

Aula 11

Git e GitHub

O que é, vantagens, práticas de uso

Atualmente, no desenvolvimento de softwares, é necessário que toda a equipe tenha conhecimento e acesso aos arquivos do projeto que está sendo realizado. Para isso, foi desenvolvido um sistema chamado Git, no qual os desenvolvedores poderiam trabalhar todos ao mesmo tempo sem interferir nos trabalhos dos demais.

Ao longo dos anos, o sistema Git foi recebendo aperfeiçoamentos e novos recursos para aumentar ainda mais seu poder de trabalho. Com isto, foram surgindo plataformas que utilizavam o sistema Git para fornecer repositórios de arquivos para os desenvolvedores, e assim em 2008, os desenvolvedores da Logical Awesome lançaram o Github.

O Github atualmente é a maior plataforma de Git do planeta, sendo utilizado em grande parte dos países do mundo por diversos desenvolvedores.

O que é

O Git, é um **sistema de versões**, no qual os desenvolvedores possuem **acesso simultâneo** aos arquivos do projeto para poderem realizar alterações e correções. Para que os arquivos sejam mantidos, é necessário a criação de um espaço chamado “Repositório”, que é um espaço digital onde as versões do projeto ficam armazenadas.

Já o GitHub, é uma plataforma que trabalha com o sistema Git. Por meio dele, é possível criar um repositório para uma equipe de desenvolvedores para que todos possam acessar os arquivos do projeto. Com isto, também é possível realizar cópias do repositório como “linhas de trabalho paralelas”, onde é possível corrigir bugs ou fazer alterações no projeto sem pôr em risco o projeto principal.

Vantagens

A plataforma GitHub, apresenta diversas vantagens para os desenvolvedores. Podendo citar algumas delas como disponibilidade dos

arquivos, possibilidade de trabalho por diversos desenvolvedores sem riscos de perda, entre outras.

Utilizando a plataforma, há possibilidade de uma equipe inteira de desenvolvedores trabalhar de maneira simultânea, apresentando uma eficácia de trabalho muito maior.

Práticas de uso

As práticas de uso se do GitHub, incluem o upload de projetos para os repositórios e a clonagem para as “IDES(Ambiente de trabalho do desenvolvedor)” dos desenvolvedores. O GitHub, também permite a criação de “Branches” para cada desenvolvedor ter uma cópia para resolução de bugs ou mesmo melhoria do código do projeto.

A partir da correção de um “bug”, ou mesmo uma melhoria no código por meio de uma “branch”, é possível realizar o processo de “merge(juntar)” no código, no qual fará os códigos se juntarem e manterem o bom resultado no projeto principal.

Outra prática consiste em realizar “commits” que se referem a atualizações realizadas no código, ou mesmo anotações de bugs que estão presentes no código.

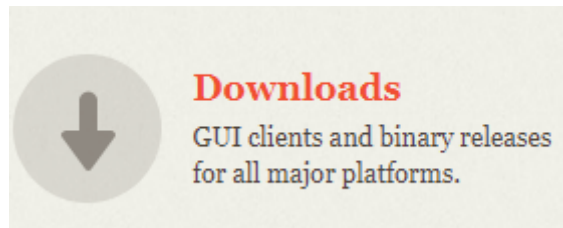
Instalação, comandos

Para que o sistema Git possa ser utilizado juntamente com a plataforma GitHub, é necessário realizarmos a instalação do sistema Git em nosso computador. Uma observação importante é frisarmos o fato de que o Git não é um sistema operacional, mas sim um sistema utilizado para controle de versões de aplicações de projetos.

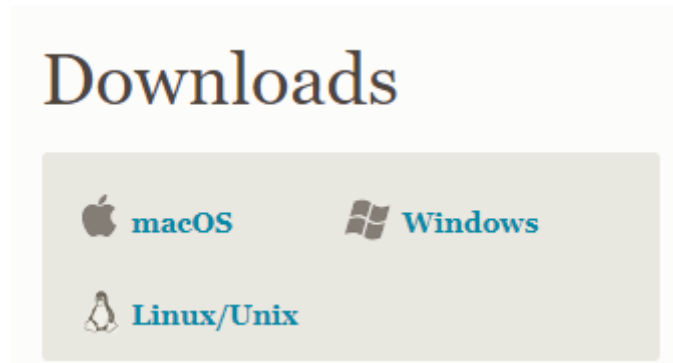
Instalação

Para realizarmos este passo, precisaremos realizar o download do sistema Git no site oficial do sistema (<https://git-scm.com/>). Logo em seguida faremos alguns passos para realizar a instalação do sistema.

1. Acessar o site <https://git-scm.com/>.
2. Clicar no botão download na cor laranja.
- 3.



4. Selecionar o seu sistema operacional.



5. Baixe o arquivo e realize a instalação conforme seu sistema operacional. Observação: Caso seu sistema operacional seja Windows, baixe uma versão standalone conforme seu sistema operacional (como na figura abaixo).

Standalone Installer
32-bit Git for Windows Setup.
64-bit Git for Windows Setup.

Comandos

O Git funciona com base em alguns comandos. Para iniciarmos nosso uso do Git, é necessário entendermos um pouco de qual a importância de comandos dentro do Git.

Os comandos são utilizados para executar funções no Git, como realizar o download ou upload do código. Eles são em inglês pois é a língua nativa da programação.

Vamos começar com dois conceitos básicos e muito importantes se tratando do Git, o "pull" e o "push".

- **Pull** em inglês, significa "puxar", isso remete a fazer o download dos arquivos do repositório, porque você "puxa os arquivos para seu computador".

- **Push** é traduzido para “empurrar”, pois remete a fazermos upload(enviar) arquivos para nosso repositório. Considere estar “empurrando arquivos do seu computador para o repositório”.

Os comandos são utilizados no console do seu editor de texto(IDE) durante a programação do código. Outro ponto importante, o comando de push pode utilizar apenas o commit.

- **Git clone**

O comando Git clone, realiza a cópia de um repositório para sua IDE. Com isto, lhe dá a possibilidade de ver e trabalhar no código e arquivos do repositório.

Clonar branch:

```
git clone <https://github.com/exemplolink>
```

- **Git branch**

O comando git branch, é utilizado para manipular nossas branches(ramificações, explicadas anteriormente).

Criar branch:

```
git branch <nome-da-branch>
```

Criar branch já realizando push:

```
git push -u <local-remoto> <nome-da-branch>
```

Ver lista de branches:

```
git branch -list
```

 (dois sinais de subtração previamente o termo “list”)

Excluir uma branch:

```
git branch -d <nome-da-branch>
```

- **Git branch**

O comando git checkout, é utilizado para “entrar” na branch na qual vamos trabalhar. O comando também é válido para trocar de branch ou mesmo verificar commits e arquivos.

Entrar/trocar e conferir arquivos e commits:

```
git checkout <nome-da-branch>
```

- **Git status**

O comando `git status` é utilizado para consultar informações como se há necessidade de realizar um pull ou push e até mesmo se a branch está atualizada.

Verificar informações da branch:

```
git status
```

- **Git add**

O comando `git add` é utilizado para adicionar uma alteração no próximo commit. Ele também pode ser utilizado de maneira a adicionar todas as alterações de uma vez.

Adicionar alteração no commit:

```
git add <arquivo>
```

Adicionar todas alterações no commit:

```
git add -A
```

- **Git commit**

O comando `git commit` é um dos mais utilizados pelos desenvolvedores. O comando define pontos de verificação ao longo do desenvolvimento do código. Um fato bem importante que deve ser ressaltado, é o fato de que o `git commit` não envia os arquivos para o repositório, apenas salva-os no dispositivo de trabalho local.

```
git commit -m "Observação do commit"
```

- **Git push**

O comando `git push` é executado após a realização do commit. Ele é utilizado para realizar o upload do commit previamente feito. É importante frisar que o `git push` realiza o upload apenas dos arquivos e códigos que já estão em um commit.

Realizar um push no código para um repositório:

```
git push <repositório-remoto> <nome-da-branch>
```

Realizar um push no código para um diretório com uma branch recém criada:

```
git push -u origin <nome-da-branch>
```

- **Git pull**

O comando `git pull` é utilizado para obter os dados de um repositório remoto, ou seja ele baixa as alterações do código e aplica imediatamente. É importante frisar que alguns conflitos podem ocorrer e você deverá resolvê-los manualmente.

Realizar o download das alterações que estão presentes no repositório:

```
git pull <repositório-remoto>
```

- **Git revert**

O comando git revert, é utilizado para voltar a uma versão anterior do projeto caso algum erro ocorra na qual você está trabalhando. Para utilizarmos este comando, é necessário primeiro utilizar o comando git log--oneline para que possamos ver as versões e os códigos respectivos de cada uma. Após isto, podemos realizar o git revert.

Realizar a consulta de versões do projeto:

```
git log--oneline
```

Realizar a reversão com o git reverse:

```
git reverse 0000000
```

 (Exemplo de número de versão)

- **Git merge**

O comando git merge é utilizado para realizar uma mescla das alterações com os arquivos originais. Este comando é utilizado após o desenvolvedor concluir uma branch com uma correção ou nova implementação que deverá ser atrelada a branch original.

Para realizar o processo, primeiro você precisa estar na branch principal, depois utilizar o comando de atualização local, git fetch e por fim realizar o merge.

Realizar a consulta se está na branch principal:

```
git checkout <nome-da-branch-principal>
```

Realizar a atualização local:

```
git fetch
```

Realizar o merge

```
git merge <nome-da-branch-com-o-recurso>
```

Criação de projetos, commits

Para que possamos realizar a criação de um projeto no GitHub, é necessário que criemos um repositório para ele. A partir da criação realizada, já será possível realizarmos o desenvolvimento do projeto com branches, commits e muito mais.

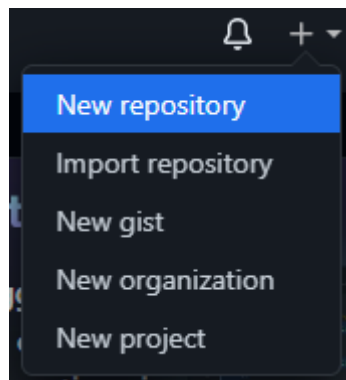
Criação de projetos

Para começarmos, iremos criar um novo repositório no GitHub para armazenarmos nosso projeto. Para isso seguiremos alguns passos.

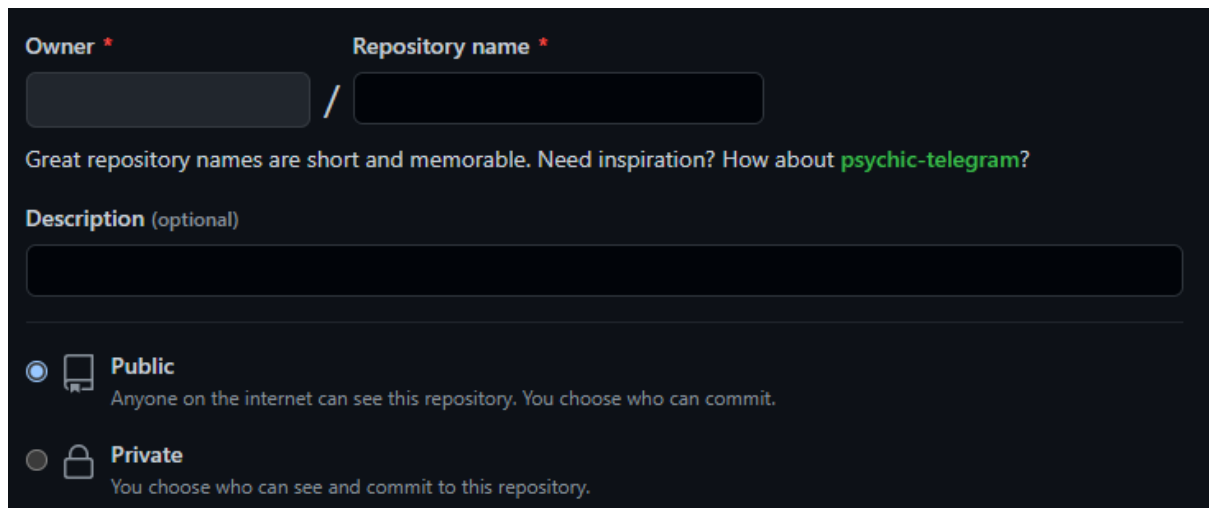
1. Acessar o site do github (github.com) e se cadastrar ou realizar o login.
2. Na tela principal, iremos clicar no ícone de adição (+) no canto superior direito da tela.



3. Vamos clicar em “New repository”




4. Agora preenchemos alguns campos. Iremos inserir o nome de nosso repositório em “repository name”. Agora criaremos uma breve descrição no campo “description”, como por exemplo “Meu primeiro projeto no GitHub”. E por último iremos definir a visibilidade no projeto, selecionando public (qualquer pessoa pode ver o conteúdo). Não se preocupe quanto aos commits, você terá a opção de decidir quem realiza os commits em seu projeto. Caso você selecionasse o private, apenas você poderia ver o projeto no GitHub.




Owner * Repository name *

Great repository names are short and memorable. Need inspiration? How about **psychic-telegram**?

Description (optional)

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

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

Create repository

5. Por fim iremos clicar no botão verde “Create repository”.

Commits

Para que possamos organizar de uma melhor maneira as versões de nosso repositório, é necessário que tenhamos registros. Para isso, é utilizado o comando commit durante o desenvolvimento do nosso código.

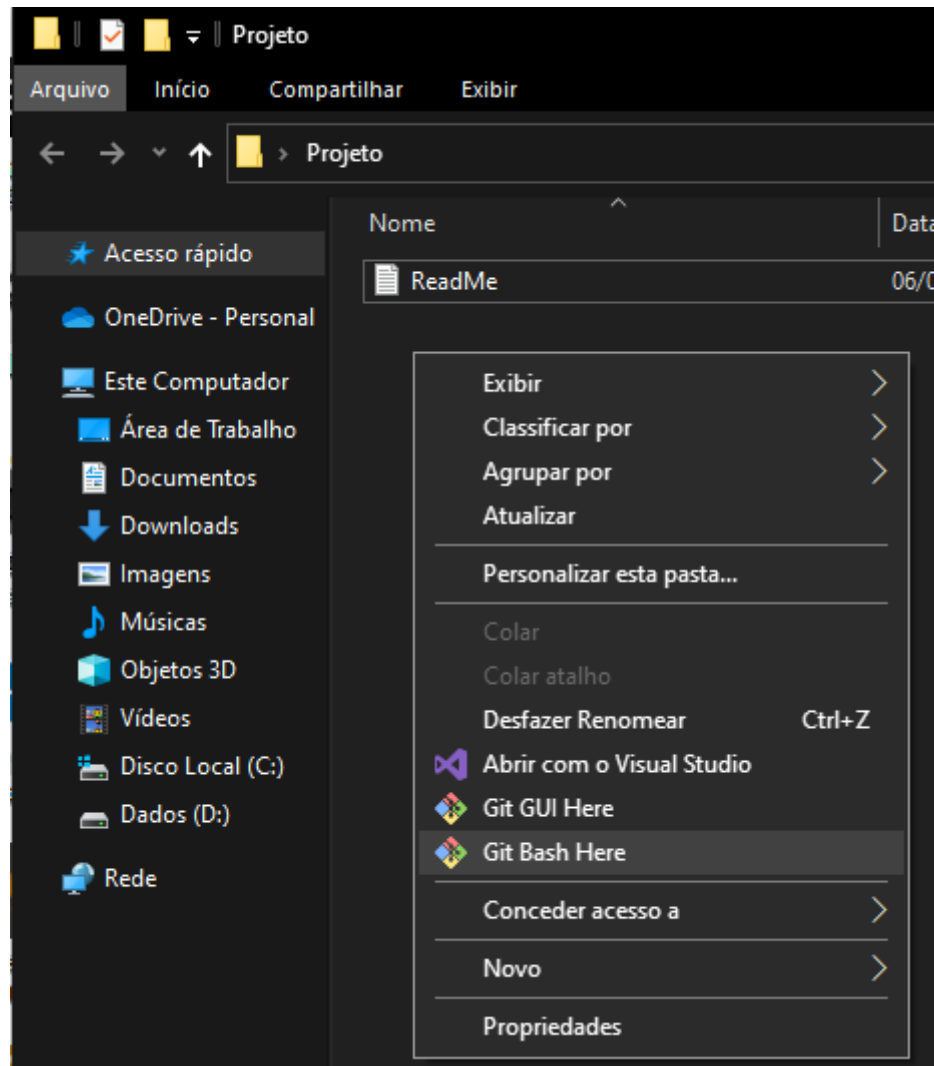
Para que possamos realizar um commit, é necessário que primeiro adicionemos os arquivos que vão sofrer o processo. Estes arquivos serão enviados ao repositório.

Exercício sobre Commit

Vamos realizar nosso primeiro commit neste exercício. Para isso, siga os passos abaixo.

1. Crie uma conta no GitHub caso ainda não tenha criado.
Link: <https://github.com/>
2. Crie um repositório no GitHub chamado “Tarefa” caso ainda não tenha um.
3. Instale o Git em seu computador caso ainda não tenha instalado.
Link: <https://git-scm.com/>

4. Crie uma pasta na área de trabalho chamada “Projeto” e dentro dela, crie um arquivo de bloco de notas (.txt) chamado “ReadMe”. Dentro dele escreva algo como “Estou aprendendo a utilizar o Git”.
5. Abra a pasta “Projeto” e clique com o botão direito do Mouse. Selecione a opção Git Bash Here.



6. Configure seu nome de usuário no terminal do Git Bash com o comando abaixo. Lembre-se de utilizar o nome de usuário que foi cadastrado no GitHub. Este processo é feito somente no primeiro uso do Git Bash.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git config --global user.name "Usuario"
```

7. Configure seu email no terminal do Git Bash com o comando abaixo. Lembre-se de utilizar o email que foi cadastrado no GitHub. Este processo é feito somente no primeiro uso do Git Bash.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git config --global user.email "email@email.com"
```

8. Inicialize o Git na pasta utilizando o comando abaixo.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git init
```

9. Vamos verificar se o Git está na pasta certa utilizando o comando abaixo. Caso.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git status
```

Caso o comando retorne o arquivo "ReadMe.txt" em letras vermelhas, o Git está na pasta correta.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git status
On branch master

No commits yet

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

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

10. Agora selecionaremos qual arquivo enviar ao GitHub com o comando abaixo.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git add ReadMe.txt
```

11. Iremos verificar se o arquivo foi adicionado para realizarmos o processo de commit com o comando abaixo.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git status
```

Caso o comando retorne o arquivo "ReadMe.txt" em letras verdes, o Git adicionou o arquivo para nosso processo de commit.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   ReadMe.txt
```

12. Vamos adicionar uma observação em nosso processo de commit. Lembre-se que esta etapa é obrigatória.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git commit -m "Meu primeiro Commit"
```

Caso o comando retorne a mensagem abaixo, o processo está correto.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git commit -m "Meu primeiro Commit"
[master (root-commit) 8b86223] Meu primeiro Commit
1 file changed, 1 insertion(+)
 create mode 100644 ReadMe.txt
```

13. Agora, identificaremos para qual repositório o Git deve enviar os arquivos utilizando o comando abaixo. Lembre-se, o link para seu repositório deve ser obtido por meio da página do mesmo no GitHub.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git remote add origin https://github.com/Usuario/Tarefa.git
```

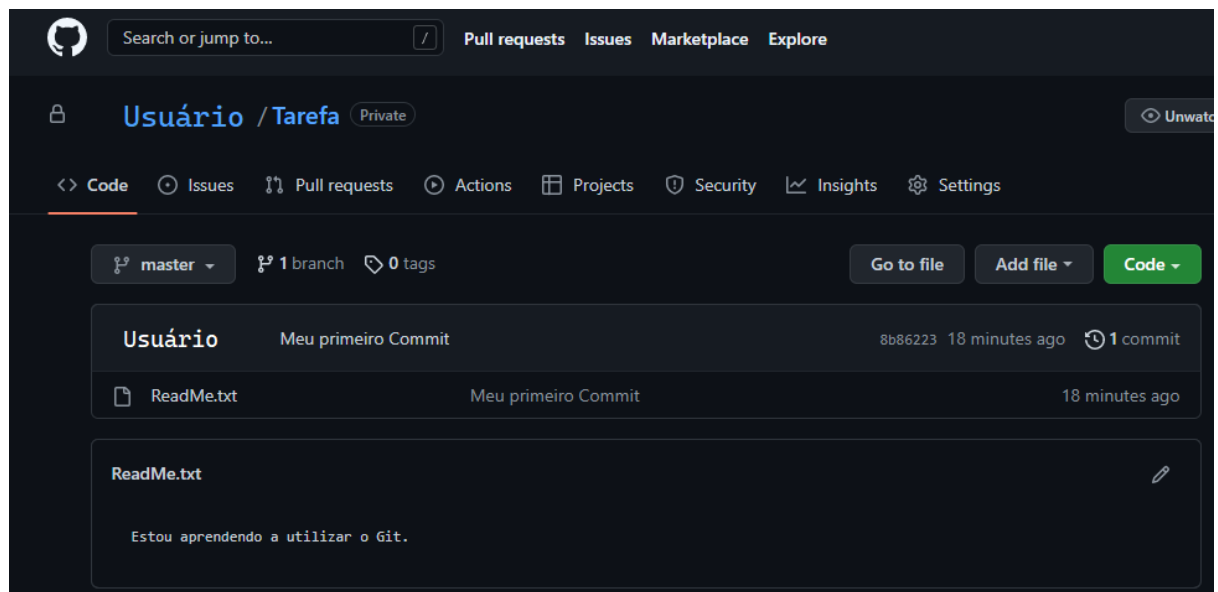
O link será semelhante ao da imagem abaixo.



14. Agora realizaremos o upload dos arquivos por meio do processo push do Git, utilizando o comando abaixo.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Projeto (master)
$ git push origin master
```

15. Recarregue a página de seu repositório no navegador e verifique se o arquivo foi enviado. Caso o processo tenha sido realizado da maneira correta, a página do repositório estará semelhante à imagem abaixo.



Clones

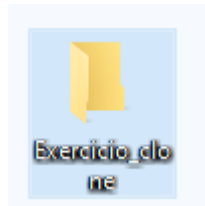
Após realizarmos um commit, os arquivos são enviados e armazenados no GitHub. Tendo isso em vista, podemos considerar que temos arquivos na nuvem e podemos acessá-los de qualquer lugar com Internet.

Considerando a possibilidade de acesso, devemos pensar em qual funcionalidade do GitHub permite realizar o acesso aos arquivos. Para isso, o sistema Git criou a funcionalidade de clonar o repositório. Com isso, podemos clonar o repositório, de maneira que baixemos os arquivos de modo que caso ocorra algum problema durante nossas correções, alterações ou implementações novas, possamos apenas deletar os arquivos locais e realizar um novo clone em nosso repositório.

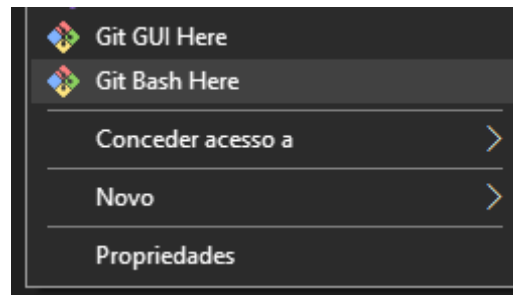
Exercícios sobre clones

Vamos realizar nosso primeiro clone neste exercício. Utilizaremos o repositório do exercício anterior. Para isso, siga os passos abaixo.

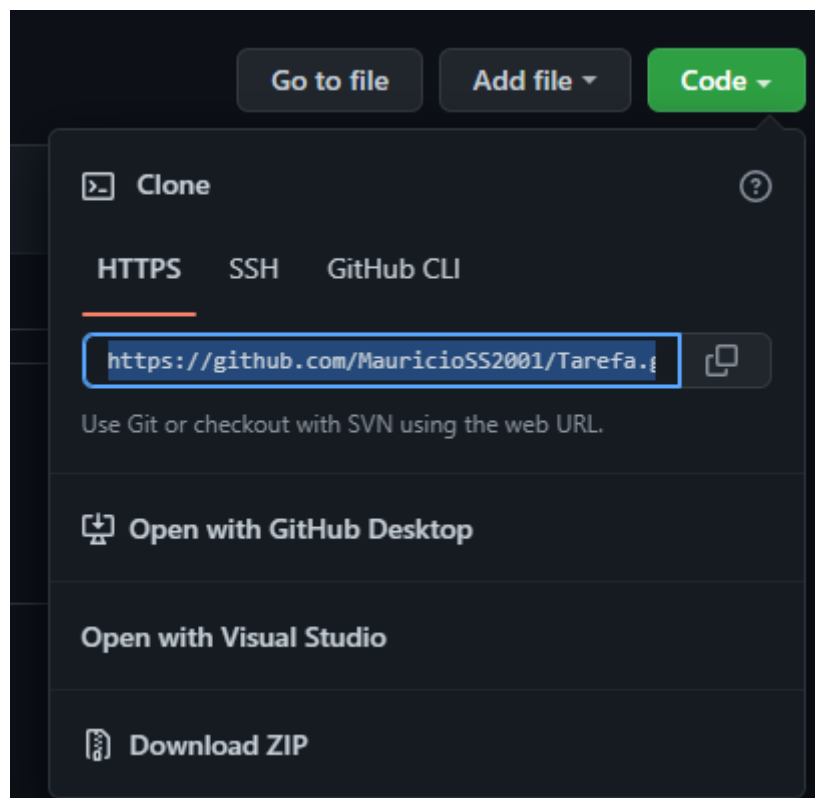
1. Crie uma pasta na sua área de trabalho, chamada "Exercicio_clone" (lembre-se de utilizar o caractere "underline" no nome da pasta).



- Abra a pasta e clique com o botão direito do mouse e selecione a opção “Hit Bash Here”



- Pegue o link de seu repositório no GitHub.



- Utilize o comando abaixo no prompt do seu Git Bash.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Exercicio_clone (master)
$ git clone https://github.com/Usuário/Tarefa.git
```

Se o Git retornar algo semelhante a imagem abaixo, o processo está correto.

```
hcmos@LAPTOP-IODL3DAR MINGW64 ~/Desktop/Exercicio_clone (master)
$ git clone https://github.com/Usuário/Tarefa.git
Cloning into 'Tarefa'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

5. Verifique na pasta se o arquivo foi baixado.

Tarefa de Casa

A tarefa de casa consiste em uma revisão dos conteúdos aprendidos em aula. Para isso, você deverá criar um novo repositório em seu GitHub com o nome “Tarefa”. Logo após sua criação, você deverá realizar um commit enviando um arquivo do tipo bloco de notas com seu nome por meio do Git Bash.