



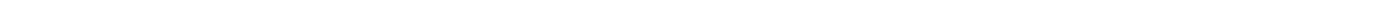
Git & GitHub

Público alvo: Desenvolvedores que queiram aprender a trabalhar com versionamento de código.

Pré-requisitos: Informática Básica.

Índice

Copyright	4
Equipe	5
Histórico de edições	5
Capítulo 01 - Git	6
O que é Git?	7
Histórico	7
Arquitetura	8
Lista de comandos	9
Instalando o Git	10
Exercícios	12
Capítulo 02 - GitHub	13
Histórico	14
Criando conta	15
Criando repositórios	15
Capítulo 03 - Enviando arquivos	18
Criando repositório local	19
Git init	20
Git add	20
Git status	21
Git commit	22
Git remote add origin	23
Git push	23
Exercícios	24
Capítulo 04 - Atualizando arquivos	25
Alterando arquivo	25
Especificando a alteração	27
Enviando para o GitHub	28
Histórico do GitHub	29
Capítulo 05 - Obtendo arquivos	30
Git clone	30
Git pull	31
Capítulo 6 - Branch e Merge	32
Branch	32
Merge	33



Copyright

As informações contidas neste material se referem ao curso de Lógica de Programação e estão sujeitas às alterações sem comunicação prévia, não representando um compromisso por parte do autor em atualização automática de futuras versões.

A Apex não será responsável por quaisquer erros ou por danos acidentais ou consequenciais relacionados com o fornecimento, desempenho, ou uso desta apostila ou os exemplos contidos aqui.

Os exemplos de empresas, organizações, produtos, nomes de domínio, endereços de email, logotipos, pessoas, lugares e eventos aqui apresentados são fictícios. Nenhuma associação a empresas, organizações, produtos, nomes de domínio, endereços de email, logotipos, pessoas, lugares ou eventos reais é intencional ou deve ser inferida.

A reprodução, adaptação, ou tradução deste manual mesmo que parcial, para qualquer finalidade é proibida sem autorização prévia por escrito da Apex, exceto as permitidas sob as leis de direito autoral.

Equipe

Conteúdos	Diagramação	Revisão
Ralf S. de Lima	Fernanda Pereira	Fernanda Pereira

Histórico de edições

Edição	Idioma	Edição
1 ^a	Português	Julho de 2020
2 ^a	Português	Dezembro de 2021

Capítulo 01 - Git

Esse capítulo irá abordar um pouco sobre o histórico, funcionalidades e a importância do desenvolvedor saber utilizar o Git em seus projetos. Essa ferramenta será utilizada em todas as aulas, sendo assim é extremamente importante que o aluno tenha os conceitos compreendidos e consiga aplicar na prática.



O que é Git?

Git é uma ferramenta adotada por muitas empresas para trabalhar com versionamento de código, isso garante realizar um backup dos dados em algum repositório, além de ter informações sobre quem realizou alguma determinada modificação, quando e o que foi alterado.

Talvez você já se deparou com uma situação, onde algum arquivo foi perdido ou deletado acidentalmente, esse tipo de coisa é normal, e por segurança é sem bom ter uma cópia, mas nem sempre realizamos essas cópias, e quanto menos a gente espera o arquivo pode ser perdido. O Git é uma maneira simples do desenvolvedor enviar projetos para algum repositório e garantir uma cópia dessas informações, além disso podemos ter um histórico detalhado das modificações realizadas, data, hora e quem realizou determinada alteração.

Histórico

O Git foi criado em abril de 2005 pelo engenheiro de software Linus Torvald, ele é conhecido pela criação do sistema operacional Linux. Inicialmente o projeto era voltado em atualizações para o Linux, a ideia do Git foi genial, fazendo com que os desenvolvedores disponibilizassem o uso do Git para qualquer espécie de projeto, independente da linguagem de programação ou sistema operacional.



Arquitetura

A arquitetura do Git é muito interessante, o desenvolvedor consegue enviar projetos e ter um histórico de cada mudança ocorrida, além de poder restaurar versões antigas dos projetos.

Inicialmente o Git enviava os projetos desenvolvidos para um servidor, onde os desenvolvedores poderiam ter acesso a tudo que foi implementado, porém com o tempo foram desenvolvidos sites com a possibilidade de criarmos repositórios e assim conseguimos gerenciar projetos de maneira online e gratuita.

Para o Git funcionar precisamos informar o local onde serão armazenados esses dados, em seguida o Git fica encarregado de enviar essas informações, disponibilizar dados de quem, quando e o que foi alterado, e também criar estruturas com cópias do projeto para diferente tipos de níveis de usuários que chamamos de branches. Talvez as branches sejam confusas, mas vamos explicar de maneira simples, imagine que você tenha um projeto com um back-end em Java Spring e o front em ReactJS, podemos estar criando uma branch que é uma ramificação do projeto para o front-end e outra para o back-end, isso deixa o projeto separado para cada branch visualizar sua estrutura. Sendo assim o back-end ficará em uma branch e o front-end em outra branch.

Por padrão o Git inicia cada projeto com uma branch chamada **master**, para quem está iniciando pode ter apenas ela, mas em projetos maiores é aconselhado separar algumas etapas em branches, assim facilita compreender cada módulo do projeto, além de cada desenvolvedor ou equipe ficar focado em uma branch e não atrapalhar o trabalho dos demais desenvolvedores.

Lista de comandos

Agora que você conhece um pouco sobre o Git, vamos compreender alguns comandos que utilizaremos nos próximos capítulos:

Comando	O que faz
git init	Inicializa um repositório local
git add	Seleciona os arquivos que serão realizadas as cópias
git commit	Cria um comentário sobre as alterações realizadas
git status	Exibe os arquivos selecionados que serão copiados
git push	Envia os arquivos do computador para o servidor
git pull	Obtêm os arquivos do servidor para o computador
git clone	Realiza uma cópia de algum diretório
git status	Retorna o nome da branch
git checkout -b minhaBranch	Cria uma nova branch e a seleciona
git checkout	Retorna para uma branch
git branch -d minhaBranch	Exclui uma branch

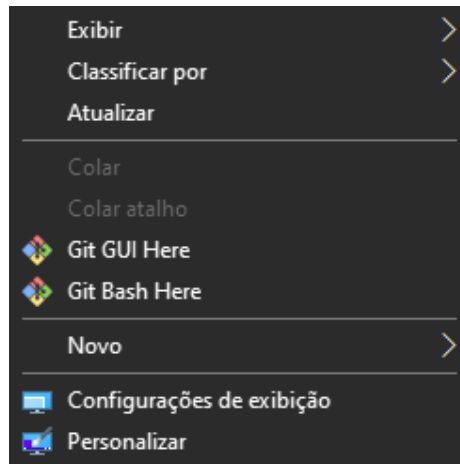
Há mais comandos que podemos utilizar, porém vamos nos focar nos principais, com essa base você poderá integrar tranquilamente uma equipe e gerenciar os projetos. Para mais informações você pode acessar a documentação oficial: <https://git-scm.com/doc>

Instalando o Git

Para podermos gerenciar nossos projetos, é necessário instalar o Git, para isso vamos baixar no site oficial <https://git-scm.com>

The screenshot shows the Git website homepage. At the top left is the Git logo with the tagline "--fast-version-control". To the right is a search bar. The main content area describes Git as a "free and open source" distributed version control system and mentions its "tiny footprint with lightning fast performance". To the right of this text is a diagram showing a branching model with stacks of code blocks connected by colored lines. Below the main text are four sections: "About" (advantages), "Documentation" (reference pages), "Downloads" (GUI clients), and "Community" (bug reporting). On the right side, a monitor displays the "Latest source Release 2.27.0" with a "Download 2.27.0 for Windows" button.

A instalação é bem simples, seu processo é apenas clicar em **next** e **finish**, você terá acesso ao Git quando clicar com o botão direito em algum local do sistema operacional, podendo ser na área do trabalho ou no interior de alguma pasta.



Teremos duas opções para utilizar o Git, sendo elas o **Git GUI Here** e o **Git Bash Here**, a diferença delas é que a versão **GUI** é uma interface gráfica para podermos gerenciar nossas aplicações, já o **Bash** é um terminal para utilizarmos os comandos do Git. Esse tutorial terá como foco a utilização de comandos através da opção **Bash**, porém fique à vontade em utilizar a versão **Gui**, é bom para o desenvolvedor ter escolhas para trabalhar com o Git.

Exercícios

Com base neste capítulo, desenvolva as questões propostas abaixo sobre os conceitos do Git:

- a) O que é Git?
- b) Quais as principais funcionalidades do Git?
- c) Como o Git pode auxiliar o desenvolvedor em seus projetos?
- d) O que faz os comandos:
 - `git init`
 - `git add`
 - `git commit`
 - `git push`
- e) Quando surgiu o Git? Por qual finalidade ele existiu?
- f) Quem criou o Git?
- g) O que é uma branch? Exemplifique sua resposta.

Capítulo 02 - GitHub

GitHub é um site que tem como finalidade servir de repositório, vale lembrar que o Git serve para gerenciarmos nossos projetos, já o GitHub é o local onde são armazenados os arquivos.

É uma distribuição *open source* que permite o desenvolver a criar projetos que estejam públicos (para qualquer um poder visualizar e baixar) ou privados (apenas usuários com permissão podem visualizar e baixar).

Essa é uma maneira simples para armazenarmos projetos e disponibilizarmos para qualquer desenvolvedor que tenha acesso a internet, além de disponibilizar o projeto, há um histórico para referenciar os desenvolvedores que fizeram determinadas modificações, além de data, hora, descrição da alteração e especificações do que foi editado.



Histórico

O projeto GitHub foi criado em 2008 em São Francisco nos Estados Unidos pelos desenvolvedores Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon, o repositório foi um sucesso, tanto que no primeiro ano de vida haviam cerca de quarenta e seis mil repositórios criados. Em 2010 foi informado que o Github estava hospedando um milhão de repositórios em um ano depois dobraram os repositórios.

O projeto estava crescendo, porém era necessário investir em infraestrutura e para desenvolvedores para melhorar suas funcionalidades, em 2015 o Github entra em uma rodada de investimento e conseguiu US \$250.000,00 dos investidores, que avaliaram a empresa em US \$ 2 bilhões.

Em 2018 a Microsoft compra o Github por US \$7.5 bilhões, a empresa diz que estava querendo investir com a comunidade *open source*, sendo assim estava adquirindo empresas do segmento e fundindo com alguns de seus produtos e serviços.



Criando conta

Para trabalhar com o Github precisamos criar uma conta, sendo assim acesse o site <https://github.com> e preencha o formulário que está na página inicial:

Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside 50 million developers.

Username

Email

Password

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

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

Criando repositórios

Para criar um repositório, basta acessar sua conta no GitHub e na lateral esquerda haverá um botão escrito **New**, clique nesse botão para podermos criar nosso repositório:

Search or jump to...

Repositories

New


Find a repository...

Quando clicado em **New**, será exibida a seguinte página:

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)


Owner * Repository name *

 ApexEnsino ▾ /

Great repository names are short and memorable. Need inspiration? How about **reimagined-octo-disco**?


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.

Skip this step if you're importing an existing repository.

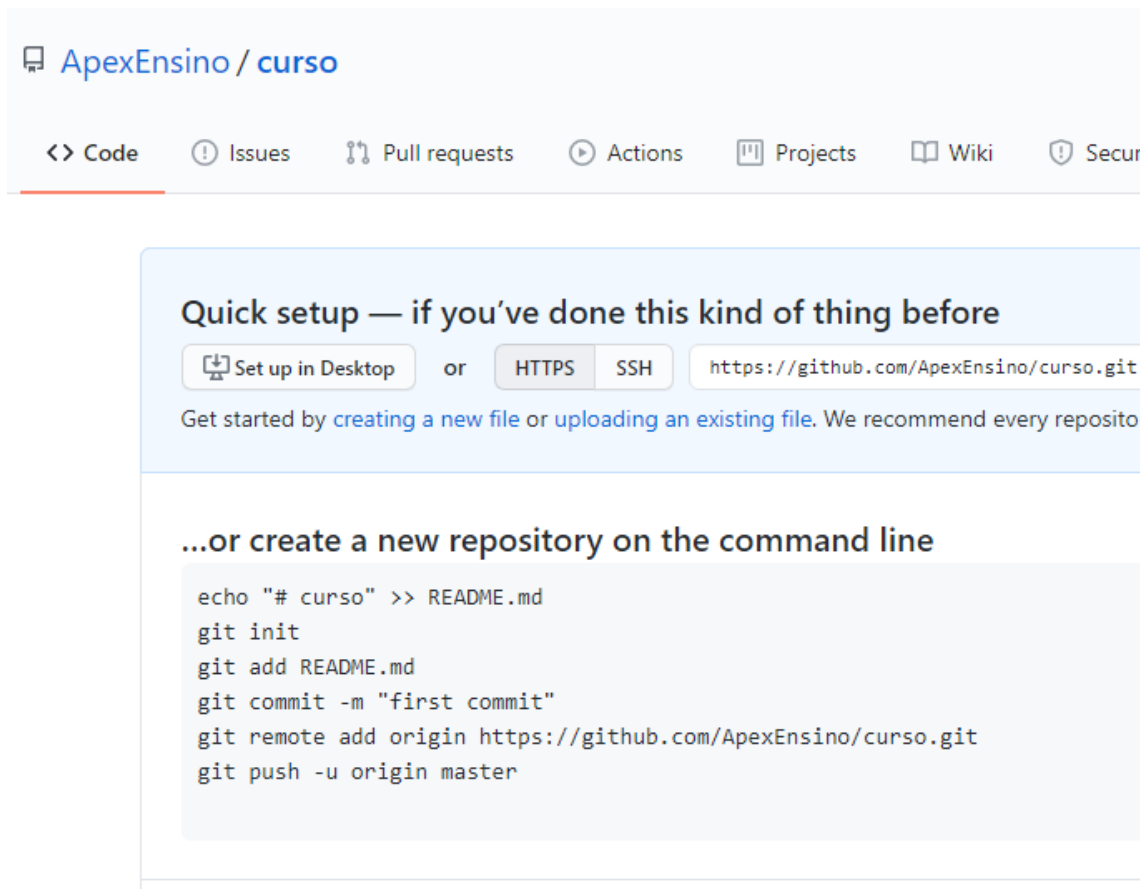
☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾ Add a license: **None** ▾ 

Create repository

Você pode informar um nome para o seu repositório, note que antes do nome do repositório irá aparecer o seu usuário, exemplo: `github.com/apexensino/curso`, o diretório pode ser escrito de qualquer maneira, porém é interessante evitar o uso de caracteres especiais e o uso de espaços. Há a possibilidade de adicionar uma breve descrição sobre o repositório, mas isso é totalmente opcional, em seguida você pode informar se este repositório criado será público ou privado, e por fim é possível iniciar o repositório com um arquivo **README**, apenas para que ele contenha algum arquivo de testes. Clique em **Create repository** e você terá seu repositório criado.

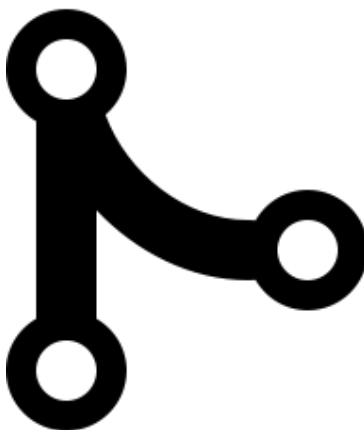
Agora teremos uma nova página como mostra a imagem abaixo:



Na parte superior temos o caminho necessário para que nosso computador consiga enviar os arquivos para o Github, neste caso a url utilizada é: <https://github.com/ApexEnsino/curso.git>, abaixo você terá um breve tutorial para enviarmos os arquivos para o GitHub.

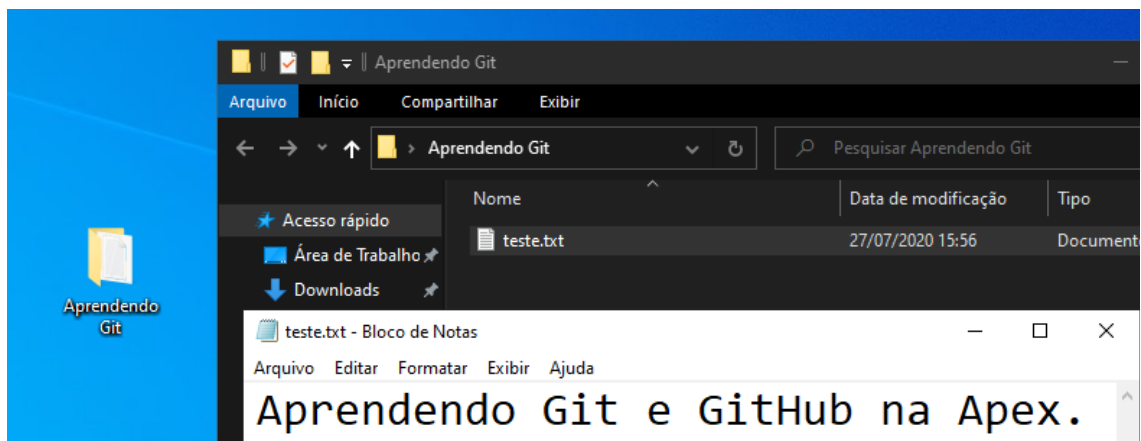
Capítulo 03 - Enviando arquivos

Para compreender o envio de arquivos para o Git, vamos exemplificar o envio de um arquivo de texto, porém podemos enviar qualquer tipo de arquivo, sendo texto, imagem, áudio, vídeo, entre outras extensões.

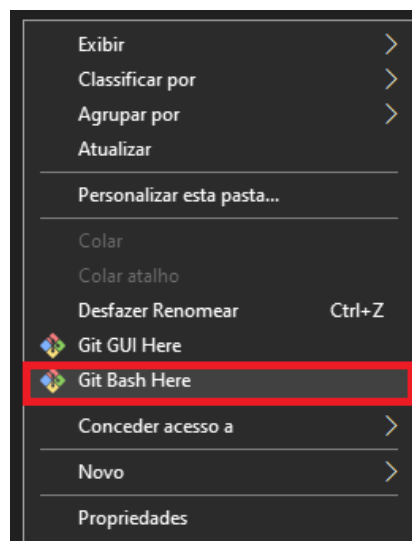


Criando repositório local

Iniciamos essa etapa do curso criando uma pasta em qualquer local do seu computador, nomeie como quiser, para esse exemplo será nomeado como **Aprendendo Git**, dentro dessa pasta será criado um arquivo de texto chamado **teste**, neste arquivo será adicionada a frase: **Aprendendo Git e GitHub na Apex.**

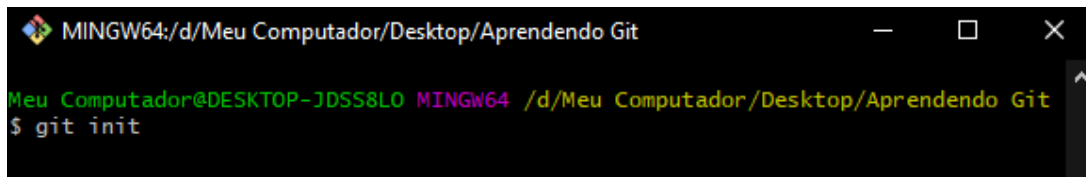


Criada a pasta e o arquivo de texto, vamos abrir o terminal do Git, para isso clique com o botão direito dentro da pasta **Aprendendo Git** e selecione a opção **Git Bash Here**:



Git init

O comando **git init** terá a finalidade de informar que nossa pasta **Aprendendo Git** será utilizada para receber e enviar projetos através de um repositório do GitHub.



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
$ git init
```

Git add

O comando **git add** serve para selecionar os arquivos que serão enviados para o repositório, podemos adicionar individualmente como por exemplo: **git add teste.txt**, esse comando ele apenas selecionará um arquivo, porém se você quiser adicionar todos os arquivos basta utilizar o ponto, exemplo **git add .**



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git add .
```

Provavelmente não irá exibir nenhuma mensagem, mas podemos verificar se os arquivos realmente foram enviados utilizando o comando **status**.

Git status

O comando **git status** exibe os arquivos que foram selecionados para envio através do **git add**, em verde estão os arquivos que serão enviados para o repositório do GitHub, em vermelho os arquivos que não foram selecionados:



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git status
On branch master

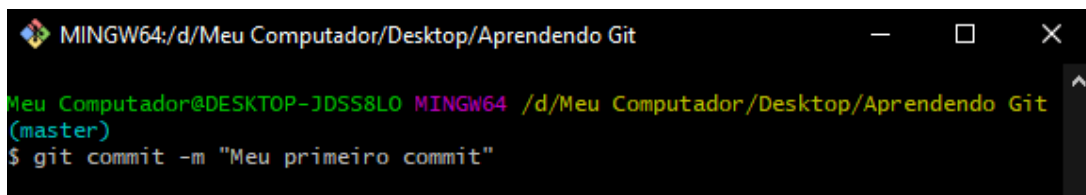
No commits yet

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

Neste exemplo apenas o arquivo **teste.txt** está em verde, porém lembre-se que você pode selecionar arquivos individualmente, e os que não foram selecionados ficarão em vermelho, sendo assim apenas os arquivos na cor verde serão enviados para o repositório no GitHub.

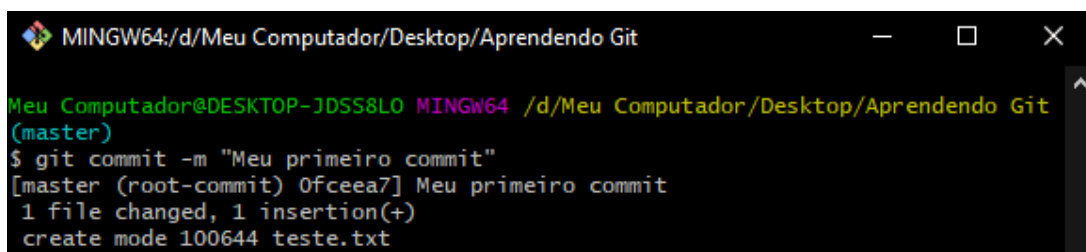
Git commit

O comando **git commit** será utilizado para realizarmos uma descrição de até 50 caracteres sobre o que estamos enviando para o GitHub, você pode informar a inclusão, alteração e remoção de arquivos. Para esse exemplo será utilizado o comando **git commit -m "Meu primeiro commit"**, o **-m** significa mensagem.



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git commit -m "Meu primeiro commit"
```

Quando executar o comando, você poderá ver uma mensagem informando a quantidade de arquivos que foram adicionadas a essa mensagem, quando por enviado para GitHub, esses arquivos terão essa mensagem para que sejam diferenciados de arquivos que não tiveram mudanças.

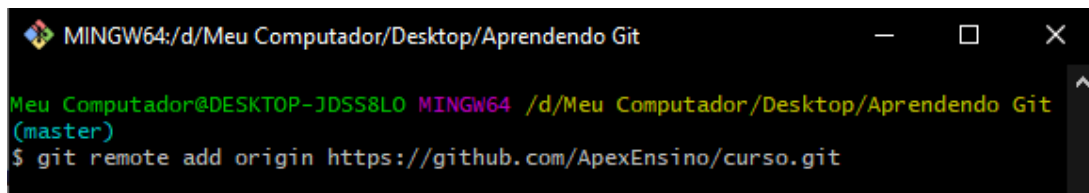


```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git commit -m "Meu primeiro commit"
[master (root-commit) 0fcee7] Meu primeiro commit
1 file changed, 1 insertion(+)
create mode 100644 teste.txt
```

Git remote add origin

O comando **git remote add origin**, especifica o diretório do GitHub que será utilizado para enviar ou selecionar os arquivos, em outras palavras, será o caminho entre os arquivos locais e os arquivos remotos.

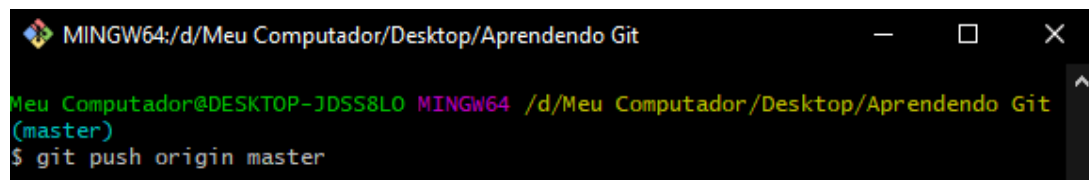
Você terá acesso a essa informação quando cria o repositório, neste exemplo o caminho é: **git remote add origin** <https://github.com/ApexEnsino/curso.git>, pode utilizar esse comando em seu terminal e executar:



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git remote add origin https://github.com/ApexEnsino/curso.git
```

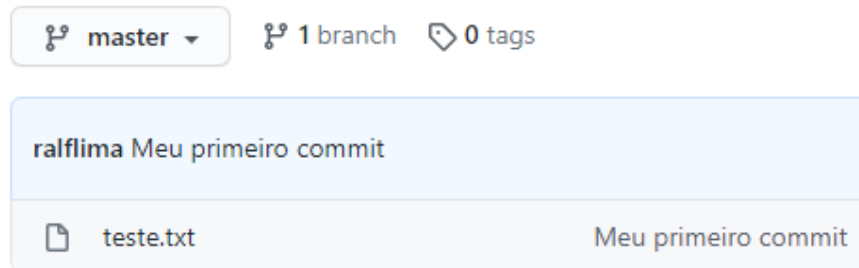
Git push

O comando **git push** é utilizado para podermos enviar os arquivos do computador para o nosso repositório do GitHub, para isso utilizamos o comando **git push origin master**, fazendo com que nossos arquivos sejam enviados para a branch master.



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git push origin master
```

Agora você pode acessar o seu repositório do GitHub, veja que o seu arquivo de texto **teste.txt** estará lá



Parabéns! Você conseguiu enviar seu primeiro arquivo para o GitHub, nos próximos capítulos será abordado sobre a atualização de arquivos e também o processo de importação.

Exercícios

Com base neste capítulo, iremos recapitular os passos para enviar um projeto pela primeira vez, sendo assim implemente as seguintes ações:

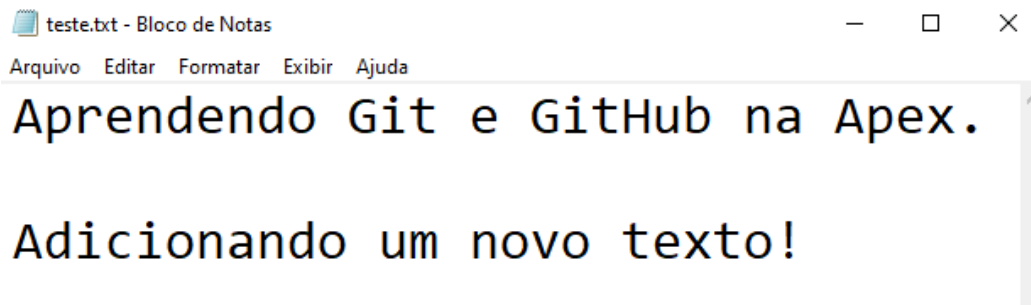
- Crie um diretório no GitHub chamado **apex**
- Em seu computador crie uma pasta chamada **apex**
- Crie um arquivo txt e escreva alguma coisa
- Envie esse arquivo para o diretório **apex** que está no GitHub

Capítulo 04 - Atualizando arquivos

Alterações nos projetos é algo muito normal, sendo assim vamos implementar o exemplo do capítulo anterior atualizando o arquivo de texto.

Alterando arquivo

Inicialmente podemos alterar texto que está no arquivo **teste.txt** e salvar, fique à vontade em fazer a alteração que achar melhor.



Vamos utilizar o comando **git status** para verificar a situação do nosso projeto, lembrando que a cor verde significa que o arquivo foi selecionado e vermelho que não está selecionado, sendo assim não será enviado para o GitHub.

```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   teste.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Note que temos em vermelho **modified: teste.txt**, agora se efetuarmos a seleção do arquivo com o comando **git add teste.txt** e em seguida **git status**, veja o resultado:

```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git add teste.txt
```

```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   teste.txt
```

Agora que temos um arquivo pronto para ser enviado ao GitHub, sempre que houver alguma alteração, será necessário utilizar o comando **git add**.

Especificando a alteração

Nessa etapa precisamos informar o que houve com o projeto, adicione um texto com até 50 caracteres utilizando o comando **git commit**, neste exemplo utilizaremos a estrutura: **git commit -m "Alterando arquivo de texto"**.

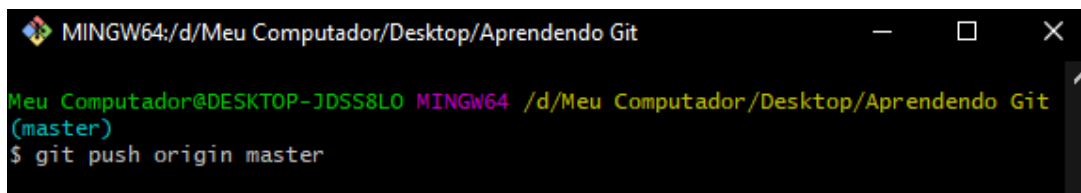
```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git commit -m "Alterando arquivo de texto"
```

Assim que realizar o **commit** será exibida uma mensagem informando os arquivos que foram alterados e receberão o texto enviado no **commit**.

```
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git commit -m "Alterando arquivo de texto"
[master 1b2168a] Alterando arquivo de texto
1 file changed, 3 insertions(+), 1 deletion(-)
```

Enviando para o GitHub

Agora podemos efetuar o envio para o GitHub através do comando **git push origin master**.



```
MINGW64:/d/Meu Computador/Desktop/Aprendendo Git
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Aprendendo Git
(master)
$ git push origin master
```

Acessando o diretório no site do GitHub, podemos atualizar a página e notar que houve uma alteração:

ralflima Alterando arquivo de texto

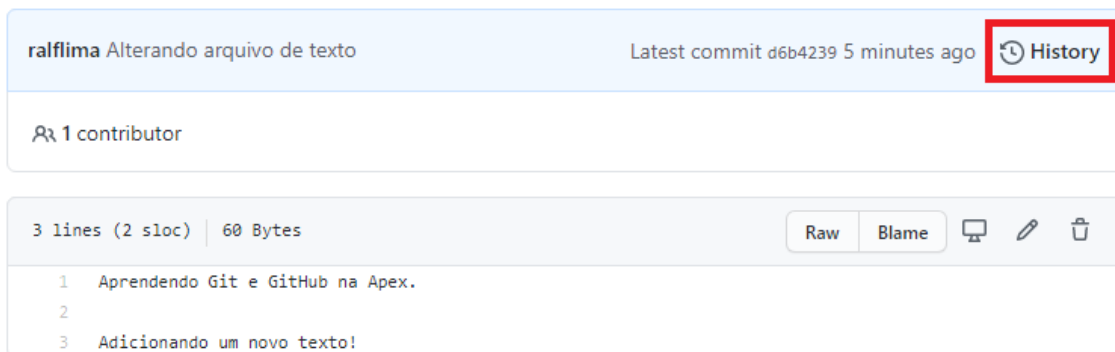
teste.txt

Alterando arquivo de texto

Sempre que houver a necessidade de alterar algum arquivo, remover ou adicionar, basta seguir esses passos. Assim sei GitHub sempre estará atualizado e disponível para você e sua equipe a trabalhar com a última versão do projeto

Histórico do GitHub

Você pode verificar tudo o que aconteceu com os arquivos de seus projetos, para isso basta clicar no nome de um arquivo, você irá conseguir visualizar a estrutura escrita, e na parte superior terá uma opção chamada **History**, onde teremos acesso a tudo que foi adicionado, removido ou alterado em cada **commit**.



The screenshot shows a GitHub file view for a file named "Alterando arquivo de texto" by user "ralflima". The latest commit is d6b4239, made 5 minutes ago. A red box highlights the "History" button in the top right corner. Below the file name, it shows "1 contributor". The file content is displayed with line numbers 1 to 3. Line 1 contains "Aprendendo Git e GitHub na Apex.", line 2 is empty, and line 3 contains "Adicionando um novo texto!". At the top right of the content area, there are buttons for "Raw", "Blame", and icons for a monitor, edit, and delete.

ralflima Alterando arquivo de texto Latest commit d6b4239 5 minutes ago **History**

1 contributor

3 lines (2 sloc) | 60 Bytes Raw Blame

```
1 Aprendendo Git e GitHub na Apex.
2
3 Adicionando um novo texto!
```

Alterando arquivo de texto
ralflima committed 15 minutes ago

Meu primeiro commit
ralflima committed 1 hour ago

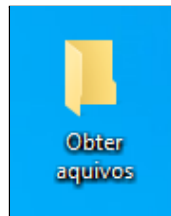
Cada **commit** realizado será criado um histórico, clicando na mensagem é possível ver o código com tons avermelhados e esverdeados, a parte em verde são as adições e em vermelho as exclusões.

Capítulo 05 - Obtendo arquivos

Vamos imaginar que você está em outro computador e precisa ter acesso aos arquivos que estão em um repositório do GitHub, como fazer isso? Há duas opções que iremos abordar neste capítulo.

Git clone

O comando **git clone** faz uma cópia do que está em um determinado repositório do GitHub, para exemplificar vamos criar uma pasta em nosso computador, podemos chamar de **Obter arquivos**.



Abra essa pasta e execute o **Git Bash Here**, onde será aberto o terminal para realizarmos os comandos do Git. Digite o comando **git clone**, seguido do diretório que deseja copiar.

```
Meu Computador@DESKTOP-JDSS8LO MINGW64 /d/Meu Computador/Desktop/Obter arquivos  
$ git clone https://github.com/ApexEnsino/curso
```

Agora você terá acesso a todos os arquivos desse repositório, porém você apenas fez uma cópia, não há vínculo entre essa pasta e o repositório, para termos acesso a realizar **commits**, precisamos utilizar um outro comando.

Git pull

O comando **git pull** serve para criarmos um vínculo com nosso repositório no GitHub, para isso podemos excluir os arquivos baixados anteriormente com o comando **git clone**.

Os processos que você deverá fazer são semelhantes a criar um repositório local, separamos em três etapas:

- **git init** - para iniciar uma comunicação entre sua pasta local e o repositório
- **git remote add origin** - para selecionar o repositório
- **git pull origin master** - para baixar os arquivos que estão no GitHub

```
Meu Computador@DESKTOP-JD5S8LO MINGW64 /d/Meu Computador/Desktop/Obter arquivos
master)
$ git pull origin master
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 1), reused 8 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), 729 bytes | 2.00 KiB/s, done.
From https://github.com/ralflima/curso
* branch      master      -> FETCH_HEAD
* [new branch] master      -> origin/master
```

Agora você terá acesso aos arquivos atualizados, além de poder realizar alterações e enviar ao GitHub tranquilamente. O comando **git pull** faz com que os arquivos que estejam em seu computador sejam atualizados com o que está no repositório do GitHub.

Capítulo 6 - Branch e Merge

Neste capítulo iremos abordar a utilização de branches e merges, algo muito importante na criação de alguma funcionalidade em um sistema. Essas duas opções do Git, permitem aos desenvolvedores evitarem sobrepor modificações no projeto principal.

Branch

Branches ou ramificações, servem para separar partes do projeto, assim é possível trabalhar mais facilmente através de módulos sem mexer na branch principal, vale lembrar que todo o repositório Git criado, há uma branch chamada Master, essa é a branch principal, onde estará todo o projeto.

Vamos supor que há um sistema onde há os módulos: financeiro, administrativo e logístico, muitas vezes é possível criar uma ramificações de um módulo e modificar especificamente aquela parte do projeto, depois de modificar é possível unificar com o projeto principal.

Em projetos onde há uma equipe grande ou uma estrutura robusta, é uma opção interessante para manter a organização da aplicação.

Para criar uma branch é muito simples, basta utilizar o comando: **git branch nomeDaBranch** e pronto, sua branch estará criada e pronta para utilizar.

Caso haja muitas branches e desejar alterar, utilize o comando: **git checkout nomeDaBranch**, o nome da branch deverá ser especificada a branch que você deseja selecionar.

Merge

Merge é a técnica utilizada para unirmos as ramificações, imagine que você tenha a branch Master, porém em um determinado momento foi criada a branch Financeiro, onde foram implementadas funções para o módulo de finanças, agora queremos unificar as branches, para isso é necessário estar com a branch master selecionada, para isso usaremos o comando: **git checkout master**, em seguida iremos utilizar os seguintes comandos:

1. `git add diretórioContendoOsArquivosModificados`
2. `git commit -m "mensagem"`
3. `git merge nomeDaBranchDoFinanceiro`

Note que basta apenas selecionarmos os arquivos, realizar um commit e por fim utilizar o comando merge e especificar a branch que queremos unir.

Vale ressaltar que o merge não funciona apenas com a branch master, também podemos utilizar outras branches para fazer a junção. Os comandos utilizados anteriormente sempre precisam ser executados na branch que irá receber aquela estrutura, então tome cuidado ao executar essa ação.

