

# **GIT e GITHUB**

**Para uso dos alunos Digicad**

**Elton e Artur  
Front End**

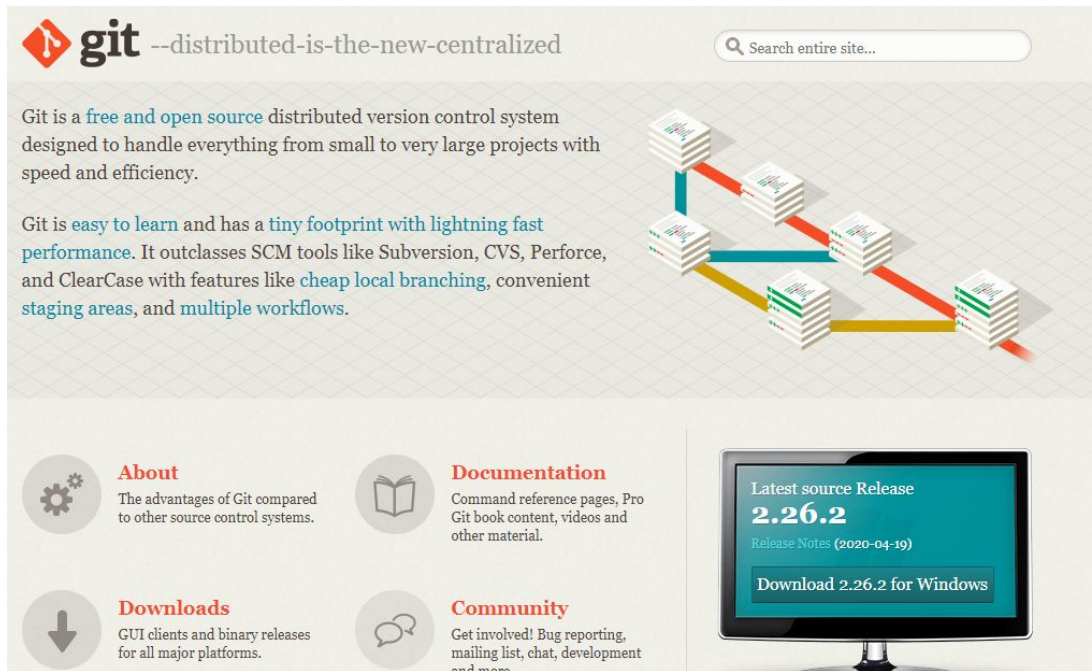
=====

Instalação: download  
Acessar o google e pesquisar por GIT

Acessar: <https://git-scm.com/>

Clicar em Downloads ou mesmo na janela que já aparece a versão para Windows..

Para quem usa Linux, algumas distribuições já vem com o Git instalado.



Clicamos agora na opção WINDOWS..



E já será iniciado o Download do arquivo do Git.  
IMPORTANTE: Verifique se seu computador é 32 ou 64 bits.

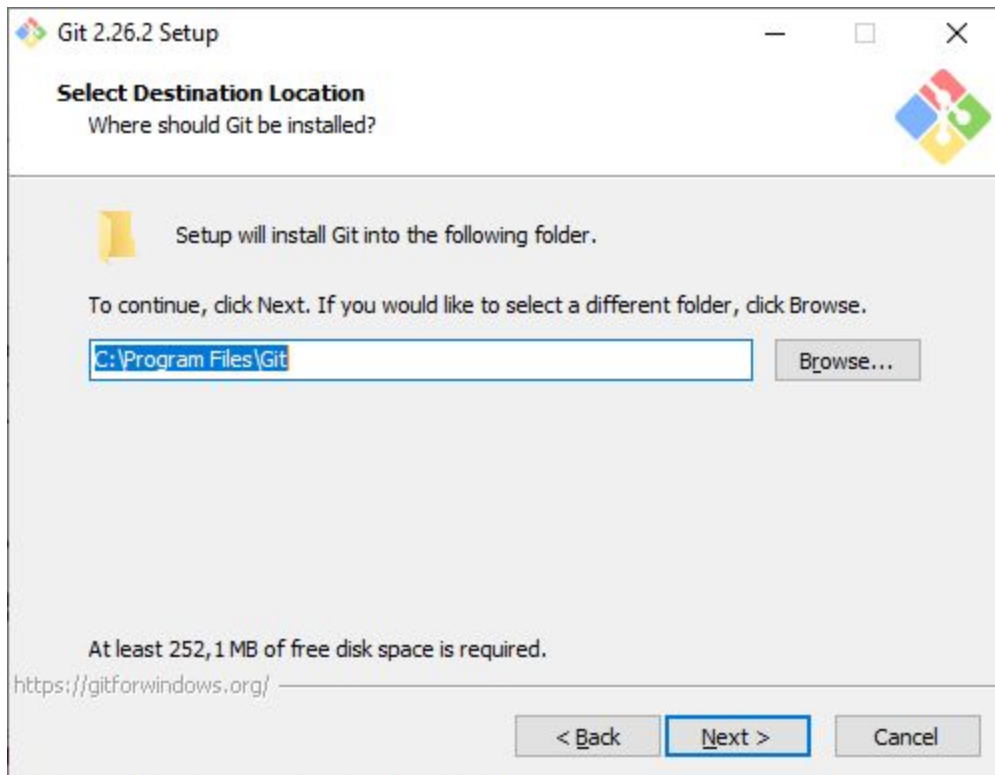
Execute a instalação.....

=====

Aceitar os termos:



Definir o local



## Selecionar os componentes...

Git 2.26.2 Setup

**Select Components**  
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ☐ Additional icons
  - ☐ On the Desktop
- ☒ Windows Explorer integration
  - ☒ Git Bash Here
  - ☒ Git GUI Here
- ☒ Git LFS (Large File Support)
- ☒ Associate .git\* configuration files with the default text editor
- ☒ Associate .sh files to be run with Bash
- ☐ Use a TrueType font in all console windows
- ☐ Check daily for Git for Windows updates

Current selection requires at least 251,7 MB of disk space.

<https://gitforwindows.org/>

< Back   Next >   Cancel

## Definir o nome na Pasta

Git 2.26.2 Setup

**Select Start Menu Folder**  
Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

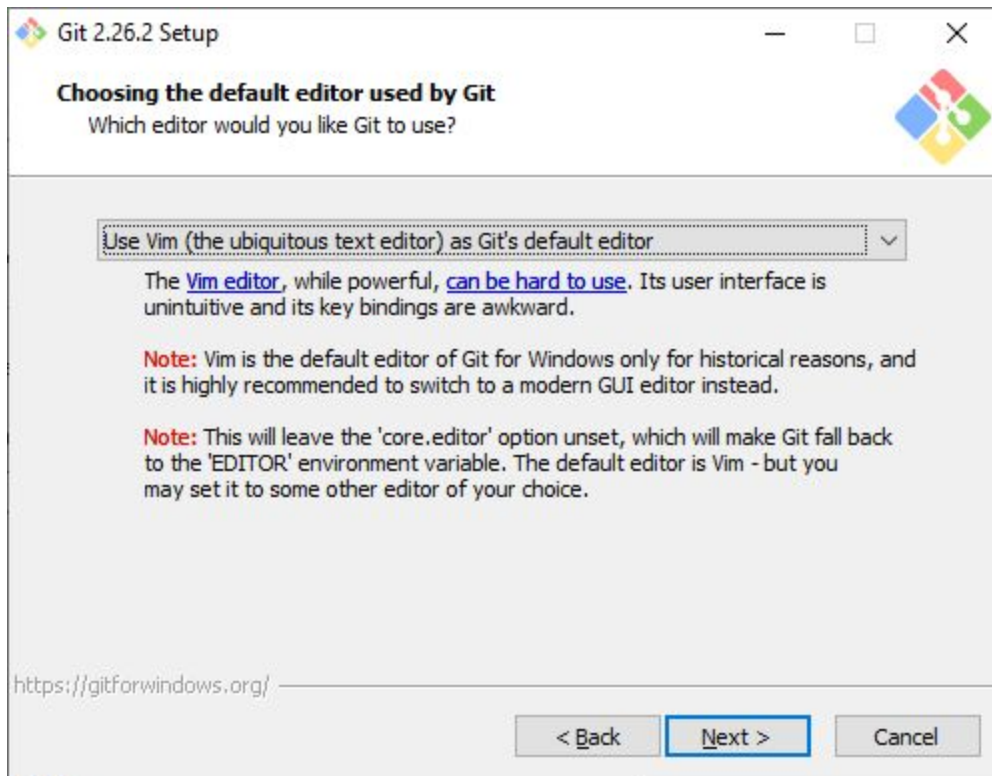
Git   Browse...

☐ Don't create a Start Menu folder

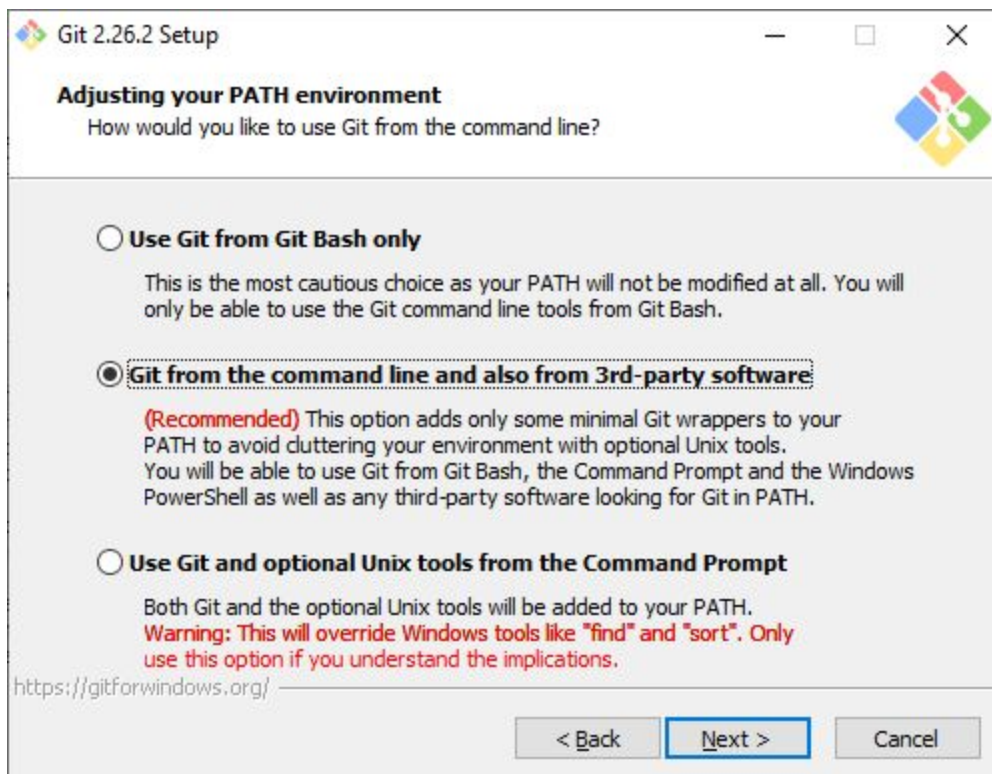
<https://gitforwindows.org/>

< Back   Next >   Cancel

Defina o editor padrão de código para ser usado.

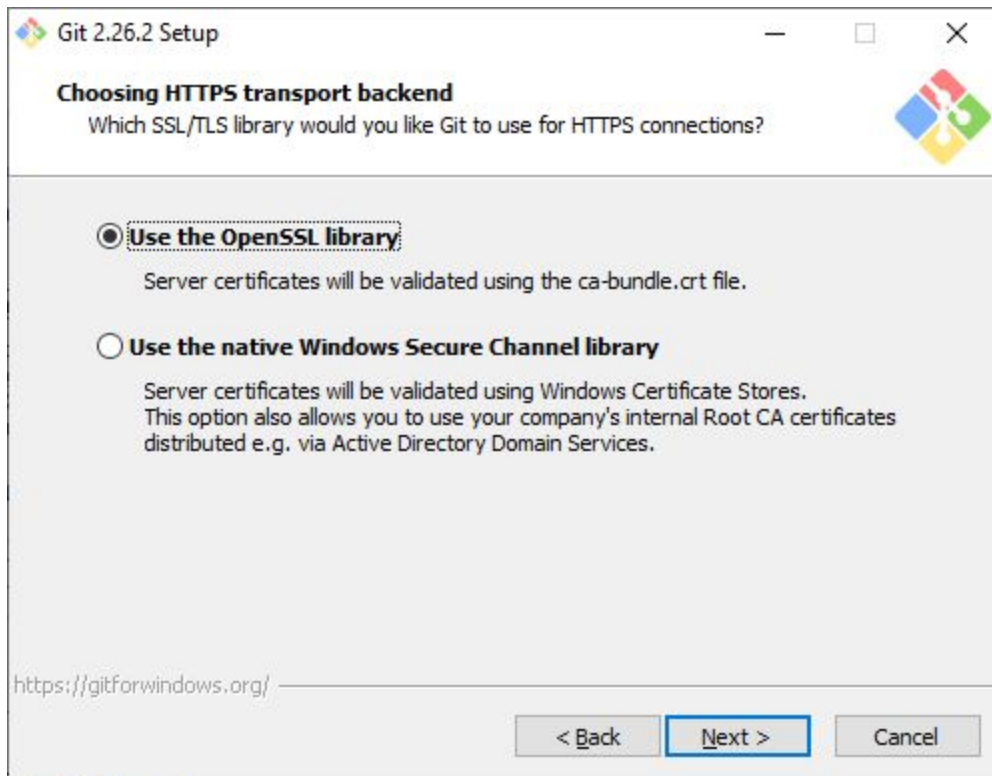


Definir a forma de uso da linha de comando..

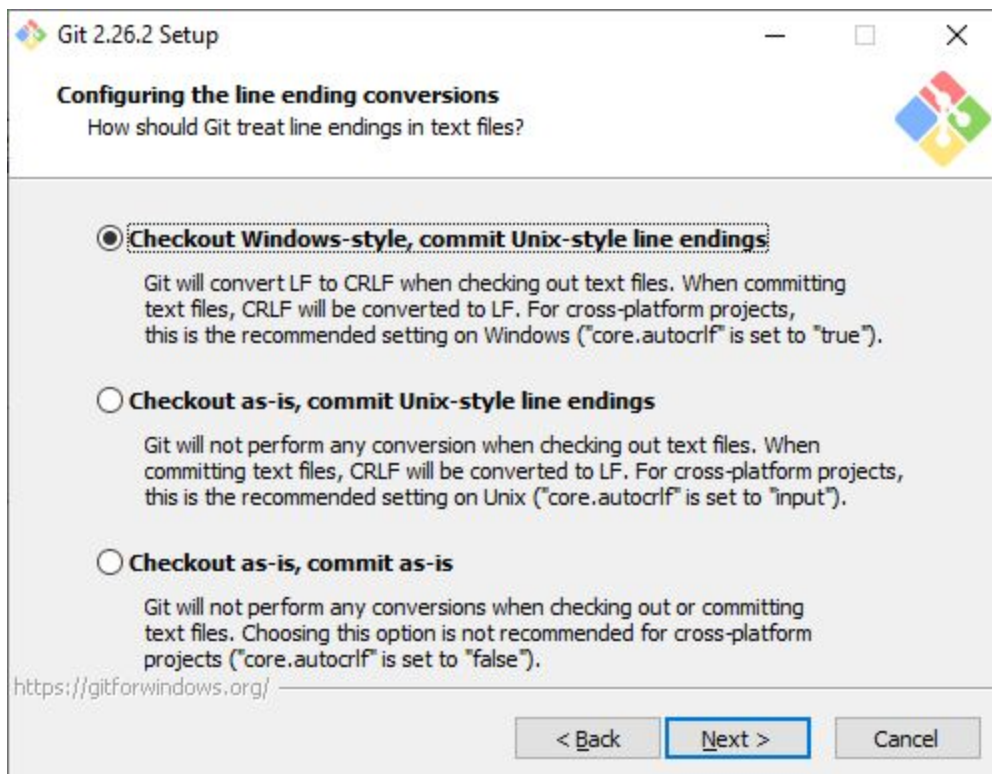




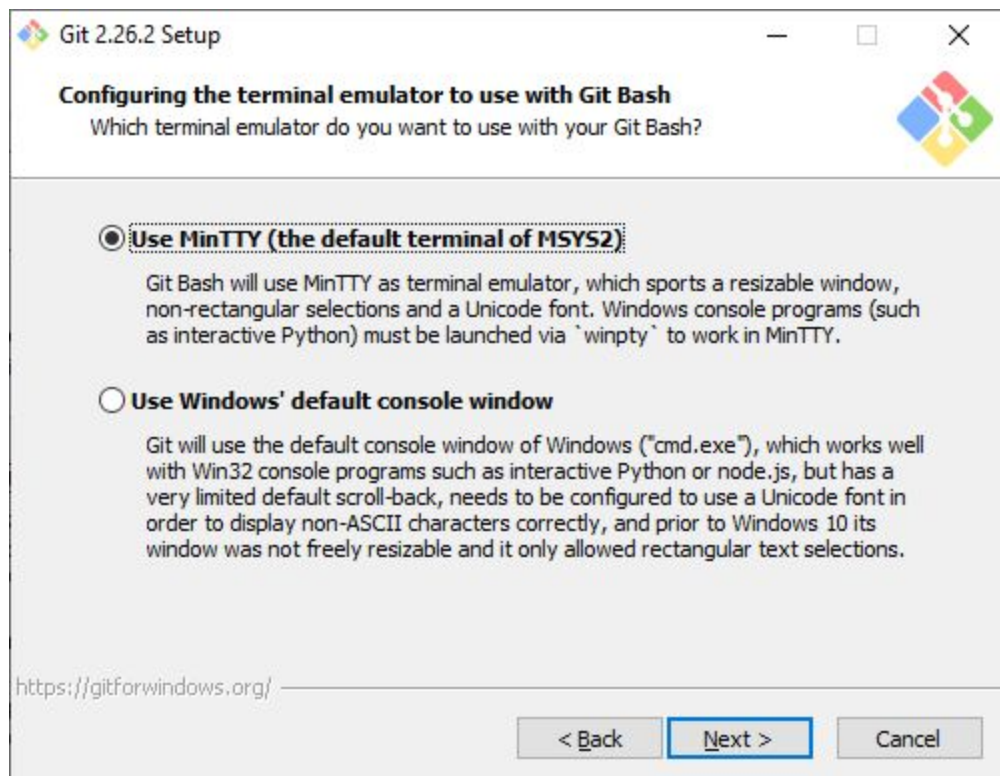
Marcar a opção padrão HTTPS transporte



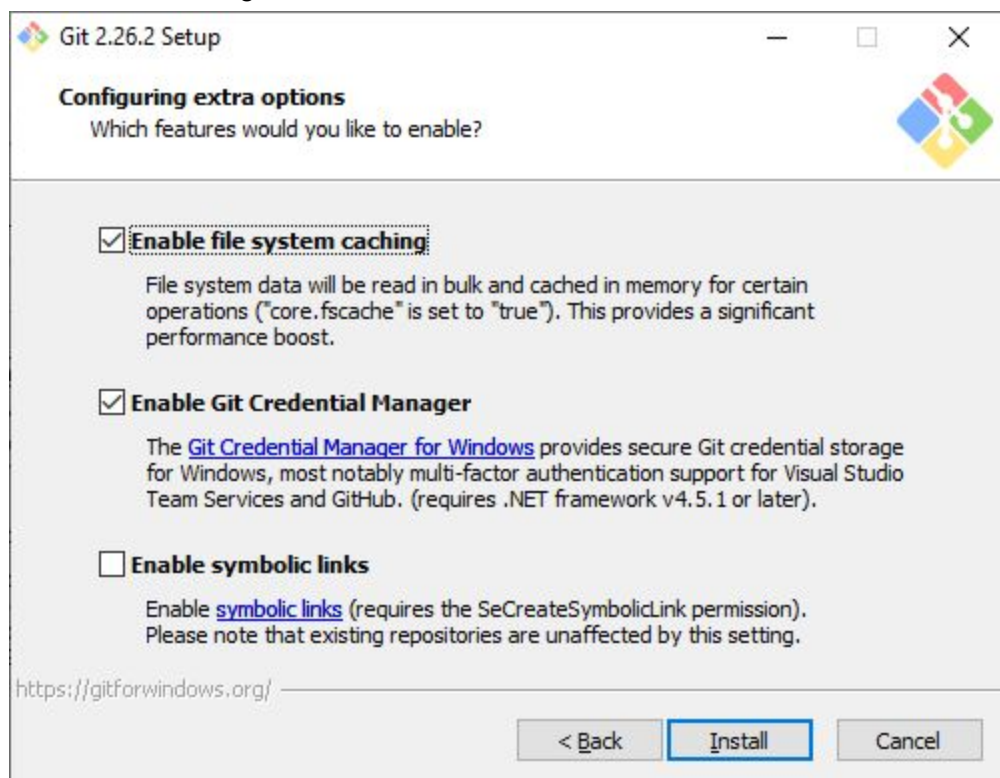
Marcar a forma de chekc in / check out



Definir a forma de emular o BASH de comandos.

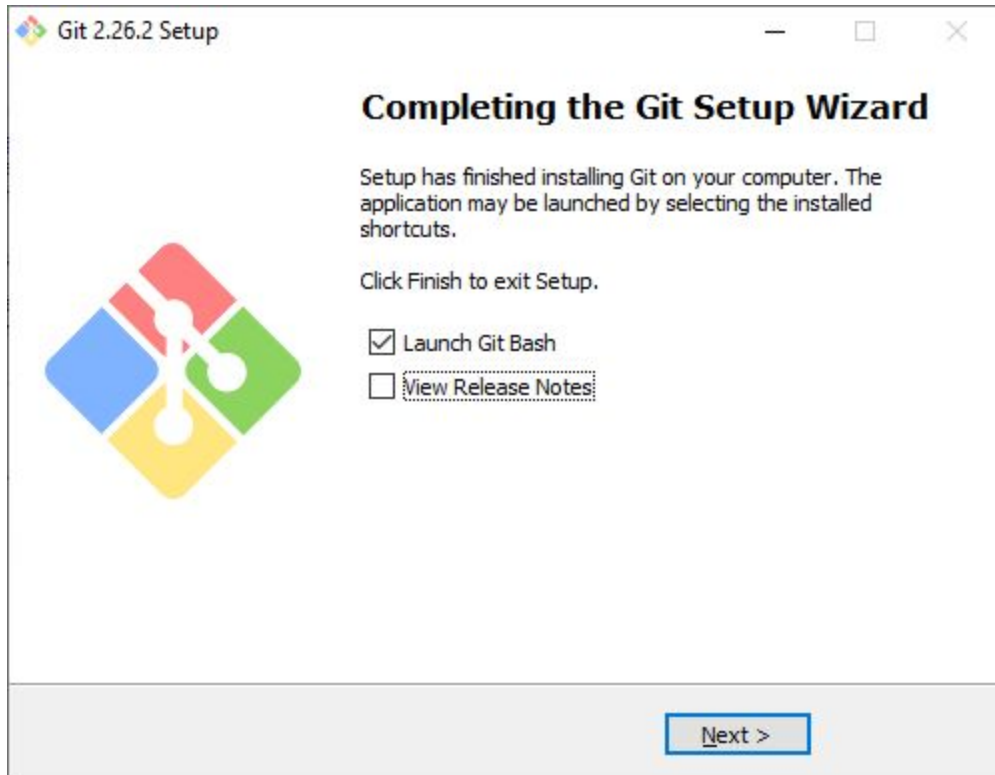


E uso de cache e gerenciamento com credenciais..



As opções marcadas são fundamentais para todo o trabalho a ser realizado.

Vamos marcar a opção de executar o Git Bash e desmarcar a opção de View Release Notes



Ao iniciar, CLICK no logo.

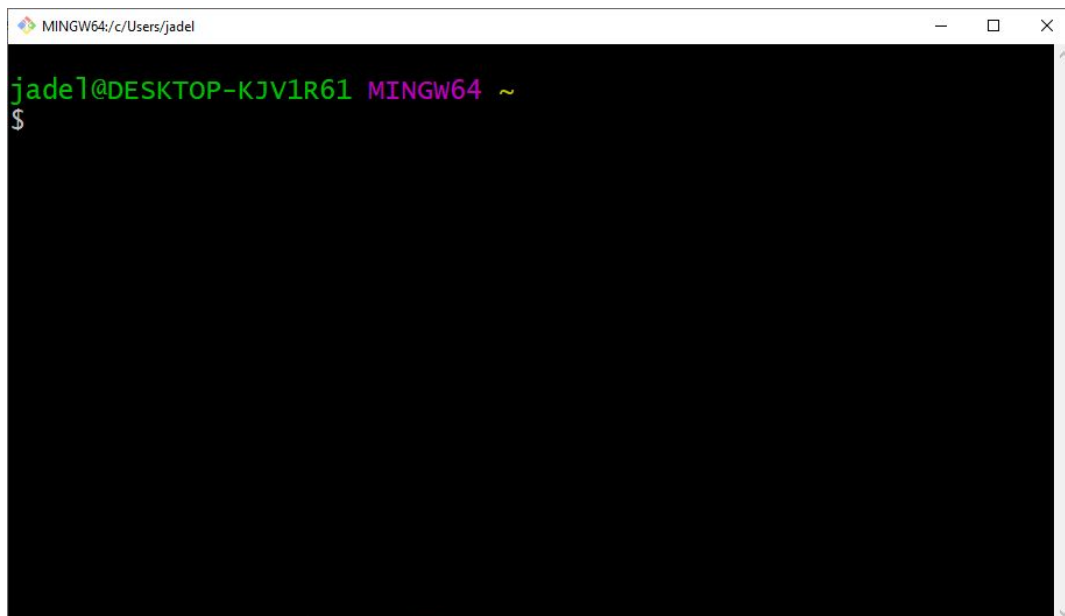
Click em OPTIONS

Click em TEXT

Na aba FONT click em SELECT e procure uma Font maior..

=====

O que eu tenho basicamente é um terminal GIT. Esse é o Git BASH





=====

Aqui posso executar alguns comandos, por exemplo:

**\$ dir**

**\$ git --version**

Para saber quais os comandos do Git, digite o comando

**\$ git**

=====

Faremos a alteração do Editor padrão do Git.

Vamos alterar para o SublimeText.

Procuramos o SublimeText na lista de Programas do botão Iniciar

Clicamos com o botão da direita e selecionamos a opção ABRIR LOCAL DO ARQUIVO

Se vc perceber, está o caminho do atalho, então clicamos com o botão da direita para ter acesso as PROPRIEDADES.

Agora, copiamos o caminho do arquivo que está no campo DESTINO. Copiar com as ASPAS.

Agora, no terminal GIT, digite o comando a seguir, e não esqueça de colocar uma ASPAS SIMPLES no nome do caminho do editor

**\$ git config --global core.editor "C:\Program Files\Sublime Text 3\sublime\_text.exe"**

Para saber se está OK, vamos digitar o comando para mostrar as configurações do git.

**\$ git config --list**

=====

Vamos acessar nosso projeto HTML 01, Hino Nacional, a pasta do projeto é: FE\_html\_01

É este projeto que vamos versionar, que vamos controlar.

Vamos abrir o projeto no SublimeText e vamos abrir o arquivo index.html

=====

Pelo terminal do Git, vamos acessar este diretório onde está o nosso projeto.

No meu caso, o diretório é: Trabalho\Jadelson\Testes\FE\_html\_01

Portanto, vamos acessar este diretório pelo Git.

**\$ cd "C:\Users\jadel\OneDrive\Documentos\Trabalho\Jadelson\Testes\FE\_html\_01"**

E o terminal deve apresentar o diretório antes do prompt

```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
$
```

=====

Vamos executar a instrução ls para confirmar o conteúdo desta pasta

```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
$ ls
bandeira_brasil.png index.html teste_folha.css
```

=====

### **Iniciando o Repositório:**

Agora vamos informar ao Git para que inicialize nesse diretório o meu repositório:

```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
$ git init
Initialized empty Git repository in C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01/.git/
```

```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

A partir de agora toda movimentação ocorrida dentro desta pasta, o Git fará o controle

- O que foi modificado
- Quem modificou
- Quando modificou

=====

Como saber se o Git está enxergando o meu repositório. Será que ele está controlando o meu Diretório?

Então usamos o comando

**\$ git status**

Será informado quais arquivos estão sendo controlados.

=====

Na tela, como resultado podemos observar algumas informações. A fundamental neste momento é perceber que os arquivos em vermelho ainda não estão sob o controle de versão do Git..

```
MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ ^C

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    bandeira_brasil.png
    index.html
    teste_folha.css

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

### Ainda não está monitorando....

=====

Precisamos fazer o GIT monitorar as mudanças de meus arquivos..

Para definir que o meu arquivo index.html seja monitorado pelo Git...

**\$ git add index.html**

Mas quero que todos os arquivos desta pasta sejam monitorados, então:

**\$ git add .**

=====

Conferindo ..... Executando git status a resposta agora é outra

**\$ git status**

```
MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ dir
bandeira_brasil.png  index.html  teste_folha.css

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git add .

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   bandeira_brasil.png
    new file:   index.html
    new file:   teste_folha.css

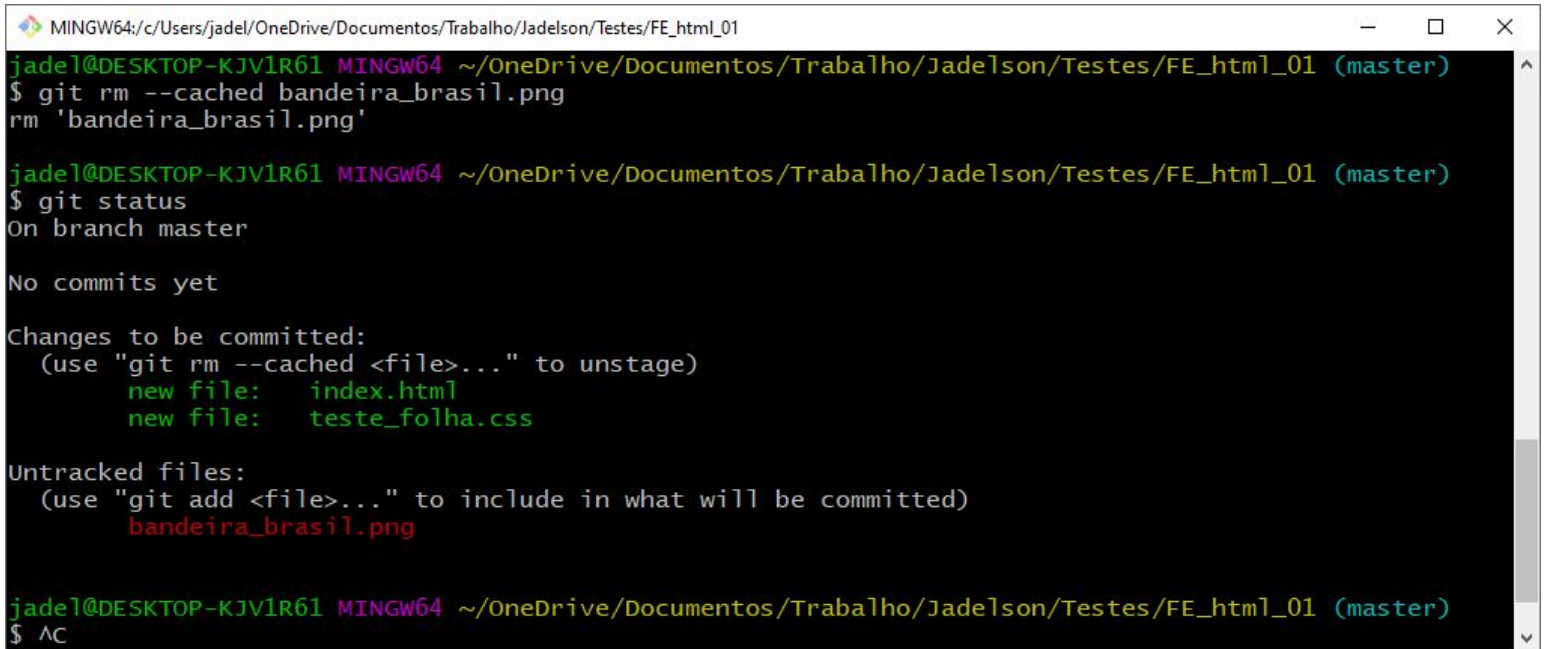
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

=====

Importante: o arquivo bandeira\_brasil.png não precisa ser monitorado.

**\$ git rm --cached bandeira\_brasil.png**

**\$ git status**

A terminal window titled 'MINGW64:/c/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01' showing the output of 'git rm --cached bandeira\_brasil.png' and 'git status'. The status shows 'On branch master', 'No commits yet', and 'Changes to be committed: new file: index.html, new file: teste\_folha.css'. It also lists 'Untracked files: bandeira\_brasil.png'. The prompt '\$ ^C' is at the bottom.

```
mingw64~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git rm --cached bandeira_brasil.png
rm 'bandeira_brasil.png'

mingw64~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html
        new file:   teste_folha.css

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

mingw64~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ ^C
```

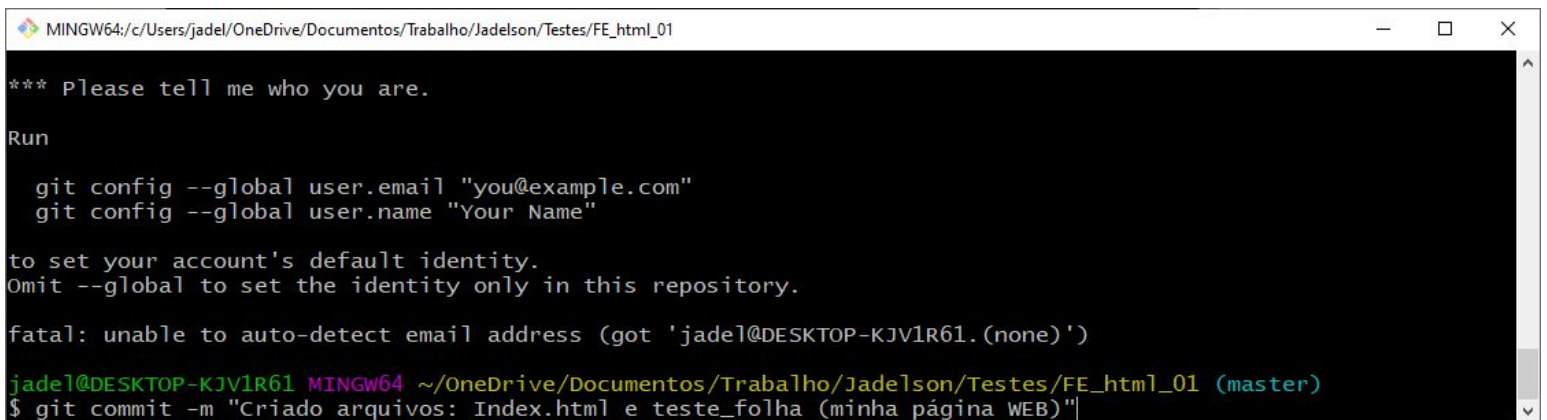
=====

Mas, apesar de todas as alterações realizadas, não validamos ainda, ou seja, não “comitamos”

Então..

**\$ git commit -m "Criado arquivos: Index.html e teste\_folha (minha página WEB)"**

Se der o seguinte erro, significa que o GIT não possui a referência da pasta com um usuário

A terminal window titled 'MINGW64:/c/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01' showing the output of 'git commit -m "Criado arquivos: Index.html e teste\_folha (minha página WEB)". It displays an error message: 'fatal: unable to auto-detect email address (got 'jadel@DESKTOP-KJV1R61.(none)')'. The prompt '\$ git commit -m' is at the bottom.

```
mingw64~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'jadel@DESKTOP-KJV1R61.(none)')

mingw64~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git commit -m "Criado arquivos: Index.html e teste_folha (minha página WEB)"
```

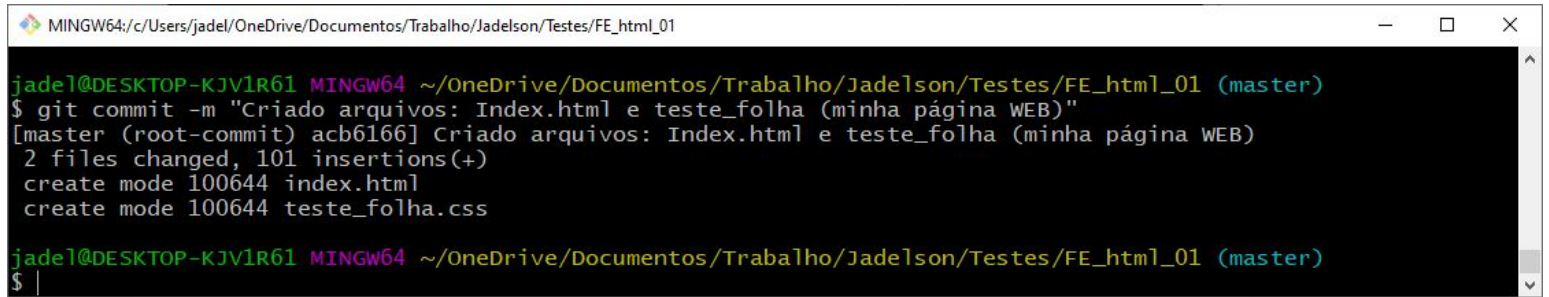
Então, precisamos adicionar o email e o nome do usuário...

**\$ git config --global user.email "meuemail@exemplo.com"**

E também o nome do usuário..

**\$ git config --global user.name "Meu Nome"**

Com estas correções executamos o commit e teremos....

A screenshot of a terminal window titled 'MINGW64: c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01'. The terminal shows a user named 'jadel' at a desktop 'KJV1R61' using 'MINGW64' to execute a git commit. The commit message is 'Criado arquivos: Index.html e teste\_folha (minha página WEB)'. The output shows the commit was successful with hash 'acb6166', indicating 2 files changed and 101 insertions. The files 'index.html' and 'teste\_folha.css' were created with mode 100644. The prompt returns to the user's shell.

```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git commit -m "Criado arquivos: Index.html e teste_folha (minha página WEB)"
[master (root-commit) acb6166] Criado arquivos: Index.html e teste_folha (minha página WEB)
2 files changed, 101 insertions(+)
create mode 100644 index.html
create mode 100644 teste_folha.css

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ |
```

Informa:

2 arquivos criados

Total de 101 linhas adicionadas (novas)

O Git monitora linha a linha...

Se executar

**\$ git status**

Não há nada a comitar.

=====

Nesse momento temos:

- Git sabe que este é um repositório
- E não existe nenhuma alteração a ser salva



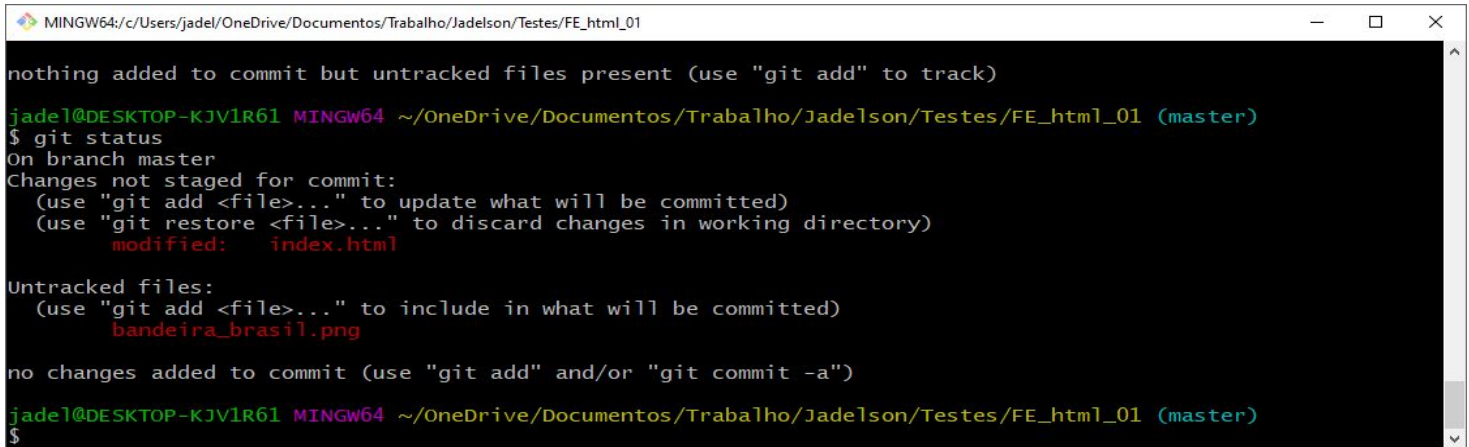
Vamos fazer uma alteração no arquivo index.html

No final, antes do </body> vamos adicionar um parágrafo: **<p>Fim</p>**

**<!-- controlado pelo Git 01 -->**

Se salvarmos e digitar..

**\$ git status**

A terminal window titled 'MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01' showing the output of the 'git status' command. The output indicates that there are untracked files, specifically 'bandeira\_brasil.png', and that the 'index.html' file has been modified. The prompt shows the user is on the 'master' branch.

```
nothing added to commit but untracked files present (use "git add" to track)
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (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:   index.html

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

no changes added to commit (use "git add" and/or "git commit -a")
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

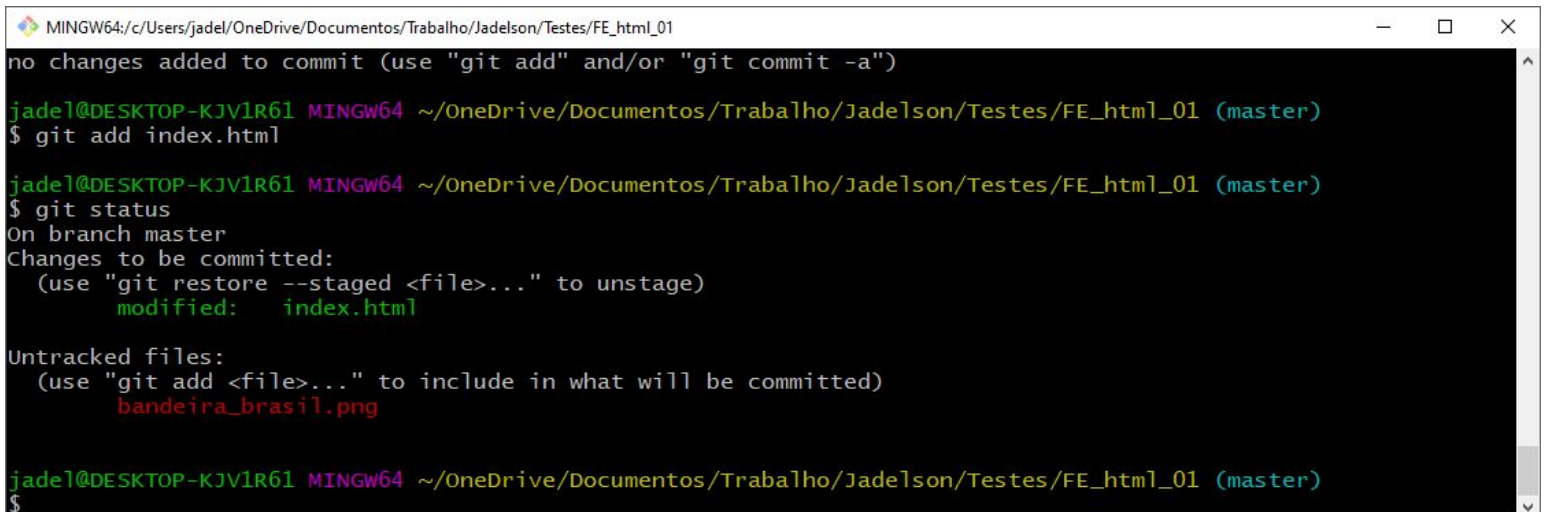
Vemos que existe uma informação sobre modificação.

Precisamos executar a instrução para adicionar novamente o index.html.....

**\$ git add index.html**

E executamos o git status para ver o que ocorreu

**\$ git status**

A terminal window titled 'MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01' showing the output of the 'git add index.html' and 'git status' commands. The output shows that 'index.html' is now staged for commit, while 'bandeira\_brasil.png' remains untracked. The prompt shows the user is still on the 'master' branch.

```
no changes added to commit (use "git add" and/or "git commit -a")
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git add index.html
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

É possível ver que precisamos também executar o commit..

**\$ git commit -m "index.html: adicionado parágrafo"**

```
MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git commit -m "index.html: adicionado parágrafo"
[master c937bb6] index.html: adicionado parágrafo
1 file changed, 1 insertion(+), 1 deletion(-)

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

=====

Vendo o Histórico de alterações.

**\$ git log**

```
MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git log
commit dd08783892bbf05c237d6c16648312e482125e30 (HEAD -> master)
Author: Jadelson <jadelsons@gmail.com>
Date:   Wed May 6 12:10:06 2020 -0300

    index.html: adicionado comentário

commit acb61663559701e543d2db745d1a8a2526171850
Author: Jadelson <jadelsons@gmail.com>
Date:   Wed May 6 11:49:47 2020 -0300

    Criado arquivos: Index.html e teste_folha (minha página WEB)

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ |
```

Entendendo a Tela:

Todo commit tem um Hash: commit **dd08783892bbf05c237d6c16648312e482125e30**  
É uma informação única de todo commit

O autor: nome e email

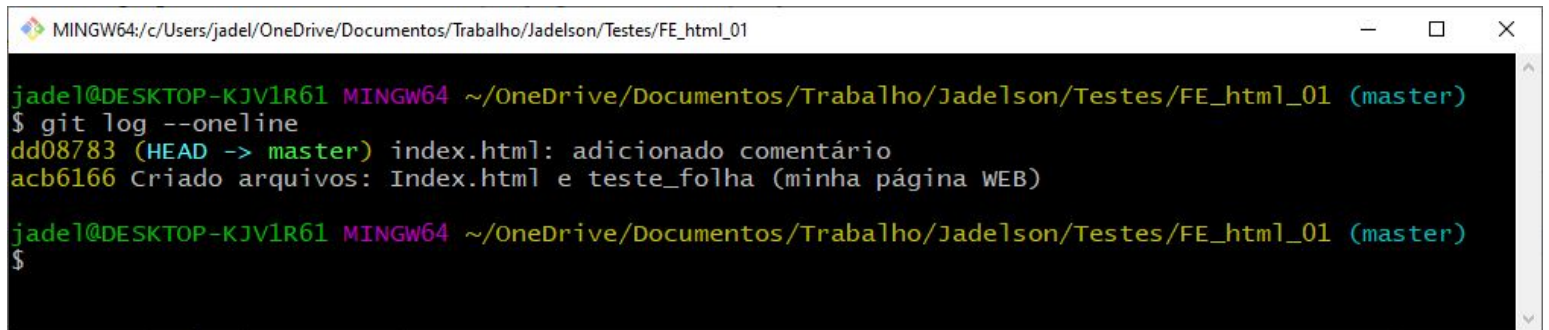
Data do commit

=====

Outras formas de visualizar os commits

Em linha: mostra apenas os arquivos comitados e o que foi feito

**\$ git log --oneline**



```
MINGW64: c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git log --oneline
dd08783 (HEAD -> master) index.html: adicionado comentário
acb6166 Criado arquivos: Index.html e teste_folha (minha página WEB)

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

Para visualizar mais informações:

**\$ git log -p**

Ele mostra commit a commit o que foi alterado, ou seja, o que foi realizado.

=====

Quais são as formas de exibir do commits.

Site: <https://devhints.io>

E pesquise por **git**

Em seguida, entre na opção **git-log**

Teremos uma série de opções possíveis, por exemplo, para git log

=====

Vamos testar algumas delas:

mostrar somente o último commit

**\$ git log -1**

Mostrar 2 último commits

**\$ git log -2**

Mostrar somente os dois último commits, só que em ordem crescente de data

**\$ git log --reverse -2**

Ou

**\$ git log --max-count=1**

**\$ git log --max-count=2**

**\$git log --reverse --max-count=2**

O --reverse aplica uma reevrsão na ordem das datas para a quantidade de linhas que desejamos trazer.

**\$ git log -3 --reverse**

Significa trazer os 3 ultimos commits, em ordem crescente de data

=====

Para obtermos todos os commits, mas pulando o último commit executado...

**\$ git log --skip=1**

Vamos pular os dois últimos

**\$ git log --skip=2**

Vamos pular o último commit e inverter a ordem mostrada.

**\$ git log --skip=1 --reverse**

=====

Existe ainda o parâmetro --pretty que possibilita especificar alguns detalhes do commit

Somente os hash

**git log --pretty="format:%H"**

Hash e autor

**\$ git log --pretty="format:%H %an"**

Hash, autor e email

**\$ git log --pretty="format:%H %an %ae"**

Hash, autor, email e data.

**\$ git log --pretty="format:%H %an %ae %aD"**

=====  
Vamos executar um git status e vamos ver a situação de nosso repositório.

## **\$ git status**

Percebemos que o arquivo bandeira-brasil.png sempre aparece em vermelho como um arquivo que, apesar de não inserido no GIT, ainda continua sendo “perseguido” por ele.

=====  
.gitignore

Fazer o GIT ignorar arquivos.

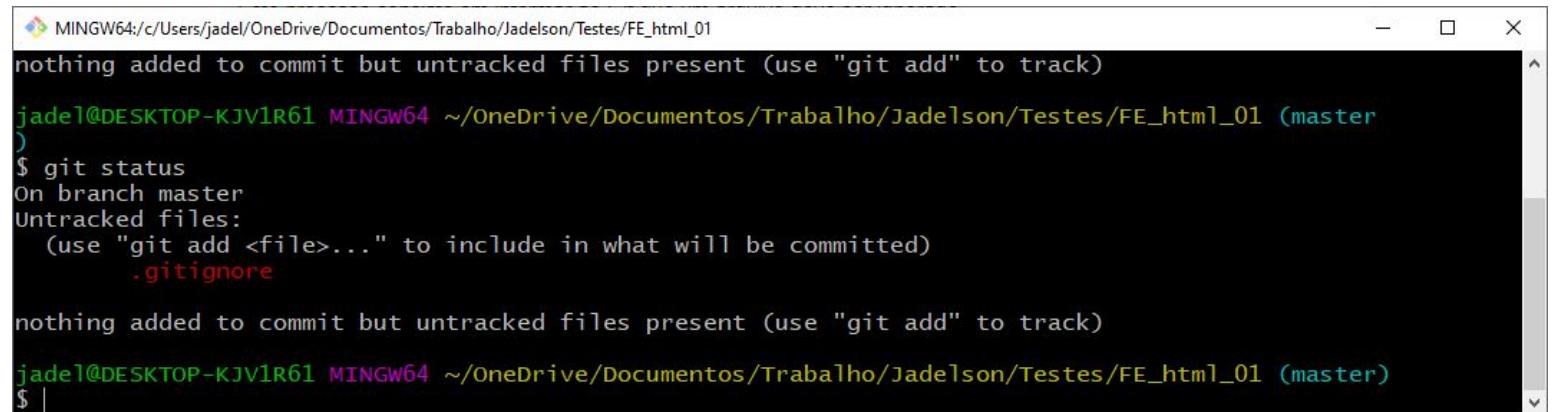
Este processo consiste em informar ao Git que um arquivo deve ser ignorado.

Vamos abrir nosso projeto e vamos criar, na raiz, um arquivo de nome **.gitignore**

- 1) Abrir o projeto
- 2) Criar o arquivo **.gitignore**
- 3) Incluir no arquivo o nome do arquivo a ser ignorado, no caso, bandeira\_brasil.png

=====  
Testando:

## **\$ git status**

A screenshot of a terminal window with a black background and white text. The window title is 'MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01'. The output of the 'git status' command is shown, indicating that the file '.gitignore' is untracked and being tracked. The prompt shows the user is on the 'master' branch.

```
MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
nothing added to commit but untracked files present (use "git add" to track)

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ |
```

Vemos que o arquivo bandeira\_brasil.png foi ignorado.

Mas vemos que ele não está monitorando o gitignore. Então vamos monitorar o gitignore.

## **\$ git add .gitignore**

## **\$ git commit -m “adicionando .gitignore”**

Podemos visualizar com git log



=====

Adicionando pastas ao gitignore: Vamos criar uma pasta e vamos informar ao git para ignorar o que existe lá dentro.

- 1) No projeto, vamos criar a pasta “**mensagens**”
- 2) Se vc estiver usando o sublime, click sobre a pasta principal e New Folder, na parte de baixo digite o nome MENSAGENS
- 3) Vá no git e digite **git status**
- 4) Vemos que não aparece o diretório porque não existe nenhum arquivo dentro dele.
- 5) Vamos criar o arquivo **skype.txt** dentro desta pasta criada, clicando com o botão da direita sobre ela e escolhendo NEW FILE
- 6) Dentro desse arquivo digitamos qualquer valor na linha 1
- 7) No Git, vamos executar o **git status**
- 8) Veremos que ele informa que existe uma pasta não monitorada

```
MINGW64: c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master
nothing to commit, working tree clean

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    mensagens/

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

- 9) Vamos inserir esta pasta dentro do arquivo .gitignore assim: **mensagens/**
- 10) Se dermos um git status veremos que ele informa que o .gitignore foi modificado e a pasta “mensagens” já não aparece mais. Ele reconhece o .gitignore mas não está controlando sua versão

```
MINGW64: c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    mensagens/

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (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:   .gitignore

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

- 11) Informa tb que precisamos adicionar e comitar este .gitignore
- 12) Então vamos adicionar e comitar
- 13) **\$ git add .gitignore**
- 14) **\$ git commit -m “Adicionando: pasta para ignore”**
- 15) Vamos ver o status: **\$ git status**
- 16) Vamos ver o log: **\$ git log**

=====

Pergunta: Quando executar um commit?

Não existem regras, existem recomendações

1. Comitar apenas códigos sem erros ou falhas
2. Quando corrigir erros
3. Quando implementar nova funcionalidade;
4. Quando finalizar um projeto

Um conjunto de commits gera toda uma rastreabilidade de código e não um único commit

---

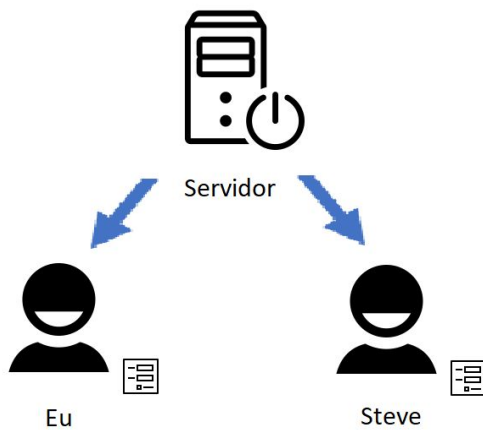
## Repositórios Remotos:

Um repositório remoto é um local onde eu posso compartilhar os meus arquivos, para que outros desenvolvedores tenham acesso.

Vamos pensar na seguinte estrutura:

- Um repositório de nome SERVIDOR;
- O repositório servidor compartilha arquivos para todos os desenvolvedores;
- O Desenvolvedor-A baixa os arquivos localmente e faz suas alterações;
- Os arquivos alterados deverão enviar suas alterações ao repositório SERVIDOR;
- O Desenvolvedor-B terá acesso a todas as alterações feitas no arquivo pelo Desenvolvedor-A

Esquema: Criar um repositório de nome SERVIDOR para compartilhar as alterações no código.



---

Primeiro, vamos identificar qual o nome da nossa pasta:

Meu projeto está na pasta: FE\_html\_01 (sou eu)

Documentos/Trabalho/Jadelson/Testes/**FE\_html\_01**

Agora, vou retornar um nível e posicionar a uma pasta anterior ao diretório que estou..

**\$ cd ..**

Documentos/Trabalho/Jadelson/Testes

=====

Vou criar a Pasta SERVIDOR:

**\$ mkdir servidor**

Posso ver o conteúdo agora

**\$ dir** ou **\$ ls**

Agora vou acessar esta pasta servidor

**\$ cd servidor**

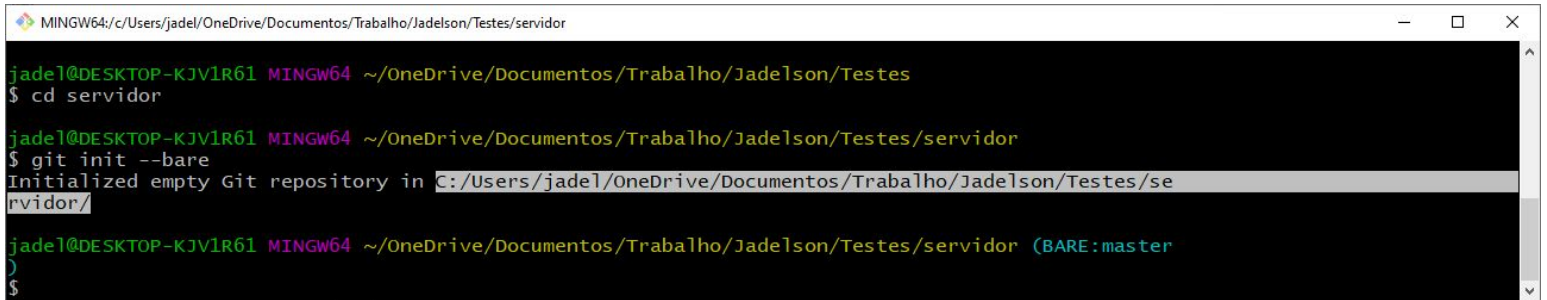
Agora vou definir que esta pasta será um repositório. Mas esse repositório é apenas para armazenar as alterações. Será um repositório “PURO” conterá somente as alterações dos arquivos. Então vou criar este repositório com o parâmetro --bare

**\$ git init --bare**

Não conterá uma cópia dos arquivos na íntegra. Somente suas alterações.

Observamos agora que o caminho do nosso repositório que serve como Servidor é:

**C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/**

A screenshot of a Windows command prompt window. The title bar shows the path "MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor". The command prompt shows the user "jadel@DESKTOP-KJV1R61" in a "MINGW64" environment. The user navigates to the "servidor" directory and runs "git init --bare". The output shows "Initialized empty Git repository in C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/". The prompt then shows the user is now in the "servidor" directory with the branch "(BARE:master)".

```
mingw64 C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes
$ cd servidor

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor
$ git init --bare
Initialized empty Git repository in C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor (BARE:master)
$
```

Vamos copiar este nome ....

Vamos agora voltar para a minha Pasta:

**\$ cd ../FE\_html\_01/**

Agora voltei para a minha pasta..

Agora, preciso informar ao meu Repositório que ele reconheça o repositório SERVIDOR como aquele para onde enviarei as alterações...

Quando executo o comando `$ git remote`, o git me informa se meu repositório está associado a algum repositório remoto..

Então, apenas para testar.. Vamos listar todos os repositórios remotos que o meu repositório conhece:

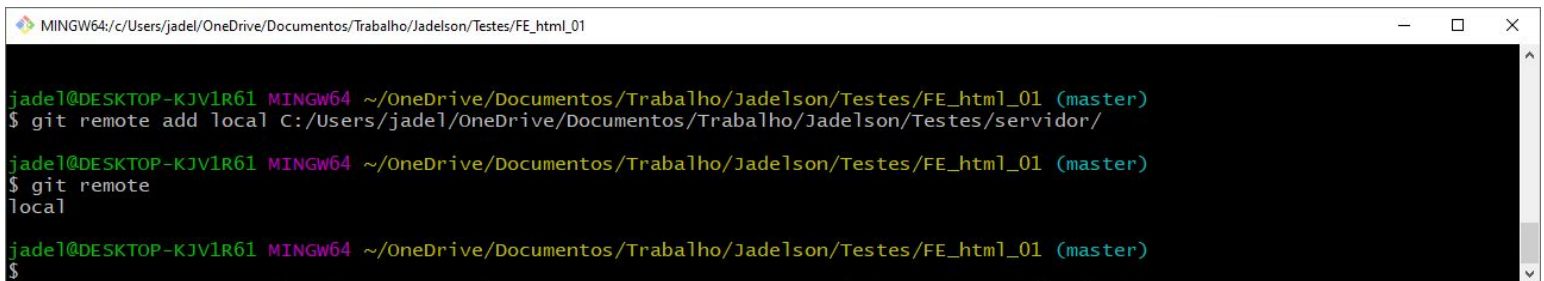
**`$ git remote`**

Vamos adicionar um repositório para que o meu repositório associe-se a ele. E vou dar o nome de “**local**” e devo associar o nome daquele caminho do repositório SERVIDOR

**`$ git remote add local C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/`**

Importante: este endereço poderia ser qualquer local da rede e até mesmo da internet.

Agora digito `$ git remote` e vejo o que exibe..

A terminal window titled 'MINGW64: c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01' shows the following commands and output:

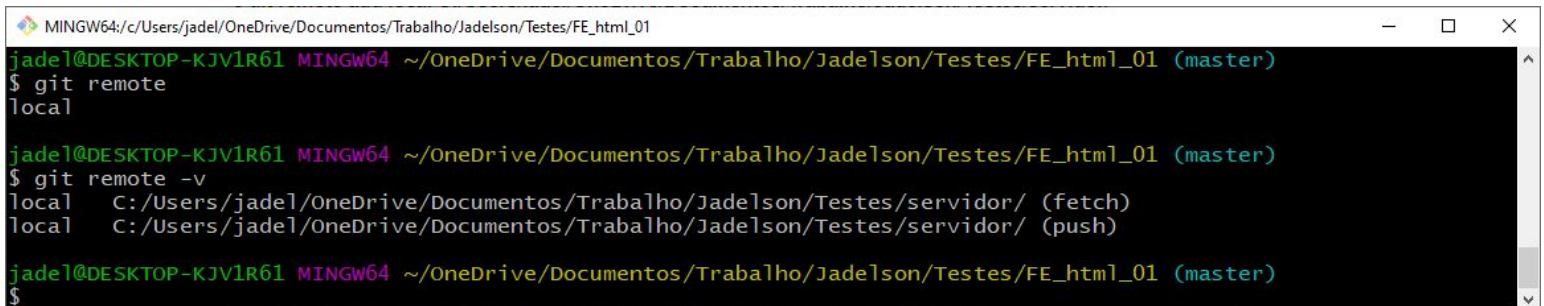
```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git remote add local C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git remote
local

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

Se eu quero ver a estrutura desse repositório, posso digitar:

**`$ git remote -v`**

A terminal window titled 'MINGW64: c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01' shows the following commands and output:

```
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git remote
local

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git remote -v
local      C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/ (fetch)
local      C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/ (push)

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

**fetch** - irá buscar dados deste caminho

**push** - irá enviar dados para este caminho



=====

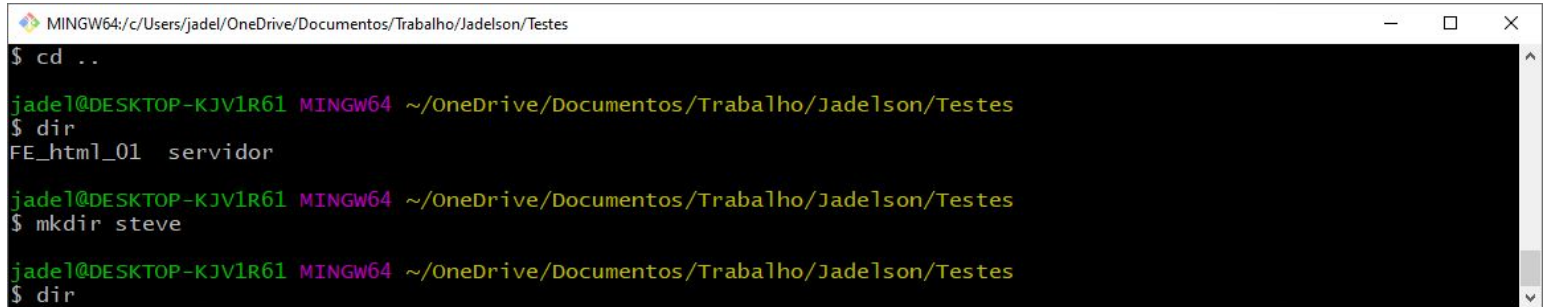
Agora vou criar a pasta de trabalho do Steve

Preciso retornar à pasta anterior...

**\$ cd ..**

E criar uma para pro Steve

**\$ mkdir steve**



```
MINGW64:/c/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes
$ cd ..

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes
$ dir
FE_html_01  servidor

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes
$ mkdir steve

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes
$ dir
```

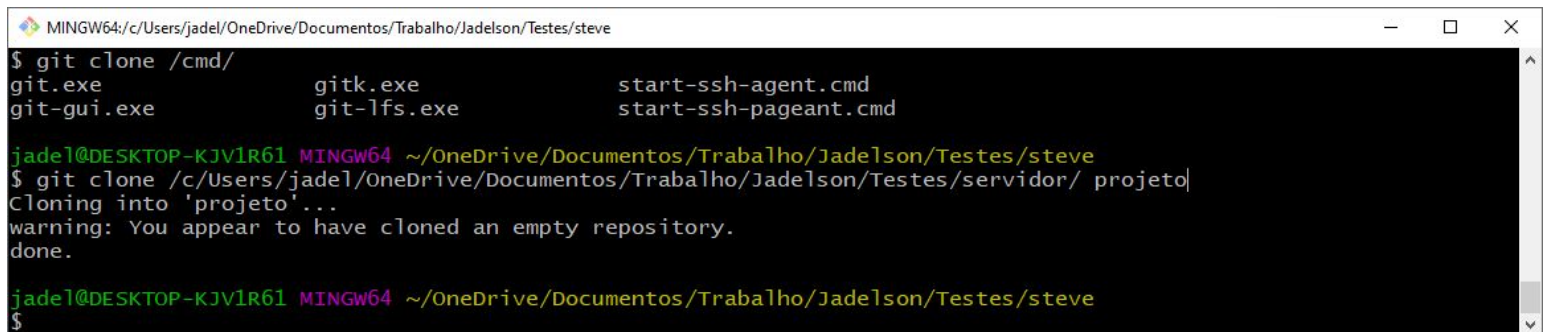
Vamos acessar a pasta do steve

**\$ cd steve**

Como o steve está iniciando seu trabalho ele não possui nada em sua pasta e também nunca acessou a pasta SERVIDOR.

Então ele precisa CLONAR a pasta SERVIDOR.

**\$ git clone /c/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/ projeto**



```
MINGW64:/c/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve
$ git clone /cmd/
git.exe          gitk.exe          start-ssh-agent.cmd
git-gui.exe      git-lfs.exe       start-ssh-pageant.cmd

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve
$ git clone /c/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/ projeto
Cloning into 'projeto'...
warning: You appear to have cloned an empty repository.
done.

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve
$
```

Clonou o repositório mas dá um aviso: Este repositório está vazio....

Apesar de termos dada a instrução a nossa pasta para enviar as alterações para a pasta servidor, ainda não enviamos os dados para esta pasta servidor..

=====

Fazendo os PUSH

Voltamos a minha pasta

```
$ cd ../FE_html_01/
```

Se executar \$ git remote, vejo que tenho um repositório associado...

```
$ git remote
```

A terminal window titled 'MINGW64; c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE\_html\_01'. The output shows a warning about an empty repository, followed by the command 'cd ../FE\_html\_01/' and 'git remote', which returns 'local'.

```
MINGW64; c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
warning: You appear to have cloned an empty repository.
done.

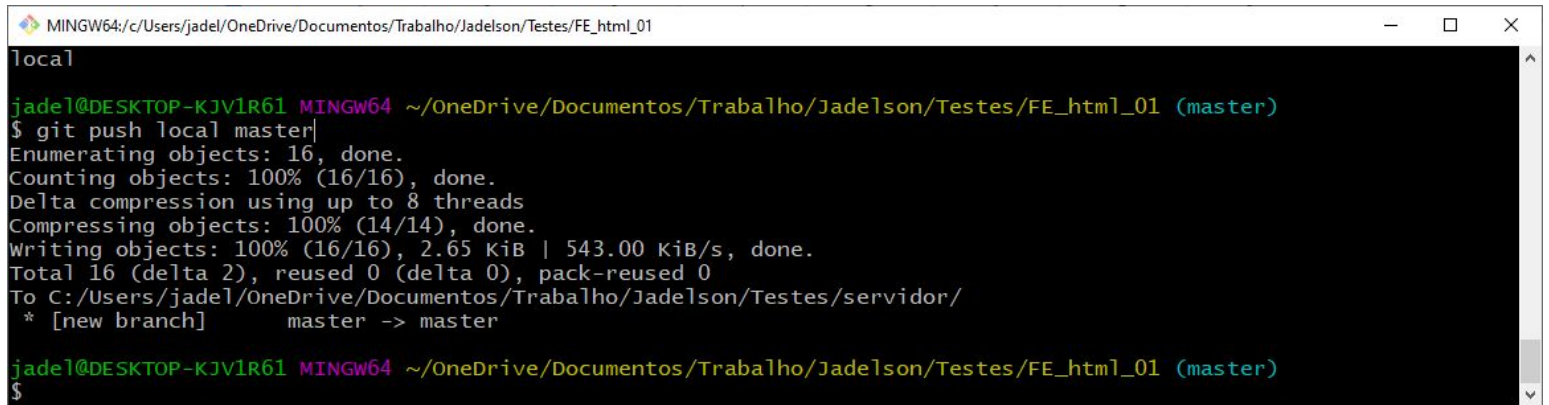
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve
$ cd ../FE_html_01/

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git remote
local

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

Para enviar os dados para o servidor: coloco git push + local + (branch = master)

```
$ git push local master
```

A terminal window showing the output of 'git push local master'. It details the enumeration, counting, and compression of 16 objects, and the successful push to the 'master' branch on the local server.

```
MINGW64; c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01

local

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git push local master
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (16/16), 2.65 KiB | 543.00 KiB/s, done.
Total 16 (delta 2), reused 0 (delta 0), pack-reused 0
To C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor/
 * [new branch]      master -> master

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

Acima vejo que realizei o PUSH com sucesso.

Agora vamos na Pasta do STEVE pra realizar o processo inverso: Baixar as alterações. Lembrando que eu criei a conexão com o SERVIDOR de dentro da pasta PROJETO do STEVE

```
$ cd ../steve/projeto
```

Se eu executar ls ou dir, verei que não existe nenhum arquivo ainda:

```
$ dir
```

Ainda não tem nada

PULL: trazendo as alterações

Na pasta steve/projeto executo o git remote para ver o nome do repositório remoto do steve

**\$ git remote**

```
MINGW64/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto
$ cd projeto

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git remote
origin

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$
```

O nome que aparece para o repositório remoto é ORIGIN.

Mas quero renomear para LOCAL.. Então

**\$ git remote rename origin local**

```
MINGW64/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git remote
origin

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git remote rename origin local

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git remote
local

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$
```

Mantemos assim uma igualdade no nome do repositório.

E para trazer os dados: git pull + local + master (nome do branch)

**\$ git pull local master**

Executo ls para testar..

**\$ ls**

```
MINGW64/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto
Unpacking objects: 100% (16/16), 2.63 KiB | 47.00 KiB/s, done.
From C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor
* branch      master    -> FETCH_HEAD
* [new branch] master    -> local/master

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ ls
index.html  teste_folha.css

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ ls
index.html  teste_folha.css

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$
```

=====  
Vamos averiguar se estão iguais:

Pelo sublimetext abrimos a pasta **FE\_html\_01**

Pelo sublimetext abrimos também a pasta **steve/projeto**

E então comparamos os arquivos:

Index.html

Teste\_folha.css

**IMPORTANTE:** Perceba que apesar de ter sincronizado o arquivo **.ignore**, o arquivo **bandeira\_brasil.png** e a pasta **mensagens** não foram sincronizados.

Eles estão dentro do .ignore

Com isso, agora, ambos conseguem trabalhar e sincronizar seus códigos.

=====  
Vamos testar esse processo.

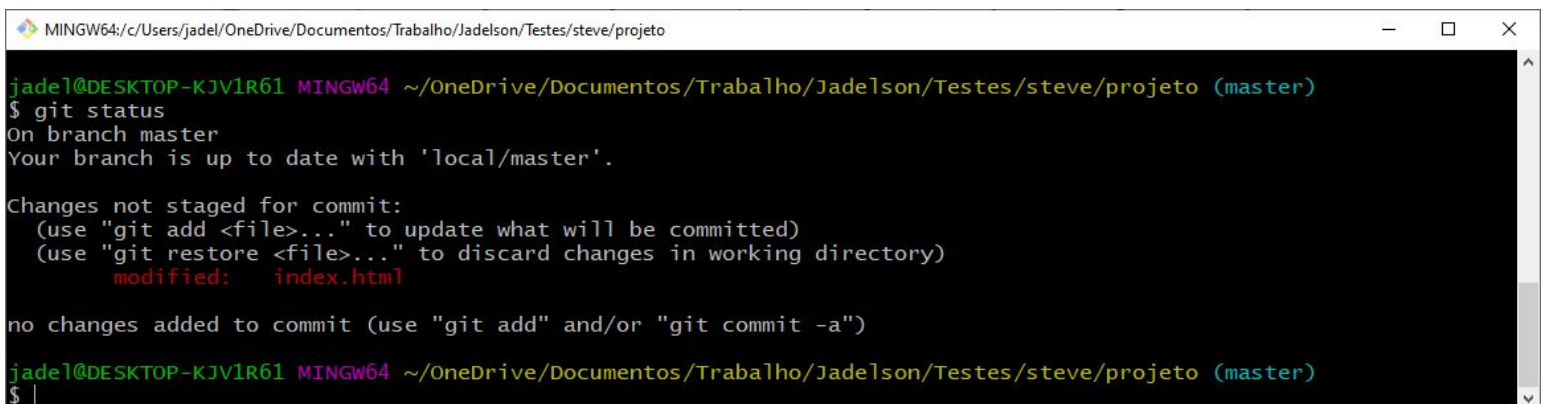
O steve fará uma modificação no arquivo index.html. Ele irá mudar a tag <title> da Página para

**<title>Brasil - Hino Nacional</title>**

Salva o arquivo e agora no GIT ele digita **git status**

**\$ git status**

Vemos que existe uma modificação...



```
MINGW64; c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git status
On branch master
Your branch is up to date with 'local/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:   index.html

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ |
```

Então: adicionar o index.html e comitar

**\$ git add index.html**

**\$ git commit -m "Alteração: alterado o conteúdo da tag <title>."**

```
MINGW64; c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

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

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git add index.html

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ git commit -m "Alteração: alterado o conteúdo da tag <title>."
[master 5bf9974] Alteração: alterado o conteúdo da tag <title>.
1 file changed, 1 insertion(+), 1 deletion(-)

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/steve/projeto (master)
$ |
```

Se eu for no meu repositório verei esta alteração?

Ainda não..... Pois preciso executar o **git push**

**\$ git push local master**

=====

Agora vou na minha pasta. Até agora era a pasta do steve

**\$ cd ../../FE\_html\_01/**

Se executar o git status verei que não tenho informação..... Mas o steve me avisou que modificou.

**\$ git status**

```
MINGW64; c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
$ cd ../../FE_html_01/

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ ^C

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git status
On branch master
nothing to commit, working tree clean

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ |
```



Então preciso dar um PULL

**\$ git pull local master**

```
MINGW64/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
jadel@DESKTOP-KJVR61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git pull local master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 381 bytes | 23.00 KiB/s, done.
From C:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/servidor
* branch      master      -> FETCH_HEAD
   728376b..5bf9974  master  -> local/master
Updating 728376b..5bf9974
Fast-forward
 index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

jadel@DESKTOP-KJVR61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

Ele informa que teve uma alteração no arquivo index.html

E se eu quiser detalhe sobre esta alteração é só digitar `git log -p`

**\$ git log -p**

```
MINGW64/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
commit 5bf9974d613e660a653de3741b2f6fb8e18e895e (HEAD -> master, local/master)
Author: Jadelson <jadelsons@gmail.com>
Date: Thu May 7 18:43:20 2020 -0300

    Alteração: alterado o conteúdo da tag <title>.

diff --git a/index.html b/index.html
index 5806cd6..a52abee 100644
--- a/index.html
+++ b/index.html
@@ -2,7 +2,7 @@
<html>
<head>
    <meta charset="utf-8">
-    <title></title>
+    <title>Brasil - Hino Nacional</title>
:|
```

E se eu abrir o arquivo da minha pasta no sublime.text, devo ter a alteração realizada...

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <title>Brasil - Hino Nacional</title>
6
7     <link rel="stylesheet" type="text/css" href="teste_folha.css">
8
9
```

## GITHUB

É um serviço que possibilita a criação de um servidor Remoto na Internet.

Vamos acessar o Portal da web `github.com` e criar uma conta

1. Acesse `github.com`
2. Click em **Sign UP**
3. Preencha o formulário e click no **Desafio**

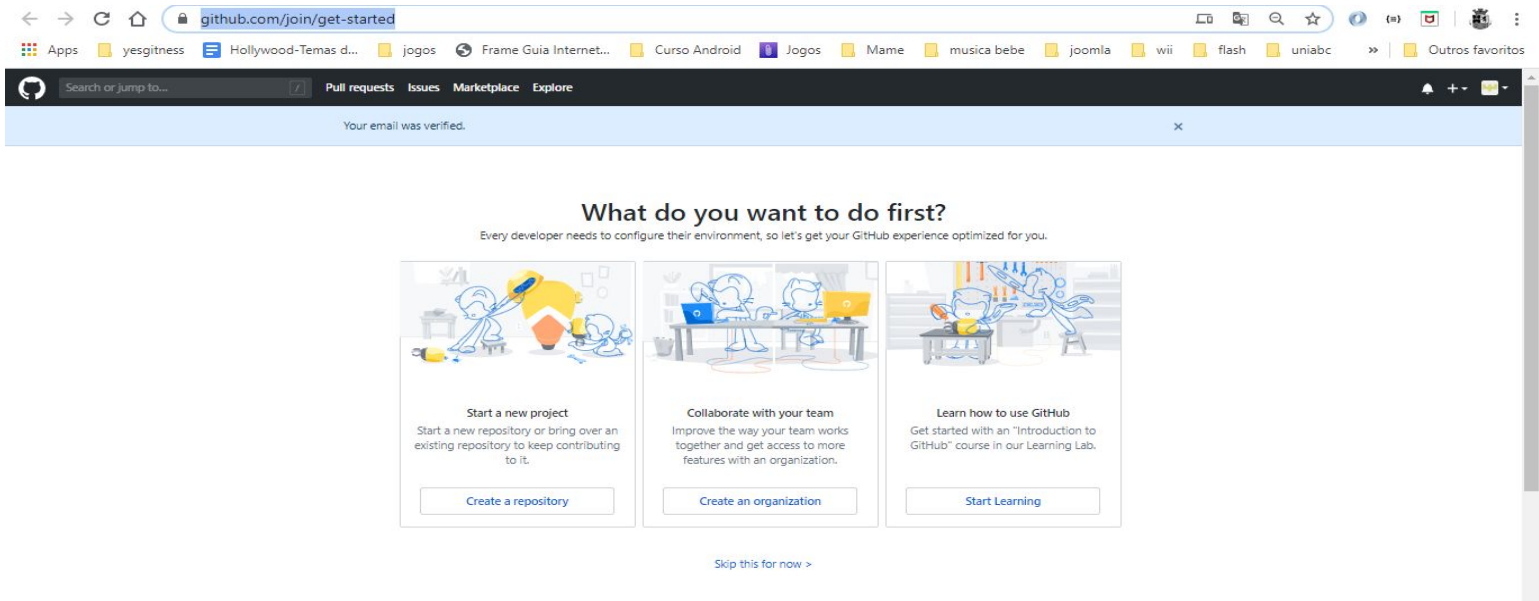
The screenshot shows the GitHub sign-up page. At the top, there's a navigation bar with a '+' icon and a URL: `join?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home`. Below the navigation bar, there's a row of links: 'id-Temas d...', 'jogos', 'Frame Guia Internet...', 'Curso Android', 'Jogos', 'Mame', 'musica bebe', 'joomla', and 'wii'. The main form has the following fields:

- Username \***: A text input field containing 'jadeisons' with a green checkmark on the right.
- Email address \***: A text input field containing 'jadeisons@gmail.com' with a green checkmark on the right.
- Password \***: A password input field with masked characters '\*\*\*\*\*'.
- Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)
- Email preferences**: A checkbox labeled 'Send me occasional product updates, announcements, and offers.' which is currently unchecked.
- Verify your account**: A CAPTCHA challenge area with the text 'Quando a imagem estiver com o lado certo para cima, toque em Feito!' and an image of a dog. Below the image is a button labeled 'Feito'. To the right of the button are icons for help (a question mark), refresh, and audio.

At the bottom of the form is a large blue button labeled 'Create account'.

4. Click em Create account
5. Agora responda “Que tipo de trabalho você faz, principalmente”
6. Responda: Quanta experiência você tem de Programação
7. Responda: Qual o seu objetivo no GitHub
8. Responda: Tenho interesse em..
9. Click em Complete Setup
10. Acesse seu email para validar a conta

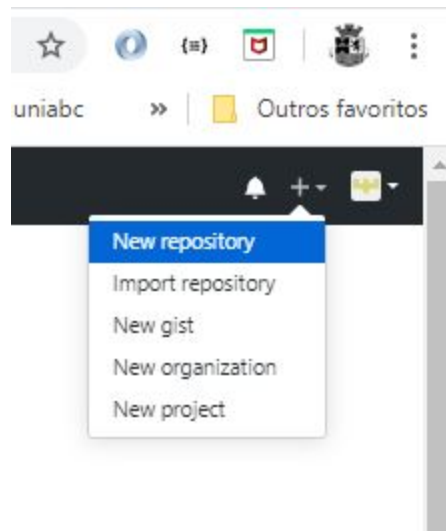
A primeira vez, o Git Hub abre a tela:



Já nos sugere que façamos: um repositório, um time ou aprendamos como usar.

Vamos iniciar um Repositório:

1. Click em **Create Repository** ou no botão **+ > New Repository**



2. Digitamos o nome do repositório: design-git
3. Digitamos a descrição desse repositório: Treinamento em Git-Hub
4. Definimos agora o tipo do repositório
  - a. Public: Para planos gratuitos
  - b. Private: Para Planos Pagos
5. Vamos escolher o repositório tipo Public
6. Click **Create Repository**


Pronto: Repositório WEB Criado

=====

O que o Git Hub nos mostra é uma tela onde existe:

A informação de como criar nosso repositório local, ou como simplesmente enviar o nosso com a ajuda de uma linha de comando

### Quick setup — if you've done this kind of thing before

 Set up in Desktop or **HTTPS** SSH

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

### ...or create a new repository on the command line

```
echo "# design-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/jadelsons/design-git.git
git push -u origin master
```

**Como criar o repositório**

### ...or push an existing repository from the command line

```
git remote add origin https://github.com/jadelsons/design-git.git
git push -u origin master
```

**... ou simplesmente enviar um repositório existente.**

### ...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

=====

A primeira coisa a ser feita é definir que a forma de acesso PUSH ou FETCH será HTTPS

### Quick setup — if you've done this kind of thing before

 Set up in Desktop or **HTTPS** SSH

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

=====

Adicionaremos o repositório do Git Hub no nosso repositório local...

1. Vamos copiar o caminho existente na orientação **git remote**



2. Vamos abrir nosso repositório FE\_html\_01 no Bash
3. Vamos colar a linha selecionada acima:
4. **\$ git remote add origin https://github.com/jadelsons/design-git.git**

Importante: estamos adicionando um repositório remoto com o nome **ORIGIN** de acordo com o caminho especificado **ORIGIN**: é um padrão para nome de repositório Principal, ou seja, como se fosse pai do seu Projeto. Então vamos manter esse nome de repositório

5. Agora podemos confirmar se o repositório remoto foi adicionado: **\$ git remote**
6. Agora, vamos fazer o PUSH, ou seja, enviar os nossos dados para o ORIGIN
7. Vamor dar o comando: **\$ git push origin master**

Importante: Por que não faremos igual está na sugestão do Git Hub `$ git push -u origin master`  
Este parâmetro **-u** indica que: sempre que eu digitar **git push** e estiver na **master**, vou mandar pro **origin**.  
Então é um pouco arriscado. É melhor sempre digitar o **remote** seguido do **branch**

8. Vamos dar ENTER e ele vai pedir o Login e Senha

A tela a seguir mostra que foi enviado....



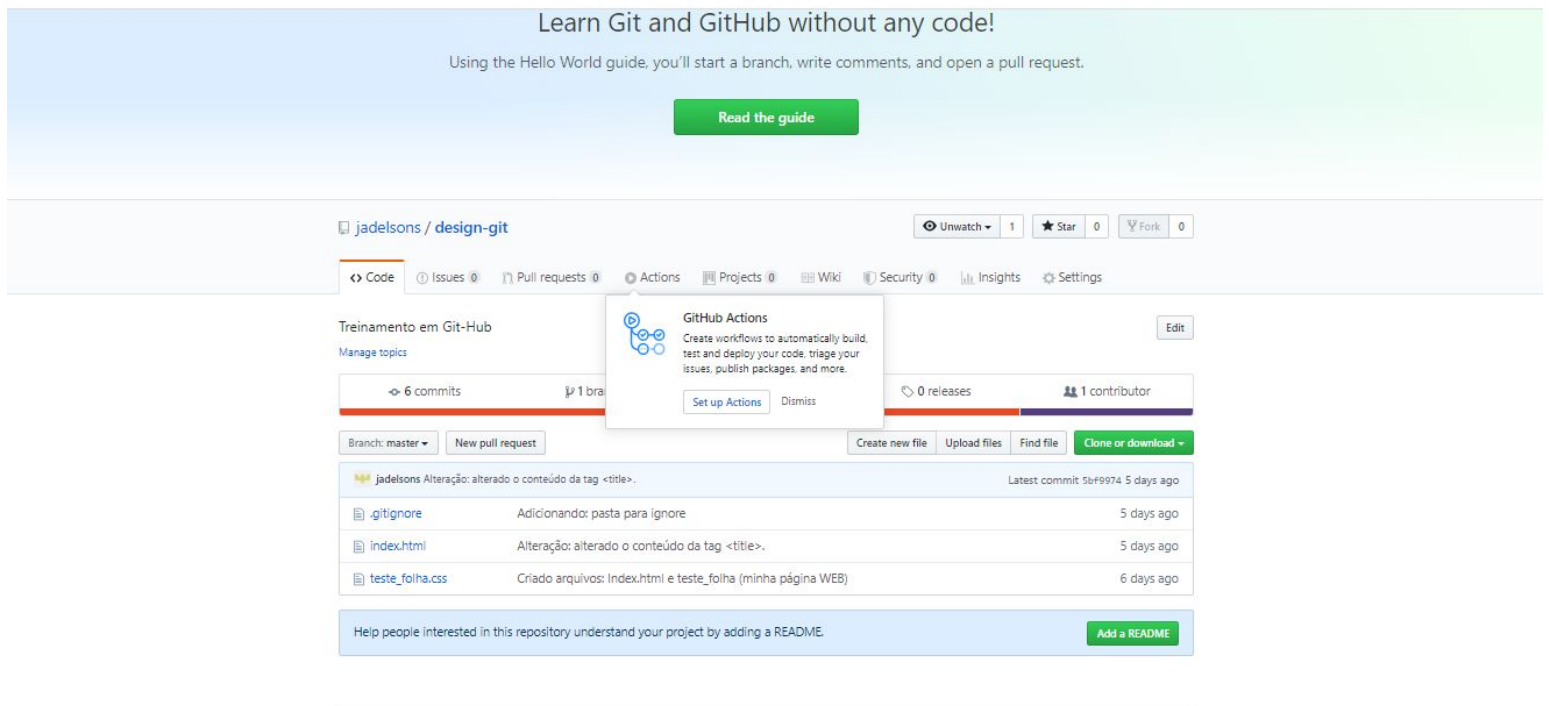
```
MINGW64:/c:/Users/jadel/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01
origin

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git push origin master
Logon failed, use ctrl+c to cancel basic credential prompt.

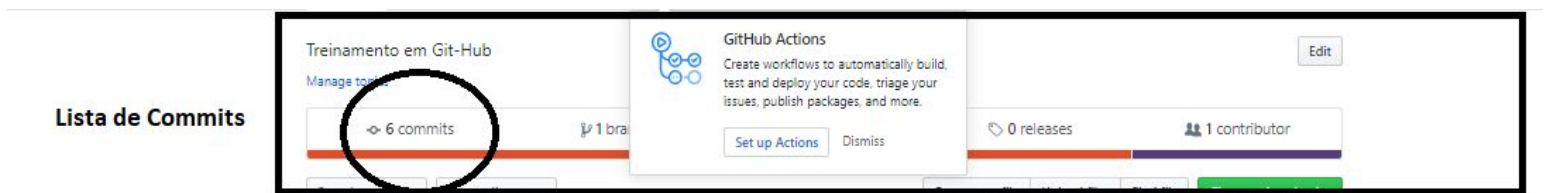
jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$ git push origin master
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 8 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (19/19), 2.99 KiB | 235.00 KiB/s, done.
Total 19 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/jadelsons/design-git.git
* [new branch]      master -> master

jadel@DESKTOP-KJV1R61 MINGW64 ~/OneDrive/Documentos/Trabalho/Jadelson/Testes/FE_html_01 (master)
$
```

Agora, volto pra tela do Git Hub e atualizo e vejo ...



Vejo a lista de Commits...




Click na lista de Commits e veja cada um deles...



Branch: master ▾

Commits on May 7, 2020

Alteração: alterado o conteúdo da tag <title>.

 jadelsons committed 5 days ago



5bf9974



Adicionando: pasta para ignore

 jadelsons committed 5 days ago



728376b



Adicionando: .gitignore

 jadelsons committed 5 days ago



001ec08



Commits on May 6, 2020

index.html: adicionado parágrafo

 jadelsons committed 6 days ago



c937bb6



index.html: adicionado comentário

 jadelsons committed 6 days ago



dd08783



Criado arquivos: Index.html e teste\_folha (minha página WEB)

 jadelsons committed 6 days ago



acb6166



Newer

Older

O que mais está disponível no Git Hub:

- Configurar colaboradores no projeto/Gerenciar Acesso
- Controle dos Branchs: definir regras para envio, evitando exclusão
- Notifica serviços externos sobre alterações
- Integra vários aplicativos
- Criptografias

=====

Suponhamos que precisamos mostrar para alguém os meus projetos

Para isso, informo o caminho do meu usuário no git Hub

**github.com/jadelsons**

Ou, posso ainda passar o caminho completo.

**https://github.com/jadelsons/design-git**

=====

Vamos fazer uma alteração localmente e enviar a alteração para gitHub e testar se um usuário não “logado” consegue visualizar..

1. Abrir o arquivo INDEX.HTML
2. Vamos acrescentar a seguinte alteração antes da tag </body> **<p> Este documento não possui Direitos Autorais </p>**
3. Agora vamos salvar a alteração;
4. Adicionar ao git: **\$ git add index.html**
5. Vamos ver se foi reconhecida a alteração: **\$ git status**
6. Vamos comitar: **\$ git commit -m "index.html: adicionado parágrafo de direitos autorais"**
7. Vamos ver o histórico de alterações: **\$ git log**

Antes de enviar ao nosso servidor remoto GitHub, vamos abrir nosso projeto lá no Git Hub e ver o conteúdo do arquivo index.html

Vemos que o arquivo está sem a alteração...

```
60         <li>Brado retumbante - Grito f
61         <li>Penhor - Usado de maneira
62     </ul>
63 </footer>
64
65     <p> Fim </p>
66
67 </body>
68 </html>
```

Agora vamos enviar a alteração...

1. Dar um git remote para ver nossos repositórios: **\$ git remote**
2. Agora dar o PUSH: **\$ git push origin master**

=====

Vamos conferir se a alteração foi realizada:

Atualize a página do Git Hub...

E veja se aparece a alteração do arquivo index.html

```
</footer>

<p> Fim </p>

<p> Este documento não possui Direitos Autorais </p>

</body>
</html>
```

---

=====



## Branchs

Vamos entender o que significa um Branch.

Antes, fazer a instalação do Visual Code:

=> arquivo: **Visual Code: Instalação**

=> Mostrar Apresentação: **GIT HUB Descomplicando Branchs**

Objetivo da Master: Ela deve conter a sua versão PRINCIPAL do seu código.

Manter ou preservar uma versão Principal significa que a qualquer momento, caso haja uma necessidade por exemplo, esta versão GUARDADA, ou preservada, poderia facilmente sofrer modificações a fim de garantir que seu projeto não pare de funcionar.

O que significa isso. Vamos ter um exemplo prático:

Vamos imaginar que seu projeto seja uma calculadora, imaginemos a seguinte situação:

1. O calculadora está pronta e em funcionamento;
2. Agora você gostaria de adicionar uma nova funcionalidade: uma conversão de numérica!
3. Então você pega o seu código principal e começa a modificar
4. Esse novo código não está pronto e impossibilita a publicação da calculadora com esta nova funcionalidade
5. De repente, a versão em produção precisa de uma correção de um BUG
6. Já era!!! Não terá como corrigir o BUG que acaba de surgir em produção, porque além deste BUG você ainda está tentando terminar aquela nova funcionalidade

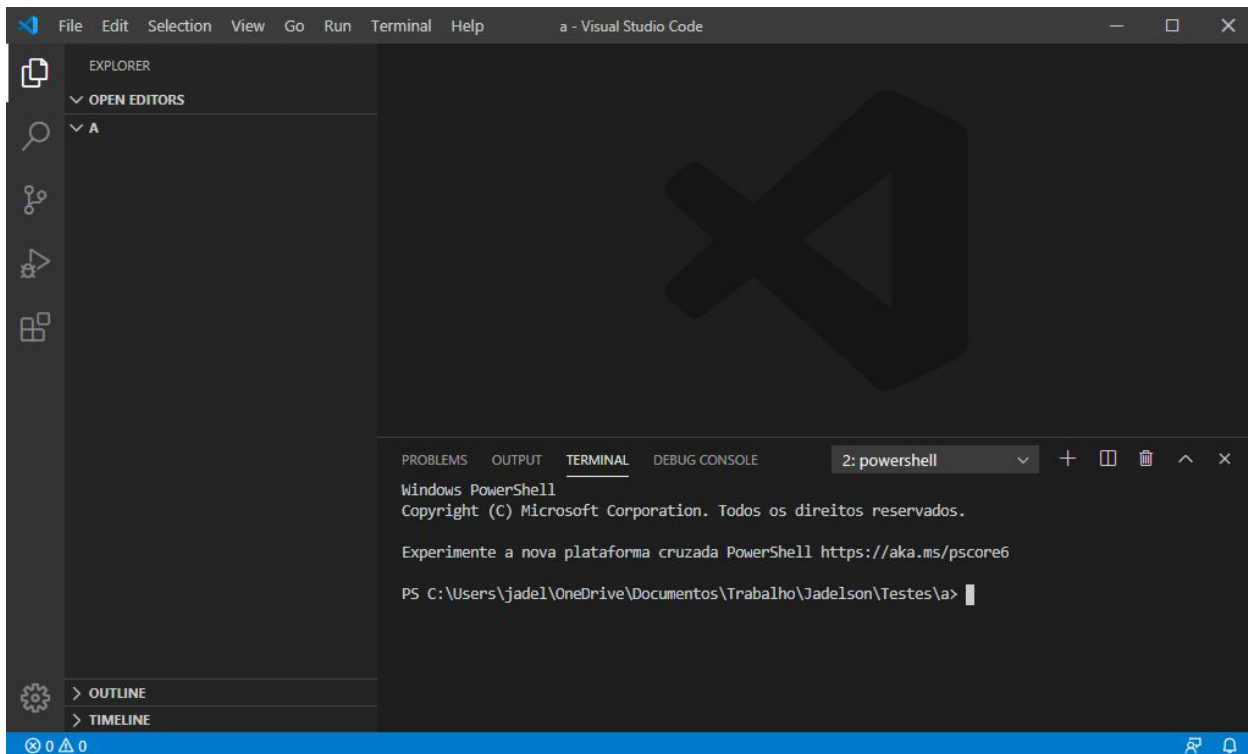
Então, o que fazer: Antes de mais nada, precisaríamos ter preservado o código ORIGINAL da calculadora, garantindo que a versão em produção poderia sofrer rápidas alterações, se necessário

Antes de iniciar o projeto, vamos criar, dentro da pasta TESTE, três pastas : A, B e Server

Nome	Status	Data de modificação	Tipo	Taman
a	✓	16/05/2020 15:36	Pasta de arquivos	
b	✓	16/05/2020 15:36	Pasta de arquivos	
FE_html_01	✓	15/05/2020 23:39	Pasta de arquivos	
server	✓	16/05/2020 15:36	Pasta de arquivos	
servidor	✓	15/05/2020 09:55	Pasta de arquivos	
steve	✓	07/05/2020 17:44	Pasta de arquivos	

Agora, vamos abrir o VS Code (Visual Studio Code) e vamos iniciar o projeto:

1. Abra o VSC e abra a pasta “A”
2. Feche a Tela de Wellcome





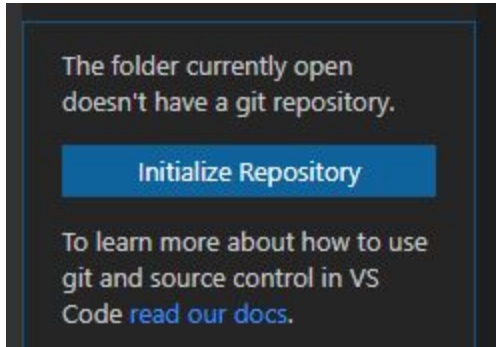
=====

Vamos dar início às definições de repositório desta pasta

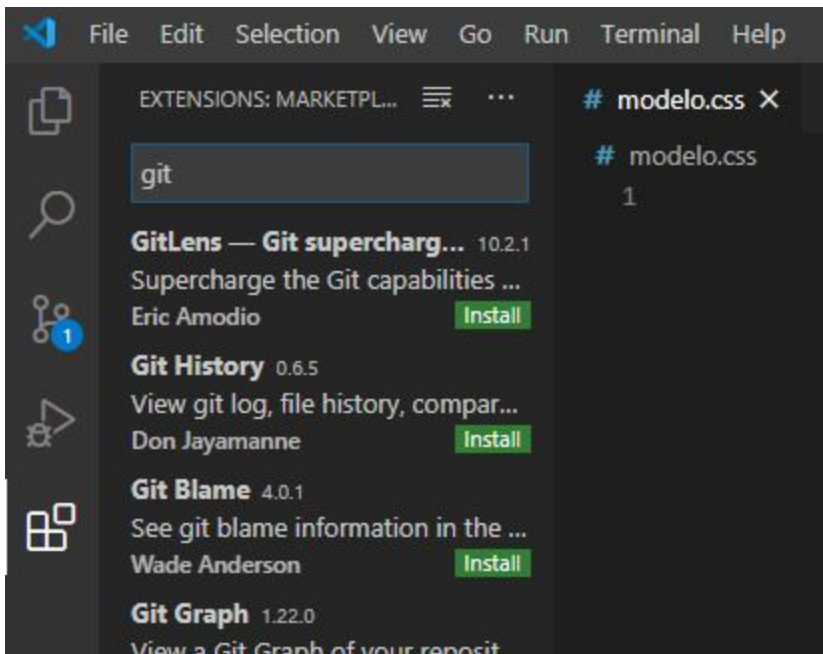
1. Click no botão SOURCE CONTROL, na barra vertical esquerda;



2. Agora clicamos em “Initialize Repository”

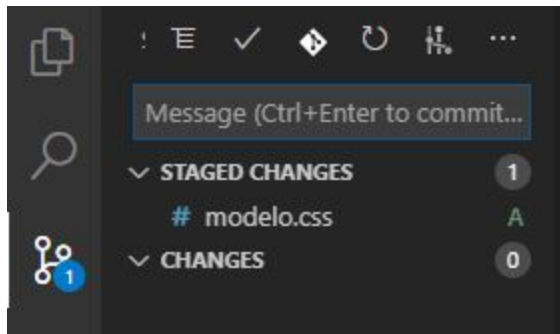


3. Neste momento, já temos nossa pasta “a” como repositório GIT
4. Vamos abrir o “bash”: TERMINAL > NEW TERMINAL
5. Se precisar enxergar quais arquivos precisam ser atualizados pelo GIT é só clicar neste botão e visualizar seu conteúdo;
6. Vamos voltar à pasta “a”, por intermédio do botão EXPLORER e vamos criar o arquivo modelo.css
7. Clicamos em **NEW FILE** e digitamos o nome **modelo.css**
8. Automaticamente o VSCODE nos informa que existe um arquivo precisando de Atualização. Na verdade ele mostra na frente deste arquivo um “U”. É o “U” de **UNTRACKED**, ou seja, arquivo não controlado pelo Git.
9. Vamos aproveitar e instalar algumas extensões do Git no VSCODE
10. Click no botão EXTENSIONS e Digite GIT

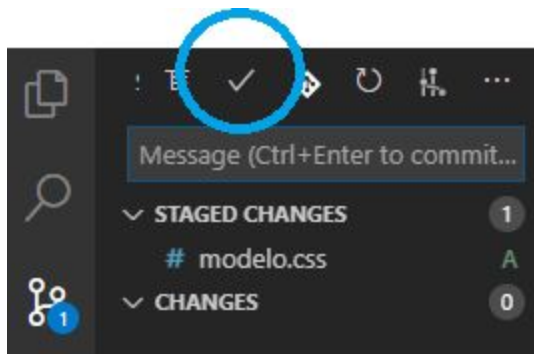


11. Vamos instalar **Git History** , **Git Graph** e **Git Lens**
12. No TERMINAL aberto, vamos testar as seguintes instruções
13. git status
14. git log
15. Vamos então adicionar o arquivo **modelo.css** ao repositório com **branch Master**
16. Click no botão “Source Control”

17. Click no botão “ + Stage Changes” (mudança de estágio)
18. Agora tenho este arquivo adicionado mas ainda falta o commit



19. Digito então o comentário do Commit: **Adicionado arquivo: modelo.css**
20. Click no botão Commit:



OK: Arquivo modelo.css criado e adicionado ao controle git

Click no Git Lens e veja o histórico de movimentações

=====

Agora vou criar a estrutura de desenvolvimento para duas situações:

Importante: Quem deseja ser rigorosamente organizado nesse ambiente pode seguir as orientações do Git Flow

Escopo: O arquivo modelo.css será desenvolvido por dois desenvolvedores:

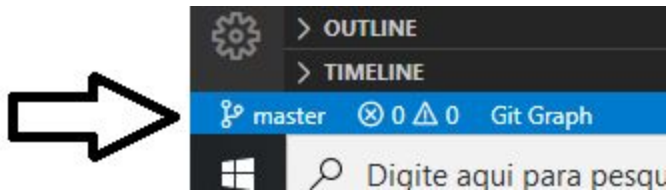
- Um desenvolverá a classe titulo { }
- O outro desenvolverá a classe body { }

Portanto, seguindo o padrão do Git Flow, teremos a branch DEVELOP e dentro dela teremos duas Features:

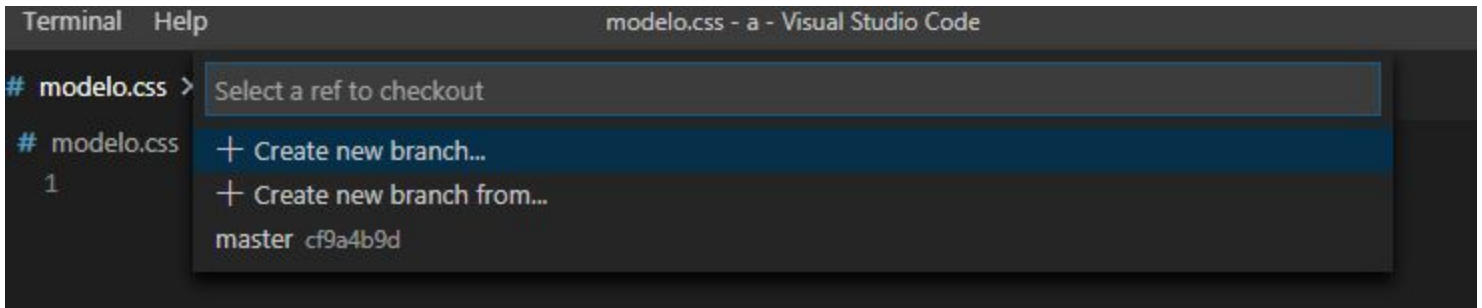
- titulo
- body

## Criando as Branchs

1. Clicamos no nome da Branch Master no rodapé

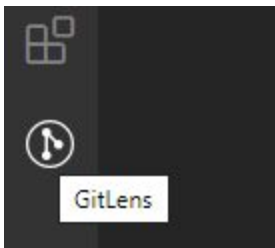


2. Fazendo isso, o VSCODE abre uma lista na área superior, onde devemos escolher CREATE NEW BRANCH



3. Agora digitamos o nome da branch: **develop**
4. Agora, tendo a certeza que estou na branch DEVELOP, faço o mesmo procedimento para criar, a partir dela, duas outras branches:
  - a. **feature/titulo**
  - b. **feature/body**

Para ter certeza que a estrutura está correta, vá para a MASTER e acesse o GitLens



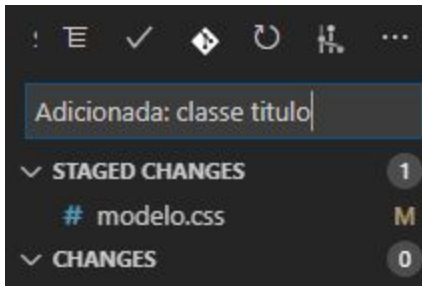
=====

Vamos criar a classe titulo no arquivo modelo.css

1. Vamos habilitar a branch feature/titulo
2. Vamos digitar o conteúdo da classe titulo:

```
.titulo {  
  background-color: #aaaaaa;  
  padding: 10px;  
  margin-top: 0px;  
  font-size: 20px;  
}
```

3. Vamos Salvar o Arquivo
4. Vamos clicar no Source Control
5. Agora vou Adicionar a o arquivo: **+ Stage Changes**
6. E agora inserimos o comentário: **Adicionada: classe titulo**
7. E clicamos no Commit



=====

Vamos criar a classe Body

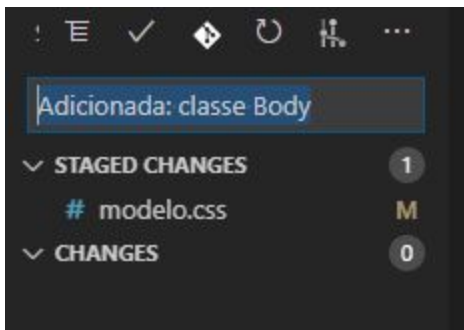
1. Vamos habilitar a branch feature/body

**IMPORTANTE: Veja que automaticamente desapareceu o conteúdo da classe titulo. Esse conteúdo não pertence a esta branch**

2. Vamos digitar o conteúdo da classe body:

```
body {  
    text-align: center;  
    background: linear-gradient(#FEFEFE, #888888);  
}
```

3. Vamos Salvar o Arquivo
4. Vamos clicar no Source Control
5. Agora vou Adicionar o arquivo: **+ Stage Changes**
6. E agora inserimos o comentário: **Adicionada: classe Body**
7. E clicamos no Commit

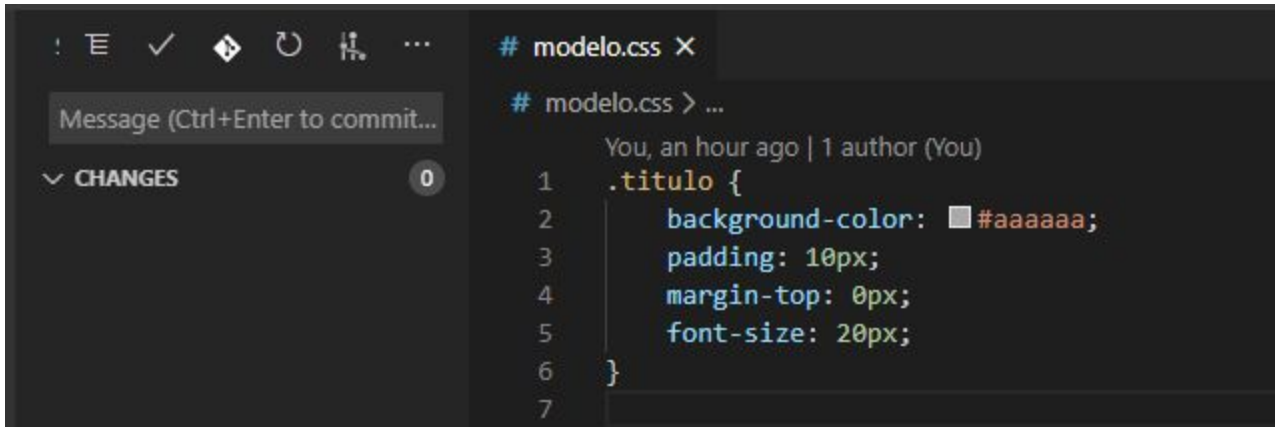


Podemos navegar pelas Branchs criadas para averiguar como está o arquivo modelo.css em todas elas.

Vou agora fazer todos os Merges para reagrupar todos os códigos que estavam distribuídos:

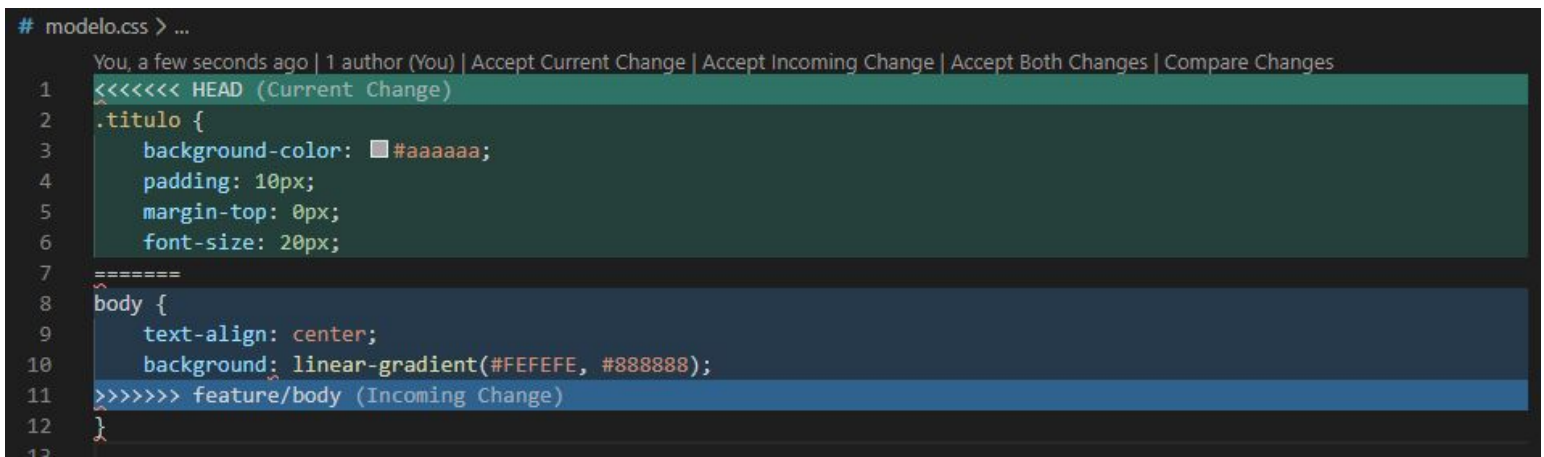
Vamos fazer via Terminal:

1. No Terminal aberto do VSCODE, vamos no posicionar na Branch DEVELOP: **git checkout develop**
2. Podemos observar que se mudamos para outra branch, o VSCODE já “atualiza” o arquivo modelo.css
3. Agora, vamos dar o chamado MERGE, ou seja, Vamos trazer o código das duas Branchs
4. Primeiro da branch feature/titulo: **git merge feature/titulo**
5. E Surge a classe título



```
# modelo.css X
# modelo.css > ...
You, an hour ago | 1 author (You)
1  .titulo {
2      background-color: #aaaaaa;
3      padding: 10px;
4      margin-top: 0px;
5      font-size: 20px;
6  }
7
```

6. Agora faremos o Merge da branch feature/body. Só que haverá um CONFLITO. O Git não saberá o que fazer e nos dará a responsabilidade para decidir o que fazer. Não é um erro, é um conflito
7. Agora o Merge da branch feature/body: **git merge feature/body**
8. E temos o resultado:



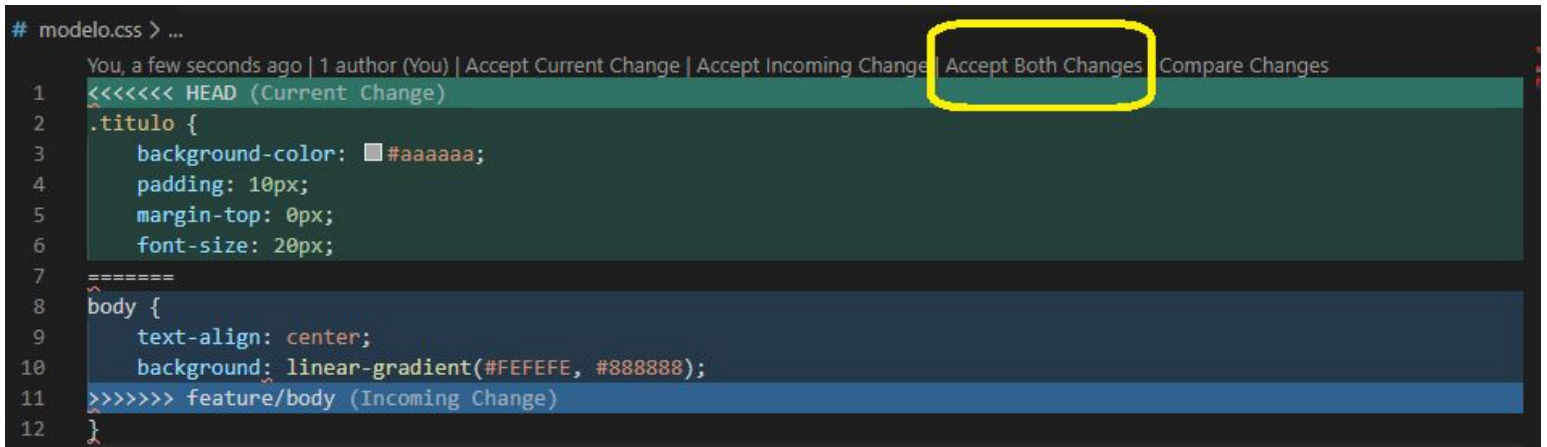
```
# modelo.css > ...
You, a few seconds ago | 1 author (You) | Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1  <<<<<< HEAD (Current Change)
2  .titulo {
3      background-color: #aaaaaa;
4      padding: 10px;
5      margin-top: 0px;
6      font-size: 20px;
7  }
8  =====
9  body {
10     text-align: center;
11     background: linear-gradient(#FEFEFE, #888888);
12 }
13 >>>>>> feature/body (Incoming Change)
```

Vamos analisar:

1. Em primeiro lugar, esse é o arquivo modelo.css
2. Em verde (current Change) é o conteúdo já existente. Esse conteúdo é o conteúdo atual, mas não existe na feature/body
3. Em azul (incoming Change), é o conteúdo que está chegando e que não existe na branch DEVELOPER
4. Podemos realizar várias ações nesse momento:
  - a. Aceitar apenas o conteúdo corrente
  - b. Aceitar apenas o conteúdo que está chegando
  - c. Aceitar AMBOS;

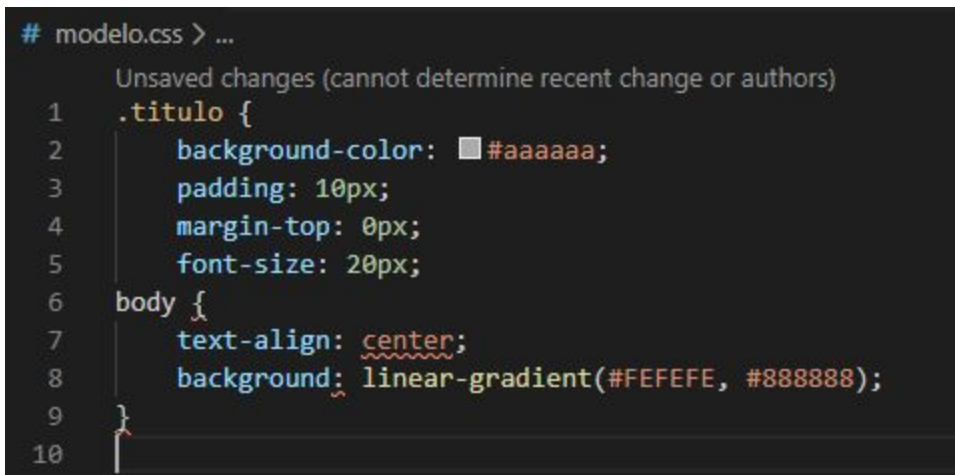


5. Vamos Clicar na opção: ACCEPT BOTH CHANGES....



```
# modelo.css > ...
You, a few seconds ago | 1 author (You) | Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<<< HEAD (Current Change)
2 .titulo {
3     background-color: #aaaaaa;
4     padding: 10px;
5     margin-top: 0px;
6     font-size: 20px;
7
8 =====
8 body {
9     text-align: center;
10    background: linear-gradient(#FEFEFE, #888888);
11 >>>>>> feature/body (Incoming Change)
12 }
```

6. E agora temos:

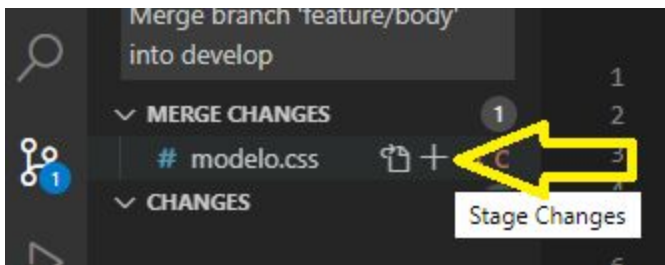


```
# modelo.css > ...
Unsaved changes (cannot determine recent change or authors)
1 .titulo {
2     background-color: #aaaaaa;
3     padding: 10px;
4     margin-top: 0px;
5     font-size: 20px;
6
7 body {
8     text-align: center;
9     background: linear-gradient(#FEFEFE, #888888);
10 }
```

7. Salvamos o Arquivo

8. Este é o momento de testar esse arquivo para homologar suas funcionalidades

9. Agora vamos confirmar o MERGE.

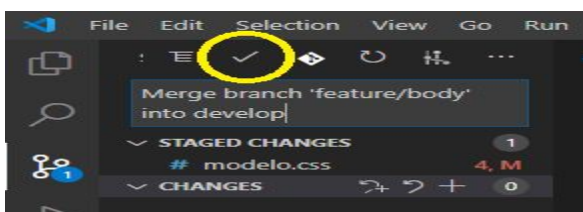


10. Temos então todas as branches atualizando o DEVELOP

11. Agora vou fazer o Commit...

12. Vamos clicar no Source Control

13. Clicar em Commit



Temos então as atualizações “Fundidas” e o arquivo finalizado na Develop

=====

Para visualizar de forma rápida os commits podemos ir no Git Lens ou podemos usar...

## git shortlog

```
PROBLEMS 4 OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\jadel\OneDrive\Documentos\Trabalho\Jadelson\Testes\> git merge feature/body
Auto-merging modelo.css
CONFLICT (content): Merge conflict in modelo.css
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\jadel\OneDrive\Documentos\Trabalho\Jadelson\Testes\> git shortlog
Jadelson (4):
    Adicionado arquivo: modelo.css
    Adicionada: classe titulo
    Adicionada: classe Body
    Merge branch 'feature/body' into develop

PS C:\Users\jadel\OneDrive\Documentos\Trabalho\Jadelson\Testes\> |
```

Vemos todas as commits realizadas.

Qual a diferença do rebase e merge: O rebase não traria este histórico todo.

=====

A partir de todos os testes homologados, vamos enviar este código para o MASTER.

1. Altere o branch para **master**
2. Veja que sumiu o conteúdo. Correto
3. Digitar: **git merge develop**
4. Digitar: **git shortlog**

OK. Pronto.

Conclusão: Não tocamos na MASTER

=====

Abrir o Git Graph e visualizar as ações de acordo com o Git Flow...

