# **Dedicatória**

Nossa missão aqui na Execute Dev é trazer para você uma maneira de pensar como podemos utilizar qualquer ferramenta para nosso dia a dia, seja para trabalho ou até mesmo de forma casual, espero que entenda como essa ferramenta pode utilizada, assim podendo usa-la no seu dia a dia agregando o máximo de valor possível. 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 necessidade, seja em no trabalho ou de forma casual. Até mesmo de forma distinta com um objetivo diferente de projeto.

Git e GitHub são conceitos e ferramentas utilizadas para versionamento de arquivos, mas existe outro ponto que essa ferramenta traz consigo, o valor que pode agregar ao trabalho dos desenvolvedores. Uma maneira de centralizar seus projetos e atividades, sendo assim, podendo efetuar versionamento de arquivos contendo versões. GitHub mostra ser uma ferramenta flexível para qualquer necessidade. Não apenas como uma ferramenta de versionamento, GitHub consegue trazer consigo diversas ferramentas como montar um Kanban para tarefas que devam ser desenvolvidas, seja de projeto ou até mesmo um trabalho de faculdade, podemos efetuar hospedagem de sites dentre outras ferramentas disponíveis no GitHub.

Uma imagem contendo computador

Descrição gerada automaticamente

Sumário

[**Dedicatória** 1](#_Toc49884799)

[**GitHub & Git** 3](#_Toc49884800)

[Git 3](#_Toc49884801)

[GitHub 4](#_Toc49884802)

[Git & GitHub 5](#_Toc49884803)

[**Requisitos** 6](#_Toc49884804)

[**GitHub e Conceitos** 11](#_Toc49884805)

[Conceito 12](#_Toc49884806)

[**Capítulo 1** 13](#_Toc49884807)

[Primeiro passo 13](#_Toc49884808)

[**Capítulo 2** 16](#_Toc49884809)

[Code 16](#_Toc49884810)

[Issues 17](#_Toc49884811)

[Pull requests 18](#_Toc49884812)

[Actions 19](#_Toc49884813)

[Projects 20](#_Toc49884814)

[Wiki 21](#_Toc49884815)

[Security 22](#_Toc49884816)

[Insights 23](#_Toc49884817)

[Settings 24](#_Toc49884818)

[**Capítulo 3** 25](#_Toc49884819)

[Code 25](#_Toc49884820)

[Iniciando seu repositório 26](#_Toc49884821)

[Git Bash 27](#_Toc49884822)

[**Capítulo 4** 34](#_Toc49884823)

[Criando Branch 34](#_Toc49884824)

[Atualizando Branch 36](#_Toc49884825)

[**Capítulo 5** 40](#_Toc49884826)

[**Capítulo 6** 47](#_Toc49884827)

# **GitHub & Git**

Para iniciarmos nosso treinamento na ferramenta GitHub 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)

Uma imagem contendo computador

Descrição gerada automaticamenteDesde 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.

Uma imagem contendo computador

Descrição gerada automaticamente

## Git & GitHub

A partir deste momento iremos dar início ao curso com foco no GitHub, nosso objetivo é de ensiná-lo há como utilizar o Git e GitHub da melhor forma possível agregando valor a ferramenta. Lembre-se que uma ferramenta bem desenvolvida consiste em dois conceitos.

* Custo
* Valor

Quando nos referimos a custo, toda ferramenta tem o seu capital que precisa ser injetado/aplicado para ser desenvolvido. Até mesmo para usufruir de uma ferramenta existente precisamos aplicar um capital.  
Mas quando falamos de valor estamos referindo-se o que aquela ferramenta irá agregar no seu dia a dia, em um trabalho, projeto ou até mesmo para usá-lo de maneira casual (Guardar documentação/Servidor de game).

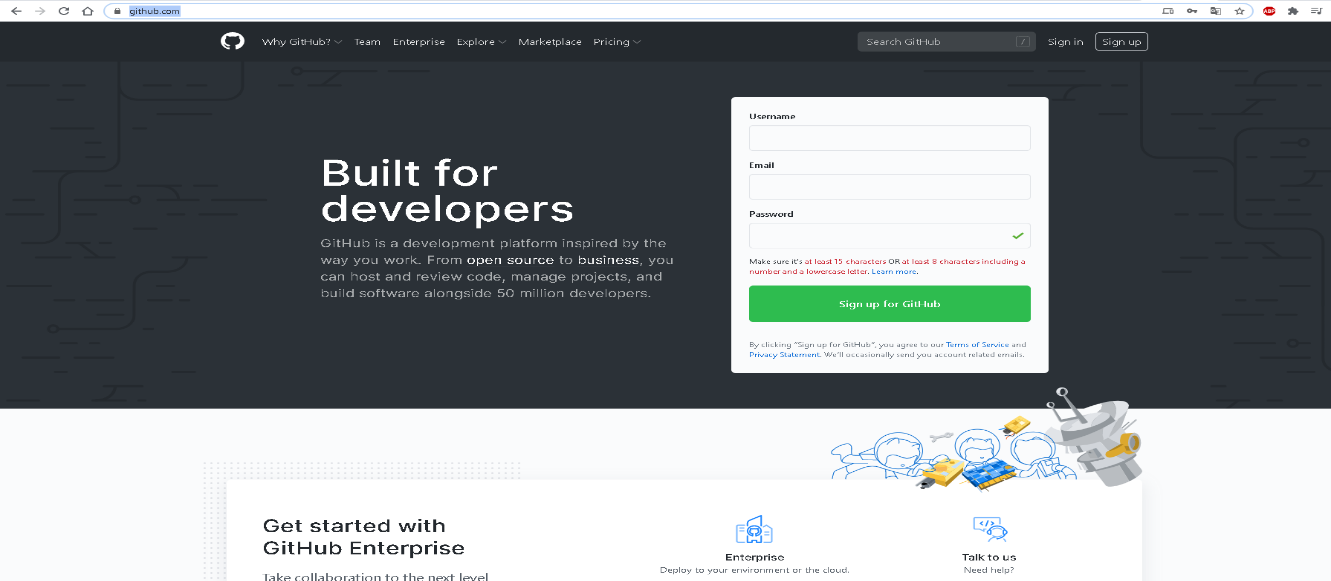
Gostaria de deixar claro que podemos usar ferramentas para nosso dia a dia de forma inimagináveis então ao concluir este curso espero que consiga aproveitar essa ferramenta e usá-la da melhor forma possível.... Conte-me depois como foi sua experiência.

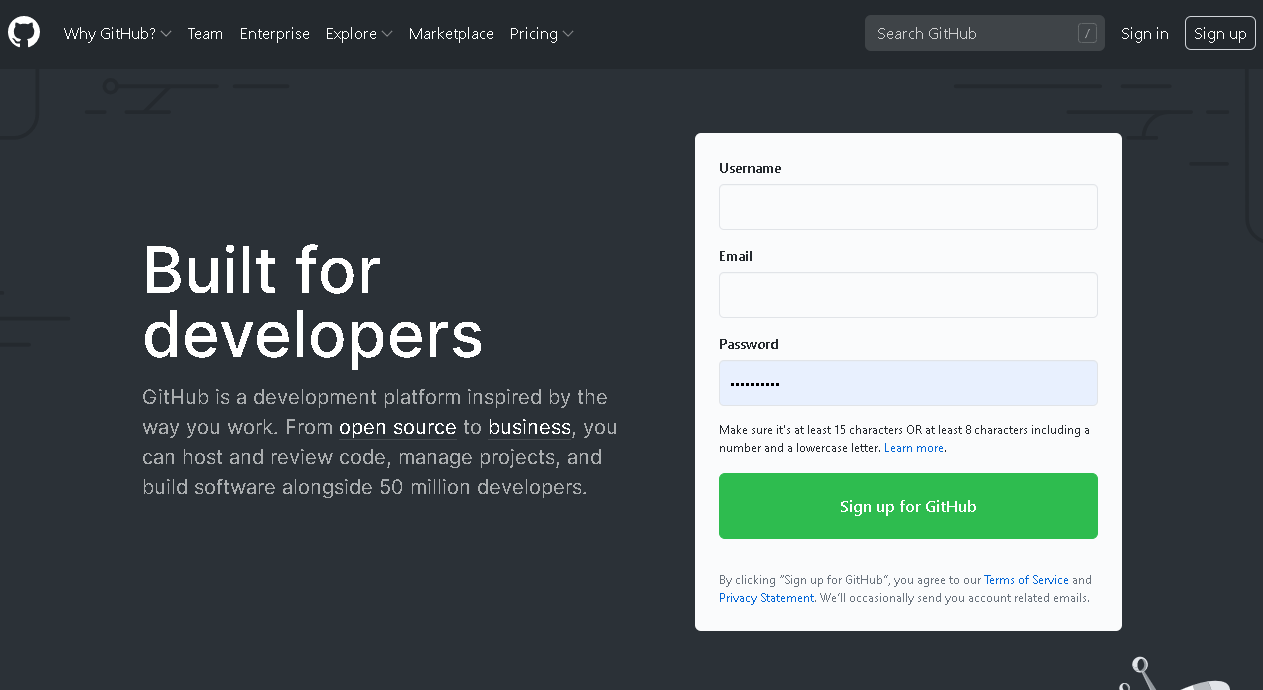
Uma imagem contendo computador

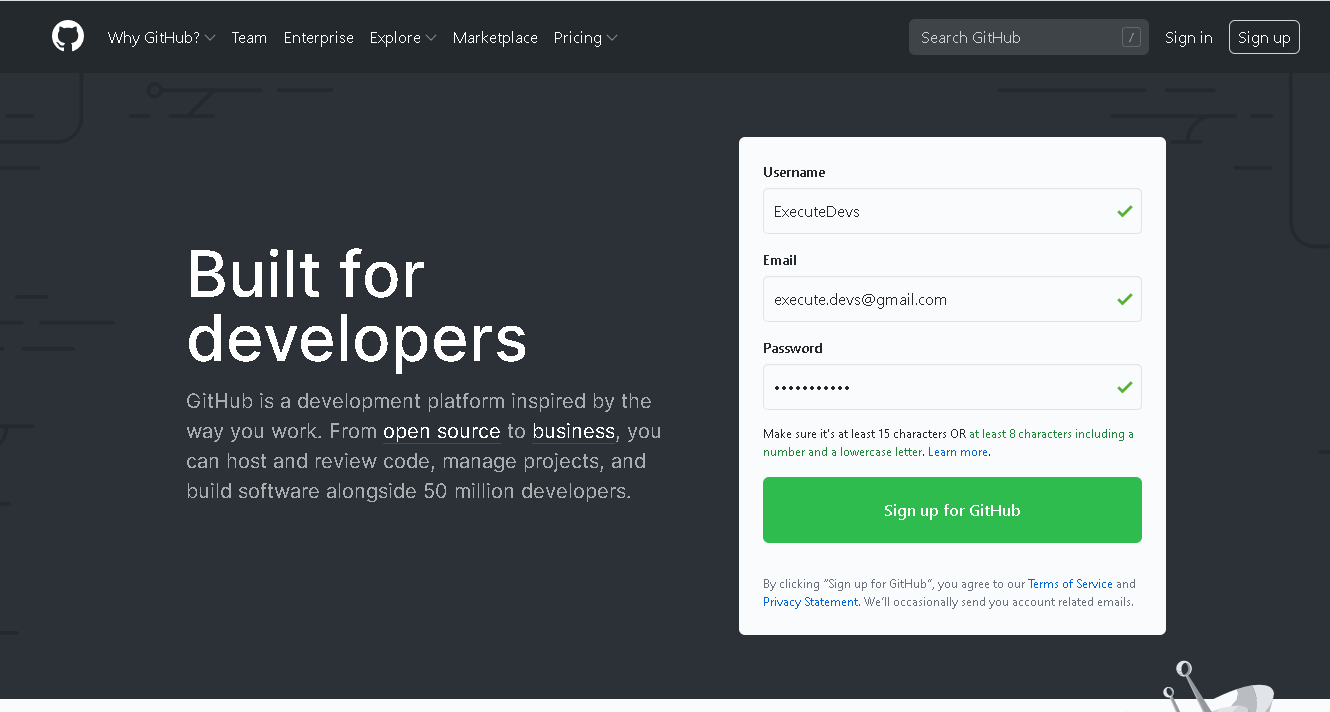
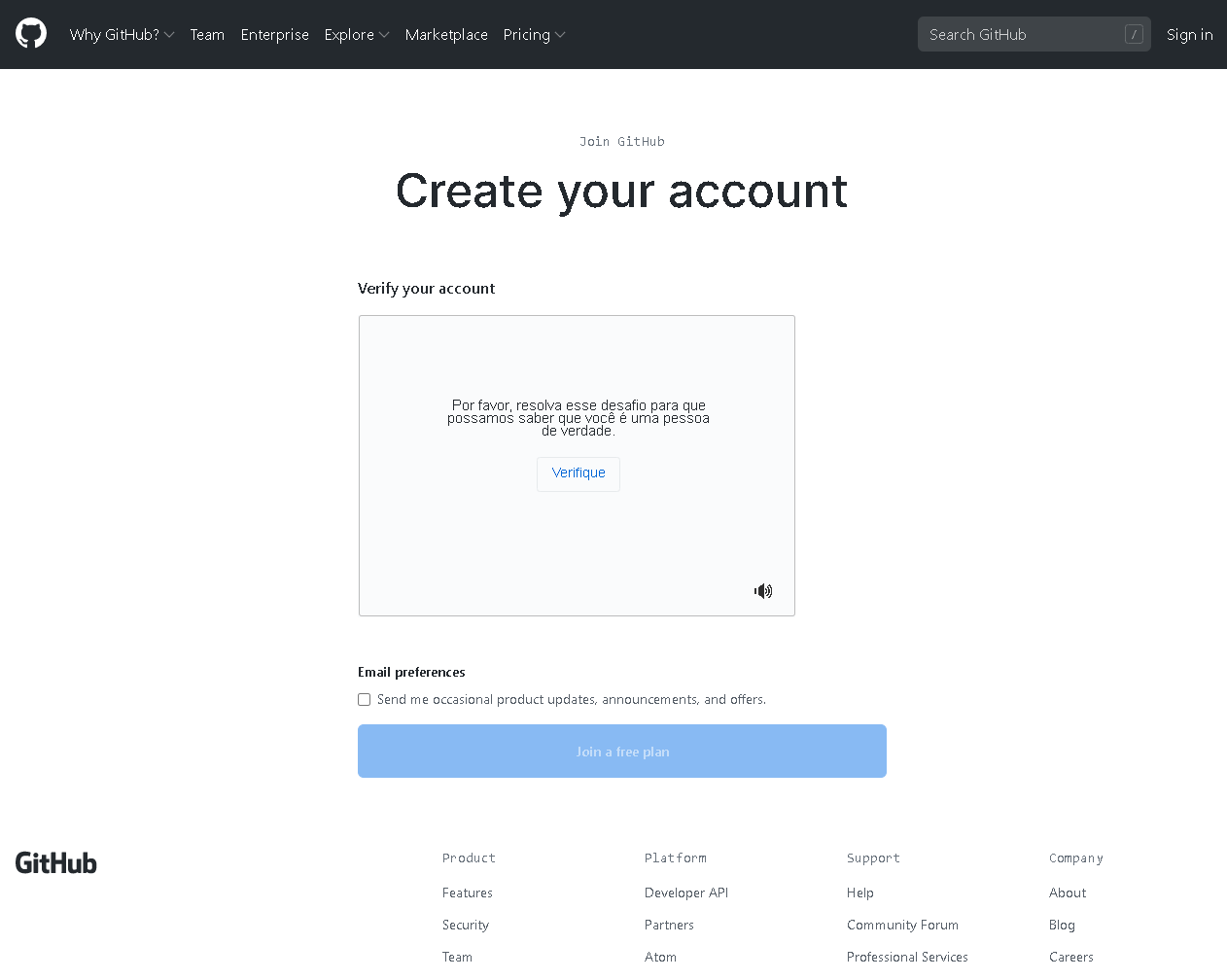
Descrição gerada automaticamente

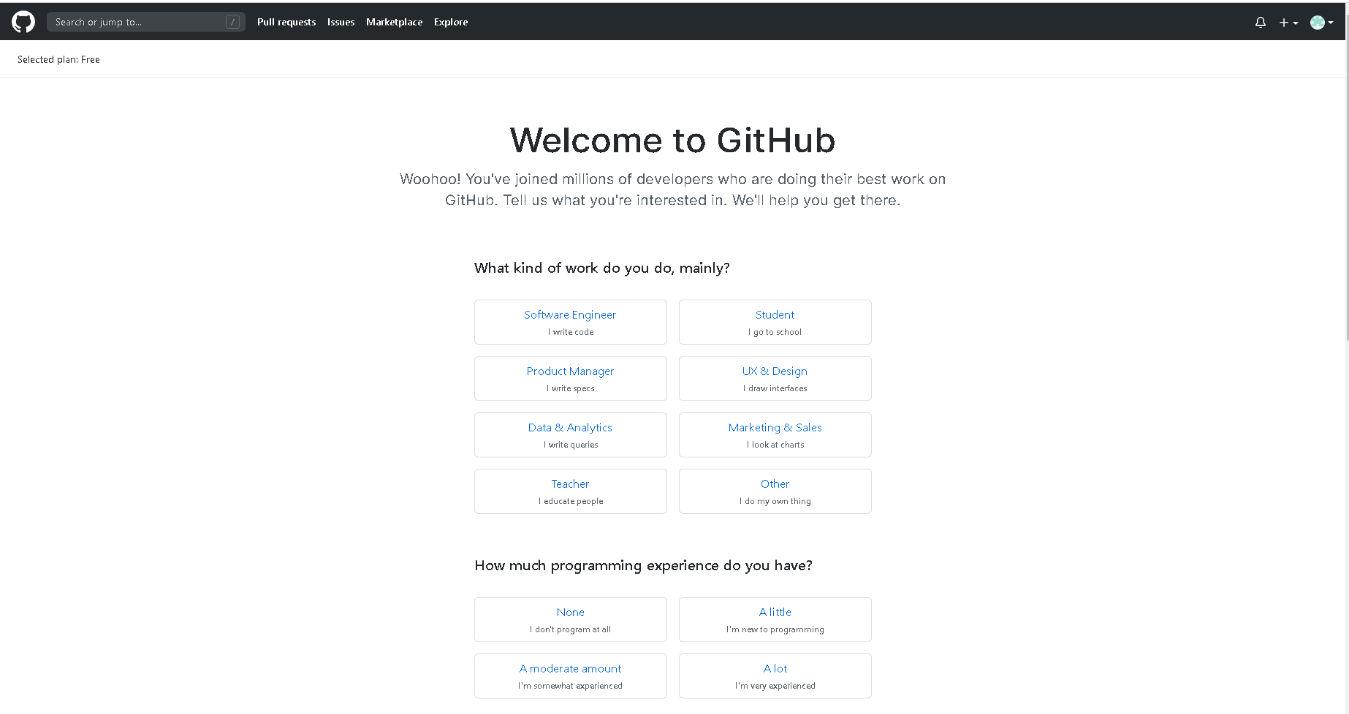
# **Requisitos**

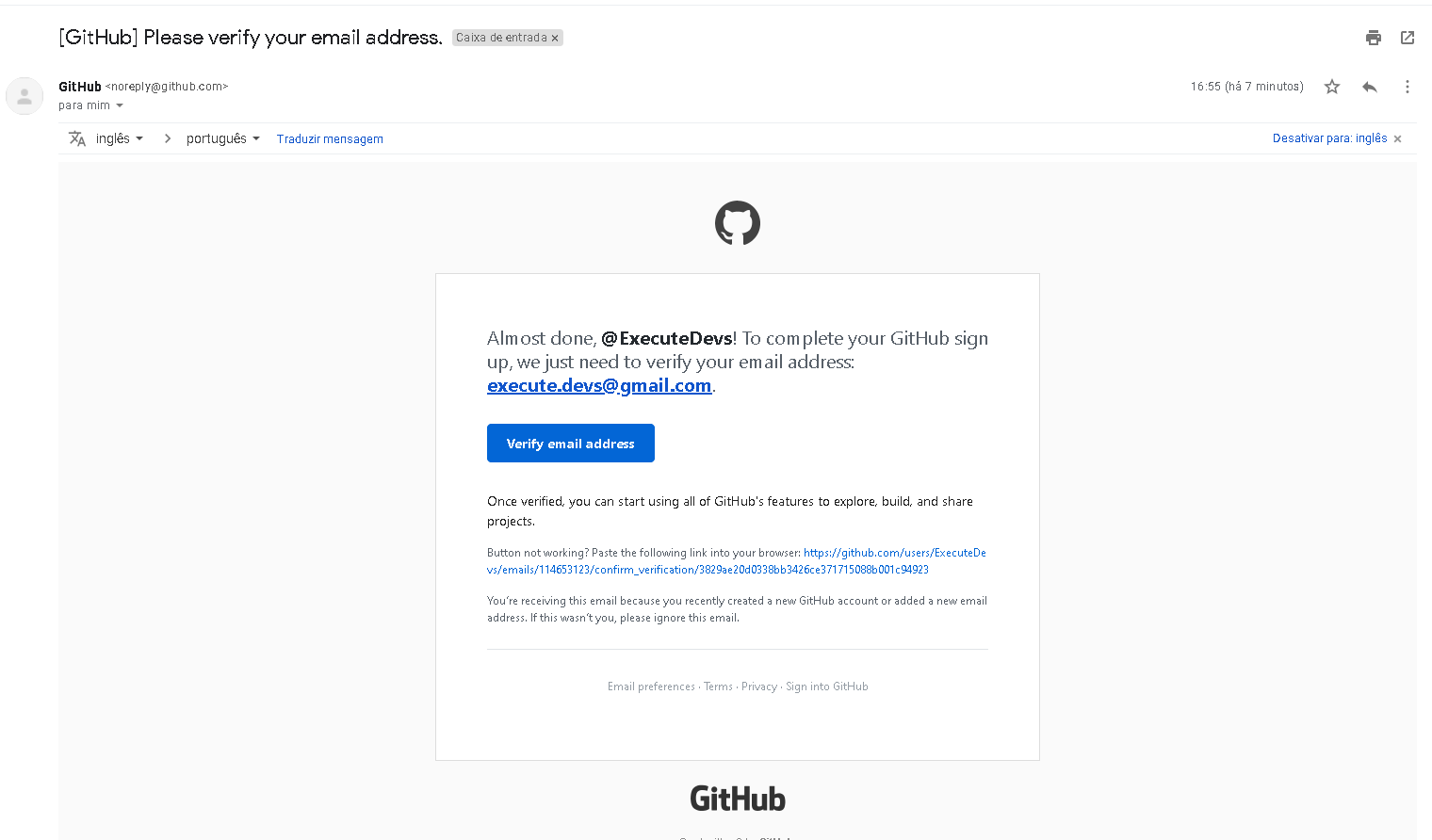
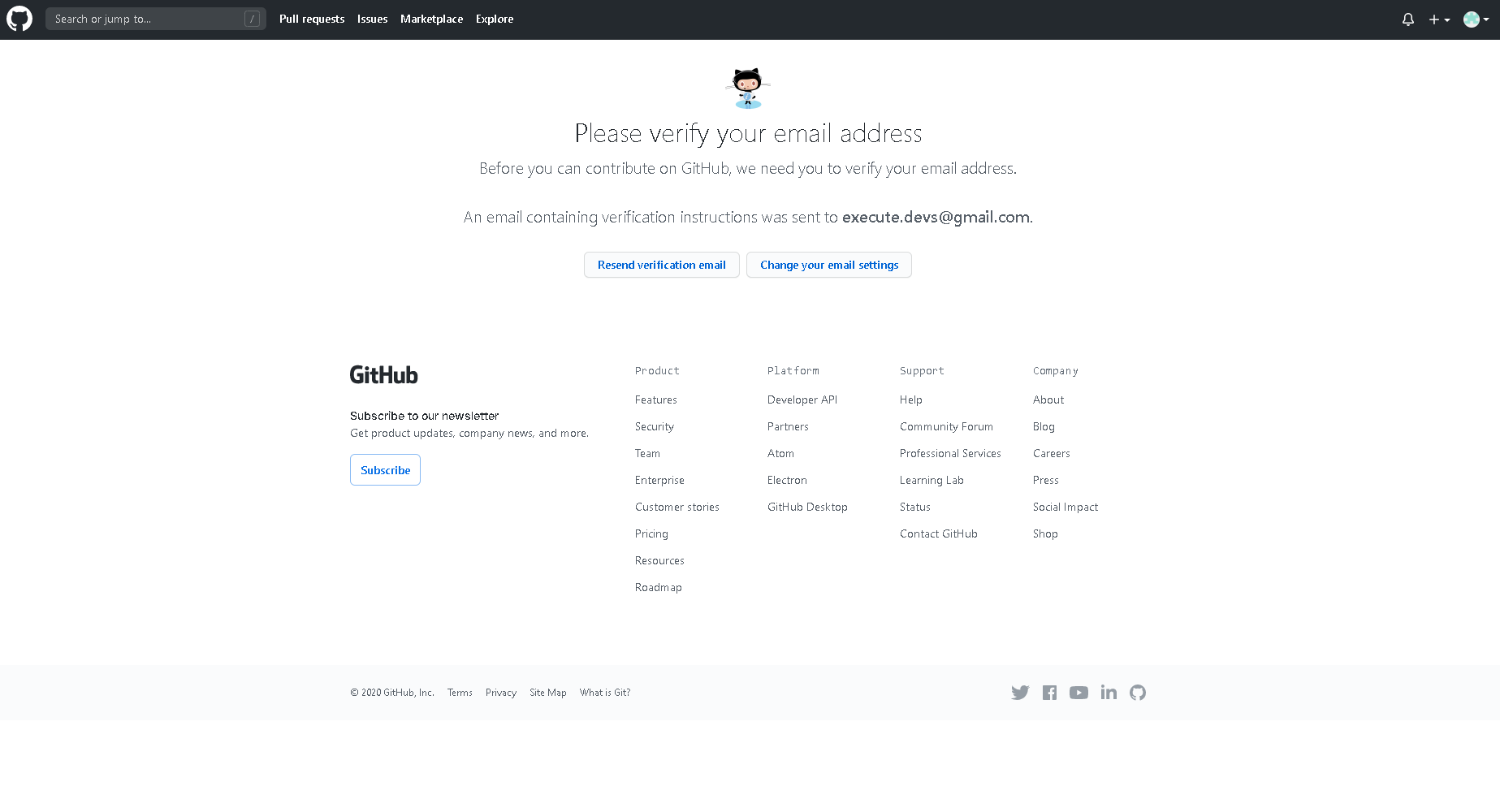
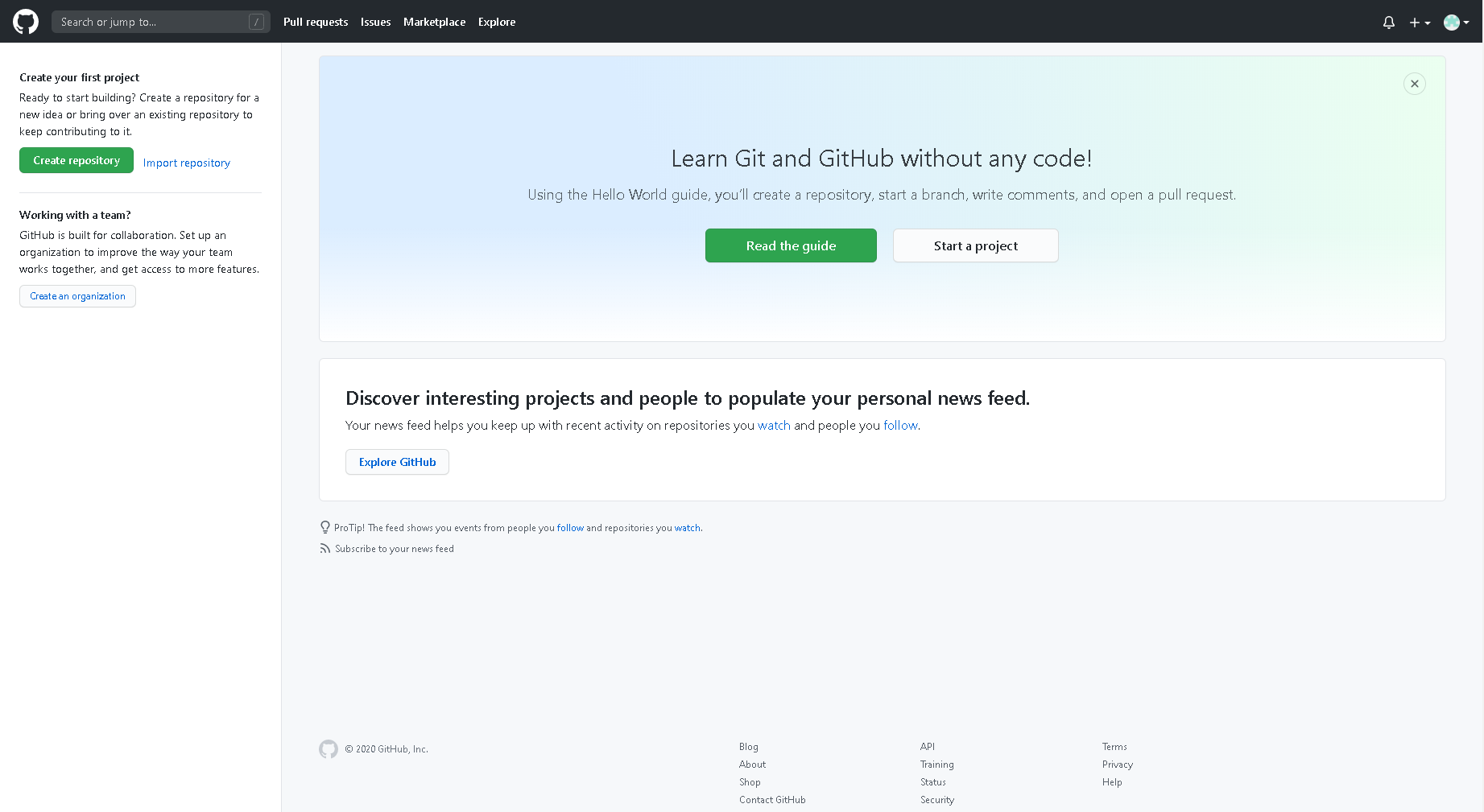
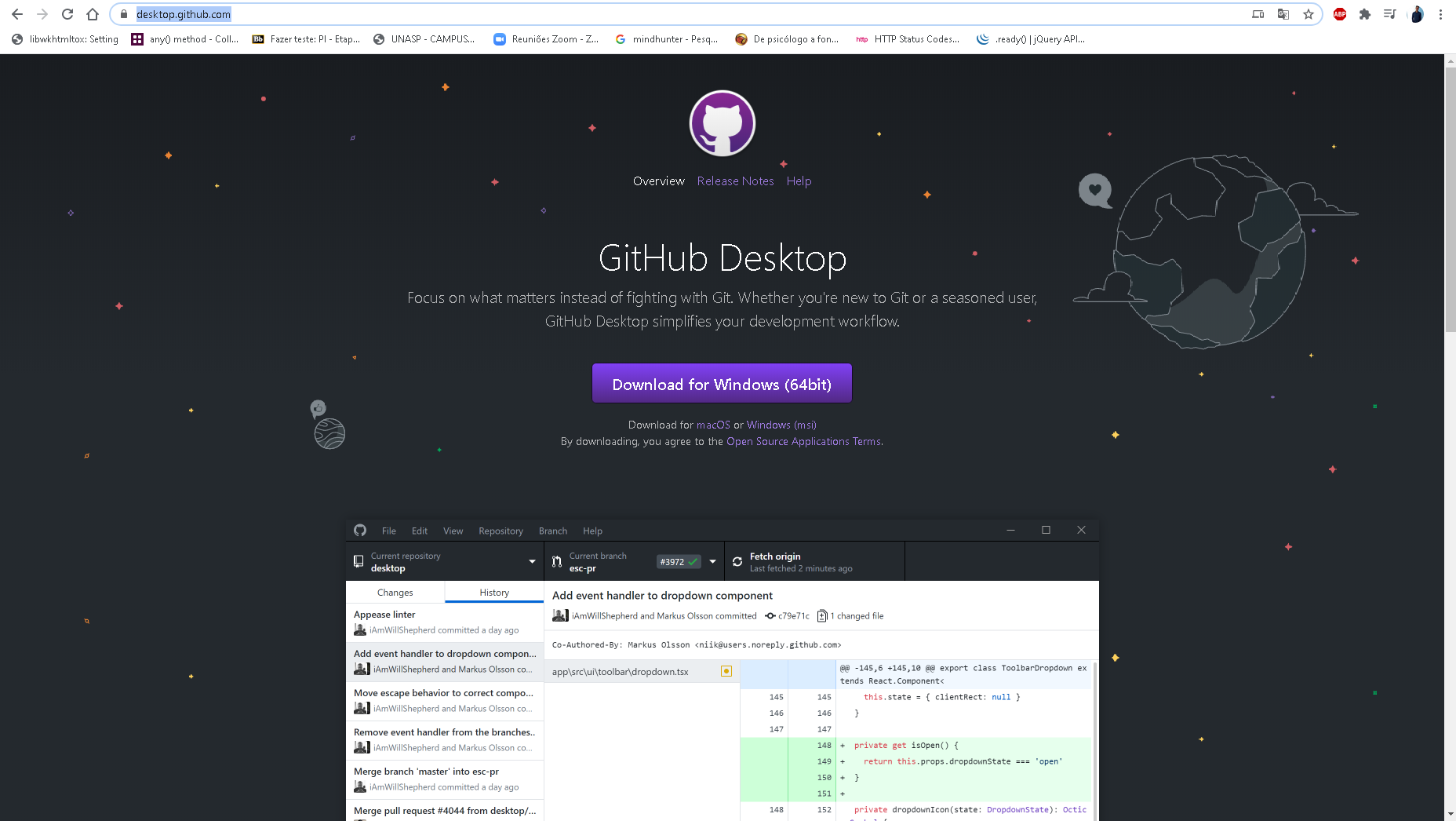
Para poder utilizar a ferramenta primeiramente vamos precisar criar uma conta, siga os passos a passos e crie sua conta.

1. Acesse o site: <https://github.com/>
2. Uma imagem contendo computador

   Descrição gerada automaticamenteAo entrar na página irá se deparar com o 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. Uma imagem contendo computador

   Descrição gerada automaticamenteConfirmação 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. Uma imagem contendo computador

   Descrição gerada automaticamenteÚltimos 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. Uma imagem contendo computador

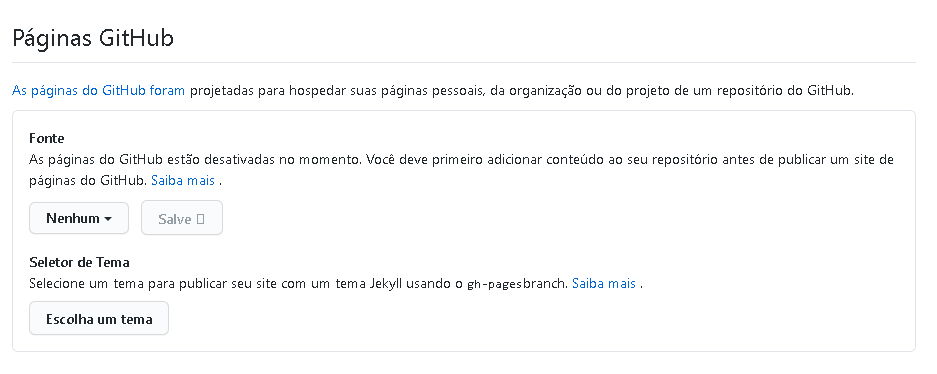
   Descrição gerada automaticamentePor último o GitHub irá precisar que você confirme em seu e-mail a conta para utilizar da melhor forma, cheque o e-mail cadastrado.
8. Agora com sua conta criada podemos partir para   
   
9. Baixe o GitHub para utilização no DeskTop  
   Link: <https://desktop.github.com/>
10. Faça a instalação de forma convencional

Uma imagem contendo computador

Descrição gerada automaticamente

# **GitHub e Conceitos**

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 serviços para oferecer, 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, para outras pessoas acessarem e analisem o seu grau de conhecimento a respeito de Web.

Uma imagem contendo computador

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

## Conceito

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.

Uma imagem contendo computador

Descrição gerada automaticamenteGitHub 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 mais atualizado. 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 desenvolvedores acabaram por ventura modificando o código aonde outro desenvolvedor está mexendo, ou seja, podemos acabar quebrando a aplicação, 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.

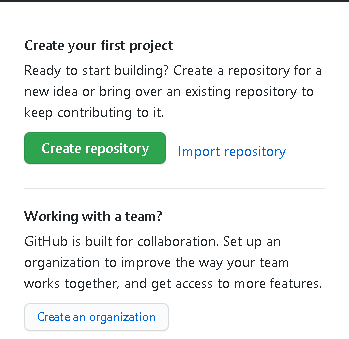
# **Capítulo 1**

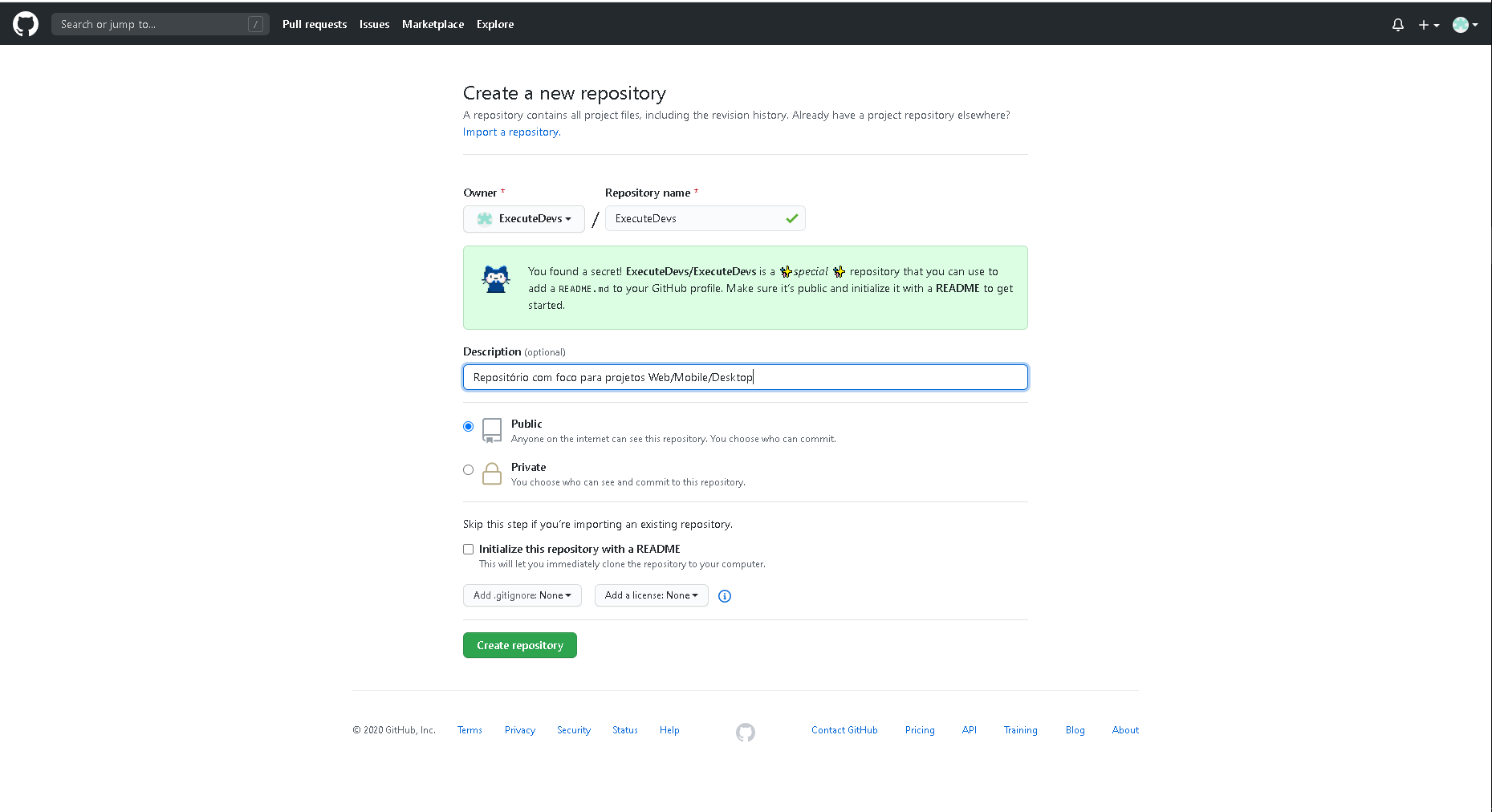
Criando seu primeiro Repositório

## 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**.

Uma imagem contendo computador

Descrição gerada automaticamente

Na etapa a seguir você irá criar e configurar seu repositório pela primeira vez, após a imagem irei descrever cada função e etapa.

Opções:

* **Owner**  
  Owner é o proprietário do repositório, na maioria dos casos quem está logado é o proprietário.
* **Repository name**  
  Em repositor name você irá colocar o nome do seu repositório que achar mais adequeado (ExecuteDevs ou ServidorJogo).
* **Description**  
  Description é uma pequena descrição para você e que outras pessoas saibam qual a finalidade deste repositório.
* Uma imagem contendo computador

  Descrição gerada automaticamente**Public / Private**  
  Neste local você irá configurar seu repositório para ele ser público 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 podem manusear estes arquivos.  
  Mas se o seu intuito 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 público, mas lembrando que se optar em deixar este repositório público qualquer pessoas poderá manipular os arquivos que você colocou. Mas futuramente você poderá alterar essa configuração também.
* **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 criou um arquivo de texto, com uma cola sua dos comandos de GitHub, você pode colocar o nome do arquivo no Gitgnore aonde ele irá sempre atualizar o repositório mas retirando este arquivo que você criou.
* **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**

Uma imagem contendo computador

Descrição gerada automaticamente

# **Capítulo 2**

Configurando seu Repositório

Neste capítulo vamos explorar melhor a ferramenta em sua parte web para que você possa entender o que pode ser contruído, configurado ou até mesmo 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, mas o GitHub explica para você como pode iniciar seu primeiro versionamento. Mas iremos abordar o versionamento no capítulo 3.

Uma imagem contendo computador

Descrição gerada automaticamente

## Issues

Nesta aba de **Issues** você consegue monitorar o que está ocorrendo em seu projeto, em relação a requisições, conflitos, versões etc.  
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 .

Uma imagem contendo computador

Descrição gerada automaticamente

## 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 árvore 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(**Pull**) ou rejeitar esse envio(**Push**) realizado pelo desenvolvedor.

Uma imagem contendo computador

Descrição gerada automaticamente

## 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

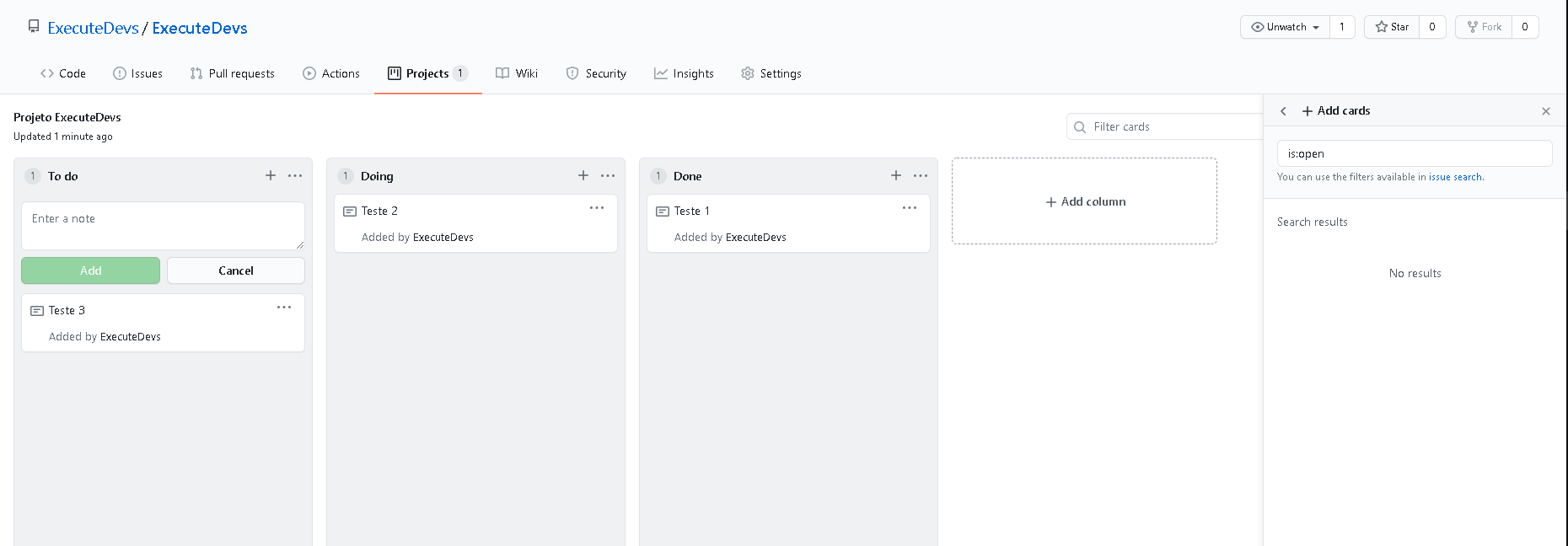
Pense da seguinte forma, você tem uma aplicação em JSP ou .Net aonde está efetuando o versionamento dela no seu repositório, diremos que você precise efetuar a publicação deste projeto agora na **AWS**, dentro de **Actions** você consegue efetuar o deploy desta aplicação na **AWS** diretamente do seu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

Uma imagem contendo computador

Descrição gerada automaticamente

## 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.

Uma imagem contendo computador

Descrição gerada automaticamente

## Wiki

Pensando mais adiante e explorando temos a aba Wiki, sua função pode ser definida de uma forma bem prática e 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.

Uma imagem contendo computador

Descrição gerada automaticamente

## 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 políticas de segurança e visualizar até mesmo falhas e visualizar se a segurança do seu projeto foi violada.

Uma imagem contendo computador

Descrição gerada automaticamente

## 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.

Uma imagem contendo computador

Descrição gerada automaticamente

## Settings

Por último mas não menos importante encontramos na última aba Settings tudo em relação a um repositório ou seja, podemos configurar ele de todas formas, alterando de privado para público e público 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.

Uma imagem contendo computador

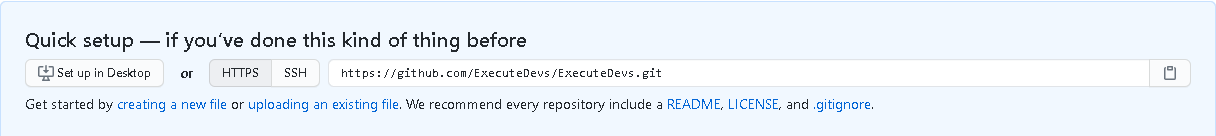
Descrição gerada automaticamente

# **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/arquivo que deseja fazer o versionamento em sua máquina. Após decidir qual projeto deseja versionar selecione um repositório clicando no nome do projeto.

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

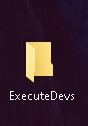
Podemos acessar o repositório de duas formas:

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

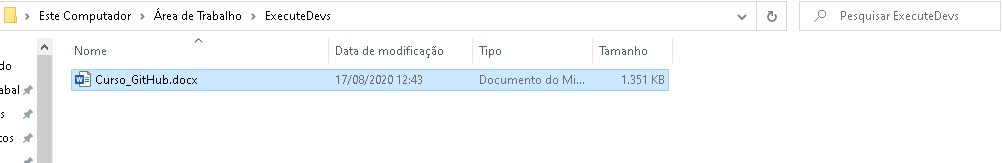
Uma imagem contendo computador

Descrição gerada automaticamenteLogo 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.

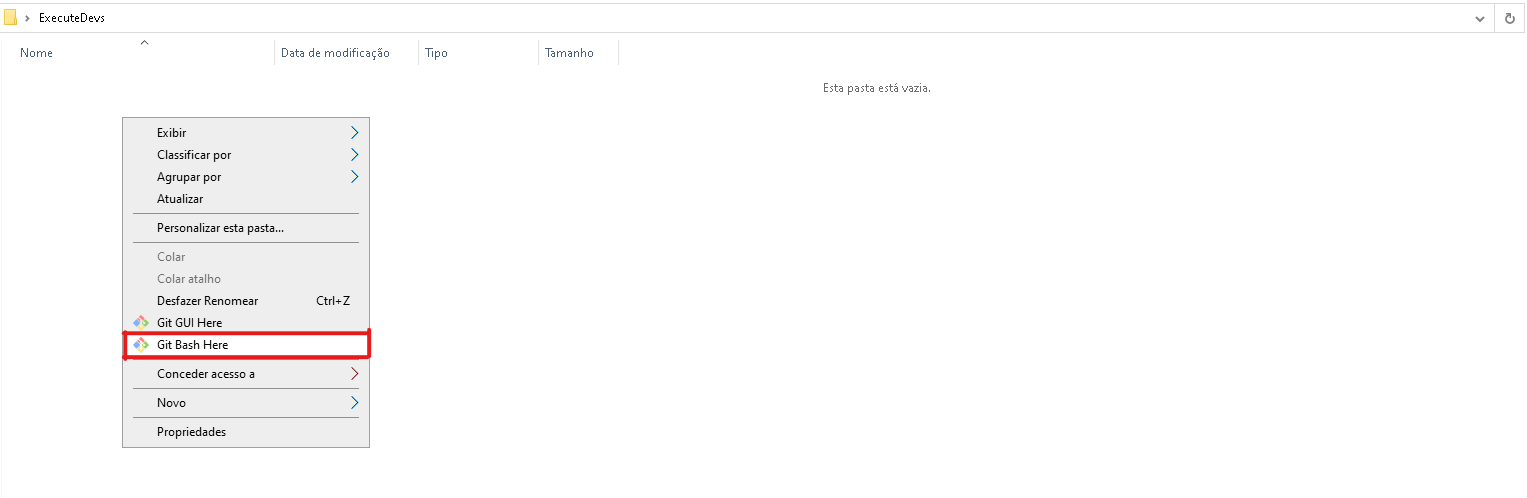
## Iniciando seu repositório



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.



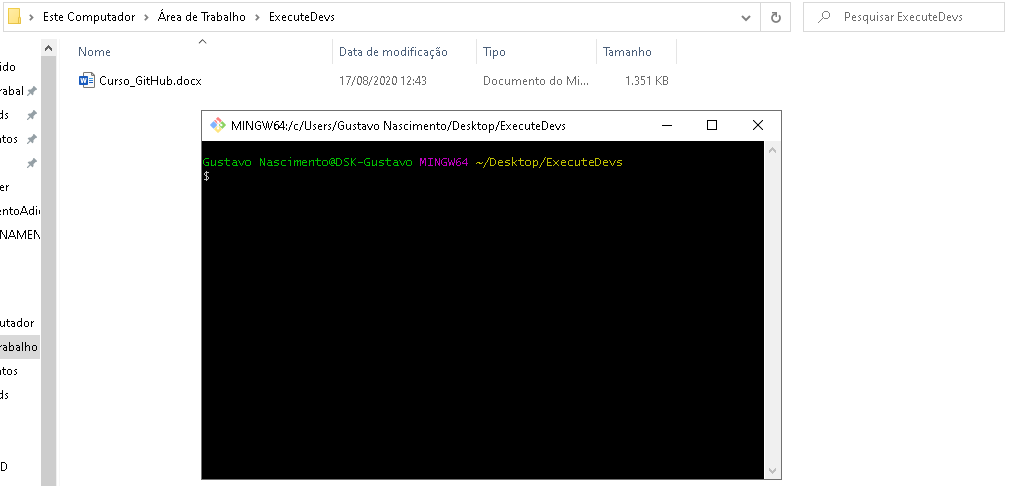
Em seguida clique com o botão direito em qualquer lugar da pasta aonde esteja vazio.

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 **Requisitos**.

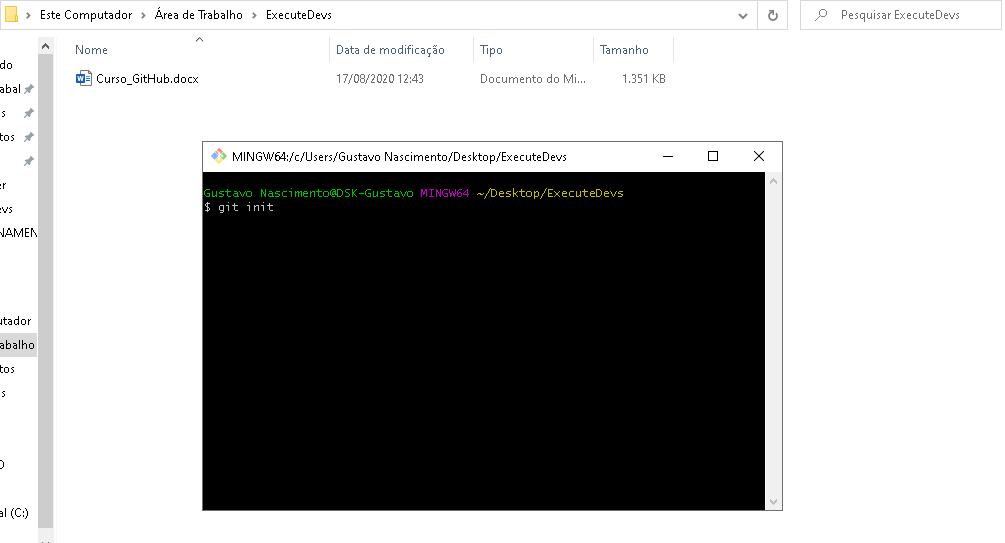
Uma imagem contendo computador

Descrição gerada automaticamente

## Git Bash

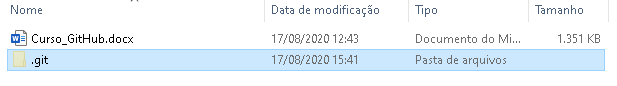


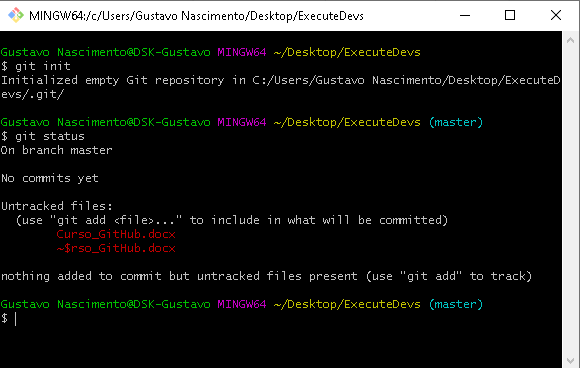
Após abrir o prompt de comando do GitHub vamos efetuar todo o versionamento via linha de comando, o GitHub tem sua parte gráfica para efetuarmos o versionamento, mas neste momento vamos lhe mostrar como utilizar via linha de comando, muitas empresas acabam optando efetuar via linha de comando o versionamento, mas não se engane, efetuar versionamento pela parte gráfica(GUI) também não contem diferença alguma.



Uma imagem contendo computador

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

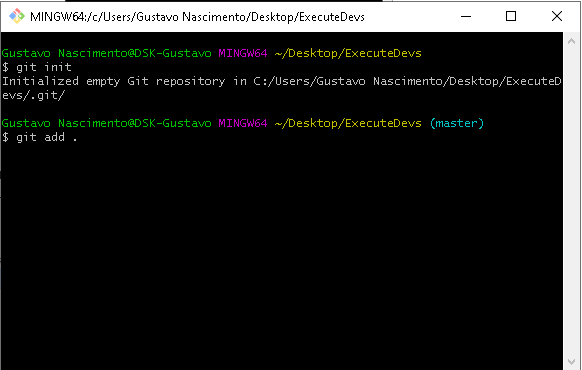




Em seguida logo após ele ter criado as configurações padrões, vamos verificar quais arquivos estão constando como alterados no seu repositório, basta executar o comando **git status**.   
Os Arquivos em vermelho são arquivos novos que ainda não existem no diretório, caso esses arquivos já existam, ele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

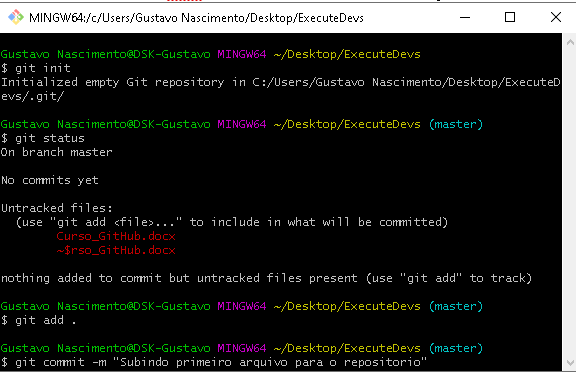
Uma imagem contendo computador

Descrição gerada automaticamenteObservação nem toda alteração significa que aquele arquivo já existe e sofreu alteração dentro dele, toda alteração é classificada da seguinte maneira:  
Se existir qualquer coisa de diferente em relação ao index (Ponteiro/Repositório) vai ser considerado uma alteração, seja adição, remoção ou alteração de arquivo.



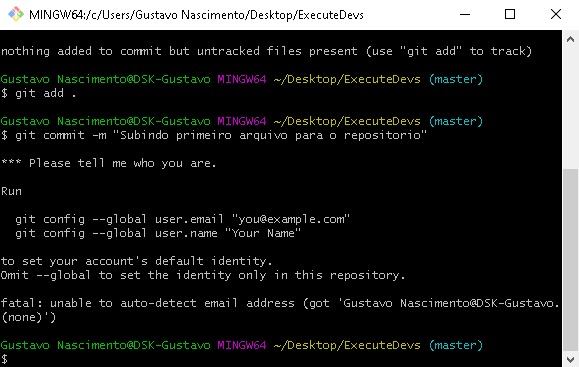
Uma imagem contendo computador

Descrição gerada automaticamenteLogo após ele ter adicionado à pasta do Git contendo suas configurações iniciais e verificar que existem arquivos pendentes para subir em seu repositório, será necessário adicionar outro comando **git add .** ou **git add all**, desta forma você irá indicar para o GitHub que deseja adicionar em um estado de subida todos arquivos acusados de diferentes 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.

Uma imagem contendo computador

Descrição gerada automaticamente

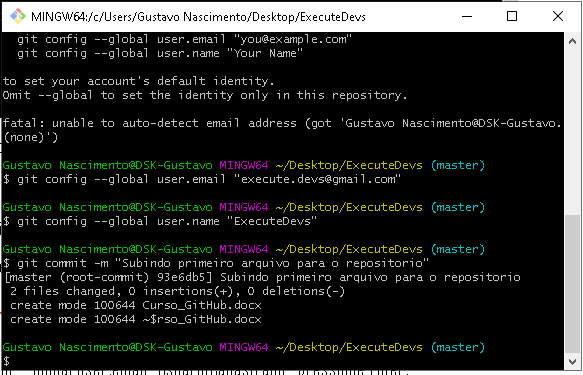
Mas agora vamos com calma, provavelmente você deve ter se deparado com uma tela pedindo para que você se identifique.  
Isso ocorre apenas a primeira vez que você está efetuando sua primeira subida via desktop.  
Vamos configurar seu usuário do GitHub Desktop com o usuário do GitHub que você efetua acesso na web, pegue 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.

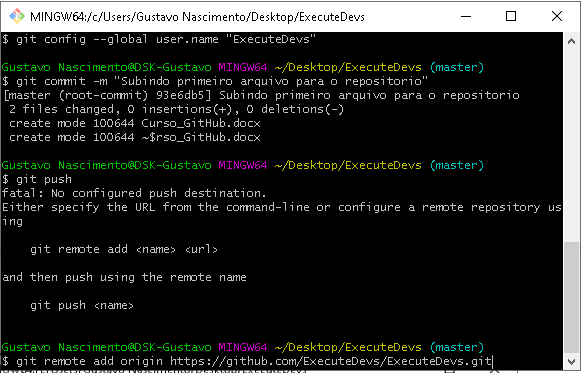


Seu resultado será semelhante a este.

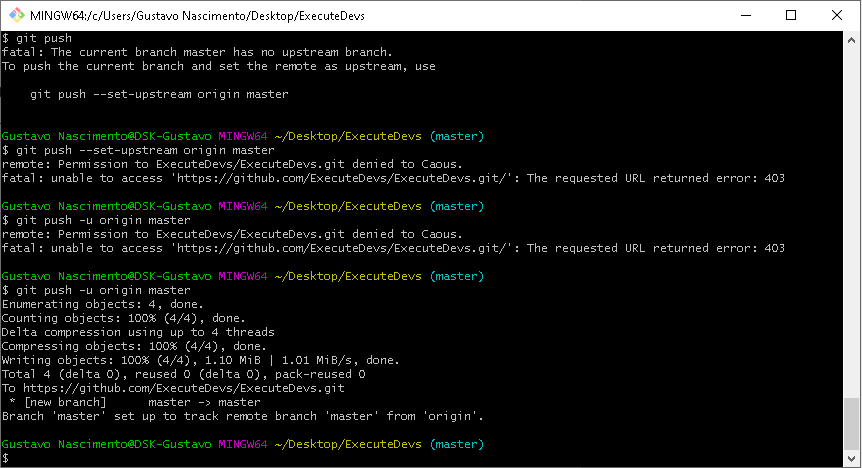
O GitHub está informando que 2 arquivos foram criados no commit efetuado.

Uma imagem contendo computador

Descrição gerada automaticamenteA 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.

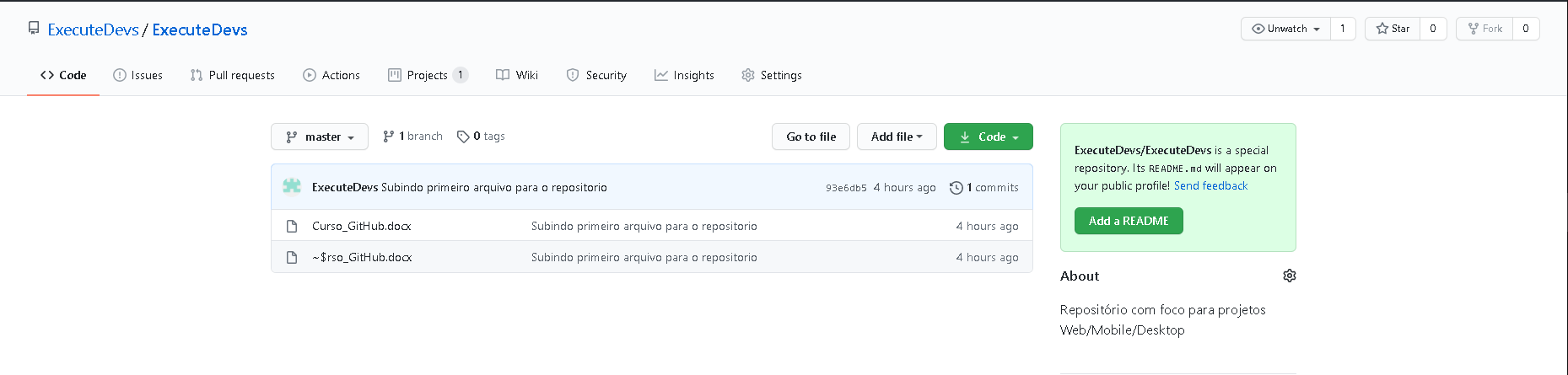


Vamos definir qual repositório desejamos que seja aplicado essas alterações consistentes. Execute o seguinte comando **git remote add origin “Link indicado na parte web no caso do ExecuteDev:" https://github.com/ExecuteDevs/ExecuteDevs.git**.



Uma imagem contendo computador

Descrição gerada automaticamenteAgora 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.

Uma imagem contendo computador

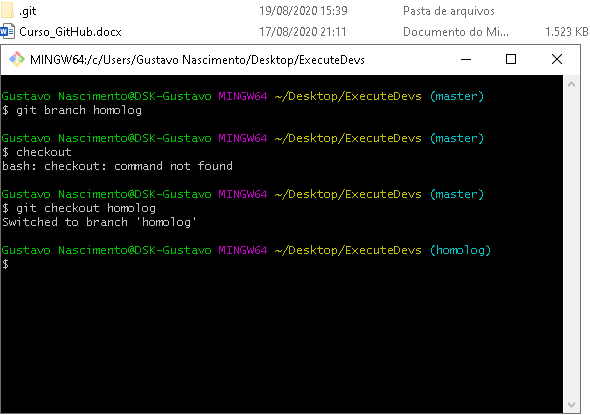
Descrição gerada automaticamente

# **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**.

Uma imagem contendo computador

Descrição gerada automaticamenteO 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 aquele desenvolvimento está de acordo com o solicitado.

O ambiente denominado **Cap\_4\_Adicao**, tem como objetivo criar um ambiente para o desenvolvimento da tarefa. **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, e está página estiver com qualquer erro, você poderá derrubar o seu site, mas também não podemos fazer diretamente em **Homolog**, porque é um ambiente para testes e apenas enviamos para a branch Homolog quando o desenvolvimento foi concluído.

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 e navegarmos entre elas vamos usar os seguintes comandos::

**git branch homolog**

**git checkout homolog**

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

Para disponibilizarmos a branch homolog, para que outros colaboradores possam, enxergar essa branch vamos utilizar o comando de push.

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

Desta forma vamos conseguir subir nossa branch homolog

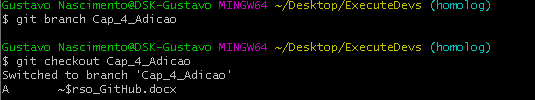
Repita o mesmo processo para criação da branch **Cap\_4\_Adicao**

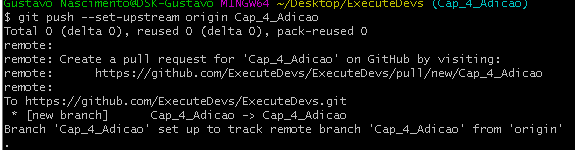
Uma imagem contendo computador

Descrição gerada automaticamente**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

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 manter as branch atualizadas e com o capítulo 4 recém criado.

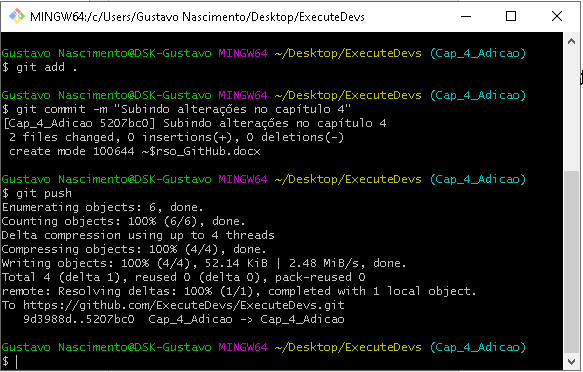
Uma imagem contendo computador

Descrição gerada automaticamente

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

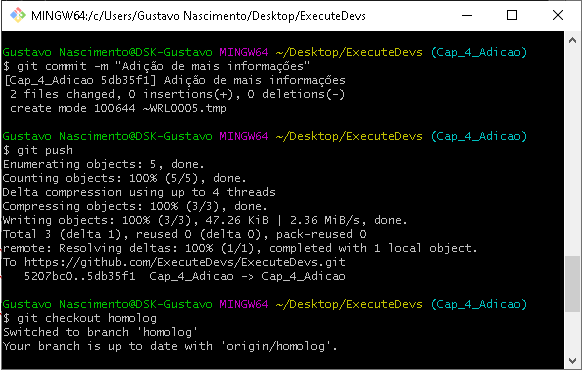
**git commit -m “Subindo alterações no capítulo 4”**

**git push**

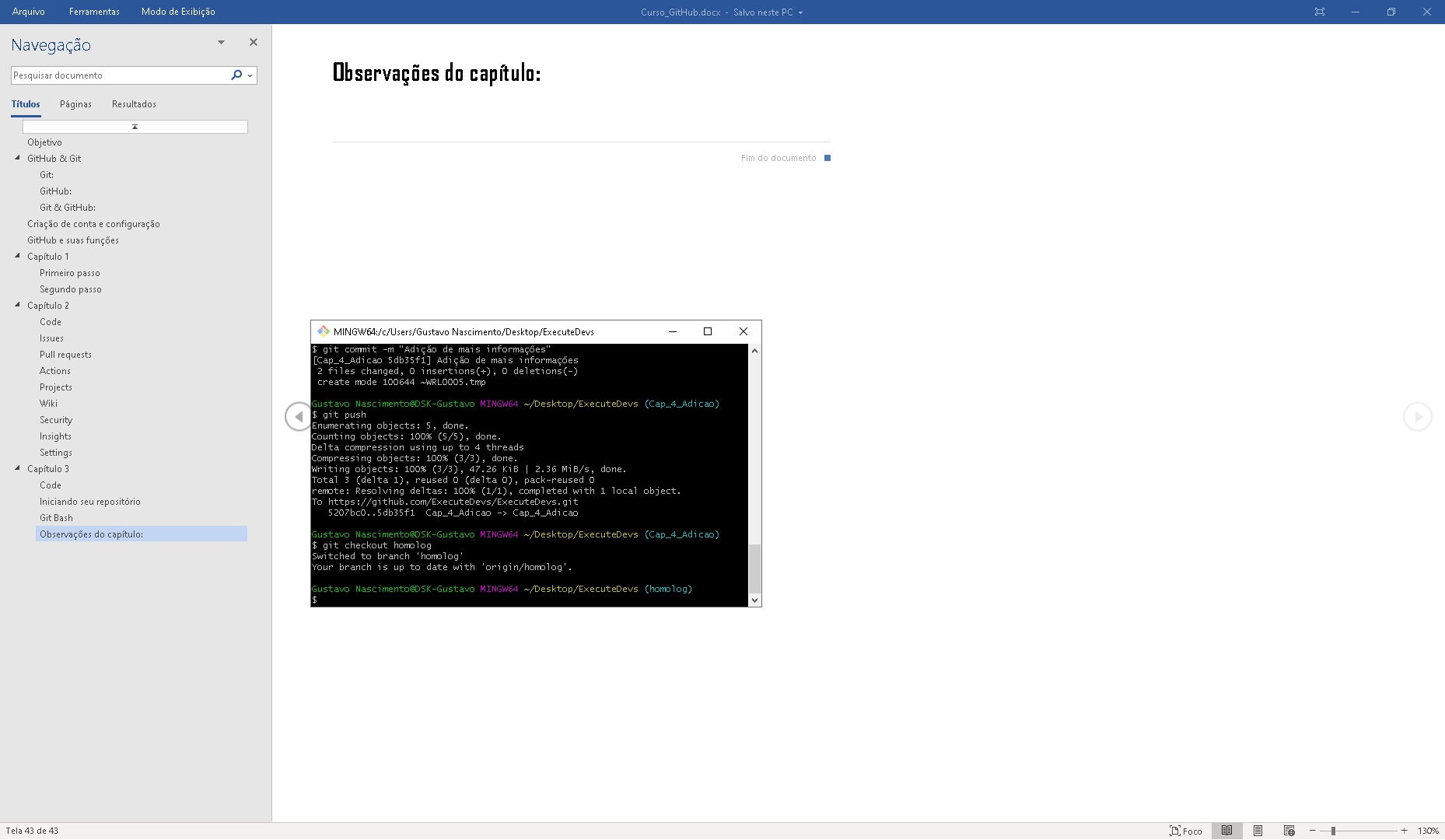
  
  
Após esses três comandos executados vamos alterar para a branch de **homolog** e verificar se essas alterações estão no arquivo da branch **homolog**. Digite:

Uma imagem contendo computador

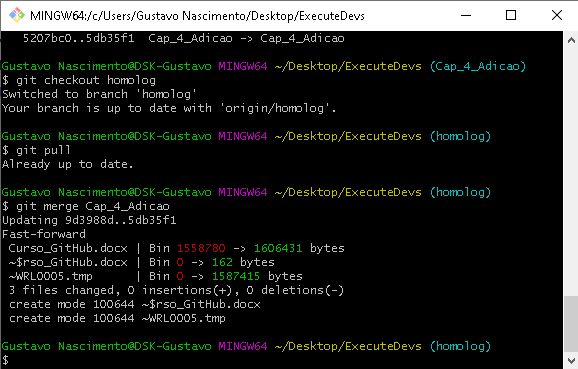
Descrição gerada automaticamente**git checkout homolog.**



Uma imagem contendo computador

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

Após verificamos, está totalmente correto não conter o capítulo 4, 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.

Comando para atualizar:

**git merge <Nome\_Branch>**

Uma imagem contendo computador

Descrição gerada automaticamente

# **Capítulo 5**

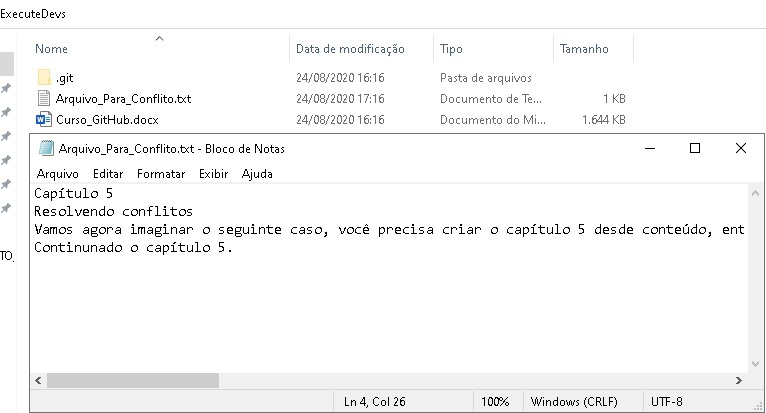
Resolvendo conflitos

Vamos agora imaginar o seguinte caso, você precisa criar o capítulo 5, então você cria uma branch chamada de **Cap\_5\_Adicao**, sendo assim você 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.

Observação: Conflitos são comuns no dia a dia de desenvolvedores que trabalham com equipe grandes, mas é necessário tomar muito cuidado, as vezes um conflito, mas resolvido pode resultar em perda de códigos, textos etc.

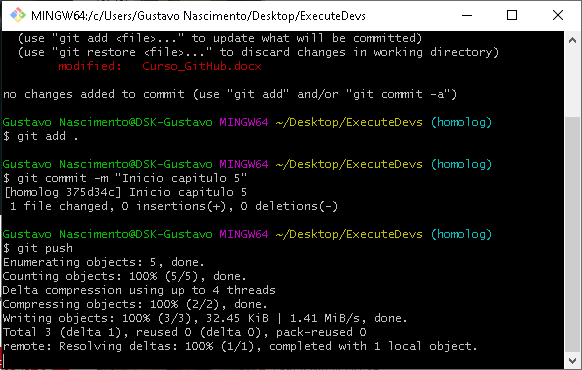
Neste caso vamos criar um arquivo separado para não corrermos risco de perder qualquer coisa no documento deste curso, vamos criar um arquivo chamado: **Arquivo\_Para\_Conflitos.txt.**

Neste arquivo vamos colocar um trecho do capítulo 5. Em seguida vamos trocar para a branch **Cap\_5\_Adicao** para alterarmos uma das linhas do trecho no arquivo.

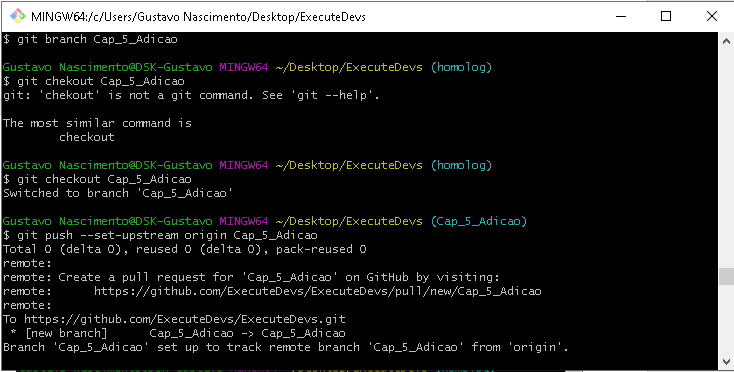


Uma imagem contendo computador

Descrição gerada automaticamenteApós adicionarmos o arquivo na branch Homolog vamos atualizar a branch Homolog e criar nossa branch Cap\_5\_Adicao contendo este novo arquivo.

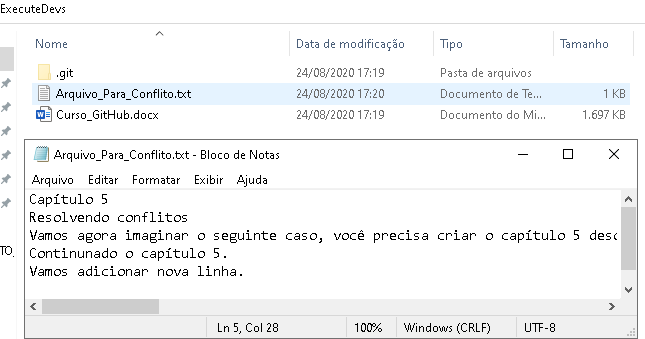


Após atualizarmos a branch **Homolog** vamos criar a branch **Cap\_5\_Adicao**

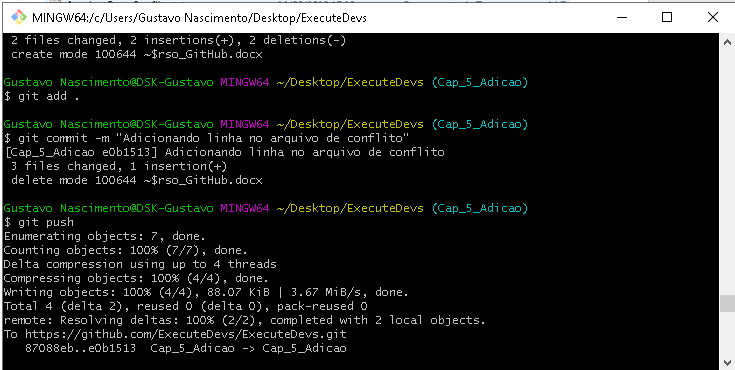


Uma imagem contendo computador

Descrição gerada automaticamenteEm seguida na branch **Cap\_5\_Adicao** vamos adicionar uma nova linha ao arquivo, reparem que a linha adicionada é de número 5.

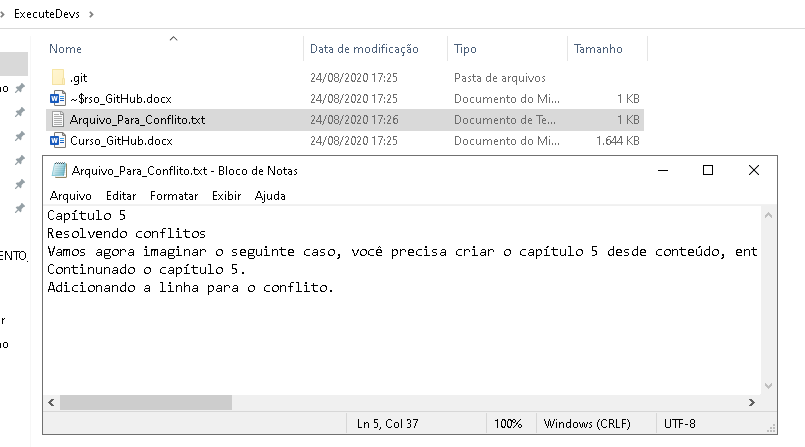


Agora vamos efetuar **commit** e **push**, mas não vamos atualizar a branch **Homolog** utilizando o comando merge.

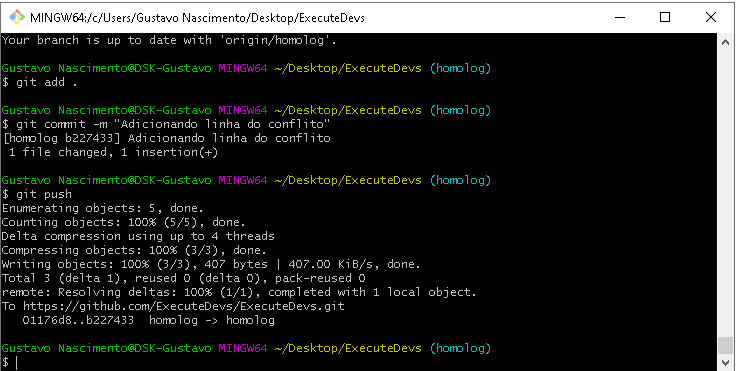


Uma imagem contendo computador

Descrição gerada automaticamenteAo invés de atualizarmos a branch Homolog com o comando merge, vamos trocar para a branch Homolog, nosso arquivo não irá conter a linha “**Vamos adicionar nova linha**”, então vamos adicionar outro texto na linha 5. Neste caso: “**Adicionando a linha para o conflito**”

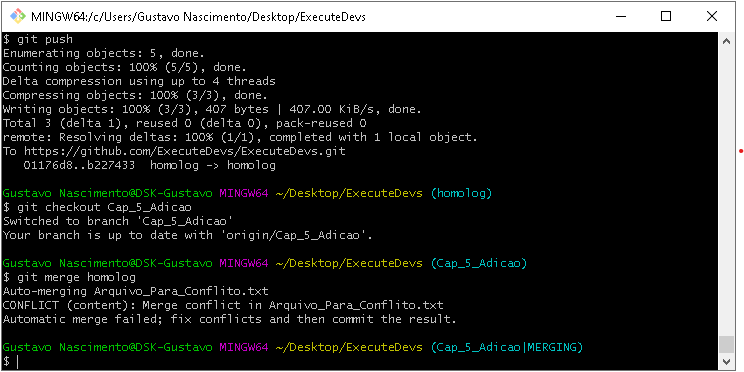


Após adicionarmos a linha para conflito, vamos apenas efetuar o **commit** e **push** em seguida alterar para branch **Cap\_5\_Adicao**.



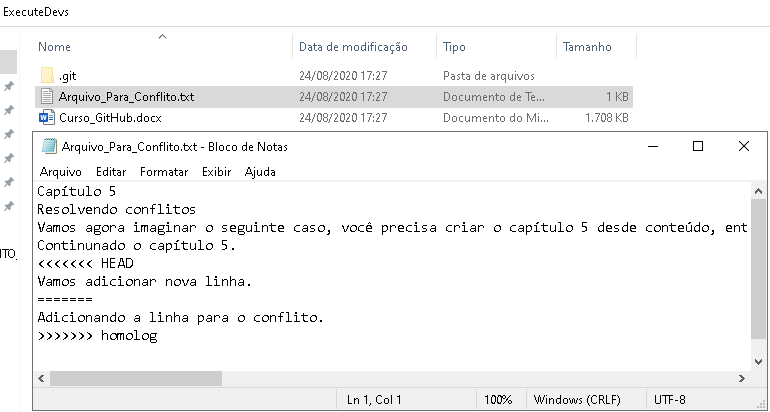
Uma imagem contendo computador

Descrição gerada automaticamenteApós a branch de **Homolog** estar atualizada vamos alterar para a branch **Cap\_5\_Adicao**, onde vamos efetuar um merge da **Homolog** para **Cap\_5\_Adicao**, normalmente efetuamos o inverso, mas desta vez vamos resolver o conflito e normalmente indica-se resolver conflitos na ramificação (branch) diferente da **Master** e **Homolog**.



Efetuamos o merge utilizando **git merge homolog** desta forma vamos conseguir trazer as alterações existentes na Homolog para Cap\_5\_Adicao. Mas não vamos conseguir ter um merge com sucesso o git irá nos informar que houve uma falha para efetuar o merge, porque existe conflitos no arquivo, aonde ele encontrou um texto diferente no arquivo Arquivo\_Para\_Conflitos.txt na linha 5 de ambos.

Após o git avisar que houve conflito, vamos entrar no arquivo para verificarmos como esse conflito é identificado e em paralelo vamos resolver o conflito.



Uma imagem contendo computador

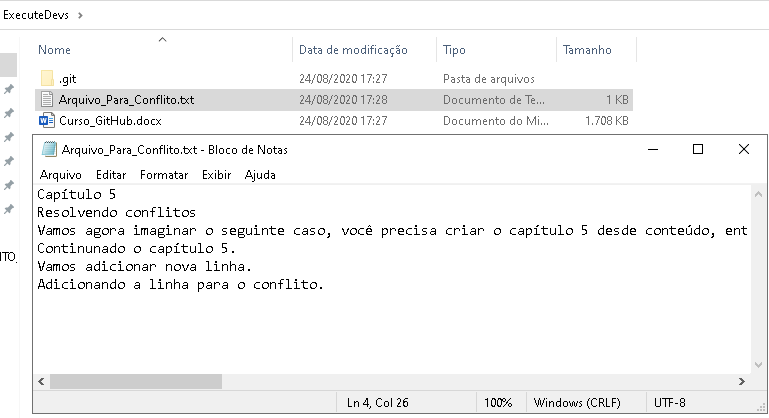
Descrição gerada automaticamente

Normalmente o Git demarcar a diferença de um arquivo para outro mostrando o Head (Aonde você está) e a branch que você solicitou, neste caso se analisarmos o **<<<<<<<< Head** está mostrando conteúdo que existe no seu arquivo: Vamos adicionar nova linha.

E separando com **=============** para mostrar abaixo a diferença: Adicionando a linha para o conflito.

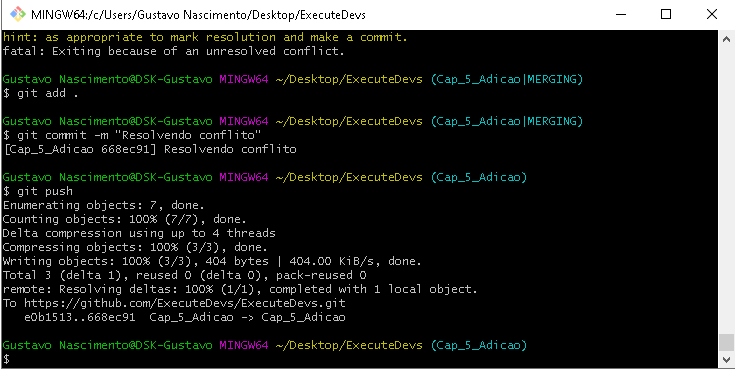
E identificando no final de onde vem essa alteração **>>>> Homolog**.

Para resolvermos o conflito vamos retirar a linha **<<< HEAD** juntamente os **==========** e **>>> homolog**.



Uma imagem contendo computador

Descrição gerada automaticamenteEntão vamos solucionar os conflitos mesclando as duas linhas, vamos deixar uma na linha 5 outra na linha 6.  
Após resolvermos conflitos vamos efetuar o commit e push.



Uma imagem contendo computador

Descrição gerada automaticamente

# **Capítulo 6**

Comandos

No capítulo 6 vamos conhecer a maior parte dos comandos existentes para serem utilizados no GitHub, mostrando suas funcionalidades e mostrando momentos que podem ser utilizados.

* Git init  
  git init é um comando que inicia um repositório vazio, onde cria a pasta oculta .git com alguns subdiretórios.
* Git push  
  git push é um comando que atualiza o repositório remoto com as alterações locais em sua máquina.
* Git pull  
  git pull é um comando utilizado para pegar todas alterações que estão na branch aonde você se encontra.
* Git checkout  
  git checkout é um comando para atualizar seus arquivos igual está no índice, ou seja, se temos uma branch que deriva da master ao efetuar git checkout todos arquivos vão estar atualizados de acordo com a master, mas note que ficará igual se você tiver clonado recentemente
* Git checkout <nome\_branch>  
  git checkout <nome\_da\_branch> utilizamos para navegar entre branchs
* Git add .  
  git add . é um comando que utilizamos atualiza o índice utilizando o conteúdo atual encontrado na árvore de trabalho para preparar o conteúdo para o próximo commit.

Uma imagem contendo computador

Descrição gerada automaticamente

* Git status  
  git status podemos verificar quais alterações encontram-se naquela branch que estamos
* Git log  
  git log utilizamos para verificar os commits efetuados dentro da branch em que se encontra.
* Git clone  
  git clone serve apenas quando temos um repositório já existente, a partir desta regra você pode obter este repositório em sua máquina para poder efetuar contribuições.
* Git merge <nome\_branch>  
  git merge <nome\_branch> atualiza seu branch com a branch seleciona.
* Git config –global user.name  
  git config –global user.name utilizamos para configurar seu usuário para efetuar subidas ao repositório.
* Git revert  
  git revert utilizamos para reverter commits efetuados na branch e podendo reverter específicos
* Git rebase  
  git rebase tem uma função parecida com git merge, mas existe uma lógica diferente nele, ele pega os commits efetuados em outra branch e adiciona na sua, os seus commits passam a serem os mais recentes.
* Uma imagem contendo computador

  Descrição gerada automaticamenteGit fetch  
  Busque as ramificações ou as tags (coletivamente, refs) de um ou mais repositórios, juntamente com os objetos necessários para completar seus históricos. As ramificações de rastreamento remoto são atualizadas (consulte a descrição de <refspec> abaixo para conhecer maneiras de controlar esse comportamento).
* Git tag  
  Da mesma forma que a maioria dos VCSs, o Git tem a habilidade de marcar pontos específicos na história como sendo importantes. Normalmente as pessoas usam essa funcionalidade para marcar pontos onde foram feitas releases (v1.0 e assim por diante). Nessa sessão, você irá aprender como listar as tags existentes, como criar tags e quais são os diferentes tipos de tags.
* Touch stash  
  Use git stashquando quiser registrar o estado atual do diretório de trabalho e do índice, mas quiser voltar para um diretório de trabalho limpo. O comando salva suas modificações locais e reverte o diretório de trabalho para corresponder ao HEADcommit.
* Touch stash pop  
  comando adiciona os arquivos

novamente na mudança da branch permitindo assim que

você efetue o commit novamente e um push

* Git cherry-pick

Aplica as alterações de commit na branch atual.

* Git blame  
  comando utilizado para mostrar alterações feitas em um arquivo, por linha de comando, mostra o autor do commit.

Uma imagem contendo computador

Descrição gerada automaticamente

* Uma imagem contendo computador

  Descrição gerada automaticamenteGit bisect  
  Faz buscas binárias no commits, utilizado normalmente para commits antigos aonde a busca pelo código não é fácil de se encotrar.