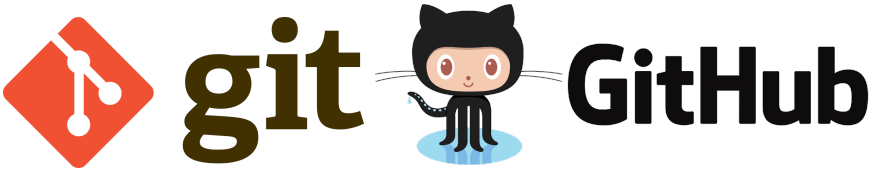
>> UNIVERSIDADE FEDERAL DO PARÁ <<
>> INSTITUTO DE TECNOLOGIA <<
>> RAMO ESTUDANTIL IEEE UFPA <<
>> WOMEN IN ENGINEERING UFPA <<

>> GRUPO DE ESTUDOS EM PYTHON <<
#WIECODEPYTHON

>> SEMANA #1 <<
>> APRENDENDO GIT E GITHUB <<
>> APOSTILA DE REFERÊNCIA <<



>> Sumário

>> O que é controle de versões?

O controle de versões pode ser descrito como uma sistemática para gerenciar as diferentes partes, versões e modificações no desenvolvimento de um documento qualquer, de forma eficiente, organizada e prática. Essa sistemática permite, mediante a necessidade do usuário, recuperar uma determinada versão deste documento. De forma tranquila e prática, pode se dizer que é como registrar todas as modificações efetuadas nos arquivos durante o seu desenvolvimento.

Este tipo de sistema é comumente utilizado em projetos onde vários integrantes contribuem paralelamente com o seu desenvolvimento. Em projetos de softwares, o controle de versões pode ser utilizado para controlar o histórico de desenvolvimento dos códigos-fontes e também da documentação do mesmo.

Ao empregar o controle de versões, você estará garantindo segurança e praticidade para colaboração em seu projeto pois, ao modificar algum arquivo ou executar instruções comprometedoras no código-fonte de um software, você terá garantia de que poderá realizar uma reversão dos arquivos, e da mesma forma verificar as alterações que os demais integrantes do projeto fizeram.

>> Uma breve história do Git

O *Git* é um software para controle de versões criado por **Linus Torvalds** em 2005. A ideia de seu desenvolvimento surgiu quando Torvalds e os desenvolvedores do *kernel Linux* optaram por não utilizar mais o software proprietário *BitKeeper*, após Larry Macvoy (detentor dos direitos autorais do *BitKepper*) remover o acesso gratuito ao software.

Torvalds tinha a intenção de desfrutar de um sistema distribuído que fosse rápido, fluído e que funcionasse de maneira similar ao *BitKeeper*. Sem muitas opções, Torvalds decidiu desenvolver o próprio software controlador de versões e assim nascia o *Git*.

Em suma, o *Git* foi desenvolvido e projetado por Torvalds para auxiliar no desenvolvimento do *kernel* do Linux, porém, com a ênfase em velocidade e praticidade, aliadas à licença *GNU*, cujos termos declaram o software como livre, o *Git* acabou ganhando vários admiradores e usuários ao redor do mundo.

Atualmente o *Git* é mantido por Junio Hamano, um dos principais colaboradores do projeto em 2005 e, responsável por entregar a versão 1.0 do software em 21 de dezembro de 2005.

>> Sobre o GitHub

O *GitHub* é um serviço web para hospedagem, gestão e compartilhamento de repositórios *Git* que oferece recursos para desenvolvimento e colaboração. *"Build software better, together."* é o principal slogan do *GitHub*, justamente para enfatizar o seu principal compromisso: dar suporte ao desenvolvimento colaborativo.

O *GitHub* permite que você acompanhe *(watch)* e favorite *(star)* repositórios. Também dá para seguir pessoas e participar de organizações (grupo de usuários) que podem ter repositórios próprios, ideal para projetos em equipe. O perfil de cada usuário registra suas atividades em todos os projetos e em cada um pode-se acompanhar as contribuições de cada colaborador. Nos repositórios pode-se navegar pelo histórico de *commits*, filtrá-los por colaborador, ver as modificações no código *(diffs)* comparando *commits* e *branches*. O *GitHub* não hospeda apenas código fonte mas sim todo e qualquer arquivo que você tenha sob versionamento. É possível hospedar dados, por exemplo, em formato texto (csv, txt), e disponibilizá-los por meio da *URL* para *download* ou leitura direta.

O *GitHub* é o serviço web para *Git* mais popular quanto ao número de projetos hospedados. No entanto, existem serviços com as mesmas funcionalidades e até com algumas que o *GitHub* não oferece no plano básico. O *GitLab* e o *Bitbucket* estão entre os 5 mais populares e permitem, por exemplo, ter alguns repositórios privados com a conta *free*. Porém, neste grupo de estudos usaremos o *Github* para hospedar e compartilhar todos os projetos.

>> Instalação do Git

Instalando o Git no Linux

Para instalar o Git no Linux, é necessário executar um comando de instalação no terminal do seu SO. O comando varia de distribuição para distribuição e vamos listar aqui os comandos que devem ser utilizados nas distribuições mais utilizadas.

Para Debian, Ubuntu e derivadas:

```
>> apt-get install git
```

Para Fedora:

```
>> yum install git (até Fedora 21)
>> dnf install git (Fedora 22 em diante)
```

Para openSuse:

```
>> zypper install git
```

Pode ser necessário realizar um chaveamento para modo root antes de executar o comando de instalação. Caso não saiba como fazer, procure na documentação da distribuição que está utilizando.

Instalando o Git no Windows

Para instalar o Git no Windows, primeiramente você deve acessar o link https://gitforwindows.org/ no seu navegador de preferência, mostrado na Figura 1. Assim, você deve estar no site da imagem e, então, precisa clicar em download.

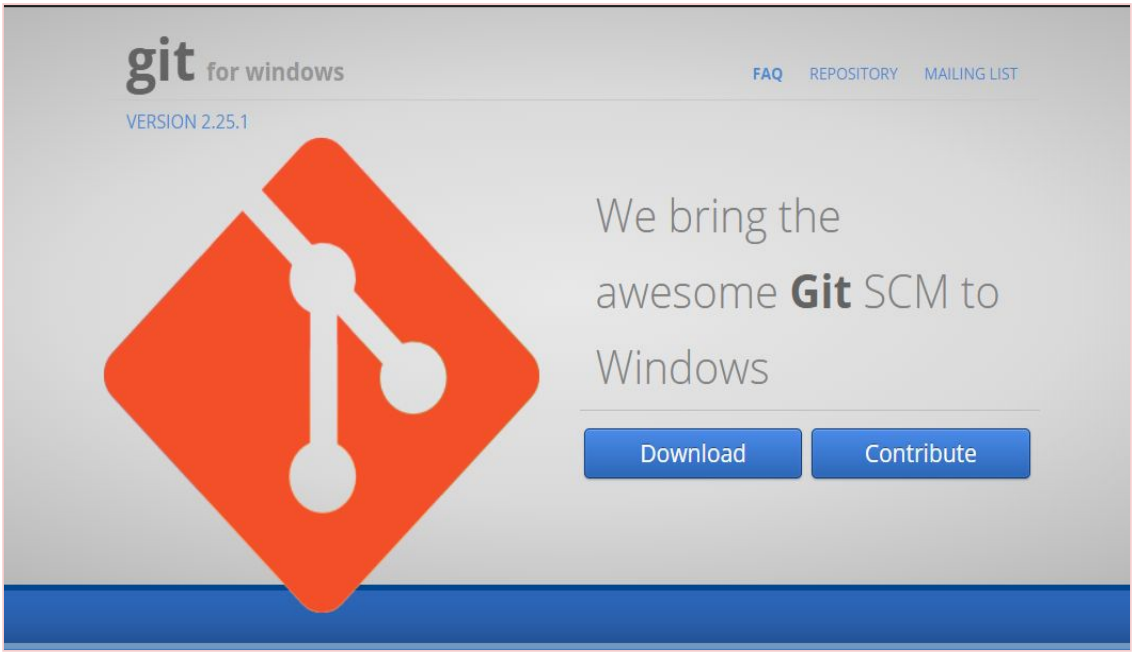


Figura 1: Página de download do git para windows.

Terminando de baixar o instalador, é necessário apenas prosseguir com a instalação normal. O pacote conta com uma interface gráfica para usuários menos experientes, mas possui também uma interface básica que emula um shell.

>> Utilizando o Git

A primeira coisa que você deve fazer quando instalar o Git é definir o seu nome de usuário e endereço de e-mail. Isso é importante porque todos os *commits* no Git utilizam essas informações, e está imutavelmente anexado nos commits que você realiza. Abra o Terminal no Linux ou CMD no Windows e execute os comandos:

```
>> git config --global user.name "wie-ufpa"
>> git config --global user.email "wie-ufpa@example.com"
```

No lugar de "wie-ufpa" coloque seu nome de usuário.
No lugar de "wie-ufpa@example.com" coloque seu e-mail.

Criando um repositório local

Repositório é como chamamos os projetos que utilizam Git.

Antes de criar o repositório você deve criar um pasta para ele.

Para criar uma nova pasta tanto no Linux quanto no Windows, abra o terminal do Linux ou CMD do Windows e execute:

```
>> mkdir nome-da-pasta
```

Você entra nela, ou melhor, vai até ela:

```
>> cd nome-da-pasta
```

Agora é só criar o repositório:

```
>> git init
```

Arquivo Readme

Quando trabalhamos numa equipe, costumamos chamar a pessoa que desenvolveu para explicar tudo para nós, mas se o desenvolvedor não estiver ao nosso alcance isso não será possível. É por isso que a comunidade adota a prática de criar um arquivo **README** para todos os projetos desenvolvidos e versionados num Sistema de Controle de Versão (VCS). Esse arquivo, como o próprio nome já diz, é feito para ser lido, pois ele ajudará o usuário interessado no projeto a entender o **seu propósito**, **como usá-lo**, **como mantê-lo**, e diversas outras informações úteis.

Adicionado e comitando

Considerando que você já criou o seu repositório Git e está na pasta do repositório. Você deve criar um arquivo qualquer, neste caso iremos criar nosso arquivo **readme**, utilize seu editor de texto ou então utilize o comando seguinte:

Linux

```
>> touch readme.md
```

Windows

```
>> dir > readme.md
```

Antes de comitar, devemos adicionar, a sequência é sempre esta:

- Adicionar (add)

- Comitar (commit -m "mensagem")

Veja os comandos:

```
>> git add readme.md
```

A principal ferramenta utilizada para determinar quais arquivos estão em quais estados é o comando:

```
>> git status
```

Dê esse comando e a saída será parecida com essa:

```
>> git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        readme.md

nothing added to commit but untracked files present (use "git add" to track)
```

Então você pode comitar seu arquivo:

```
>> git commit -m "meu primeiro commit"
```

Execute git status novamente e a saída deve ser esta:

```
On branch master
nothing to commit, working tree clean
```

Para ver seu histórico de commits execute o comando:

```
>> git log
```

A saída deve ser parecida com esta:

```
Author: wiecodepython <wiecodepython@gmail.com>
Date:   Mon Mar 23 17:24:27 2020 -0300
    meu primeiro commit
```

Criando um repositório no GitHub

Primeiro, crie um conta no GitHub.

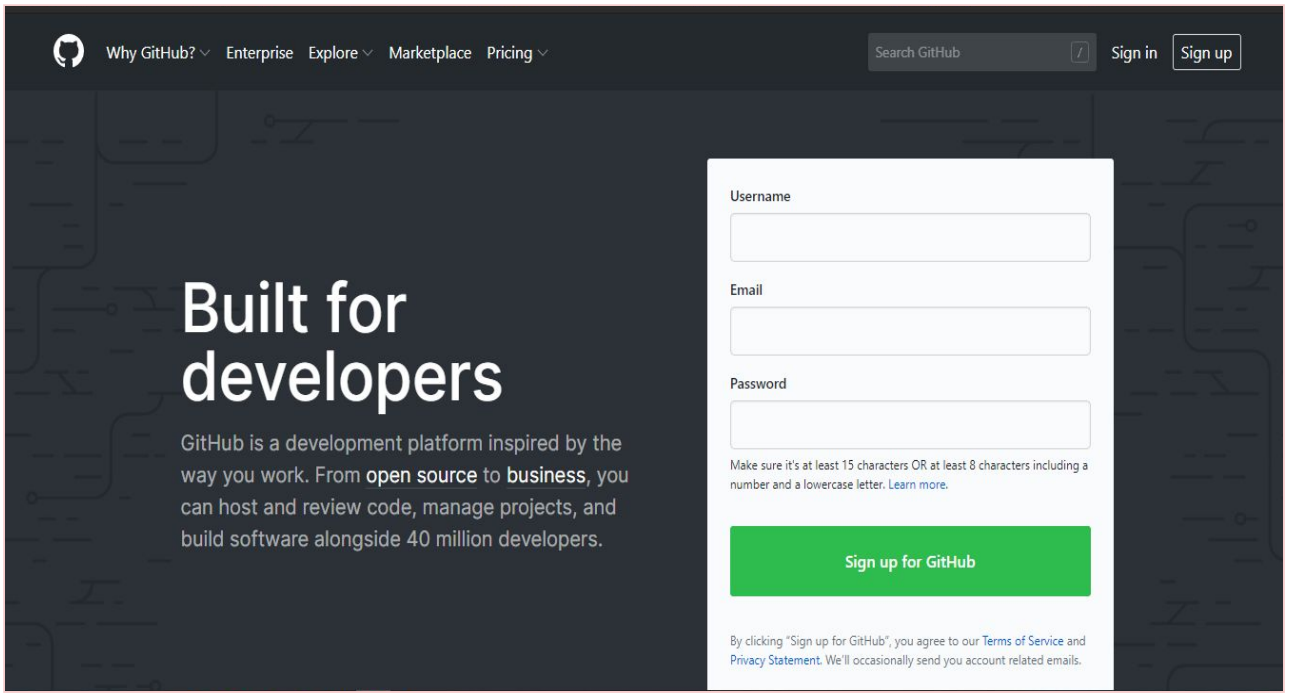


FIgura 2: Página inicial do Github.

Utilize um nome de guerra que transmita seriedade, ele será visto por todos, inclusive seu futuro empregador. Agora procure o ícone chamado "New".

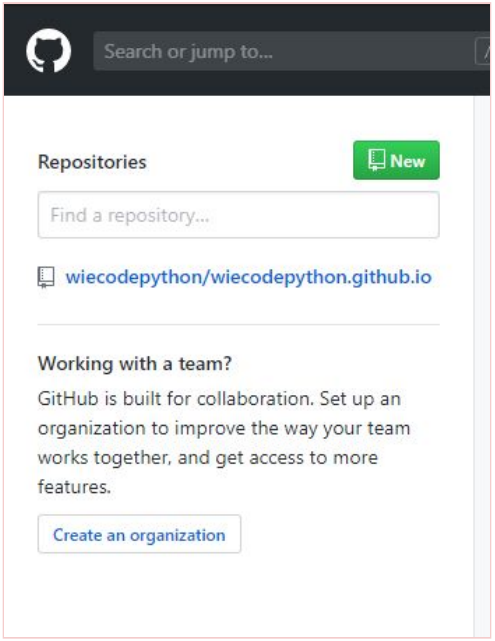


Figura 3: Criar novo repositório no Github.

Defina um nome para seu novo repositório.

Por enquanto, ignore as outras opções. Depois, com calma, você mesmo aprenderá sobre elas.

Como seu repositório está vazio, o GitHub lhe dará algumas dicas, sua tela deve parecer-se com esta:

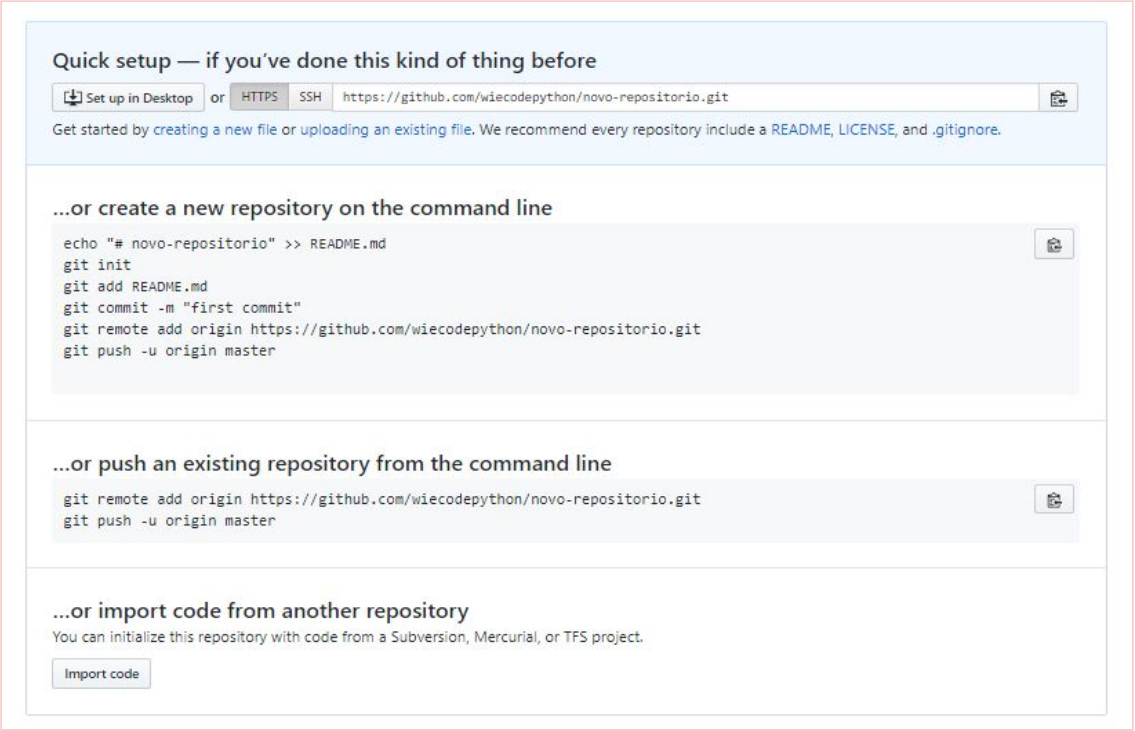


Figura 4: Repositório vazio.

Apenas anote a *URL* exibida no campo *HTTP*, vamos precisar dela daqui a pouco.

Agora, temos um repositório no GitHub e um localmente, então vamos conectá-los.

Conectando o repositório local com o da web

O seu QG (quartel general) é o seu repositório local. A partir dele, você enviará ou receberá informações do repositório na web (no GitHub).

Para "conectar" os repositórios executamos o comando abaixo:

```
>> git remote add origin https://github.com/wiecodepython/novo-repositorio.git
```

Onde **origin** significa um apelido para seu repositório, poderia ser qualquer outro nome.

E no lugar da minha URL você deve utilizar a sua URL, eu avisei que precisaríamos dela!

Sincronizando os repositórios

Você já criou um repositório local e outro no GitHub.

Já adicionou alguns arquivos e "comitou" algumas modificações (localmente).

O seu repositório local já está conectado com o da web (possui uma referência para ele).

Agora só falta enviar as informações do repositório local para o repositório na web (no GitHub):

```
>> git push origin master
```

Lembrando que origin é o apelido para seu repositório na web e master é o seu **branch** principal.

Branches são linhas independentes de desenvolvimento nas quais podemos trabalhar livremente, versionando quando quisermos, sem atrapalhar outras mudanças no código.

Fechando a conta

Se tudo deu certo, acesse novamente (ou atualize) a página de seu repositório, você deve estar vendo algo parecido com a figura abaixo.

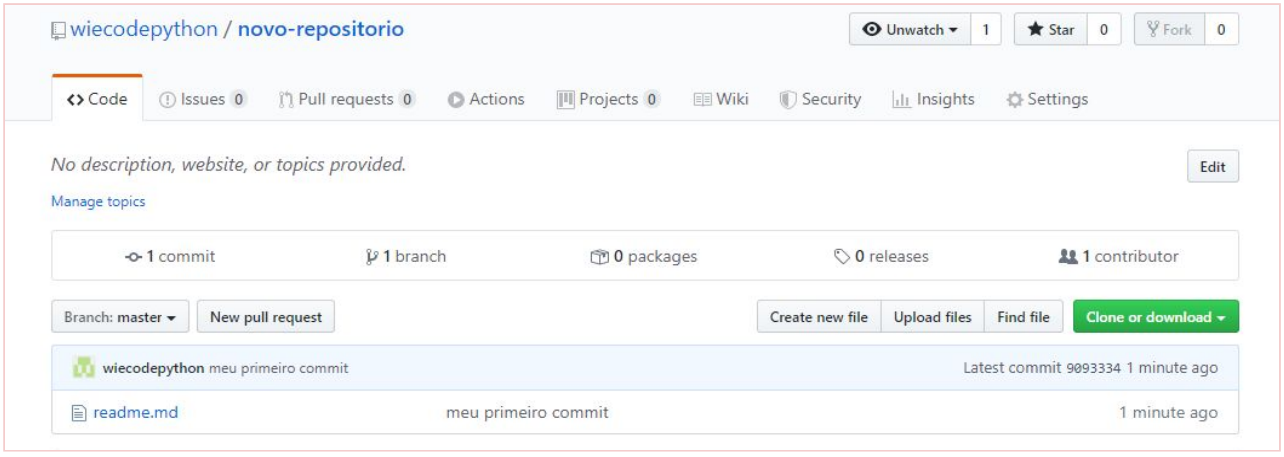


Figura 5: Repositório após o commit.

Agora estamos prontos para avançar de fase.

Passos para adicionar o seu código no repositório compartilhado do WieCodePython

Vá até o repositório Exercícios em https://github.com/wiecodepython/Exercicios e faça um Fork no repositório, como indicado na figura :

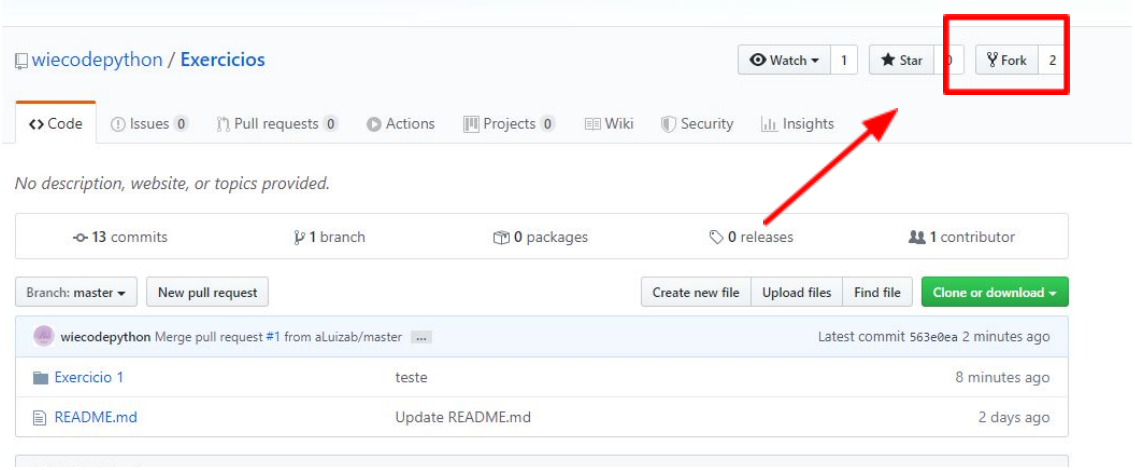


Figura 6: Botão de Fork.

Quando fizer isso, o repositório é copiado para o seu perfil no GitHub para que você possa modificá-lo sem que mexer no repositório principal. Como mostrado a seguir:

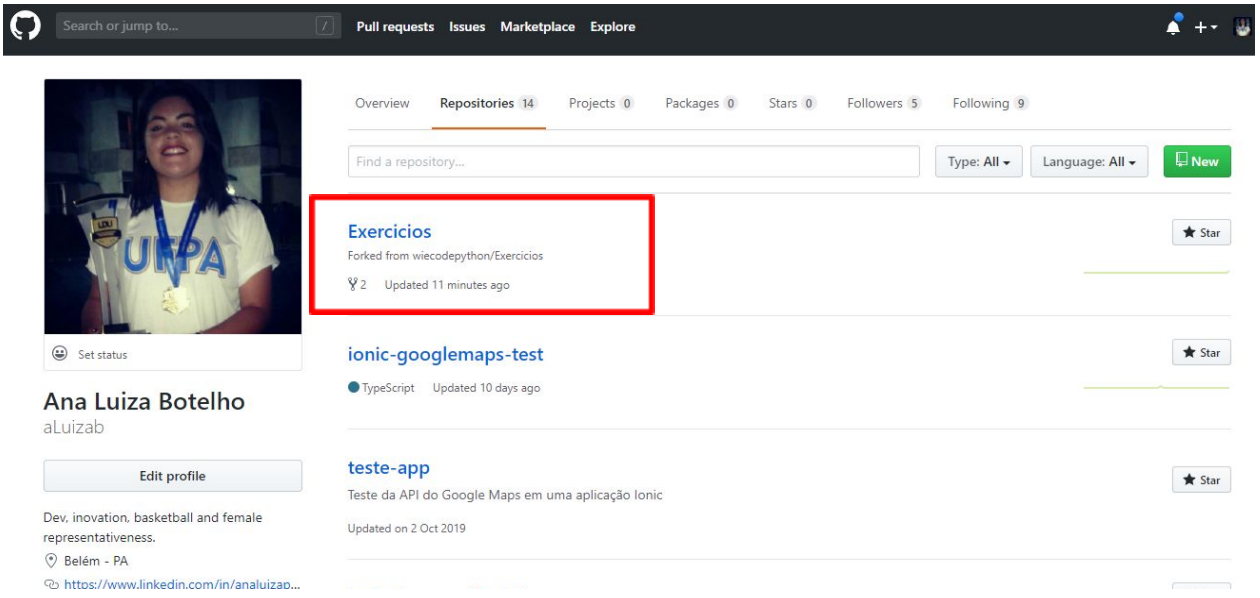


Figura 7: Repositório no perfil após o fork.

Desta forma, entre no repositório e faça o clone para sua máquina para que seja possível modificá-lo.

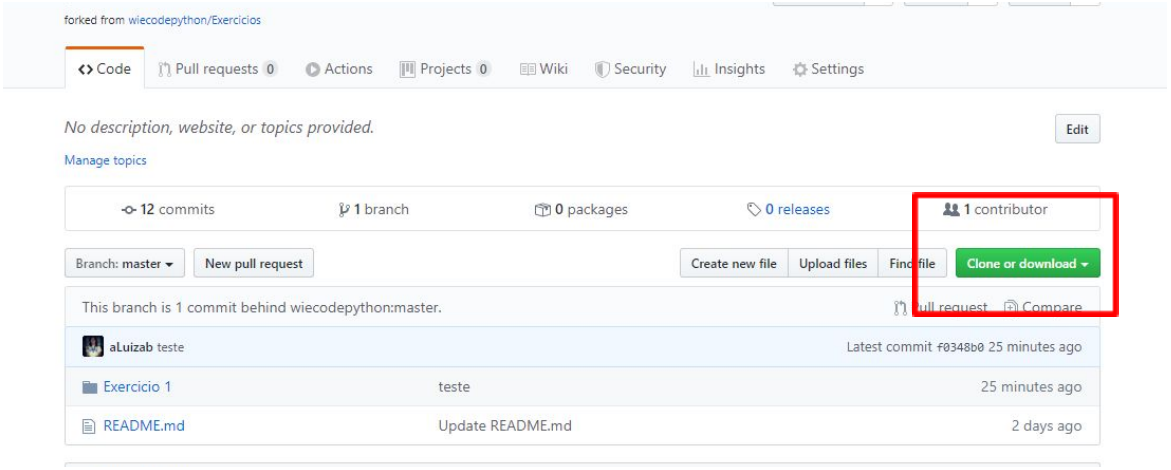


Figura 8: Botão de Clone ou Download.

Clique no botão "Clone or download" e copie a URL que aparecerá. Guarde esta URL, ela já será usada.

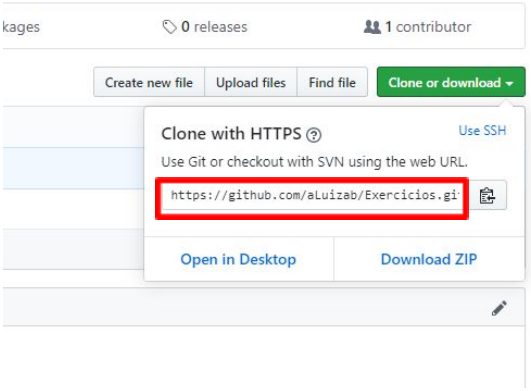


Figura 9: URL do repositório.

Escolha uma pasta do seu computador para fazer o clone do projeto.

Se você estiver utilizando qualquer distribuição do Linux é só executar os mesmos comandos descritos a seguir no seu terminal.

Se você está utilizando o Windows, siga os passos mostrados nas figuras.

Na pasta clique com o botão direito do mouse e vá até a opção git bash here.

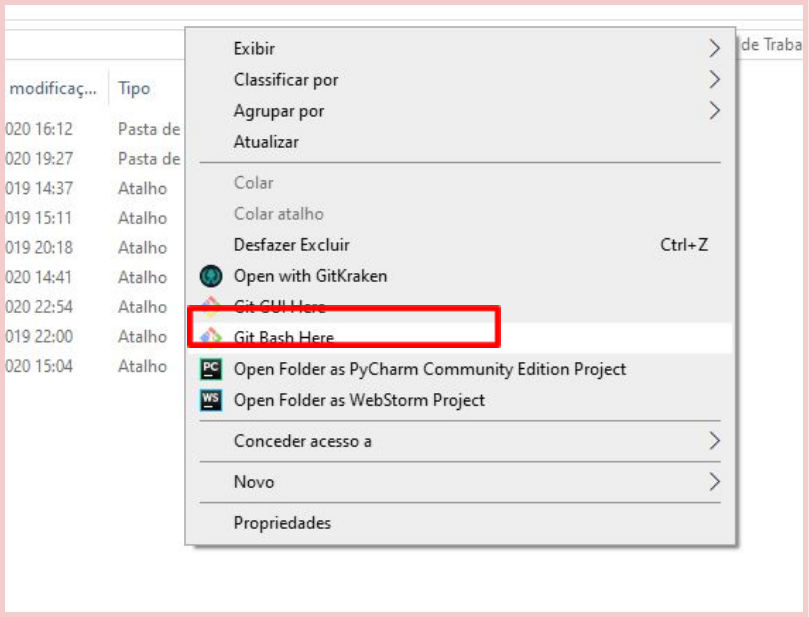


Figura 10: Abrir o Git Bash no windows.

Será aberto o terminal do git, nele podemos executar todos os comandos do git.

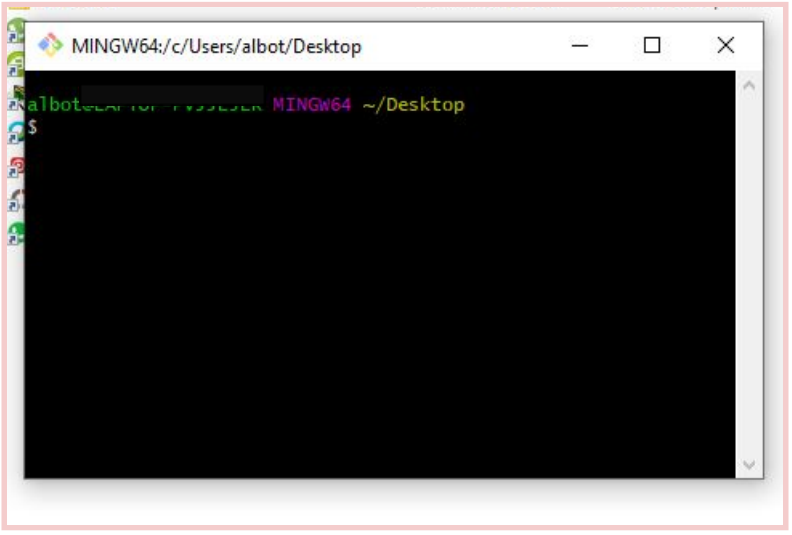


Figura 11: Interface do Git Bash.

Agora vamos lá!

Execute o comando:

```
>> git clone https://github.com/SeuUsuário/Exercicios.git
```

Lembre-se, coloque a URL que você copiou anteriormente do seu repositório. Este comando fará um clone do projeto para sua pasta escolhida, a saída deverá ser esta:

```
$ git clone https://github.com/aLuizab/Exercicios.git
Cloning into 'Exercicios'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 42 (delta 7), reused 4 (delta 0), pack-reused 0
Unpacking objects: 100% (42/42), done.
```

Vá até o diretório clonado e crie dentro da pasta do **exercício 1** uma pasta com seu **nick do github** e coloque nela suas respostas e códigos. Feito isso execute:

```
>> git status
```

para verificar se houve alteração.

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

        aLuizab/

nothing added to commit but untracked files present (use "git add" to track)
```

Agora adicione a modificação com `git add .` e execute o `git status` novamente para verificar se realmente adicionou.

```
$ git add .
```

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   aLuizab/exercicio1.txt
```

Se a saída estiver parecida com a de cima é só comitar:

>> git commit -m “resposta exercício 1”

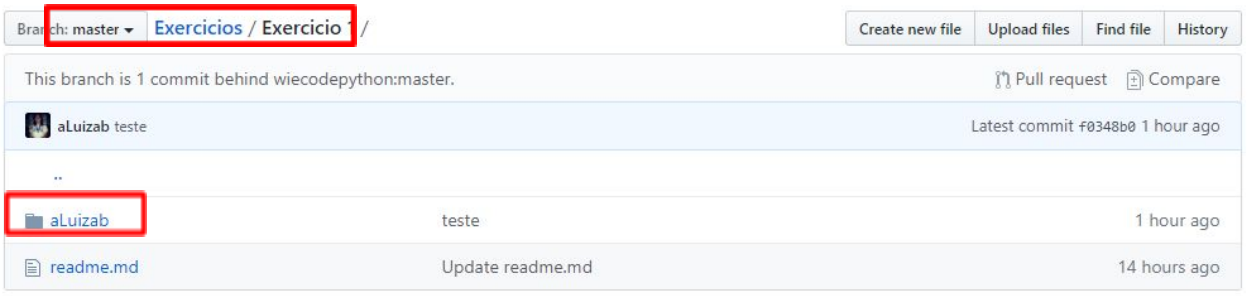
```
$ git commit -m "teste"
[master f0348b0] teste
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Exercicio 1/aLuizab/exercicio1.txt
```

E enviar para o repositório remoto:

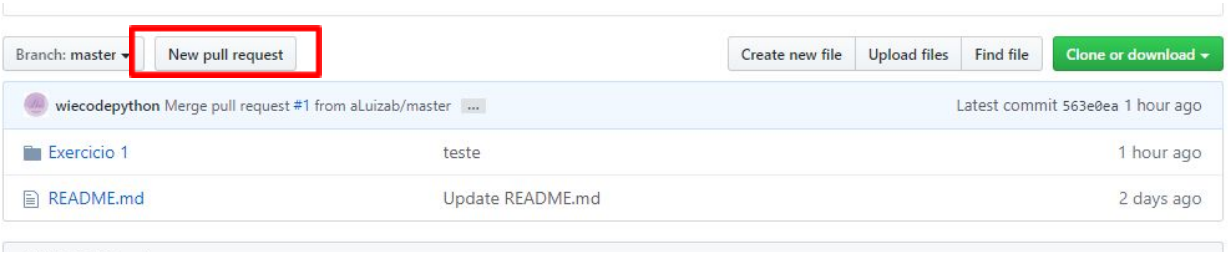
>> git push -u origin master

```
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 409 bytes | 136.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/aLuizab/Exercicios.git
   46d94d7..f0348b0  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

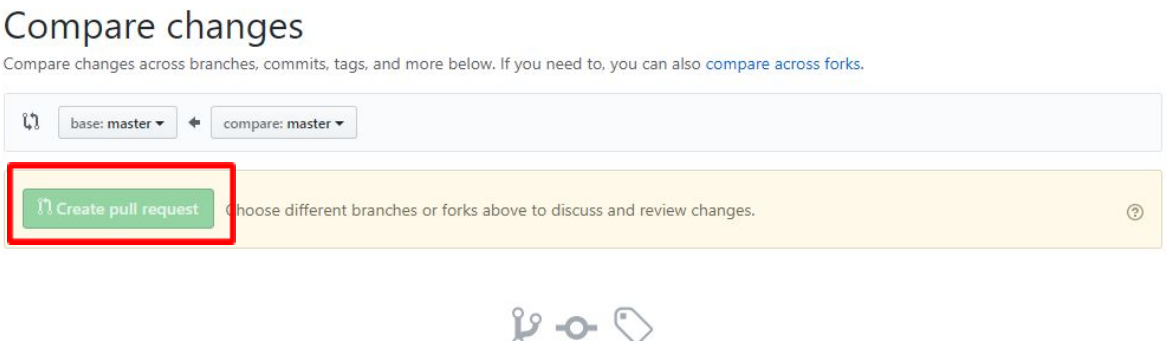
Após estes passos, faça um refresh no seu repositório no GitHub e verifique se foi feito o upload da pasta e do arquivo.



Bom, você conseguiu adicionar suas modificações no seu repositório pessoal. Para adicionar elas ao perfil do grupo de estudos é só fazer um Pull Request:



Clique no botão "New pull request" e crie um novo.



Siga os passos e pronto, seu pedido para editar o projeto foi enviado, vamos aprovar e em seguida você verá a pasta com o seu nome no perfil do WieCodePython.

Segue abaixo a descrição de alguns dos principais comandos do Git.

>> git add: Adicionar os arquivos (modificados, novos ou deletados) na área da staged

>> git commit: Consolidar todos os arquivos. Antes de commitar você deve informar uma mensagem clara de commit.

>> git pull: Sincronizar o seu repositório local com o remoto. Este comando realiza também automaticamente o merge das alterações (git fetch + git merge)

>> git fetch: Apenas baixar as atualizações sem realizar o merge. Os arquivos ficam guardados no arquivo FETCH_HEAD

>> git push: Enviar as atualizações para o servidor remoto

>> git branch <nome_da_branch>: Criar uma ramificação do projeto. Muito usado quando vc possui versões finais em Produção (boa prática)

>> git checkout <nome_da_branch>: Muda para uma nova branch já criada

>> git checkout -b <nome_da_branch>: Cria uma branch local e automaticamente já muda para ela

>> git log: Verificar histórico de commits

>> git blame: Verificar em detalhes o histórico de cada arquivo individualmente

>> git revert: Reverter um arquivo para o commit anterior, mas criando um novo commit

>> git reset: Desfaz os commits. Por padrão ele executa com o argumento **--soft** que desfaz todos os commits mas mantém as alterações. Já o argumento **--hard** desfaz os commits e também as alterações

>> git merge: Usado para merge entre branchs. Cria um novo commit de merge

>> git rebase: Usado para merge entre branchs, porém não cria um novo commit de merge

>> git bisect: Faz uma pesquisa binária na árvore do Git, procurando no histórico e elege o commit good e bad

>> git cherry-pick: Adiciona apenas o commit de outra branch na sua branch atual

>> git config --global core.editor <editor_de_sua_preferencia>: Coloca o editor de mensagens do commit, com o editor de sua preferência

>> git config --global user.email <seu_email>: Configura seu email

>> git config --global user.name <seu_nome>: Configura seu nome

>> git clone <link_do_projeto>: Baixar o projeto do repositório (baixará por default da branch master)

>> git clone -b <nome_da_branch> <link_do_projeto>: baixar o projeto de uma branch específica

Lembramos aqui que este é um material de referência, está longe de ser o melhor e o mais completo, o objetivo é a troca de conhecimento. Por isso referenciamos todas fontes e na internet você pode encontrar muitos materiais disponibilizados pela comunidade.

>> Referências

Roger Dudler, **Git - guia prático** - http://rogerdudler.github.io/git-guide/index.pt_BR.html - Acessado em março de 2020.

PET Estatística UFPR, **Apostila Git** - https://www.academia.edu/29891173/Apostila_Git_Universidade_Federal_do_Paran%C3%A1 - Acessado em março de 2020.
I. Corrêa, C. Araújo, A. Medina, **Tutorial Git** - http://coral.ufsm.br/pet-si/wp-content/uploads/2017/02/Consult%C3%B3rio-de-Software-Git.pdf - Acessado em março de 2020.

DevFuria, **Tutorial para iniciar com o Git e o GitHub** - http://devfuria.com.br/git/tutorial-iniciando-git/ - Acessado em março de 2020.

Thiago Pascoal, **Exercicios Git** - https://github.com/thiagopaschoal/exercicios-git - Acessado em março de 2020.

Diego Fernandes, **Git & Github: O que é? Por que? Como iniciar?** - https://blog.rocketseat.com.br/iniciando-com-git-github/ - Acessado em março de 2020.

Caelum, **Trabalhando com Branches** - https://www.caelum.com.br/apostila-java-testes-spring-design-patterns/primeiras-interacoes-com-o-novo-projeto/ - Acessado em março de 2020.