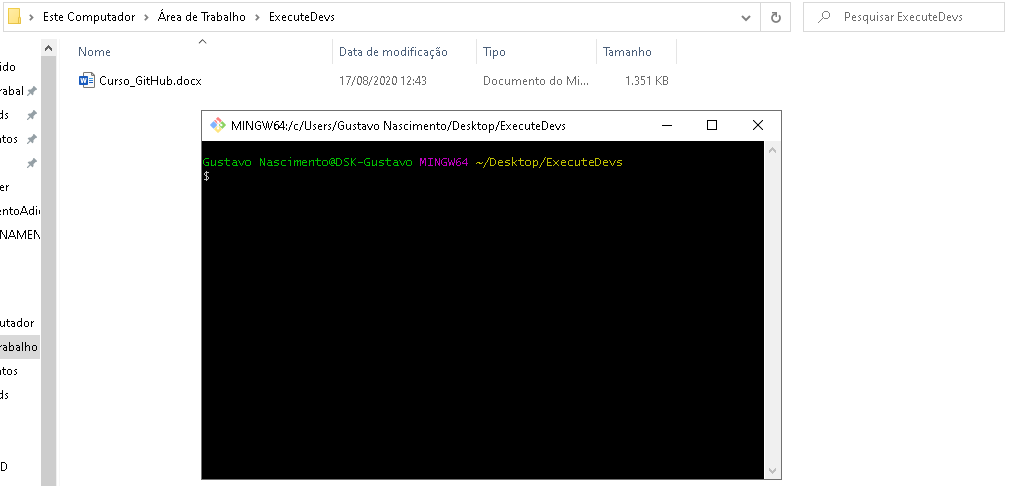
Em seguida como segunda etapa adicione o arquivo que deseja na pasta, para que ele seja disponibilizado em seu repositório, neste caso vamos demonstrar com o próprio arquivo do curso que estamos criando para disponibilizar.



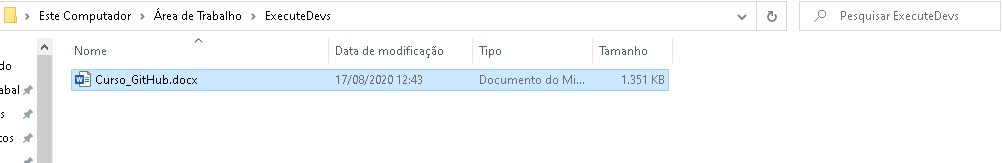
Em seguida clique com o botão direito em qualquer lugar da pasta aonde esteja vazio.

Um menu com algumas opções estará disponível para você procure pelo menu Git Bash Here, caso não encontre o menu certifique-se que você seguiu o passo a passo de pré-requisitos.

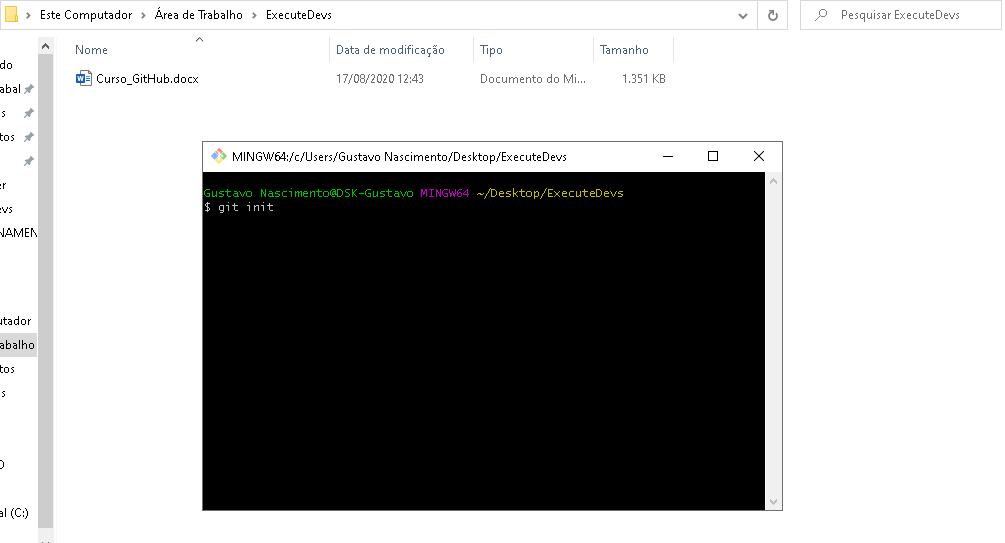
## Git Bash



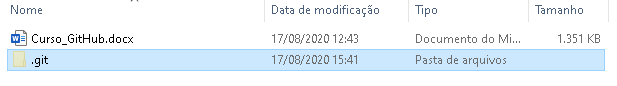
Para iniciarmos seu repositório de maneira adequada, primeiro crie uma pasta na sua área de trabalho com o nome que achar adequado.

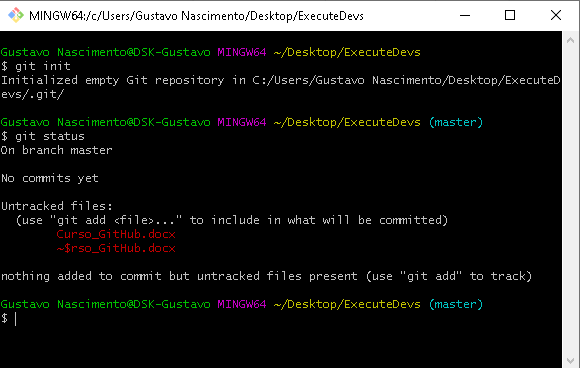


Em seguida como segunda etapa adicione o arquivo que deseja na pasta, para que ele seja disponibilizado em seu repositório, neste caso vamos demonstrar com o próprio arquivo do curso que estamos criando para disponibilizar.

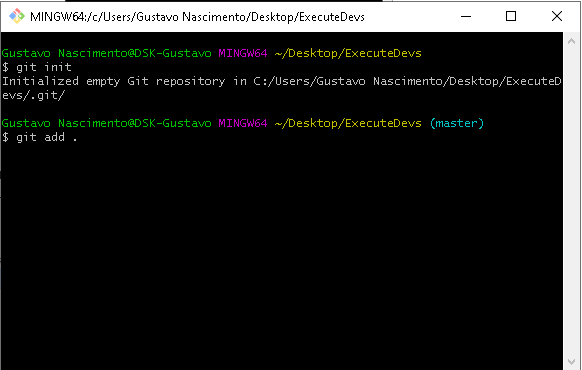


Para iniciarmos a subida dos arquivos ao repositório devemos digitar o seguinte comando **git init** (tudo em minúsculo), com esse comando o Git irá criar uma pasta oculta para você com as configurações iniciais básicas.

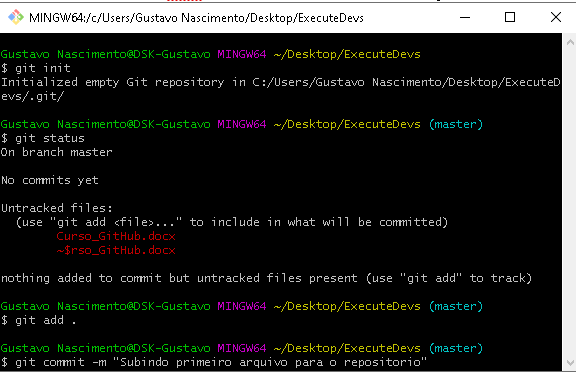




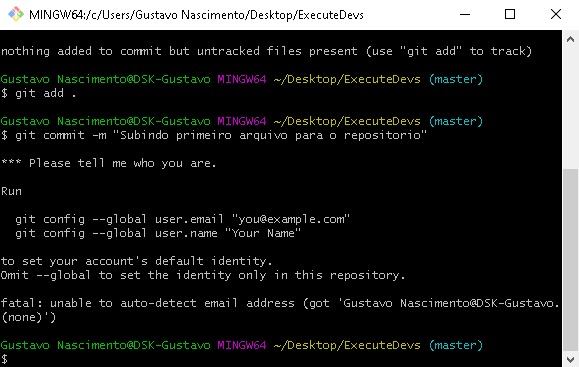
Em seguida logo após ele ter criado as configurações padrões, vamos verificar quais arquivos estão constando para subida do seu repositório, basta executar o comando **git status**. Os Arquivos em vermelho são arquivos novos que ainda não existem no diretório, caso esses arquivos já existam, ele apenas trocaria de cor caso esse arquivo consta-se qualquer mudança.



Logo após ele ter adicionado a pasta do Git contendo suas configurações iniciais e verificar que existem arquivos pendentes para colocar em seu repositório, será necessário adicionar outro comando **git add .** ou **git add all**, desta forma você irá indicar para o GitHub que deseja subir todos arquivos existentes naquela pasta, caso deseje subir apenas um arquivo específico coloque o comando com o nome do arquivo **git add Curso\_GitHub.docx**. Pressione Enter, caso não mostre nenhuma mensagem de erro, significa que o arquivo foi adicionado com sucesso, mas isso significa que ele está em um estado de “pendente”, ainda será necessário indicar com um **Commit** (Comentário) o que está ocorrendo com essa mudança.



Para adicionar um comentário a uma mudança será necessário adicionar o seguinte comando **git commit -m “Mensagem de texto que achar adequado”**. Toda mudança na pasta do seu repositório precisar ser adicionada para subida e deve conter um comentário de identificação, para que quando analisado no seu repositório contenha as informações explicando aquela alteração, seja um arquivo novo adicionado, alterado ou até mesmo excluído. Logo em seguida pressione Enter.



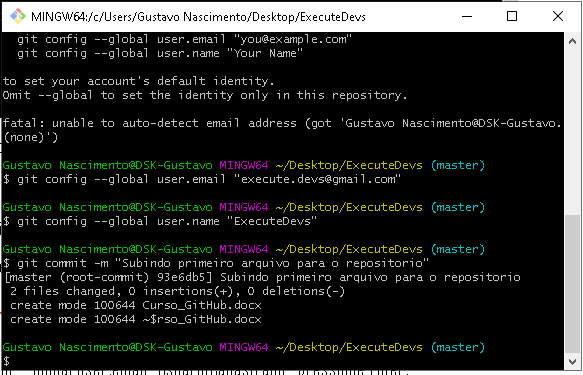
Mas agora vamos com calma, provavelmente você deve ter se deparada com uma tela pedindo para que você se identifique, agora vamos configurar seu usuário do GitHub com o usuário do GitHub Desktop, pega seu e-mail e nome de usuário cadastro no GitHub.

Adicione a seguinte linha

**git config -–global user.email “**[**emailcadastrado@email.com**](mailto:emailcadastrado@email.com)**”** pressione Enter.

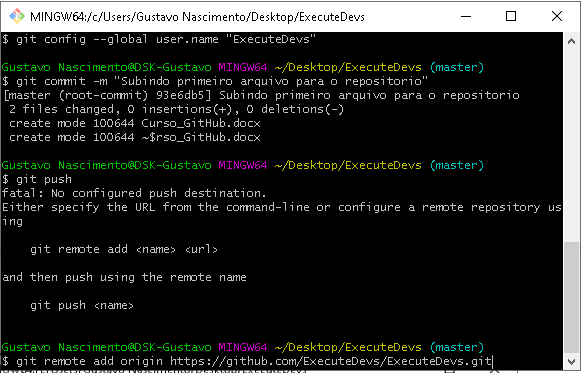
**git config –-global user.email “UsuarioCadastrado”** pressione Enter.

Após isso repita o comando de git commit -m “Texto” e pressione Enter.

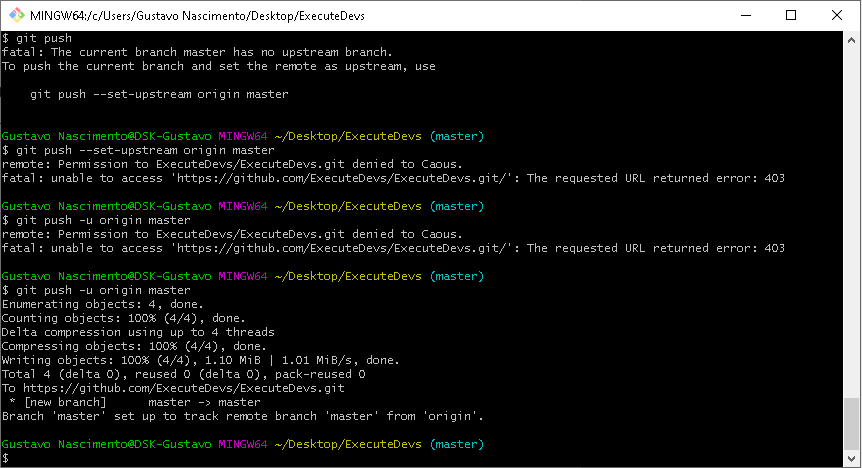


Seu resultado será semelhante a este.

A partir deste momento você deixo seus arquivos em estado de “pronto para subida”, basta finalizarmos apontando para qual diretório desejamos fazer o upload projeto e por último efetuar o upload.

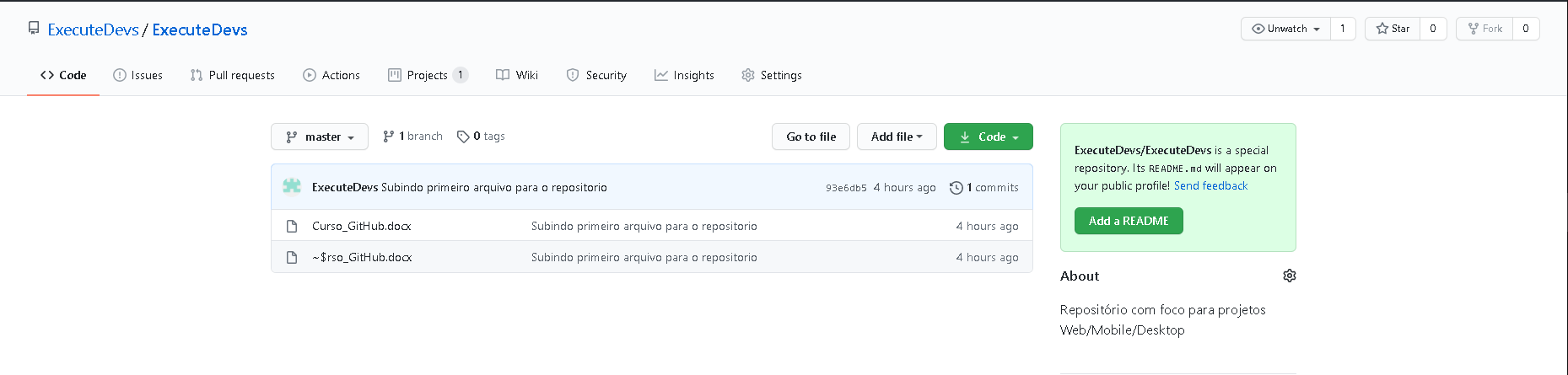


Vamos definir qual repositório desejamos que seja aplicado essas alterações consistentes. Execute o seguinte comando **git remote add origin https://github.com/ExecuteDevs/ExecuteDevs.git**.



Agora para finalizarmos a subida com êxito digite o seguinte comando **git push -u origin master,** este comanda irá empurrar para o seu repositório suas mudanças sendo assim irá persistir os arquivos que estão na máquina no repositório.

Agora volte ao seu navegar no seu repositório na aba **<>Code**, você vai se deparar com seus arquivos e o comentário(**commit**) efetuado, com a data.



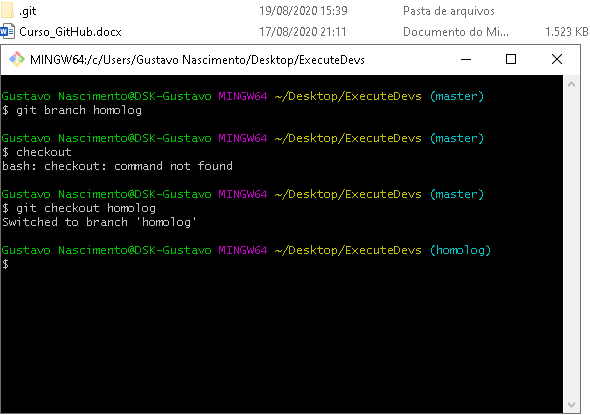
## Observações do capítulo:

# **Capítulo 4**

Versionamento de arquivos

No capítulo 4 vamos simular que este repositório será tratado por diversas pessoas, onde o documento que se encontra no repositório ExecuteDevs será alterado e tratado.

## Criando Branch



A partir deste momento vamos passar a utilizar as **Branch**, como mencionado ao início do curso, imagine o GitHub como uma árvore, desta forma temos o tronco da árvore que vamos denominar de Master. Uma árvore contém diversas ramificações, onde essas ramificações podem ser alteradas e enviadas novamente para a Master, podemos criar qualquer nome que acharmos compatíveis com nossas tarefas, neste caso vamos criar duas **Branch**, uma vamos denominar de **Homolog** e outra de **Cap\_4\_Adicao**.

O ambiente denominado **Homolog** será um ambiente aonde vamos validar todas alterações antes de **commitar** e efetuarmos **push** para a **Master**, normalmente encontramos esse processo em diversos projetos. Criamos uma função em um projeto, mas antes de publicarmos para todos passarem a usufruir desta função precisamos que pessoas responsáveis validem se aquela informação está de acordo com o solicitado.

O ambiente denominado **Cap\_4\_Adicao**, tem como objetivo separa o desenvolvimento da tarefa. Antes de enviarmos as alterações do capítulo 4 para branch **Homolog**, imagine em um projeto Web, você é responsável por criar uma nova página HTML, desta forma você precisará desenvolver esta página, mas não poderá desenvolver na **Master**, porque se você publicar essa alteração e sua página estiver com qualquer erro, você irá derrubar o seu site, mas também não podemos fazer diretamente em **Homolog**, porque é um ambiente para testes.

Então neste caso criamos uma Branch nova, onde ela irá conter todas as informações da Branch **Homolog**, para que seu projeto esteja atualizado, assim também evitando que você não desenvolva e solte está versão diretamente na Homologação da sua aplicação Web.

Para criarmos uma nova branch vamos utilizar o seguinte código:

**git branch homolog;**

**git checkout homolog;**

Desta forma você irá identificar no fim da linha que entre () está a palavra homolog.

Para subirmos uma homolog vamos utilizar um commit

**git push -–set-upstream origin homolog;**

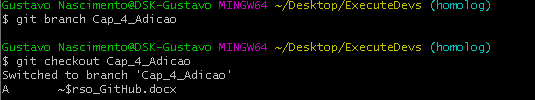
Desta forma vamos conseguir subir nossa branch homolog

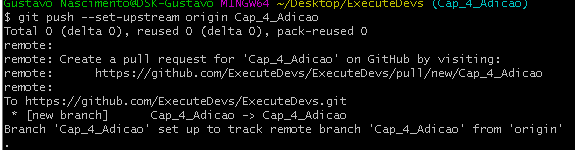
Repita o mesmo processo para criação da branch **Cap\_4\_Adicao**

**git branch Cap\_4\_Adicao;**

**git checkout Cap\_4\_Adicao;**

**git push -–set-upstream origin Cap\_4\_Adicao;**





## Atualizando Branch

Vamos manusear as branch com o arquivo atual, a proposta será da seguinte forma.

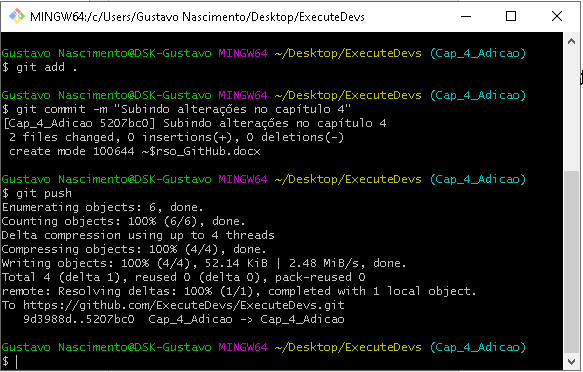
Iremos adicionar novos conteúdos ao capítulo 4, no arquivo **Curso\_GitHub**, mas faremos essas alterações apenas na branch **Cap\_4\_Adicao**. Após a adição dos conteúdos no capítulo 4, vamos efetuar commit e push para enviarmos essas alterações ao repositório. Em seguida vamos trocar para branch **Homolog**, com o intuito de verificar se na branch **Homolog** contém esse novo conteúdo adicionado no capítulo 4.

Mas quando conferirmos vamos ter a certeza, que as alterações ficaram apenas na branch **Cap\_4\_Adicao**, então como consequência vamos atualizar a branch **Homolog** e **Master**, desta forma vamos conferir se as alterações realmente foram aplicadas desta vez.

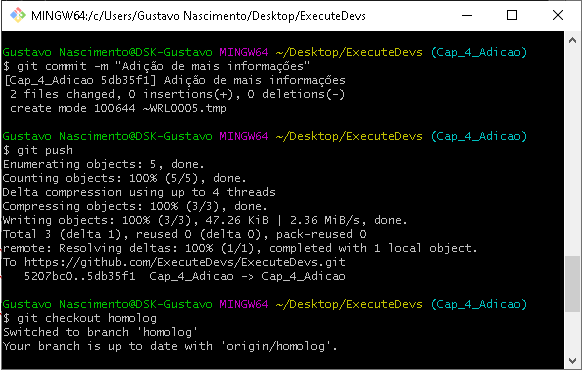
Para isso vamos efetuar os dois seguinte comandos:  
**git add .**

**git commit -m “Subindo alterações no capítulo 4”**

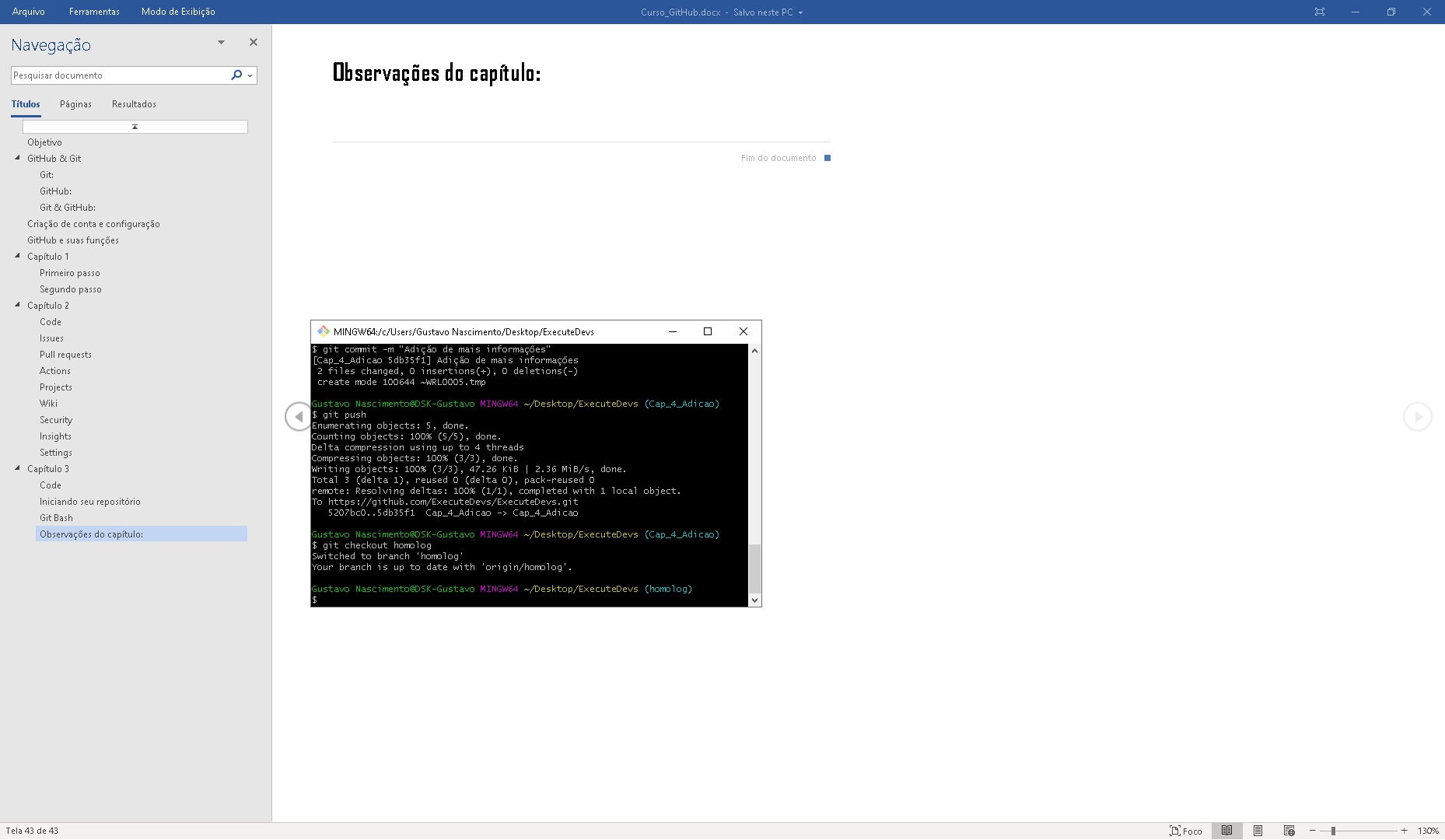
**git push**

  
  
Após esses três comandos executados vamos alterar para a branch de homolog e verificar se essas alterações estão no arquivo da branch homolog. Digite:

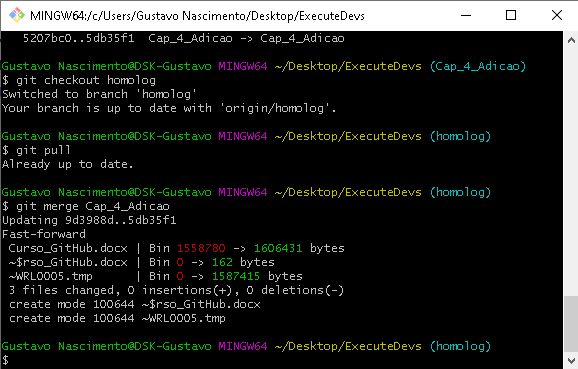
**git checkout homolog.**



Ao abrir o arquivo novamente vamos nos deparar que a branch de homolog não contém nem o capítulo 4.



Verificando isso está totalmente correto, nós criamos um ambiente para desenvolver o capítulo 4, então no ambiente de Homolog (testes), não deve conter alterações que ainda não foram concluídas. Mas agora vamos dizer que o capítulo 4 esteja completo, então devemos mandar para o ambiente de teste para que aprovem as alterações. Sendo assim para atualizar o ambiente de Homolog vamos utilizar o comando **Git Merge**, este comando irá mesclara as alterações que foram efetuadas no arquivo da branch Cap\_4\_Adicao para sua branch **Homolog**.



Assim sucessivamente faremos o mesmo método para a branch Master, mas não neste momento porque lembre o ambiente Master é seu ambiente de produção, no momento temos muito ainda para produzir antes de efetuarmos merge para master.

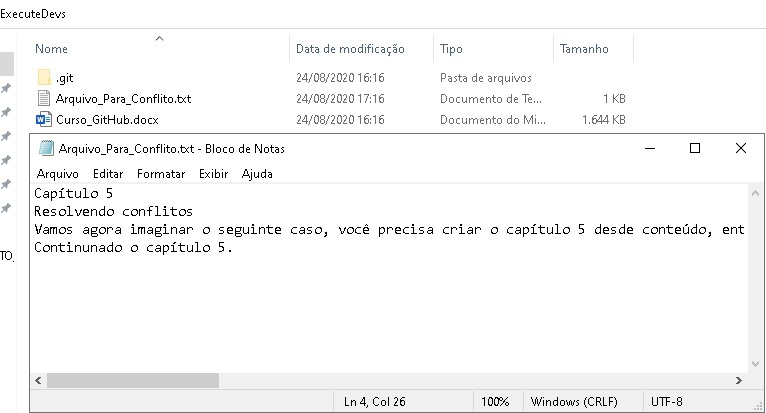
# **Capítulo 5**

Resolvendo conflitos

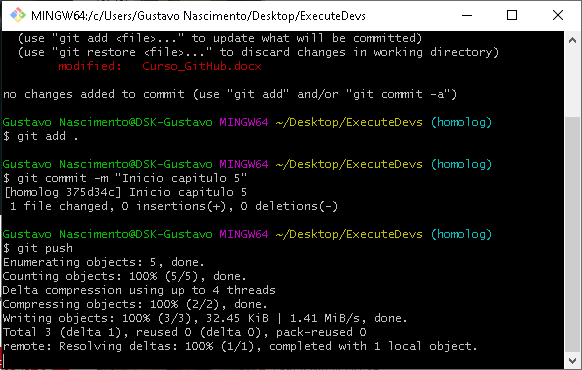
Vamos agora imaginar o seguinte caso, você precisa criar o capítulo 5 desde conteúdo, então você cria uma branch **Cap\_5\_Adicao**, cria seu capítulo, mas em paralelo um colega também criou o capítulo 5 em outra branch, na hora que você foi efetuar o merge da **Cap\_5\_Adicao** para **Homolog** ele irá conflitar avisando que ambos lugares no arquivo foram editados.

Neste caso vamos criar um arquivo separado para não corrermos risco de perder qualquer coisa no documento deste curso, vamos criar um arquivo chamado: **Arquivo\_Para\_Conflitos.txt.**

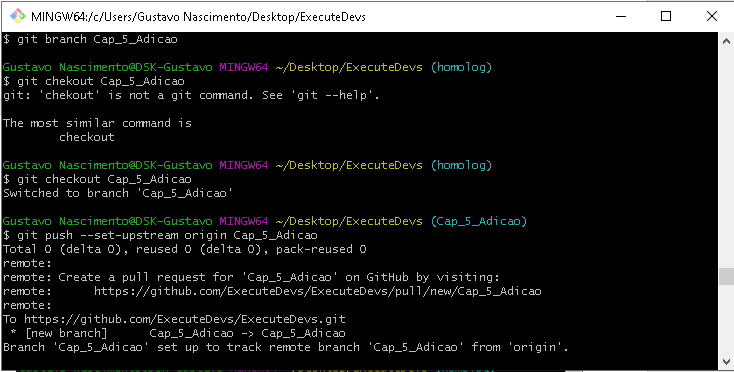
Neste arquivo vamos colocar um trecho do capítulo 5. Em seguida vamos trocar para a branch Cap\_5\_Adicao e vamos alterar uma das linhas do trecho.



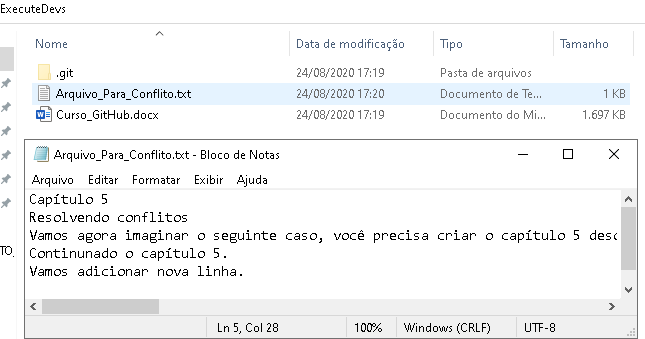
Após adicionarmos o arquivo vamos atualizar nossa branch neste caso fizemos a adição deste arquivo na branch Homolog.



Após atualizarmos a branch Homolog vamos criar a branch Cap\_5\_Adicao



Em seguida na branch Cap\_5\_Adicao vamos adicionar uma nova linha.



Agora vamos efetuar commit e push, mas não vamos atualizar a branch Homolog. Ao invés de atualizarmos a branch Homolog vamos trocar para a branch Homolog, nosso arquivo não irá conter a linha “Vamos adicionar nova linha”, então vamos adicionar um outro texto.