



# GIT e GitHub

---

**Angelo Cesar Colombini**

Mas, afinal do que se trata e  
por quê preciso conhecer  
isso?



# Introdução

- A primeira coisa a nos chamar a atenção é que Git não é um apelido para GitHub
- Trata-se de duas ferramentas que trabalham juntas com o propósito de versionar o seu projeto
- Vamos conhecer ferramentas e conceitos
- O que são e como podemos participar de projetos colaborativos



# Fundamentos do GitHub

Criando e configurando uma conta no GitHub



# Git & GitHub

- Git
  - Também chamado de VCS (Version Control System)
- GitHub
  - Permite a hospedagem de repositórios



# Git & GitHub

- Quando é que precisamos usar um ou outro ou ambos?
- Pense no “seu projeto”
  - Pode ser seu TCC
  - Sua dissertação
  - Um projeto de graduação
  - Um site
  - Uma aplicação computacional para solucionar um problema de engenharia na sua empresa, etc



# Git & GitHub

- Toda vez que você salva seu trabalho para continuar no dia seguinte, ou na semana seguinte OU ...
- Bate sempre a dúvida, opa será que está tudo ok?
- Se eu precisar recuperar o que fiz anteriormente, por algum problema de consistência futuro, vou conseguir?
- Neste momento você se lamenta por não ter criado um projeto com os nomes, PROJv1, PROJv2, PROJv3, ..., PROJvn
- Opa, em qual deles esta aquele parágrafo decisivo????

# Git & GitHub



- Isso sem falar em projetos que envolvem dezenas, centenas e por que não milhares de arquivos!!!!
- É parece exagero, no mundo real são bem comuns e você já passou ou passará por eles
- Quando estiver nesta situação, lembre-se do GIT → ele estará à sua disposição para garantir que todo histórico de seu projeto, seja lá qual for sua dimensão esteja a seu alcance sem estresse

# Git & GitHub

- O Git é um software que te permite fazer a versão de cada parte de seu projeto
- Permite a você rastrear todo histórico de alterações, sem preocupação da criação dos famosos PROJv1, PROJv2, PROJv3, ..., PROJvn
- Além disso lhe fornece uma série de ferramentas que lhe permite trabalhar em colaboração com diversas outras pessoas



# Git & GitHub

- Quando falamos em **colaboração**, estamos entrando no terreno do **GitHub**



- Seu objetivo, centralizar o repositório do projeto e permitir seu compartilhamento com toda equipe do projeto e assegurar o controle das versões

# Instalando o GIT

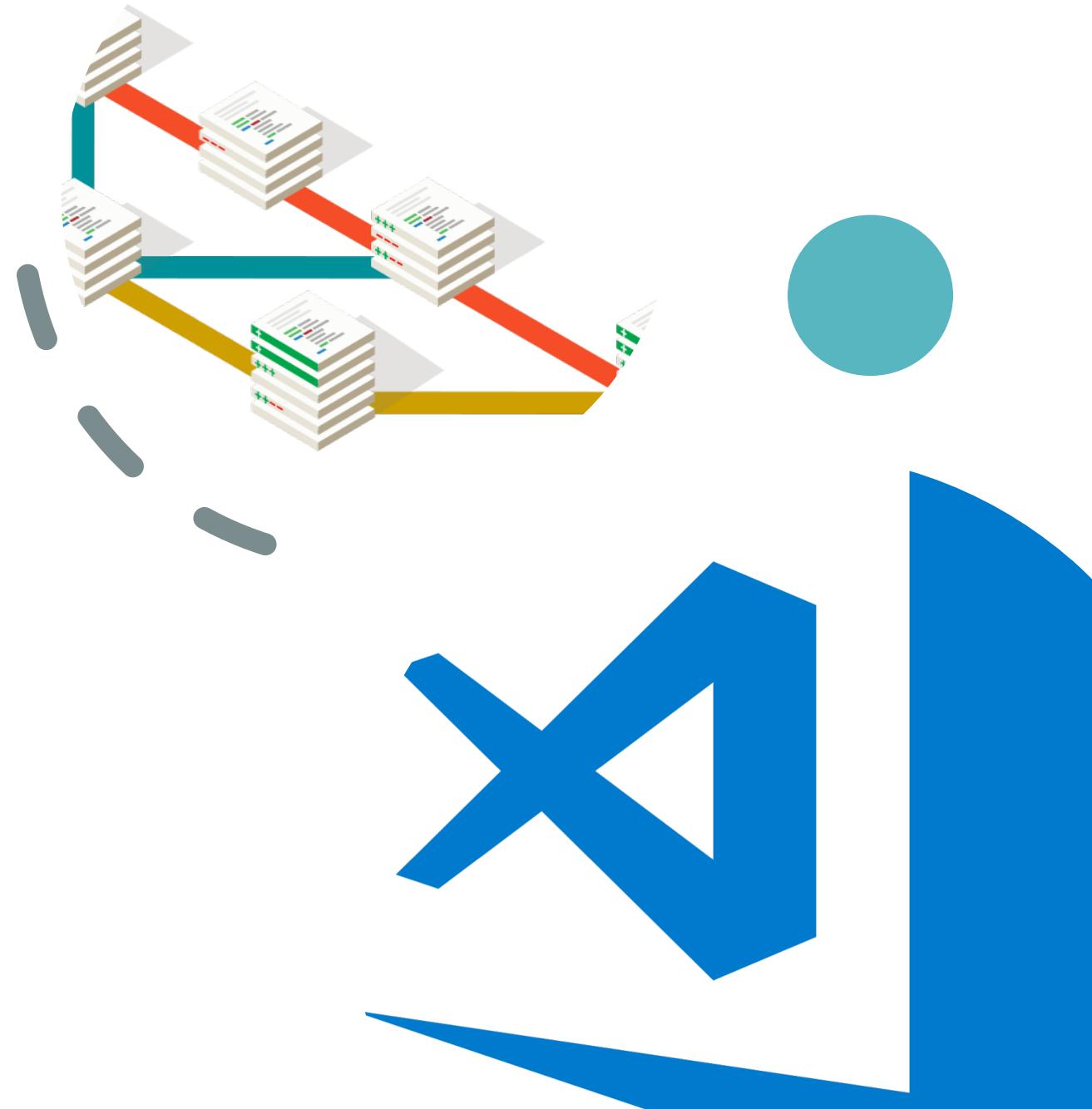
Preparando seu ambiente!



Esta Foto de Autor Desconhecido está licenciado em [CC BY-SA-NC](#)

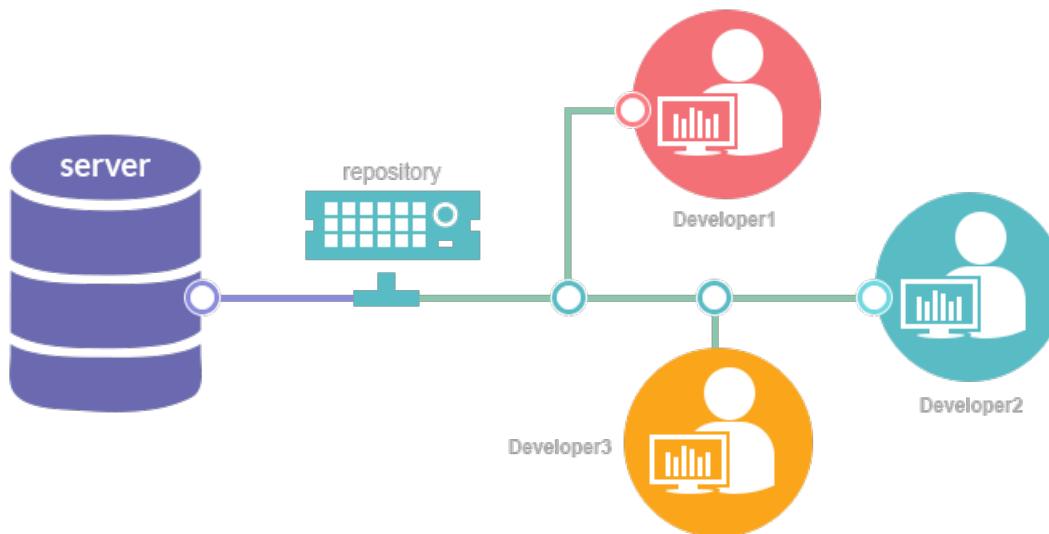
# Git & GitHub → Instalando e configurando o GIT

- Passo 1: baixar o Git no site → [git-scm.com](http://git-scm.com)
- Passo 2: baixar o Visual Studio Code → [code.visualstudio.com](http://code.visualstudio.com)
- **Iniciemos pelo Git**



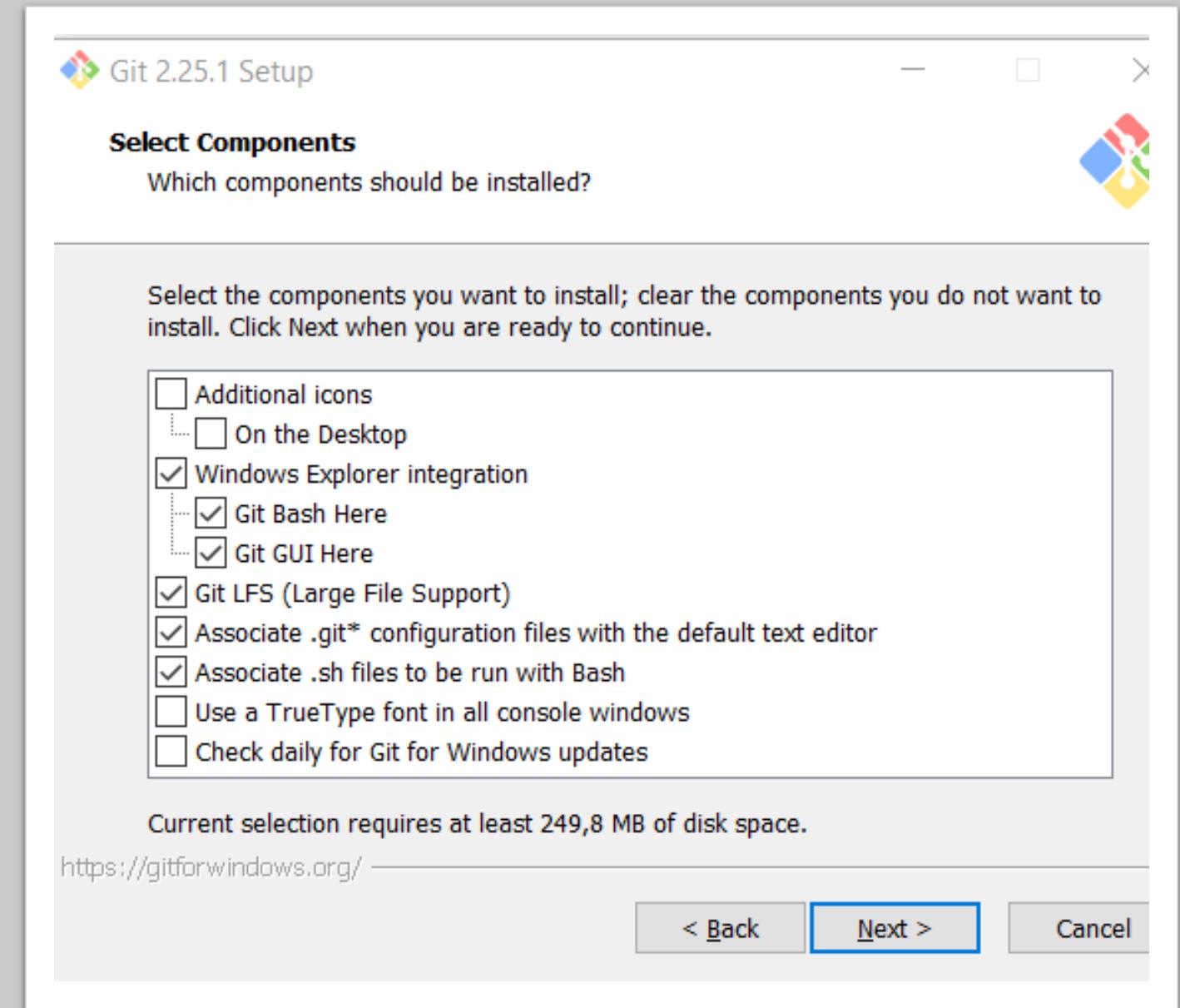
# Git & GitHub → Instalando e configurando o GIT

- Passo 3: Iniciando a instalação do Git



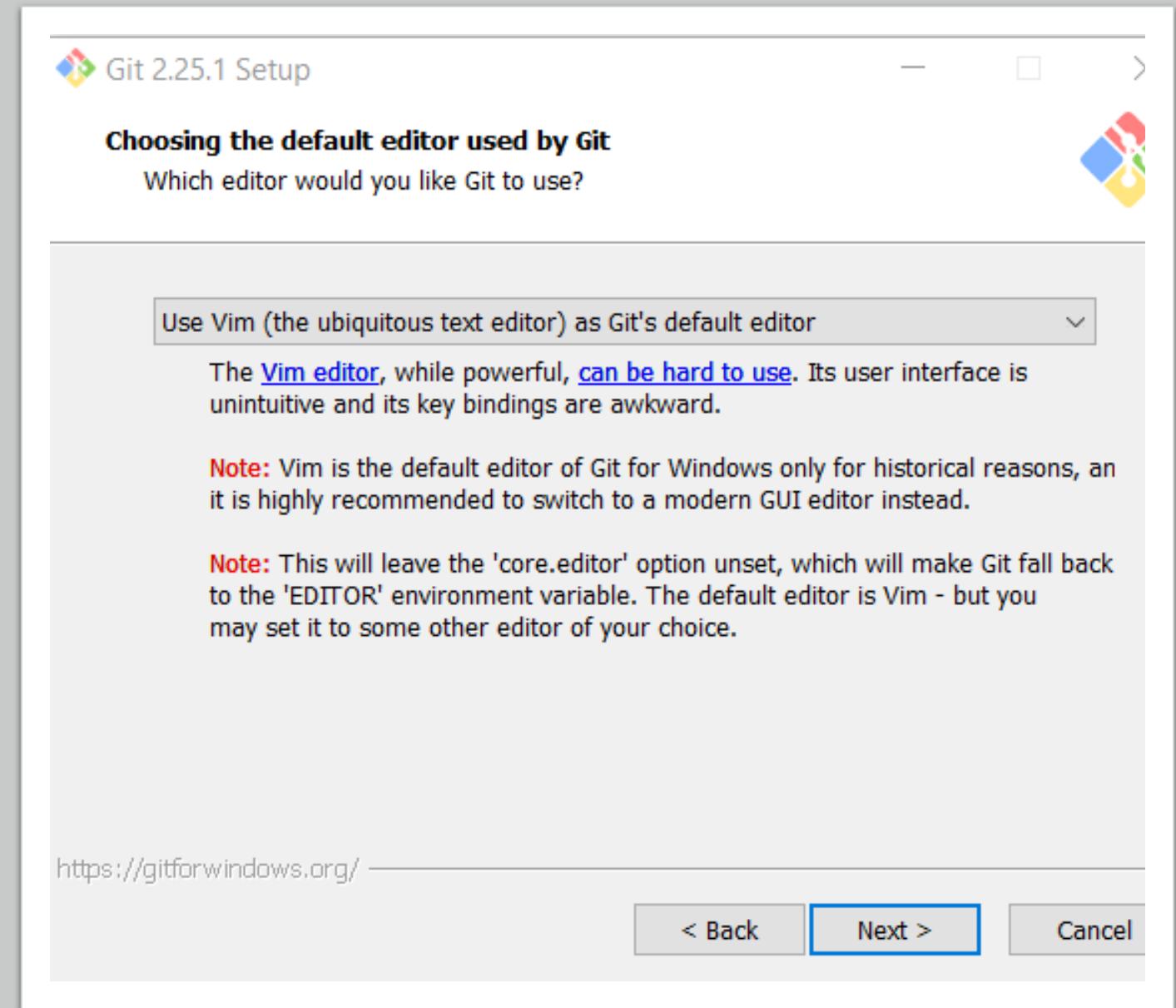
# Git & GitHub → Instalando e configurando o GIT

Passo 4: após  
aceitarmos os  
termos da licença →  
estamos prontos  
para iniciar.



# Git & GitHub → Instalando e configurando o GIT

Passo 5: Vamos  
manter a  
configuração de  
Editor padrão →  
poderá ser alterada  
mais tarde, caso  
queira.



# Git & GitHub → Instalando e configurando o GIT

Passo 6: o Git é um software de linha de comandos, então fiquemos com o padrão.

Git 2.25.1 Setup

— X



**Adjusting your PATH environment**

How would you like to use Git from the command line?

**Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

**Git from the command line and also from 3rd-party software**

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

**Use Git and optional Unix tools from the Command Prompt**

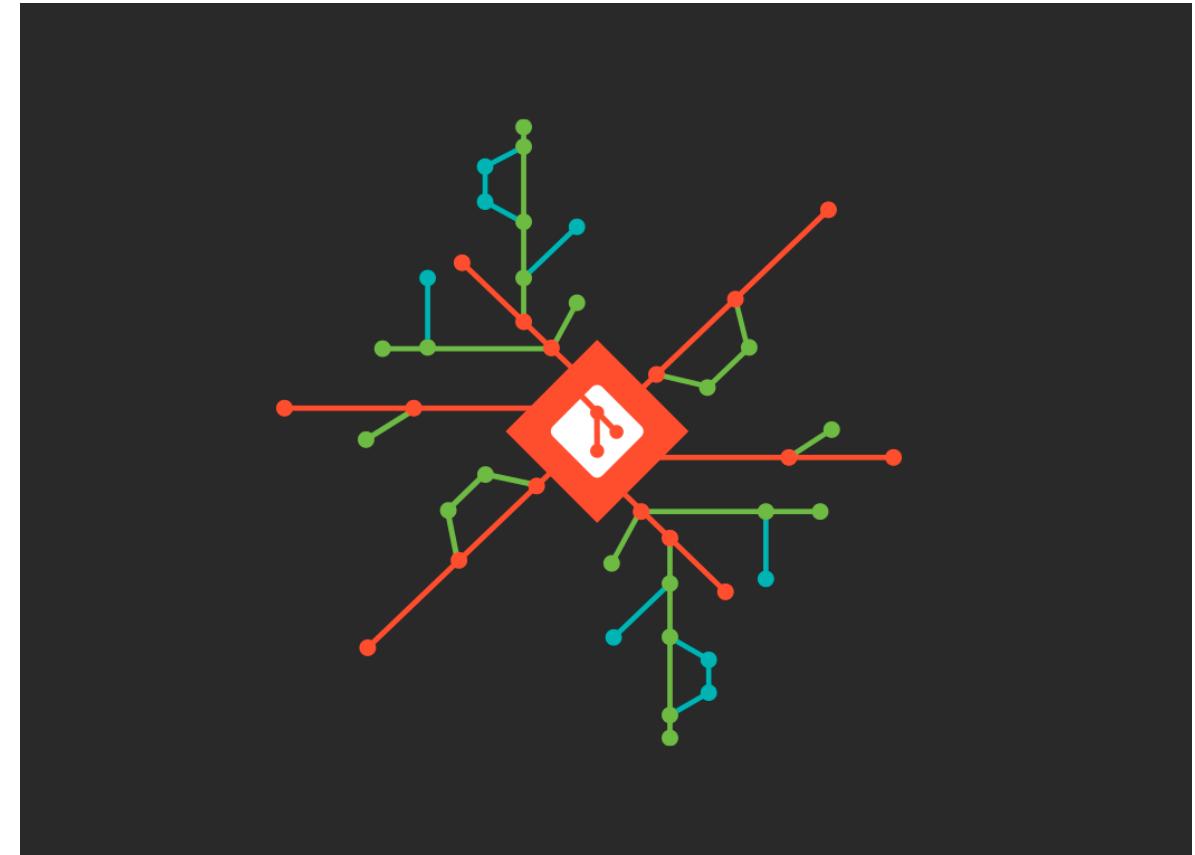
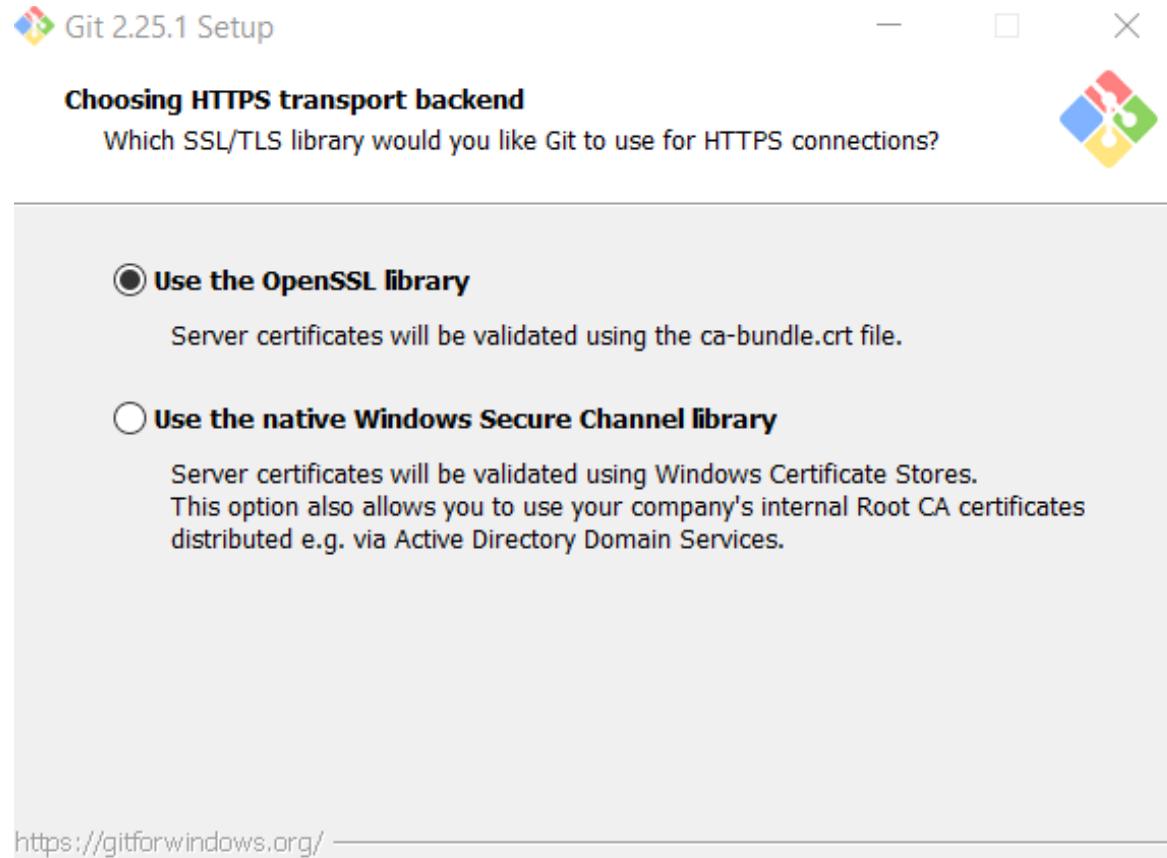
Both Git and the optional Unix tools will be added to your PATH.  
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

<https://gitforwindows.org/>



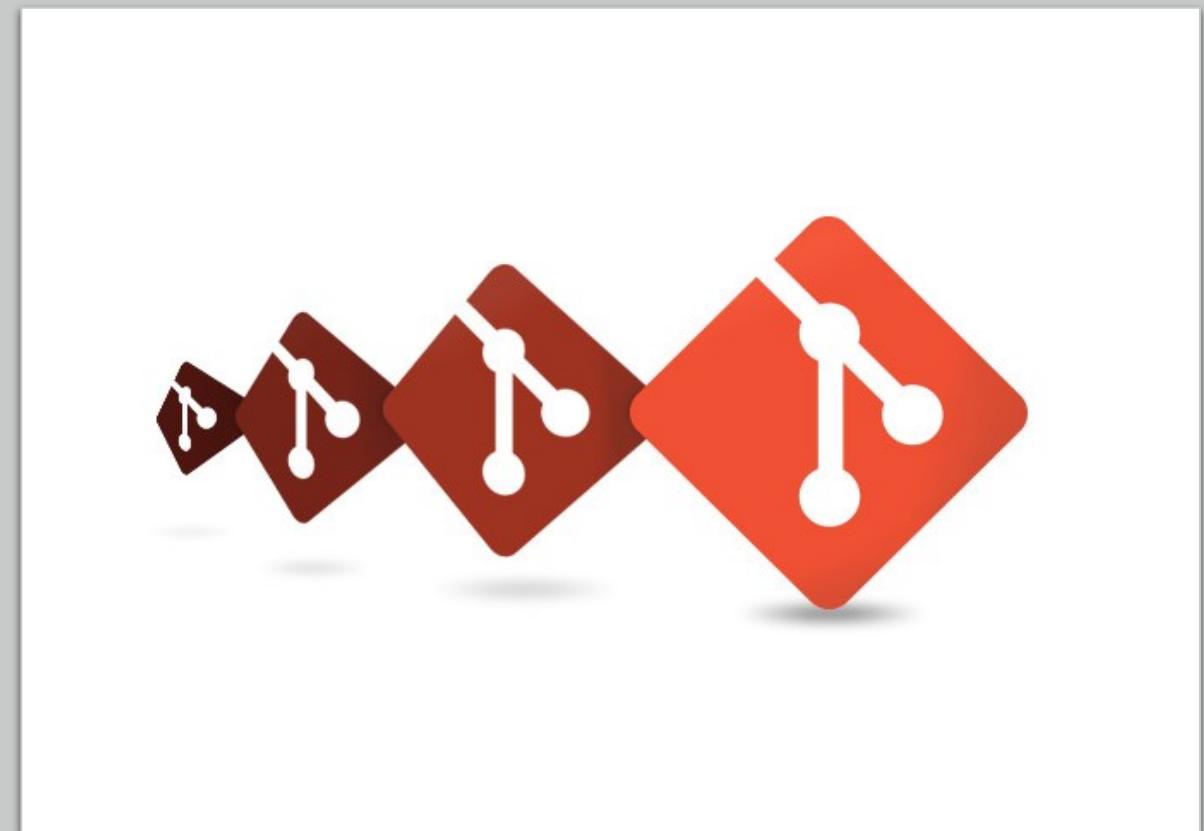
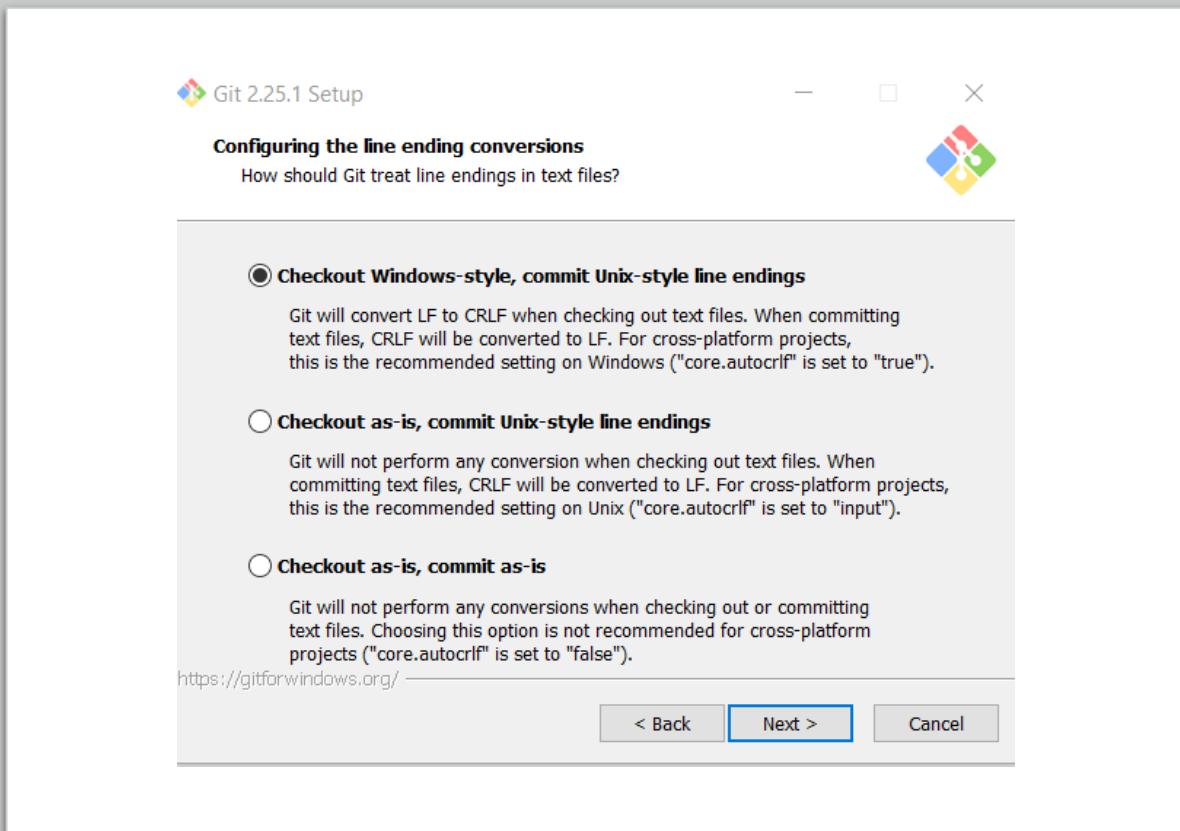
# Git & GitHub → Instalando e configurando o GIT

Passo 7: Precisamos do SSL para fazermos conexões seguras



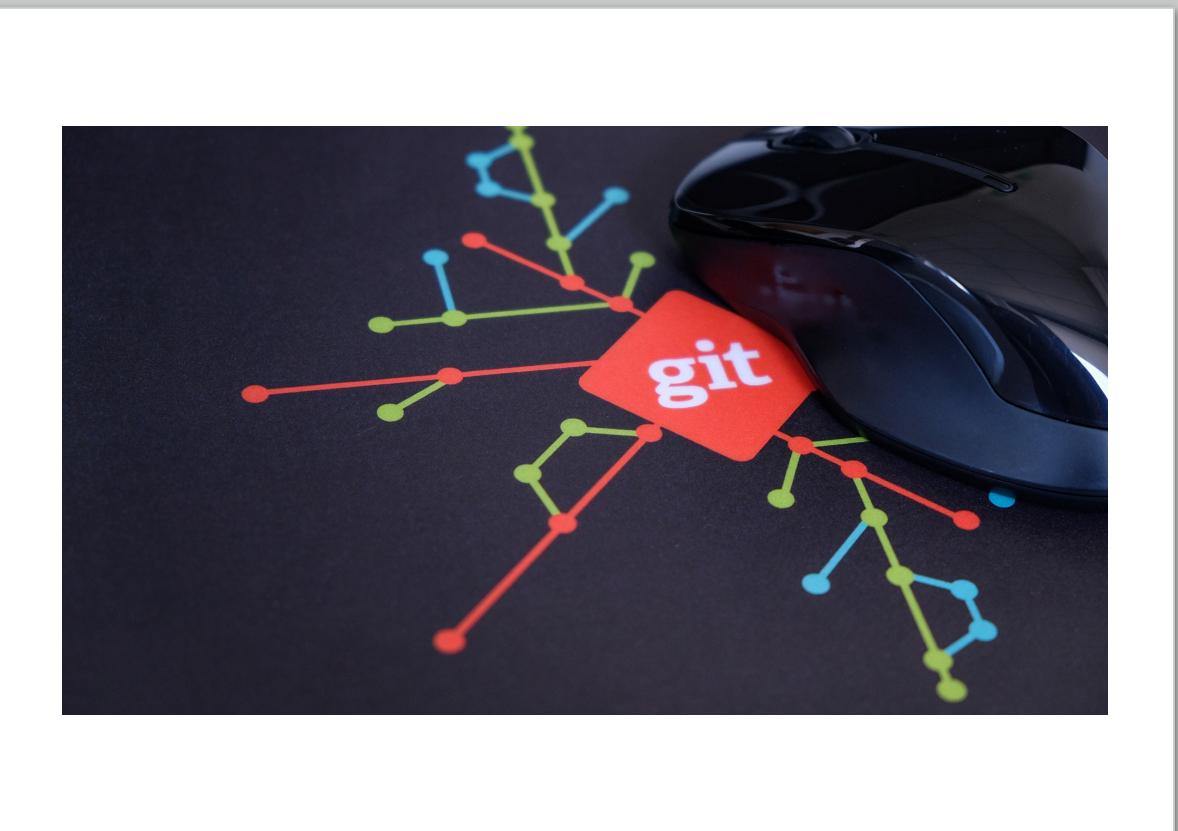
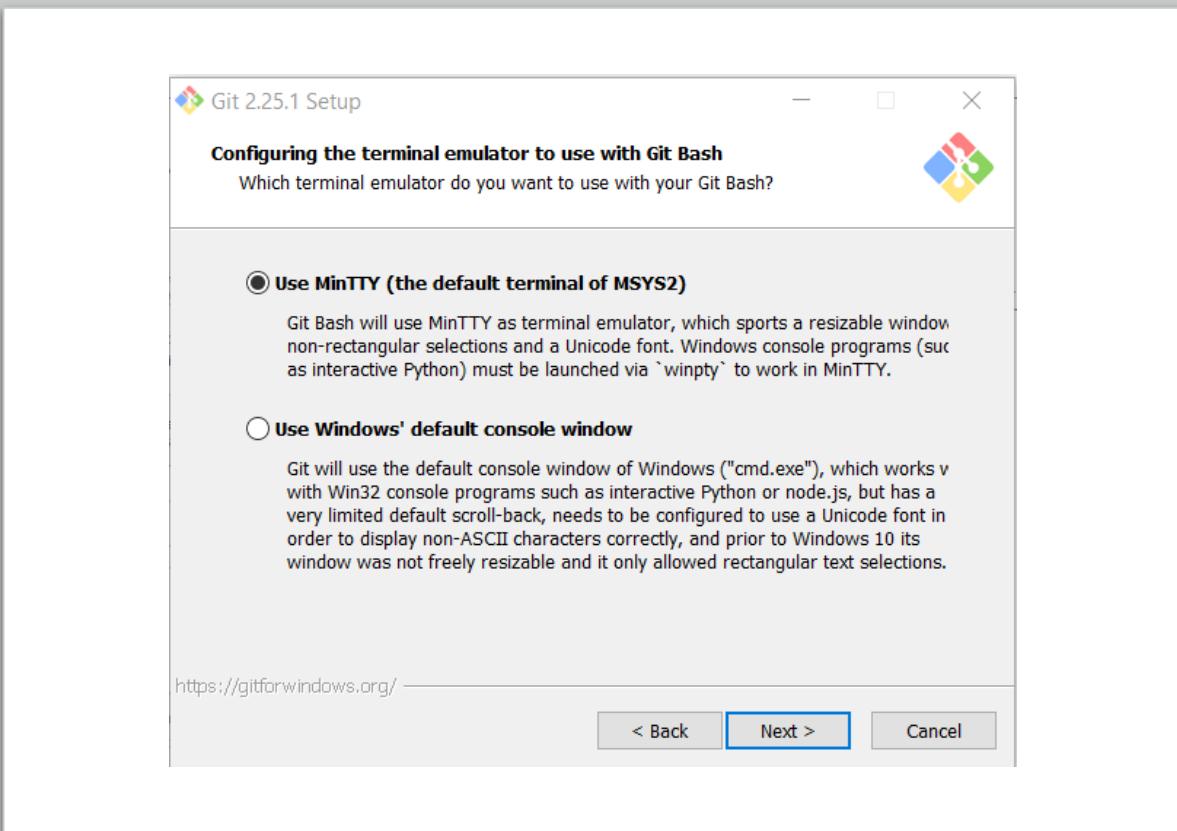
# Git & GitHub → Instalando e configurando o GIT

Passo 8: Aqui é um ponto que o Windows se diferencia do Linux e do MAC → é preciso definir como o Git deverá interpretar final de uma linha de código



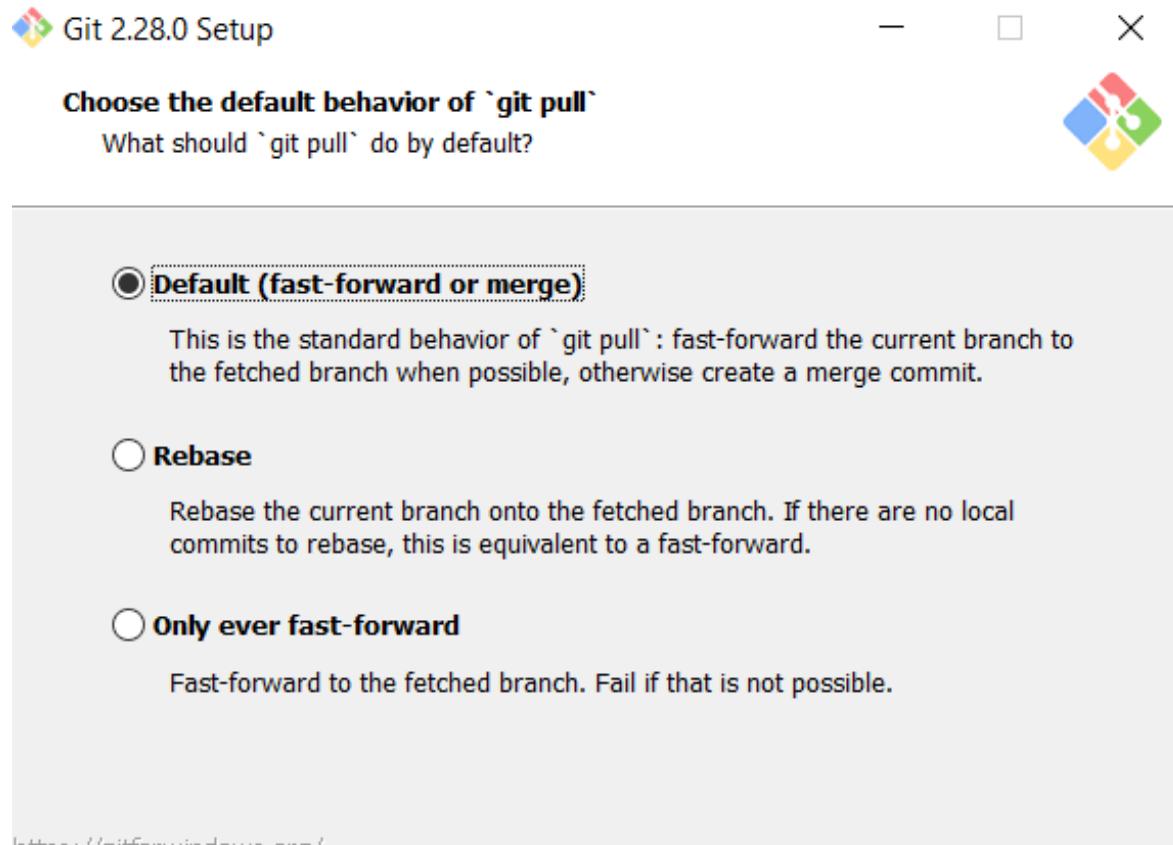
# Git & GitHub → Instalando e configurando o GIT

- Passo 9: seguimos no default poderemos usar o git bash como terminal, veremos isso no Visual Studio Code



# Git & GitHub → Instalando e configurando o GIT

Passo 10: nada a acrescentar aqui



## Git 2.28.0 Setup

### Choose a credential helper

Which credential helper should be configured?

**None**

Do not use a credential helper.

**Git Credential Manager**

The [Git Credential Manager for Windows](#) handles credentials e.g. for Azure DevOps and GitHub (requires .NET framework v4.5.1 or later).

**Git Credential Manager Core**

(NEW!) Use the new, [cross-platform version of the Git Credential Manager](#). See more information about the future of Git Credential Manager [here](#).

//gitforwindows.org/

Only show new options

< Back

Next >

## Passo 11: Siga no default

Git & GitHub →  
Instalando e  
configurando o  
GIT

# Git & GitHub →

## Instalando e configurando o GIT



Passo 12: Selecionamos as três opções a seguir.

**Configuring extra options**  
Which features would you like to enable?



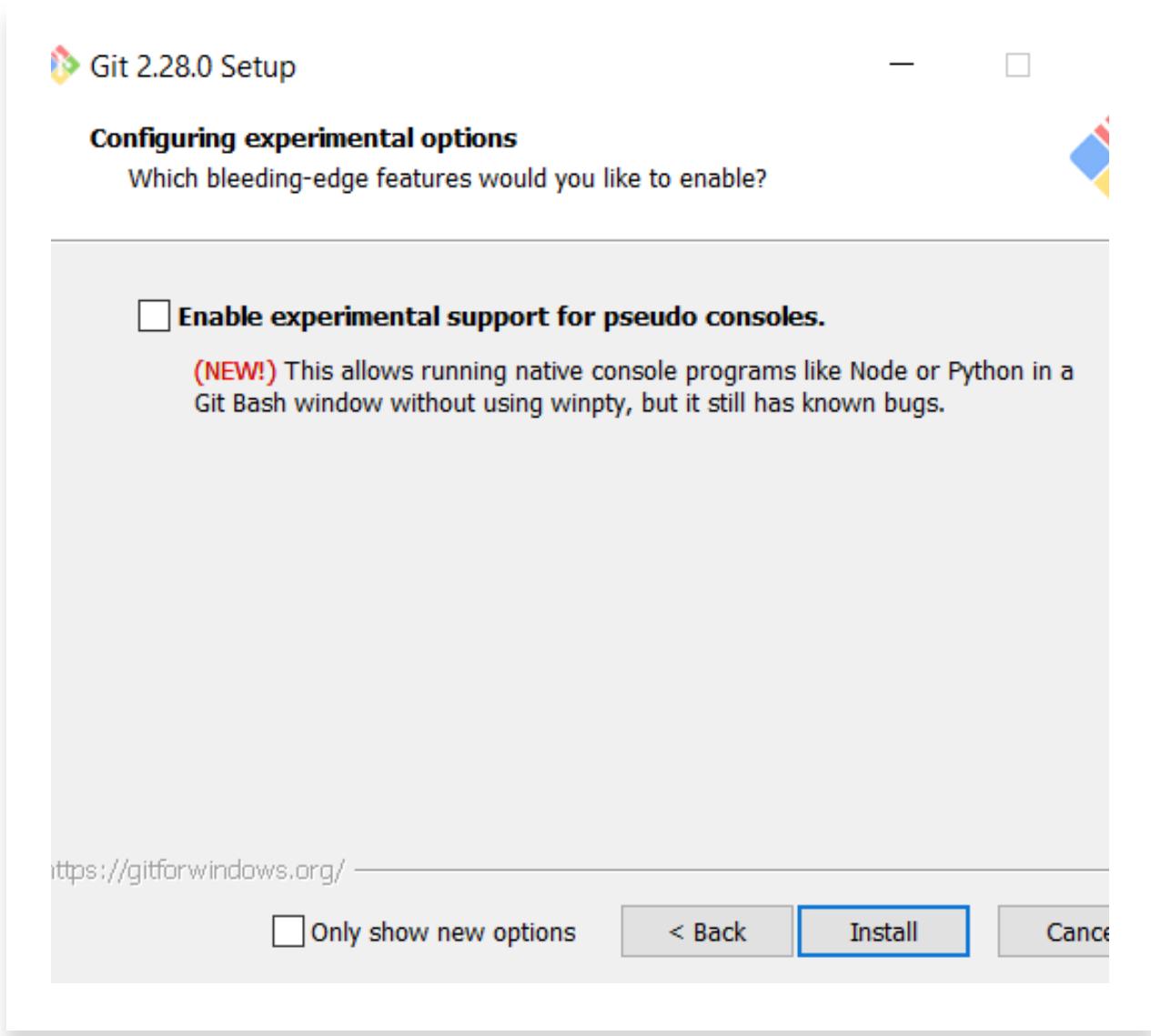
**Enable file system caching**  
File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

**Enable Git Credential Manager**  
The [Git Credential Manager for Windows](#) provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or later).

**Enable symbolic links**  
Enable [symbolic links](#) (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

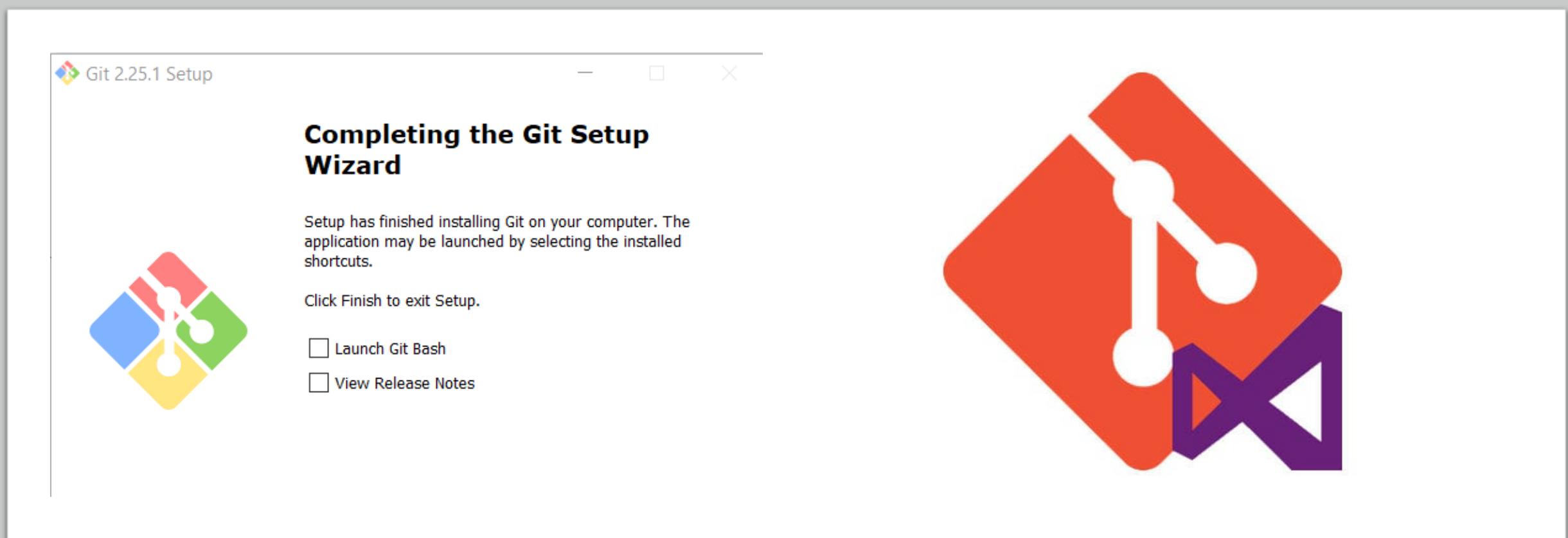
## Passo 13: Deixe em branco e siga em frente

Git & GitHub →  
Instalando e  
configurando o  
GIT



# Git & GitHub → Instalando e configurando o GIT

Passo 14: Fim → a seguir  
configurando o **Visual Studio**  
**Code**. Atenção você pode usar o terminal de  
sua preferência



## Git & GitHub → Instalando e configurando o GIT

Baixar e instalar o Visual Studio Code

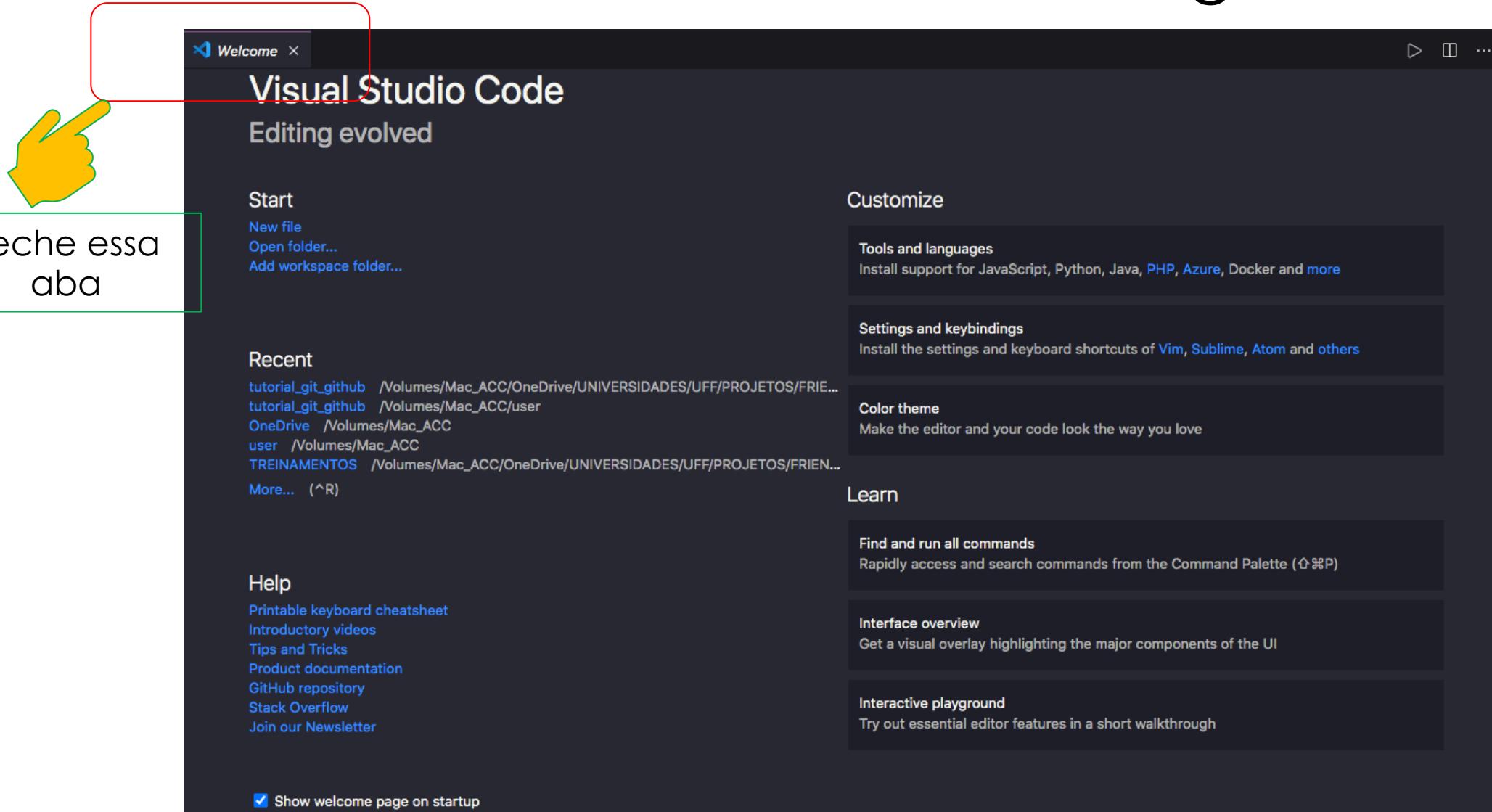
Processo de Instalação normal ... Padrão Windows

Lembrando → o Git é um software de linha de comando, o que demanda o uso de um **terminal**. É aí que entra o **Visual Studio Code**

Preparar o **Terminal** para trabalhar com Git, não é obrigatório, mas facilita bastante, veja como:

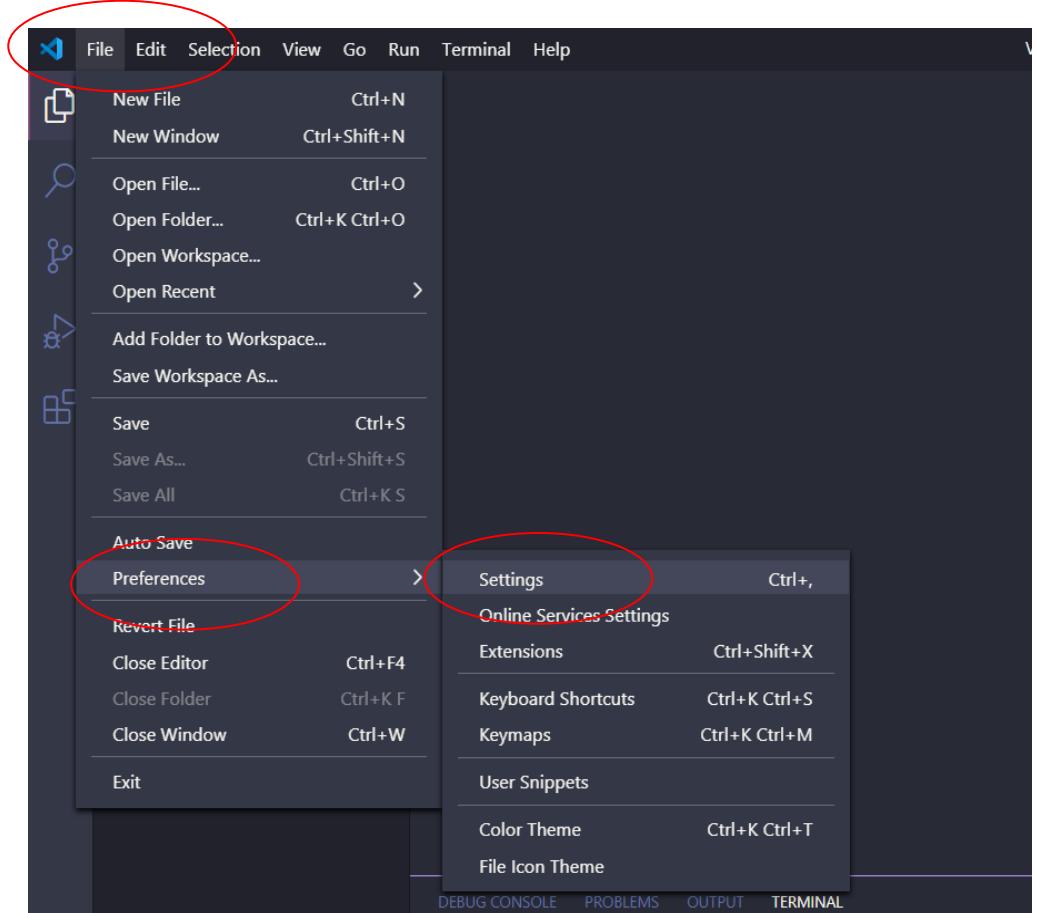
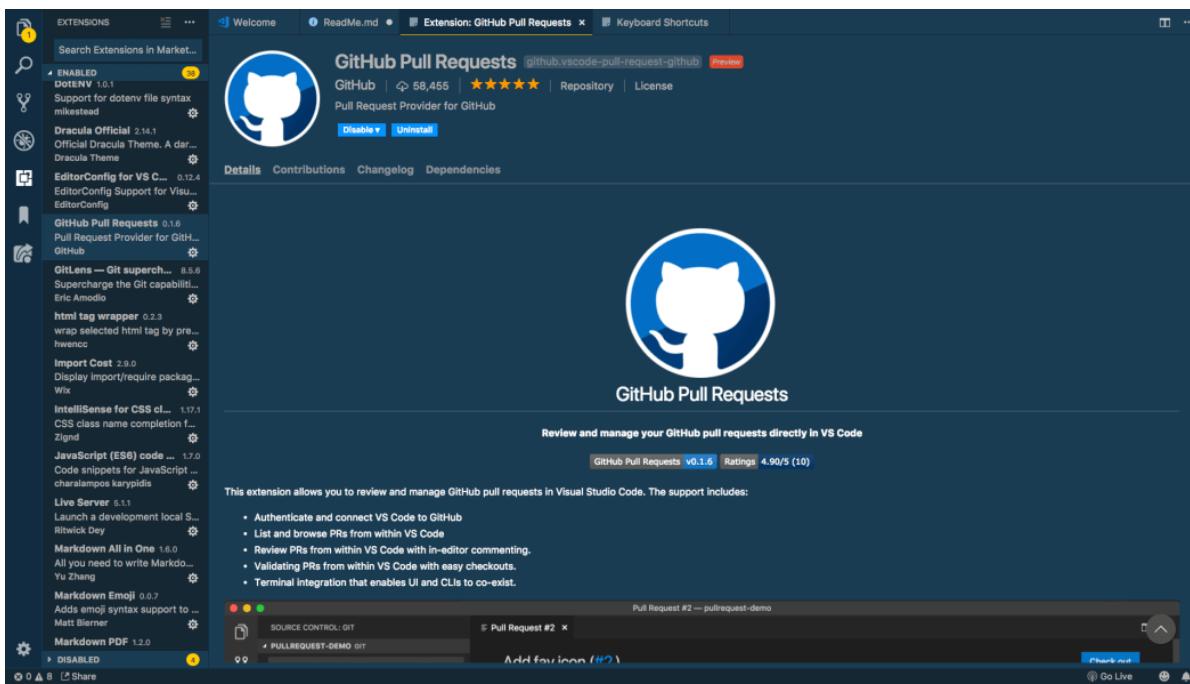


# Ao abrir o VsCode você terá a seguinte tela



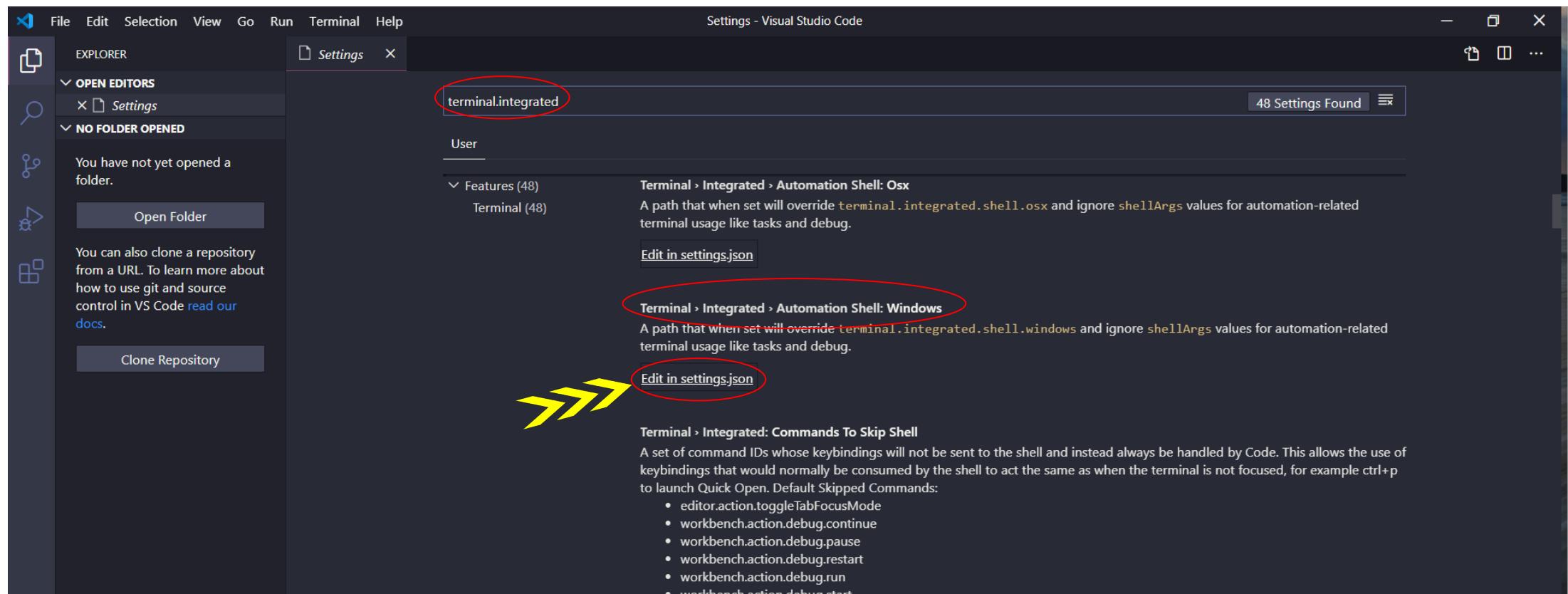
# Git & GitHub → Instalando e configurando o GIT

- Passo 15: Primeiro, vá em: File → Preferences → Settings



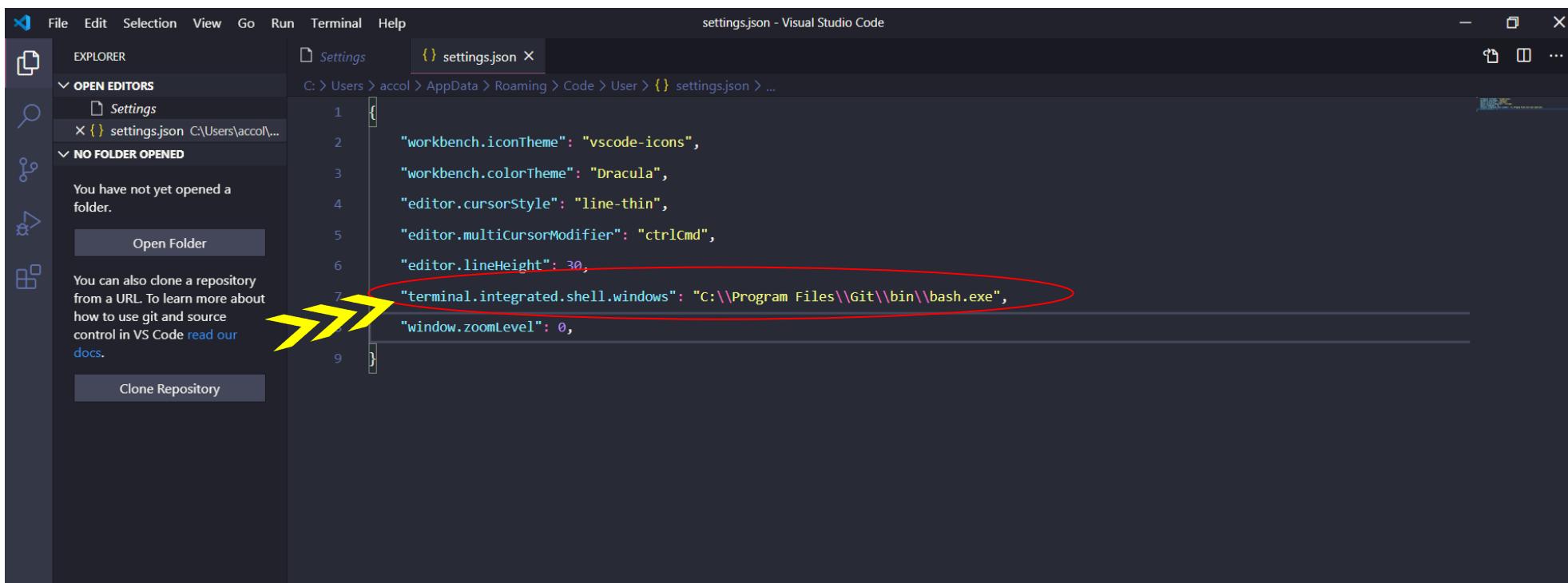
# Git & GitHub → Instalando e configurando o GIT

- Passo 16: Digite → terminal.integrated → veja abaixo:  
procure por Shell Windows e selecione Edit in settings.json



# Git & GitHub → Instalando e configurando o GIT

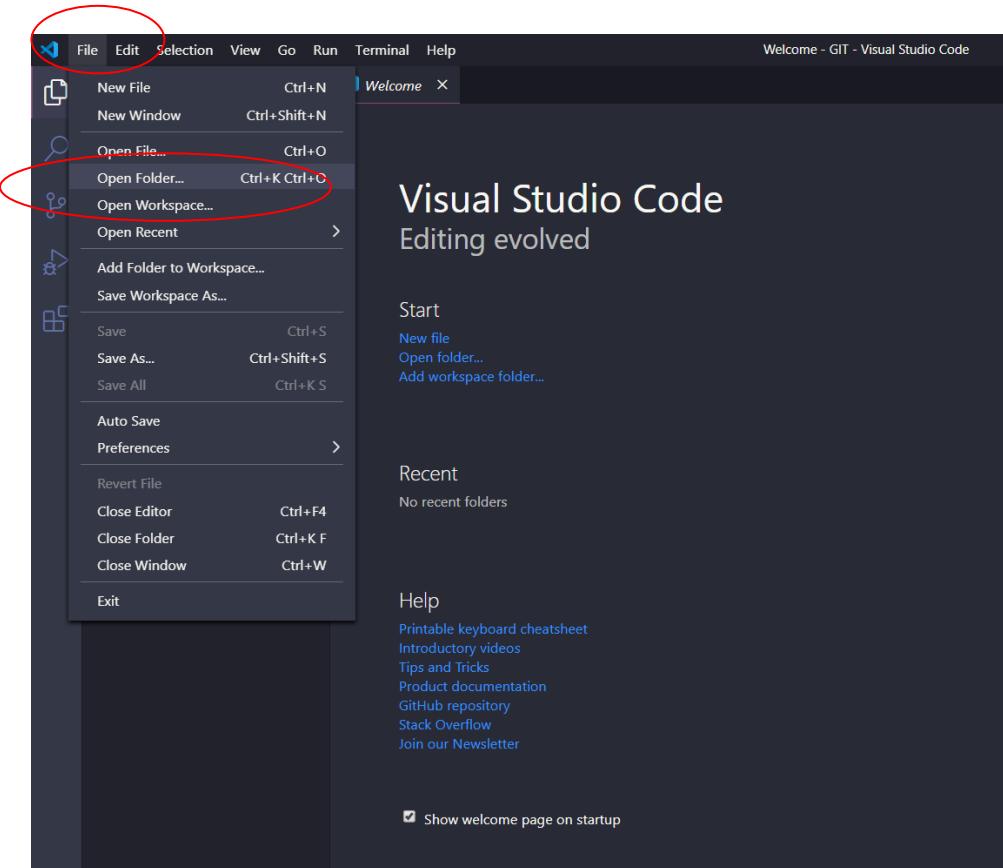
- Passo 17: Na tela de edição acrescente a linha assinalada. Note que, no caso o Git esta instalado em C:\Program Files\Git → verifique o seu caso. Salve e reinicialize o Visual Studio Code.



```
settings.json - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER Settings settings.json X
C: > Users > accol > AppData > Roaming > Code > User > { } settings.json > ...
1 {
2   "workbench.iconTheme": "vscode-icons",
3   "workbench.colorTheme": "Dracula",
4   "editor.cursorStyle": "line-thin",
5   "editor.multiCursorModifier": "ctrlCmd",
6   "editor.lineHeight": 20,
7   "terminal.integrated.shell.windows": "C:\\Program Files\\Git\\bin\\bash.exe",
8   "window.zoomLevel": 0,
9 }
```

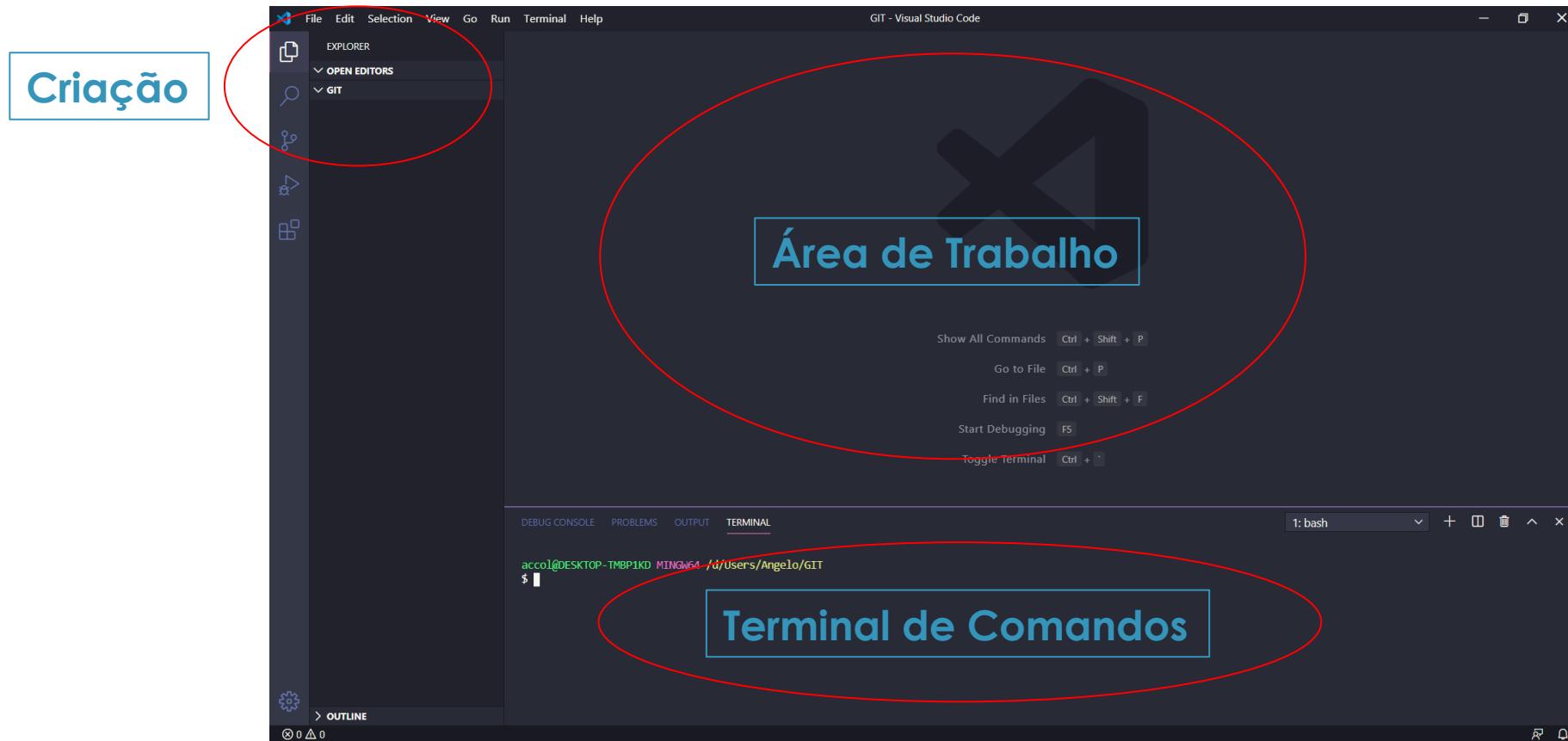
# Git & GitHub → Instalando e configurando o GIT

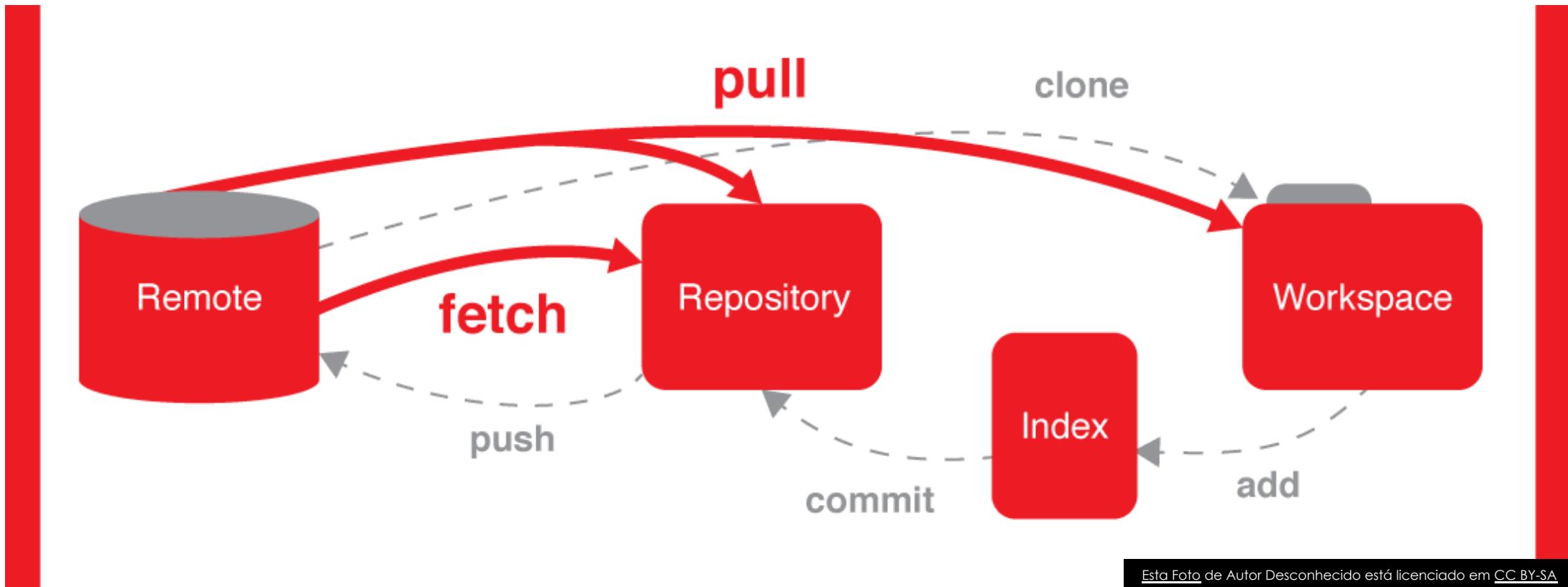
- Passo 18: O próximo passo é criar uma pasta de trabalho. No meu caso, criei como exemplo, a pasta GIT. Feito isso, vamos abri-la no Visual Studio Code.



# Git & GitHub → Instalando e configurando o GIT

- Passo 19 Agora estamos prontos para trabalhar criando novos arquivos e acessando o terminal do Visual Studio Code.





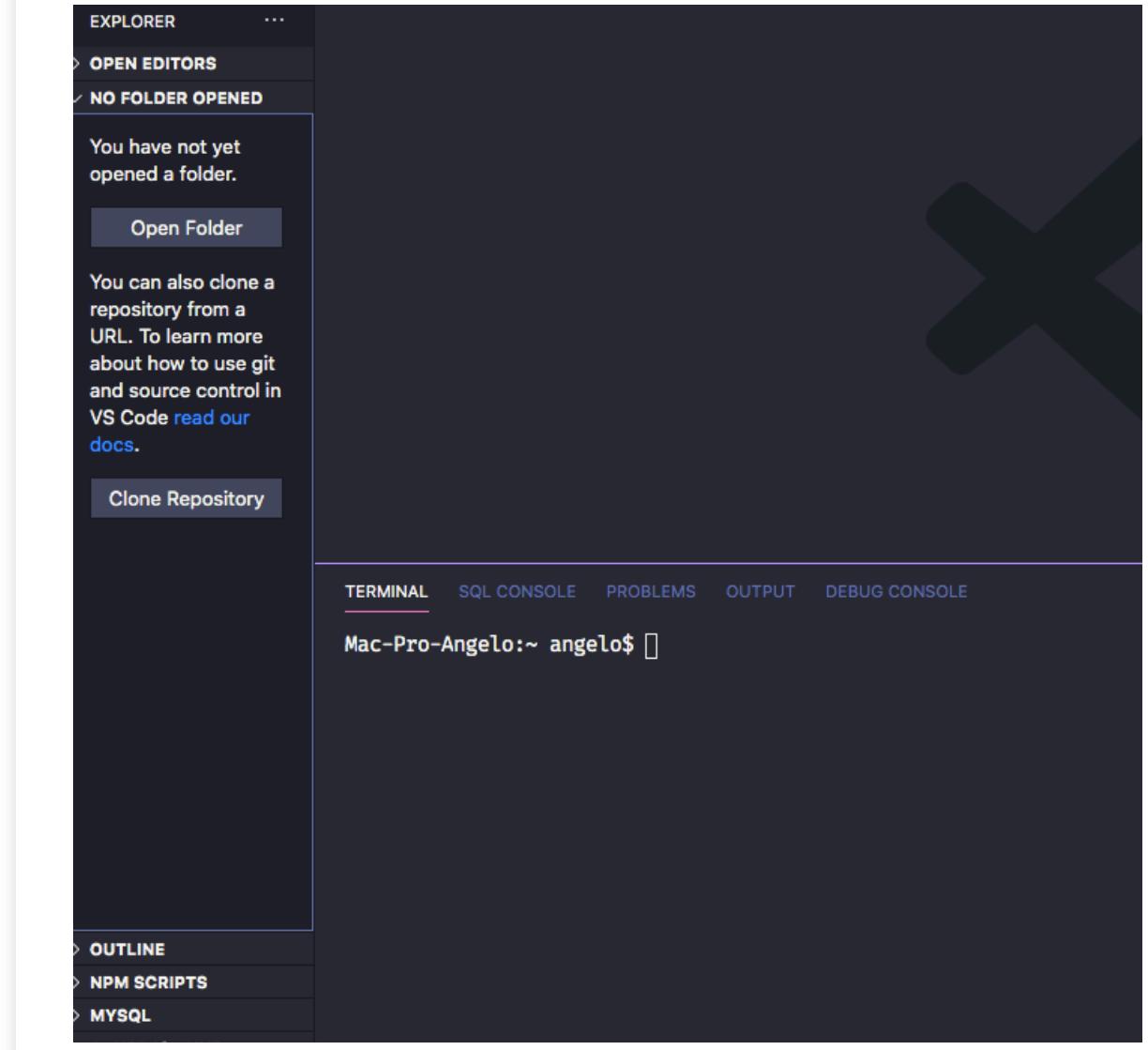
Esta Foto de Autor Desconhecido está licenciado em CC BY-SA

## Git & GitHub → Instalando e configurando o GIT

- Como sabem o git é um software de linha de comando, logo precisa de um terminal. Como sugestão apontamos para o VsCode, mas poderia tranquilamente ser utilizado outro, não recomendo.
- Para acessa o terminal integrado e iniciar seus trabalhos, vamos descrever de forma resumida alguns passos, avaliem se ajuda.

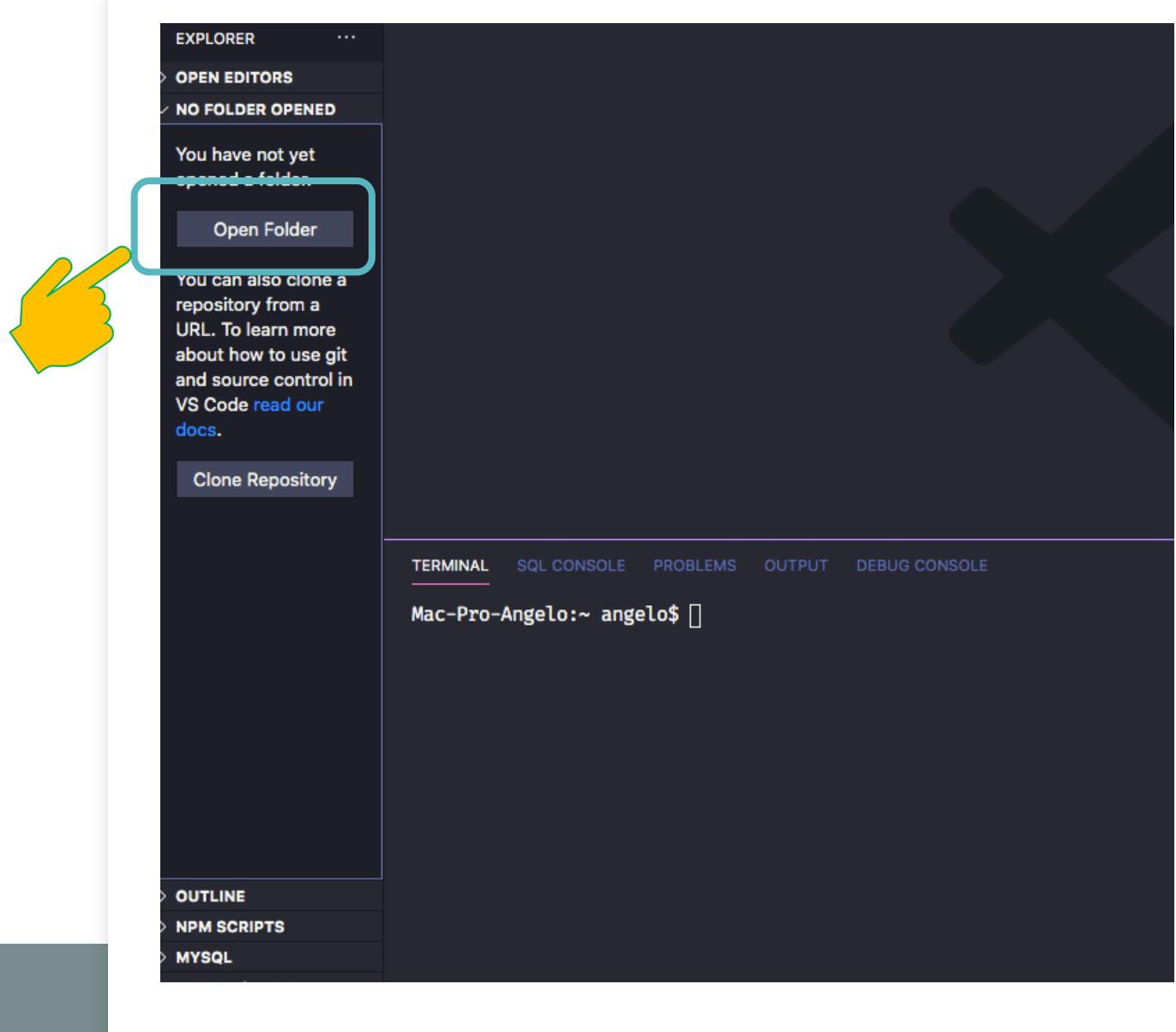
# Git & GitHub → Instalando e configurando o GIT

- Abra seu VsCode, fecha a aba Welcome.
- Pressione CTRL + SHIFT + ACENTO CRASE `
- Você terá algo como:



# Git & GitHub → Instalando e configurando o GIT

- Neste momento você deverá abrir seu diretório de trabalho, no caso, criei um diretório chamado tutorial\_git\_github, mas lembre-se, você terá seus diretórios de trabalho.
- Use essa tela de início como atalho, veja:
- Processo normal, escolha seu diretório de trabalho



[Find and run all commands](#)

TERMINAL

SQL CONSOLE: MESSAGES

PROBLEMS

OUTPUT

DEBUG CONSOLE

1: sh

Mac-Pro-Angelo:tutorial\_git\_github angelo\$



# Git & GitHub → Instalando e configurando o GIT

- Aberto seu diretório, terá algo como:
- Observe que no terminal tenho exatamente meu diretório de trabalho
- A partir daí aplique tudo que aprender neste tutorial

# Git & GitHub → Instalando e configurando o GIT

- Antes de finalizar, uma última palavra sobre o uso do terminal.
- Pode acontecer de ao usar um comando **git** ele necessite mais de uma tela do terminal.
- Quando isso acontecer aparecerá na tela o sinal de dois pontos :
- Você deverá digitar **<Enter>** para continuar rolando a tela, ou digitar a letra **q** para sair da rolagem de tela e ter o terminal de volta a seu controle

TERMINAL    SQL CONSOLE: MESSAGES    PROBLEMS    OUTPUT    DEBUG CONSOLE

```
credential.helper=osxkeychain
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
remote.origin.url=https://mfake0323@github.com/mfake0323/tutorial_git_gi
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
:
```



Git & GitHub →  
Instalando e  
configurando o GIT

- Neste ponto, digite **enter** para rolar a tela, ou **q** para sair da tela.
- Atenção se optar por digitar **enter** chegará a um ponto onde terá **END**: neste momento terá que digitar **q**

TERMINAL

SQL CONSOLE: MESSAGES

PROBLEMS

OUTPUT

DEBUG CONSOLE

```
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
remote.origin.url=https://mfake0323@github.com/mfake0323/tutorial_git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
user.name=mfake0323
user.email=mfake0323@gmail.com
```

(END)

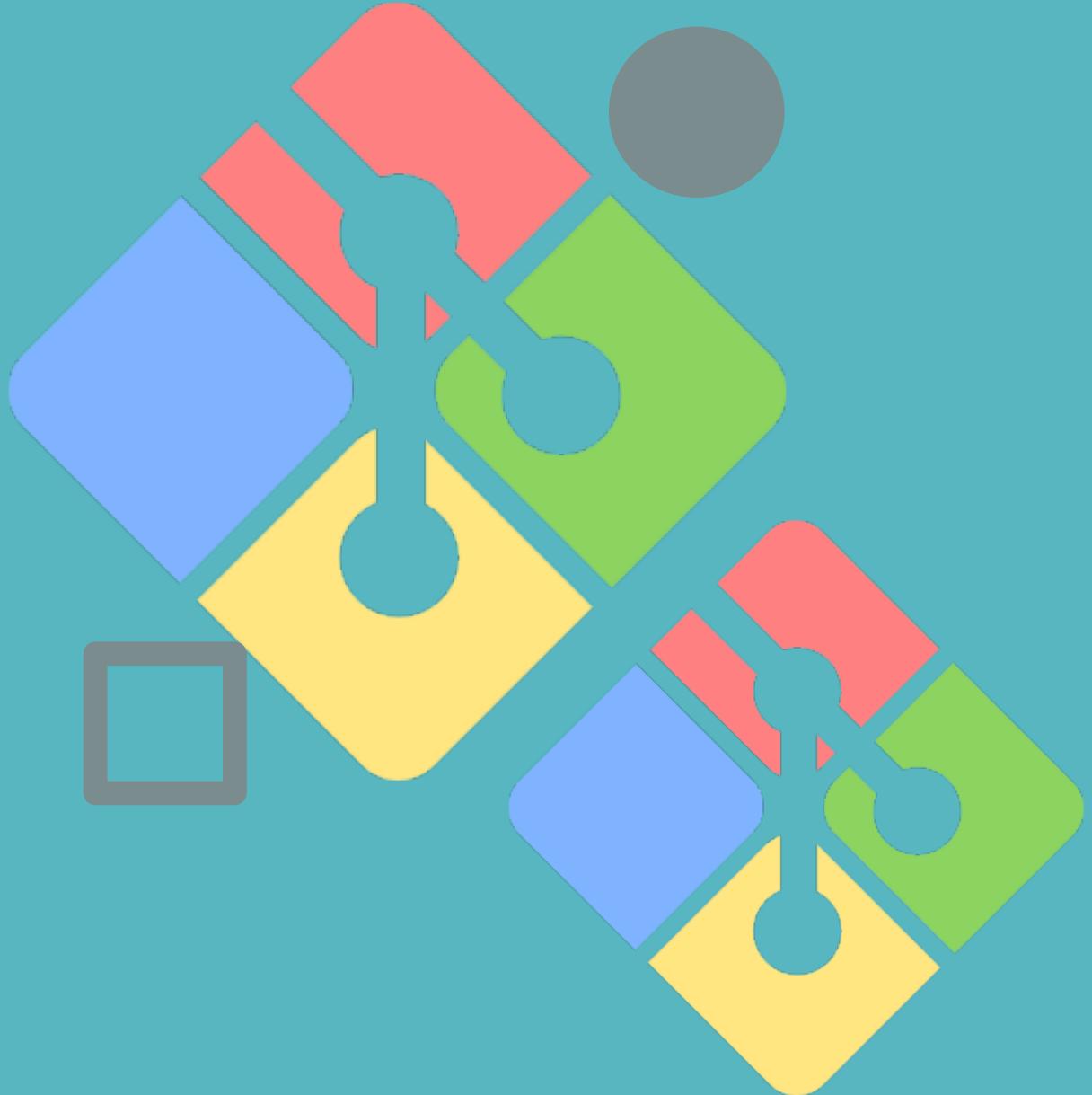


Git & GitHub →  
Instalando e  
configurando o G

- Neste ponto, necessariamente terá que digitar a letra **q**

Vamos agora  
entender como  
iniciamos o processo  
de versionamento

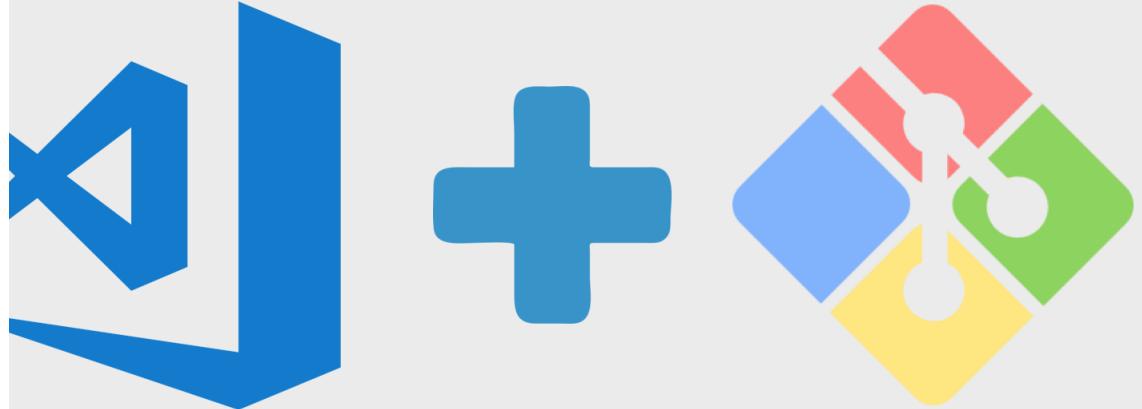
Os comandos Git Init e Git  
Config



# GIT INIT & GIT CONFIG

Para iniciarmos o versionamento de um projeto, tudo o que você precisa é do comando → git init

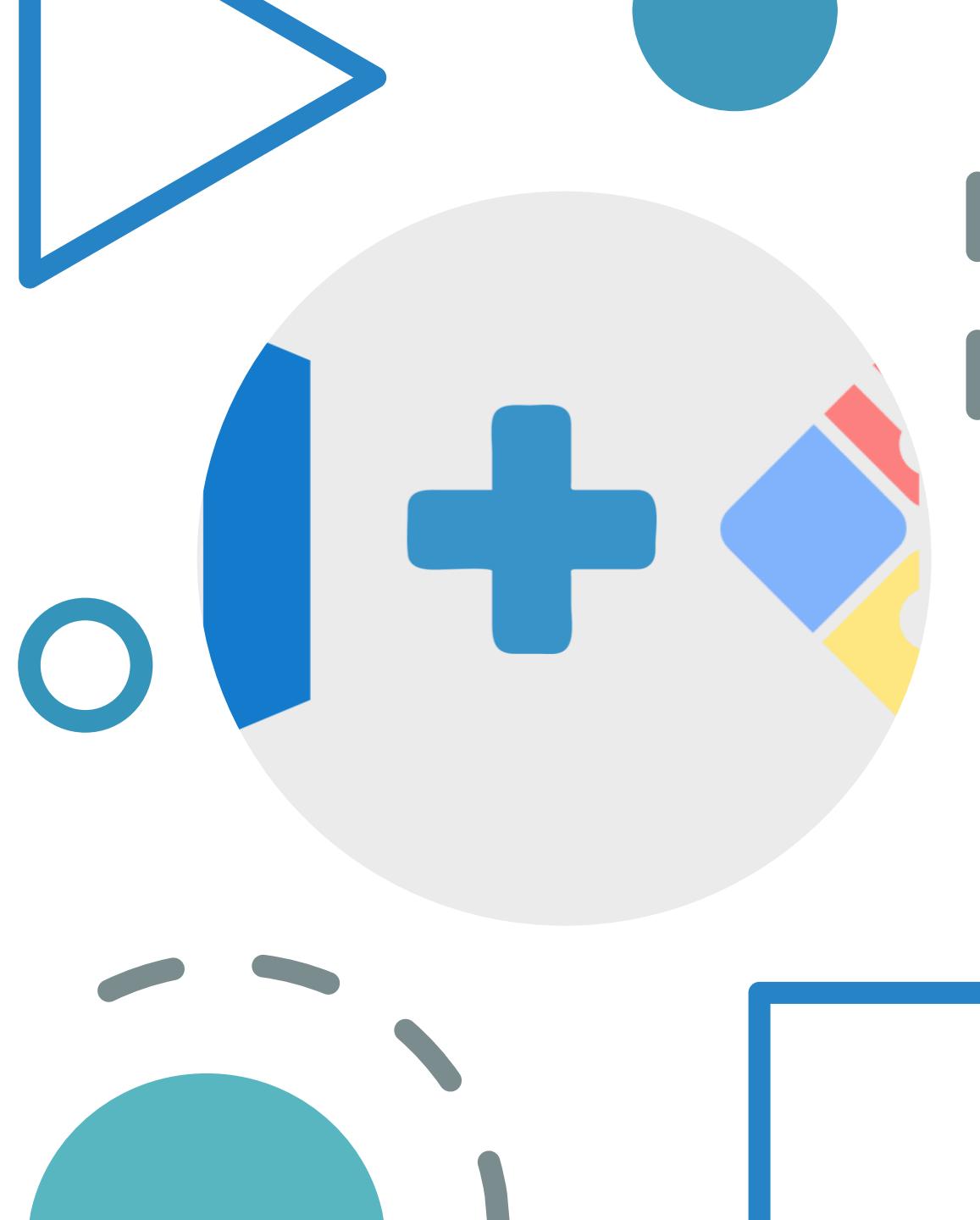
Não importa se a pasta está vazia, ou se arquivos de seu projeto já estejam salvos, o git irá habilitar seu repositório para o versionamento



```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL
ccol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT
git init
    initialized empty Git repository in D:/Users/Angelo/GIT
ccol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
[]
```

# GIT INIT & GIT CONFIG

- O Git irá criar uma pasta **.git** no diretório principal de seu projeto (esta pasta por default é oculta)
- **Atenção, não mexa, não altere, não apague esta pasta**
- A partir daí, tudo o que estiver presente ou que você venha a acrescentar, ou ainda ambos, podem ou não serem versionados pelo Git
- O que precisamos a partir daqui é de algumas configurações a mais, vamos a elas
- Chegou o momento de conhecermos o **git config**



# GIT INIT & GIT CONFIG

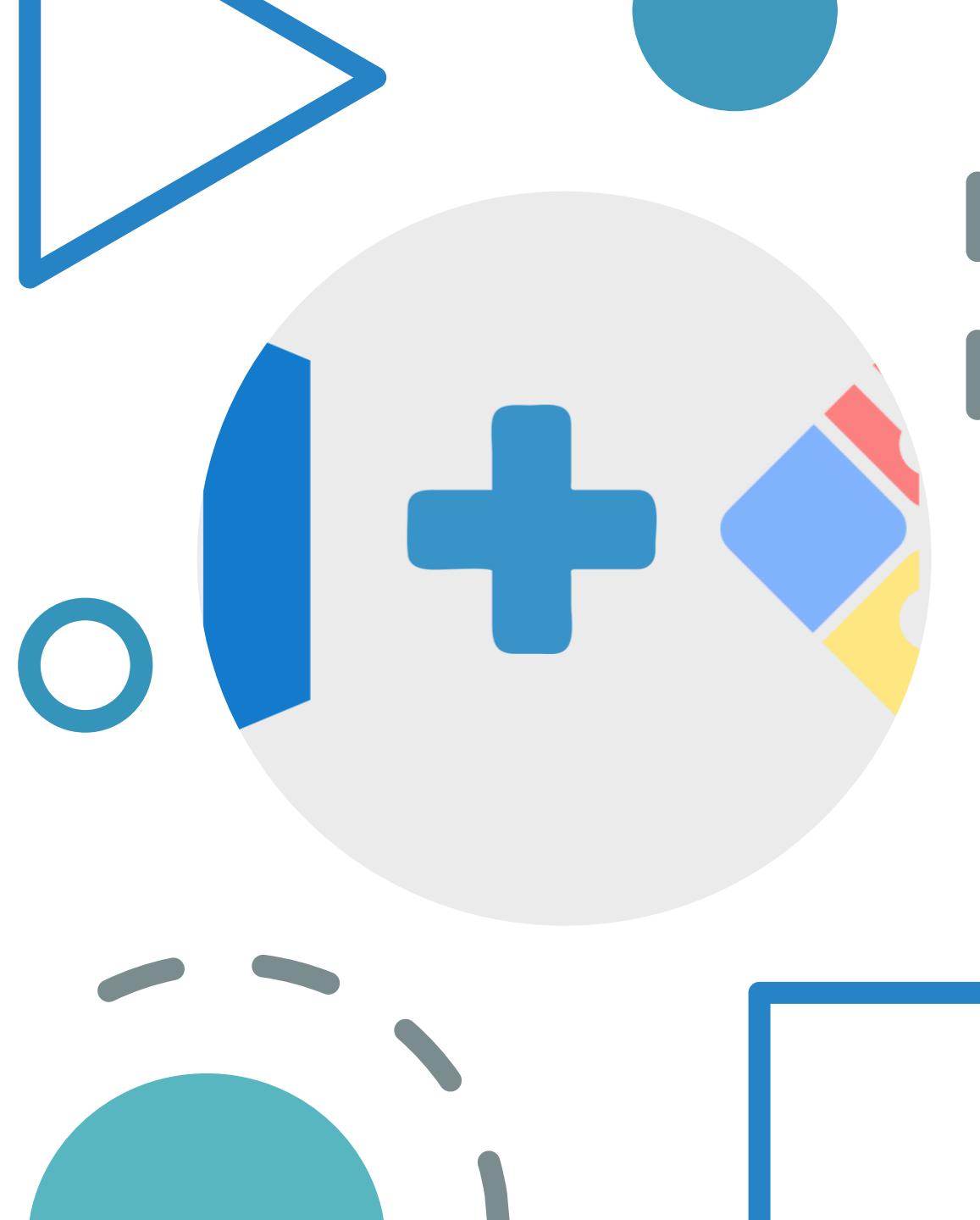
- Através do **git config** poderemos controlar quem estará trabalhando no projeto
- Sempre que iniciar um repositório git. Logo após o git init, faça os passos abaixo:

**Defina o user name e o emial de trabalho:**

```
git config user.name <seu_nome_usuário>
git config user.email <seu_email_de_trabalho>
```

**Para conferir se sua configuração está ok, digite (é menos l de list):**

```
git config -l
```





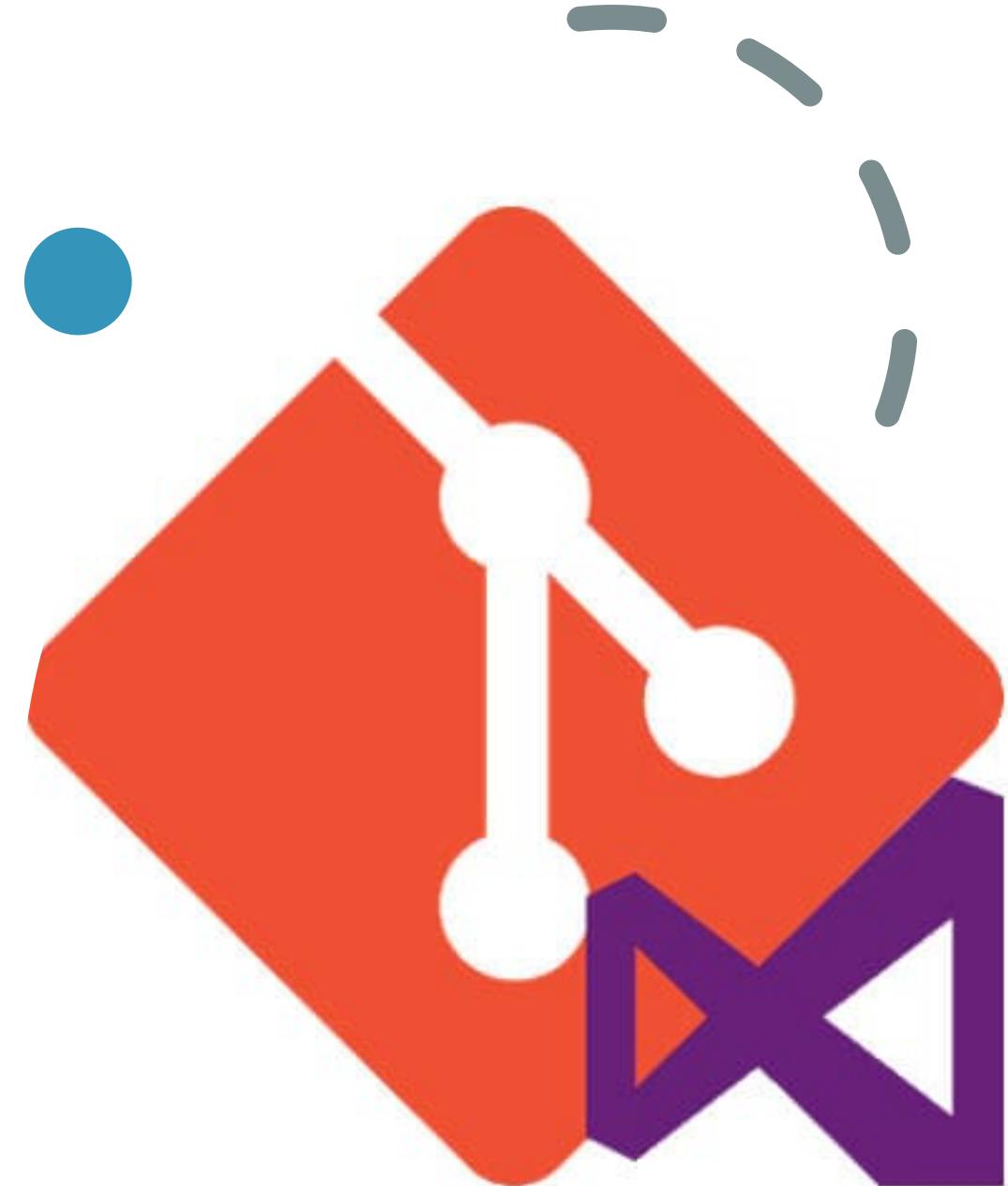
Agora é hora de  
entendermos  
como o Git  
trabalha

O Fluxo de Trabalho do Git



# Fluxo de Trabalho do Git

- Quando digitamos o **git init** e o **git config** numa pasta de Projeto, o que na verdade estamos fazendo?
- Estamos dizendo para o Git rastrear e controlar todas as alterações que forem realizadas no projeto
- As subpastas de seu projeto também serão controladas, o que é muito bom, você pode, caso queira definir que determinada pasta ou arquivo não deva ser rastreado pelo Git, isso veremos em breve, aguarde.



# Fluxo de Trabalho do Git

## Mas atenção!!!

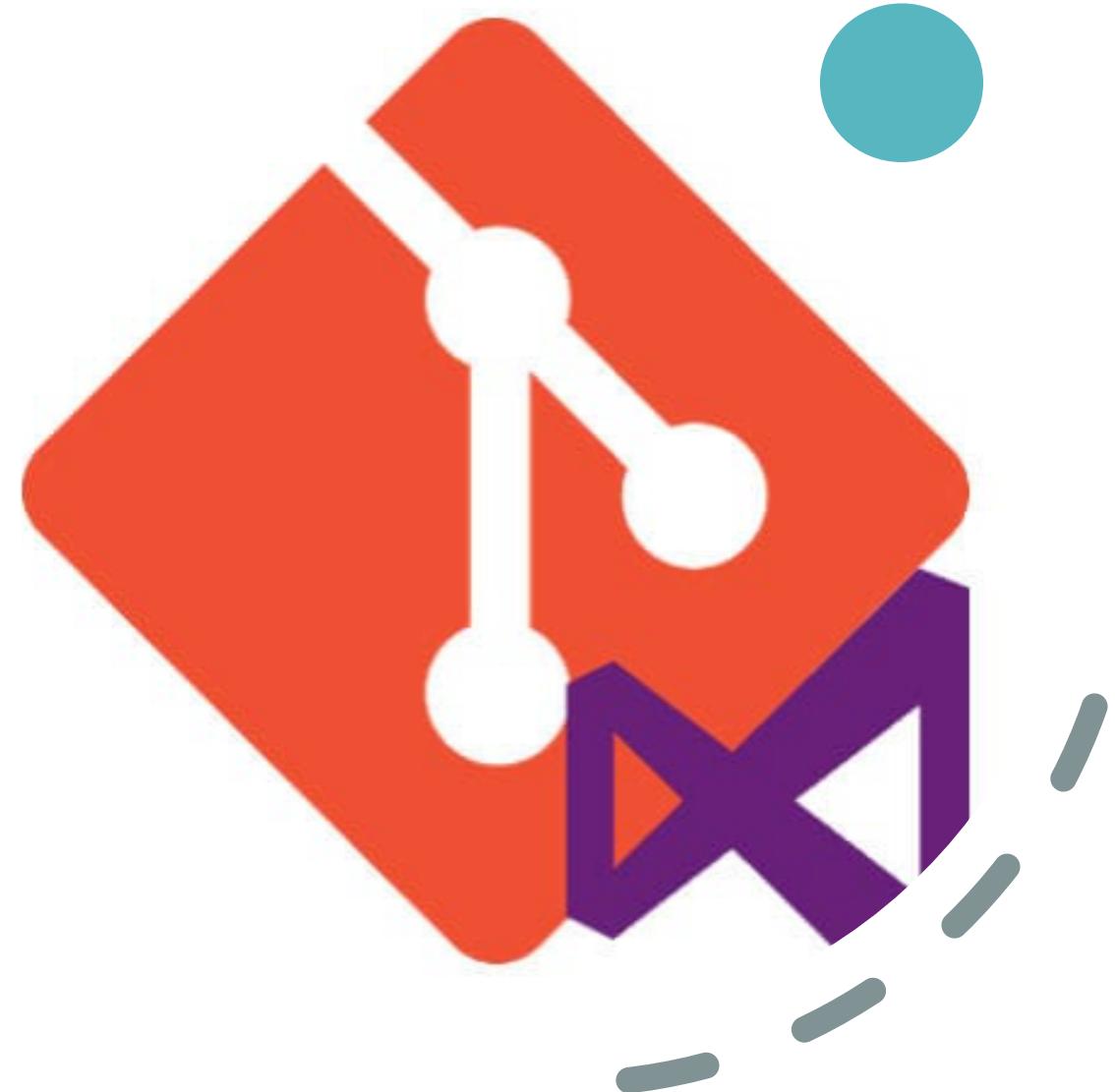
Sempre que você cria um arquivo ou uma pasta no seu diretório do projeto, o Git irá considerá-lo **untracked**, isto é, ele está em um estado não rastreável pelo Git

É comum você ter arquivos que não deseja rastrear, por exemplo, caso tenha baixado algum texto para completar seu projeto e usado apenas uma pequena parte, você **poderá** preferir descartar, apagar esse arquivo a controlar seu histórico, seria o equivalente a um arquivo temporário

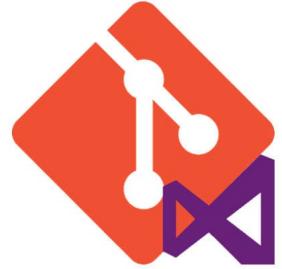


# Fluxo de Trabalho do Git

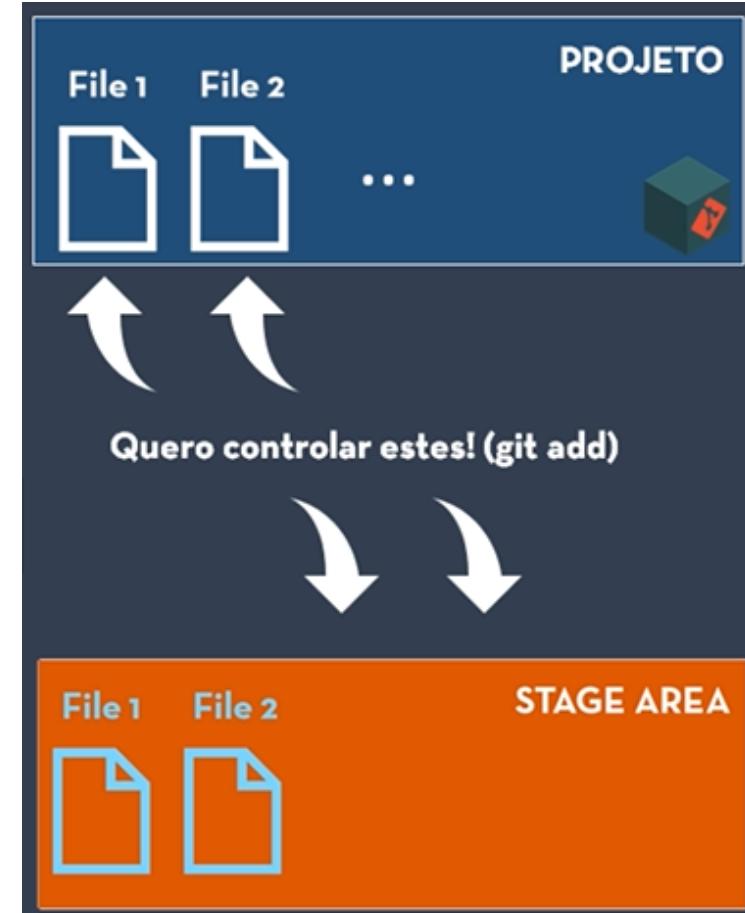
- Ok, mas vamos direcionar nosso foco para os arquivos que queremos rastrear, queremos controlar seu histórico, como dizemos isso para o Git? Como colocamos o Git para trabalhar para nós?
- Para conseguirmos isso, vamos precisar de um comando que colocará nosso arquivo numa área chamada **STAGE**
- O Stage para o Git é como se fosse uma área de transferência, tudo o que temos ali ainda não está sendo controlado, mas poderá ser controlado pelo Git



# Fluxo de Trabalho do Git



- Entendendo o STAGE:
- Esta área existe para que você possa aos poucos definir o que deseja que seja controlado, seu projeto cresce, outras pessoas podem estar contribuindo e você precisa de alguma forma ir controlando isso

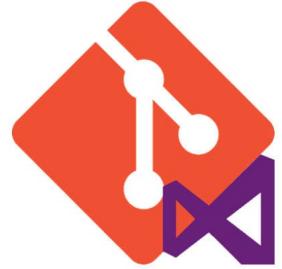




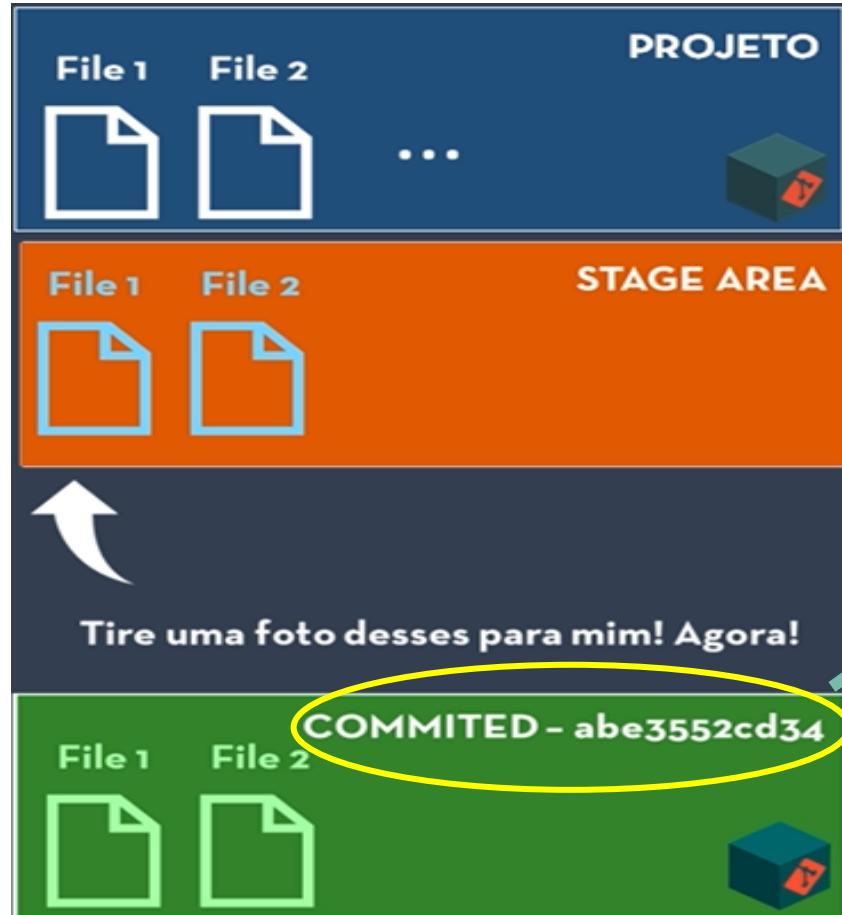
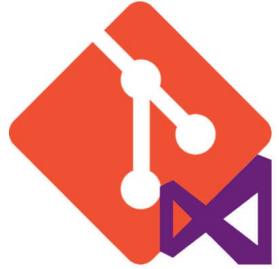
# Fluxo de Trabalho do Git

- O próximo passo é gravar o histórico desses arquivos que estão na Stage
- Pense nisso como o Git tendo a capacidade de tirar uma foto instantânea destes arquivos que estão na Stage, guardando exatamente o estado destes arquivos naquela condição exata de sua incorporação na Stage
- Não é só isso, além de tirar a foto **o Git guarda seu histórico sequencialmente para que você possa retornar a ele sempre que quiser, e ou precisar**

# Fluxo de Trabalho do Git



# Fluxo de Trabalho do Git

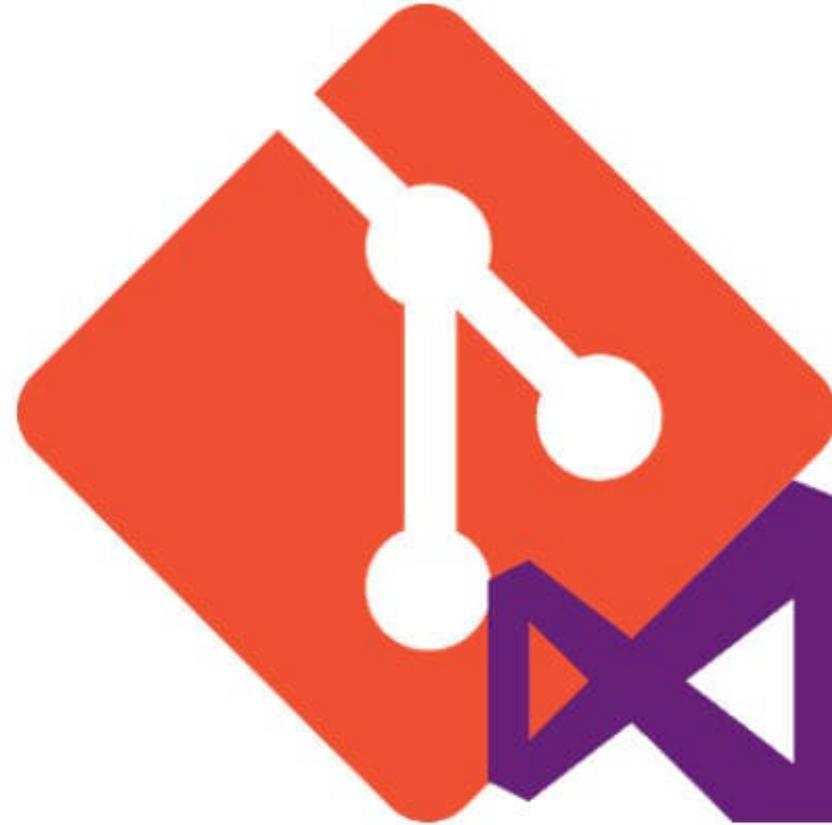


Este histórico é criado através do commit(commited).

Sempre que um commit é realizado é criado um **hash code** do seu snapshot naquele momento, no caso (`abe3552cd34`)

Através dele seu histórico estará assegurado e controlado

Mas o que achou? Pouco amigável não é mesmo?



# Fluxo de Trabalho do Git

- Quando tiramos uma foto estamos fazendo um commit
- **Fique atento, você só faz commits em arquivos que foram adicionados na Stage, não esqueça, este estágio é muito importante**
- Podemos resumir tudo isso como: “snapshot (foto) do estado atual do projeto (aquilo que você adicionou na área de Stage)
- Quando o commit é realizado os arquivos passam para o estado de committed, ou seja, já está no seu histórico de versão, você poderá seguir em frente sem preocupações