

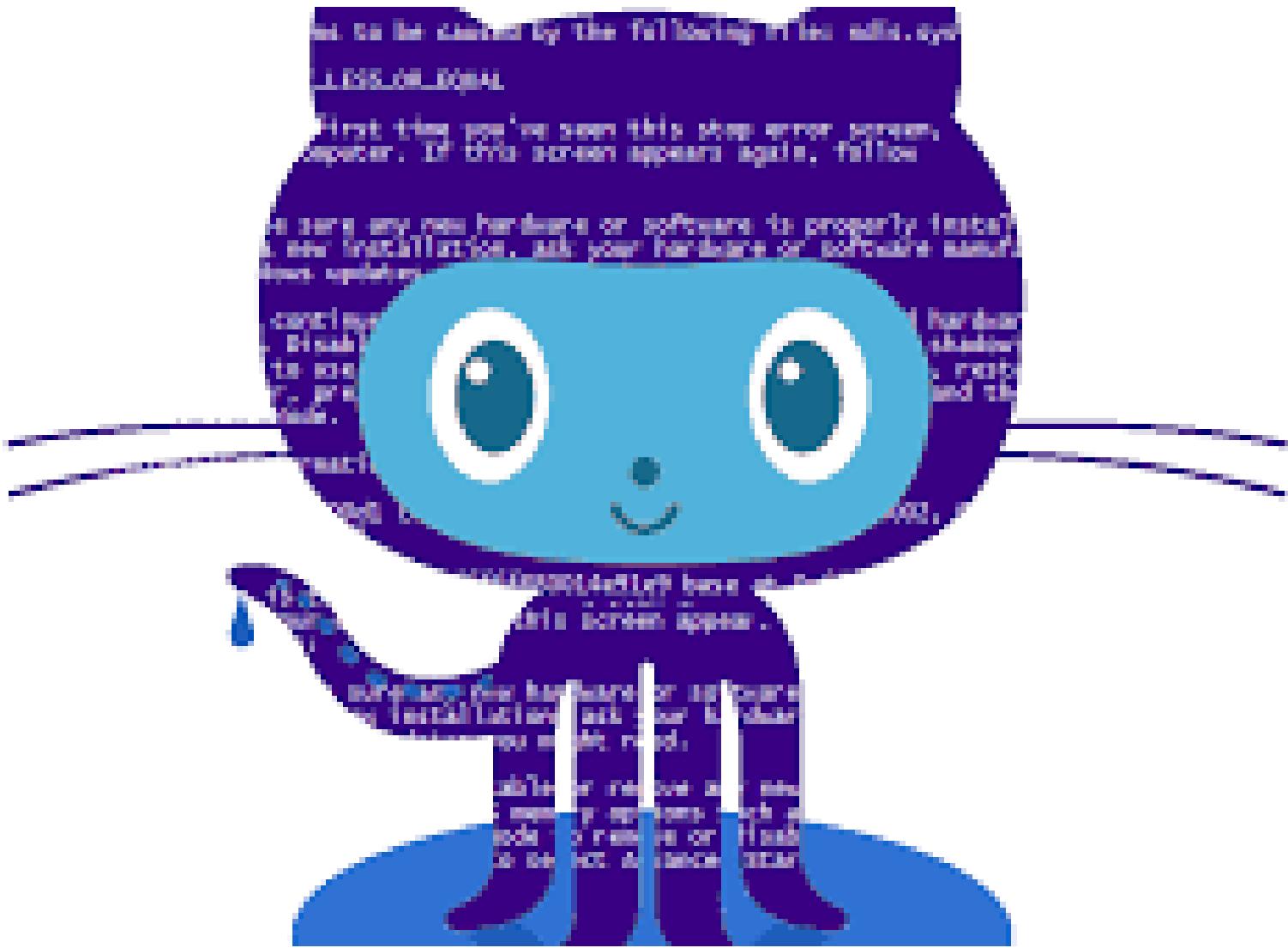
GIT e GitHub

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

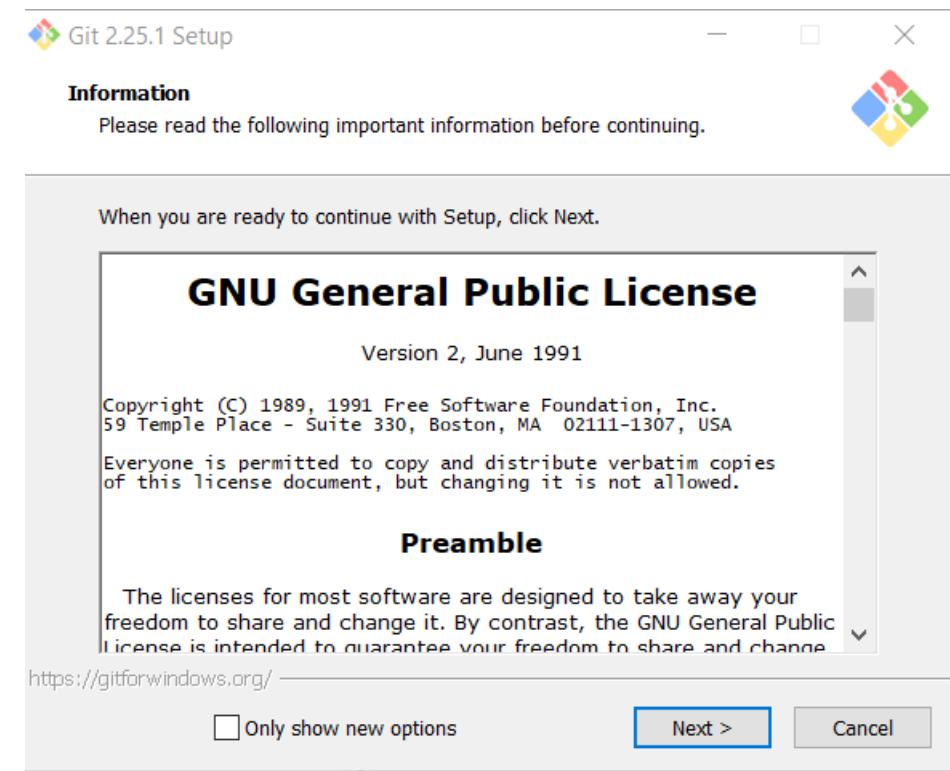
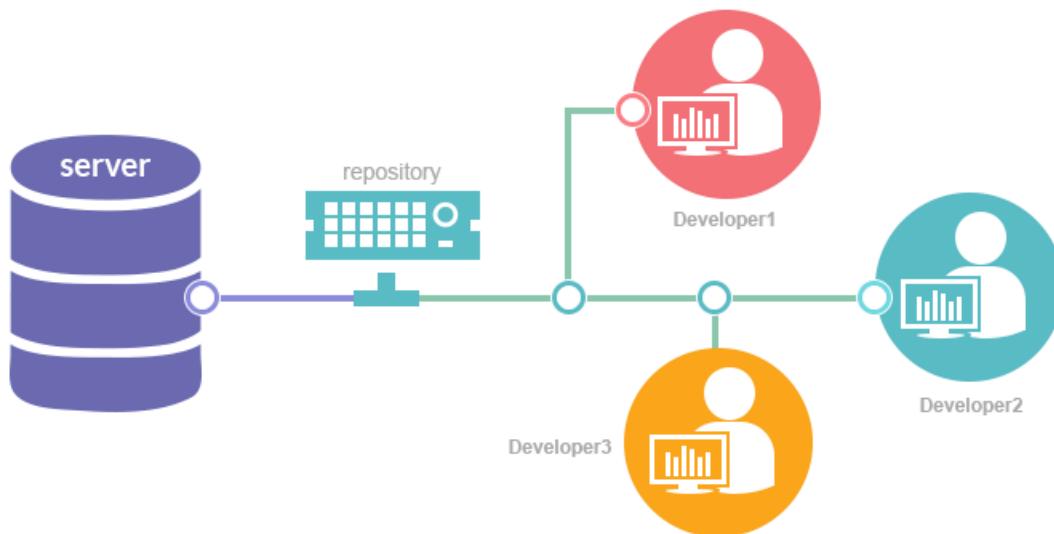
Git & GitHub → Instalando e configurando o GIT

- Passo 1: baixar o Git no site → git-scm.com
- Passo 2: baixar o Visual Studio Code → code.visualstudio.com
- Iniciemos pelo Git



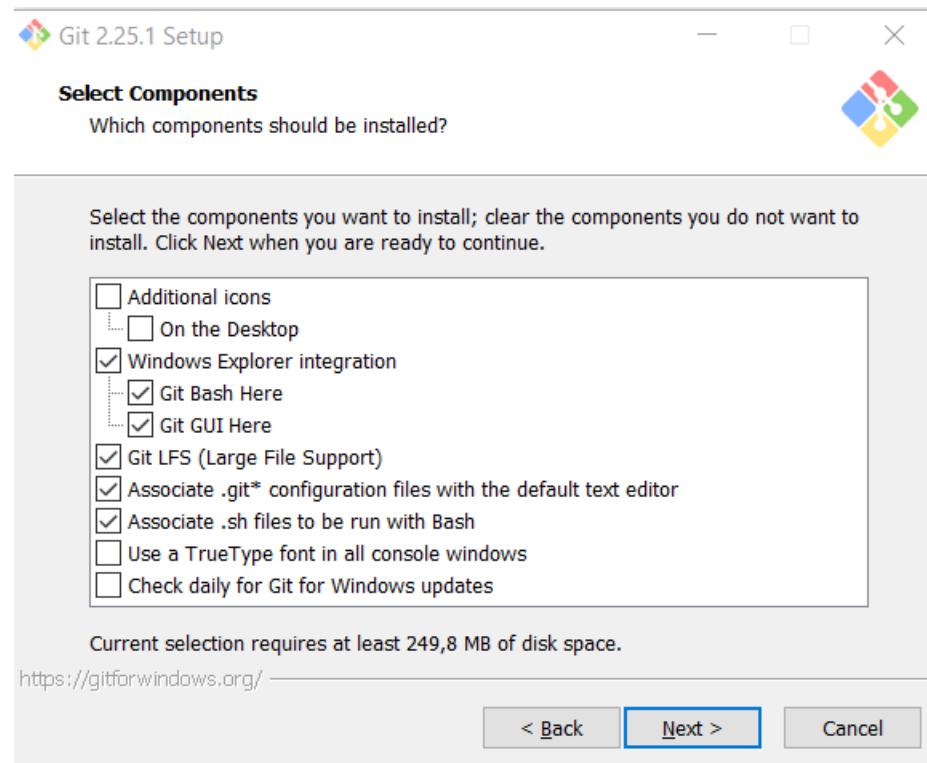
Git & GitHub → Instalando e configurando o GIT

- Passo 3: Iniciando a instalação do Git



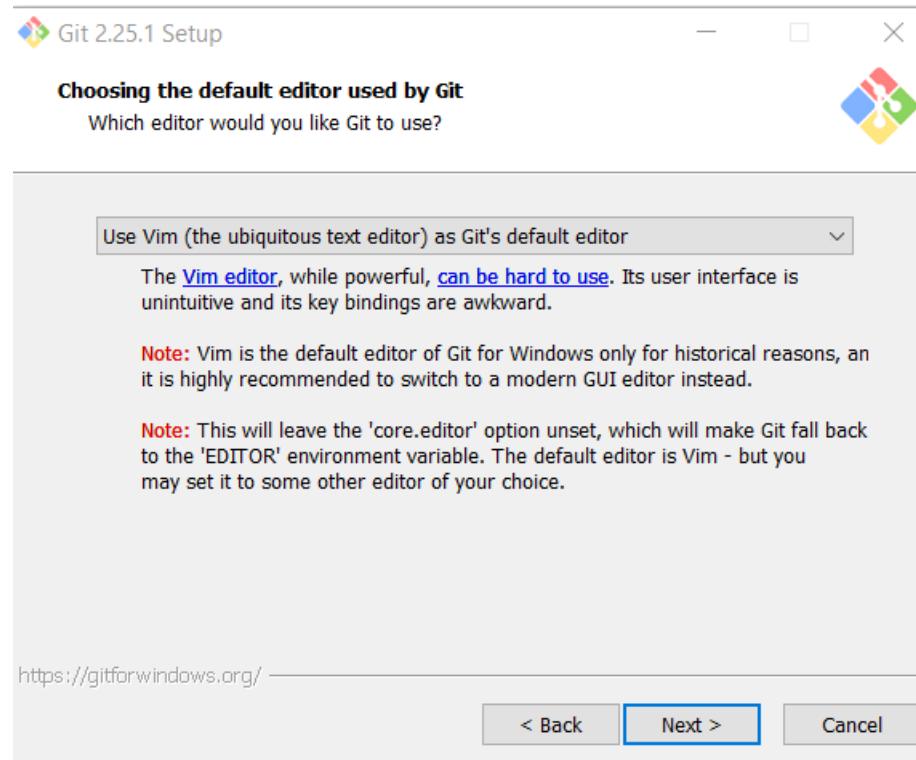
Git & GitHub → Instalando e configurando o GIT

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



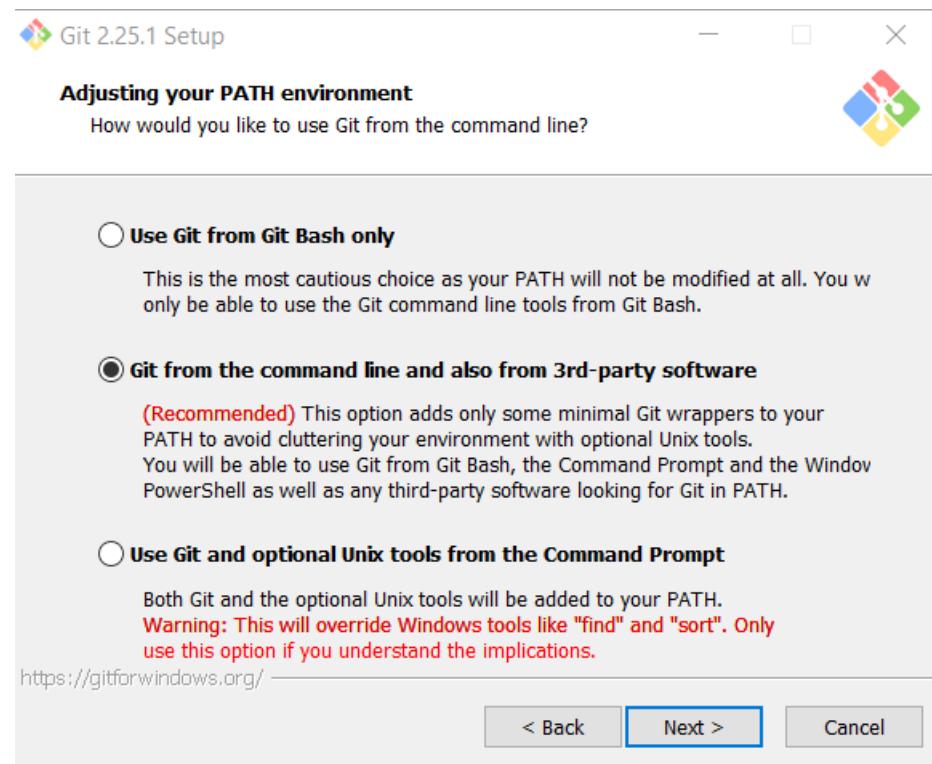
Git & GitHub → Instalando e configurando o GIT

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



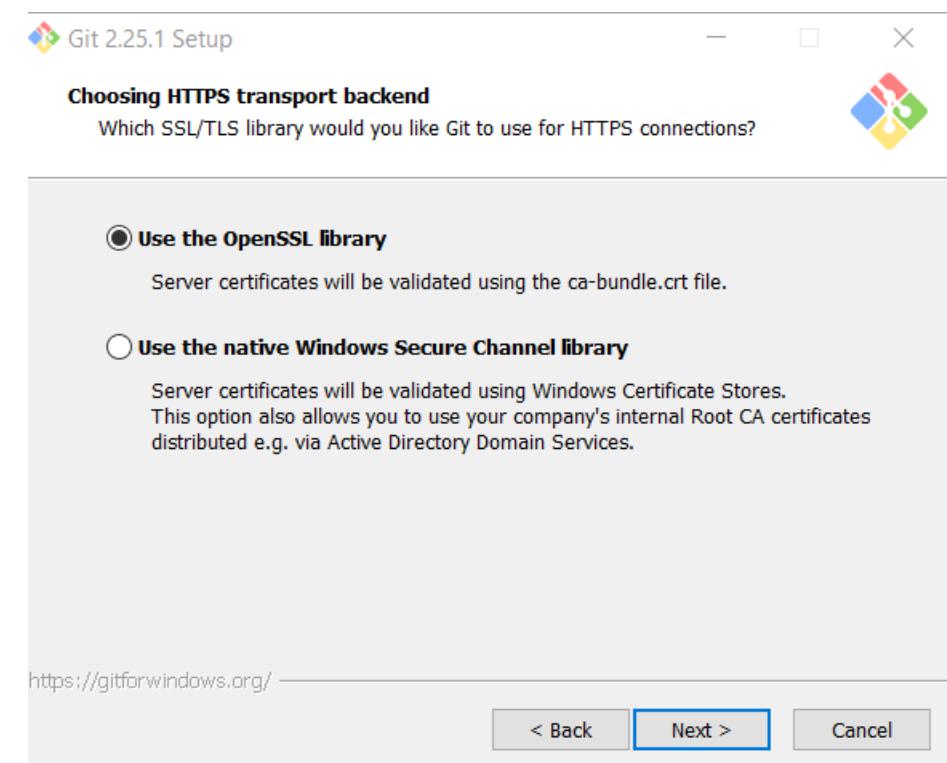
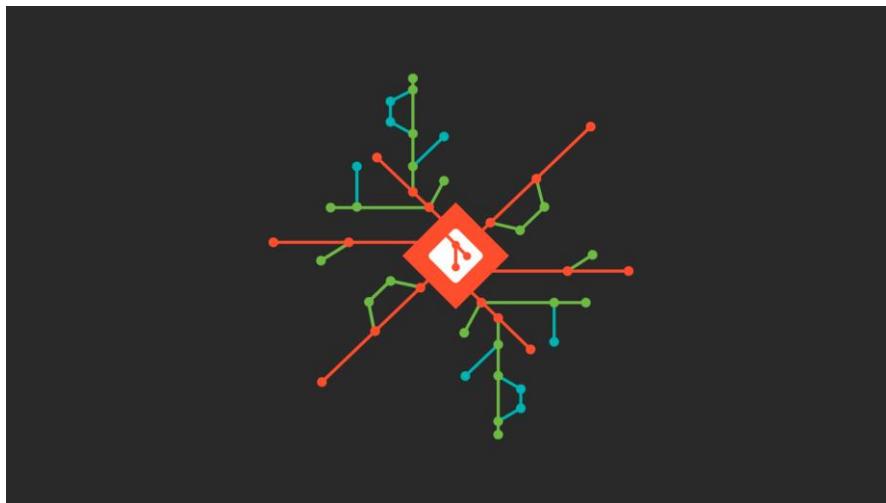
Git & GitHub → Instalando e configurando o GIT

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



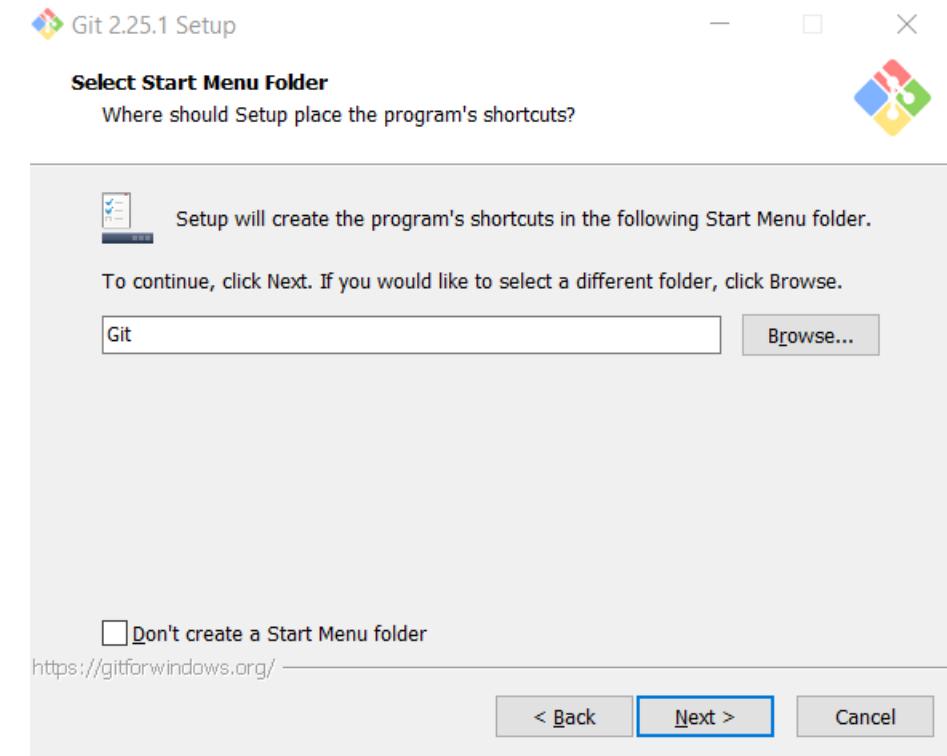
Git & GitHub → Instalando e configurando o GIT

- Passo 7: Precisamos do SSL para fazermos conexões seguras



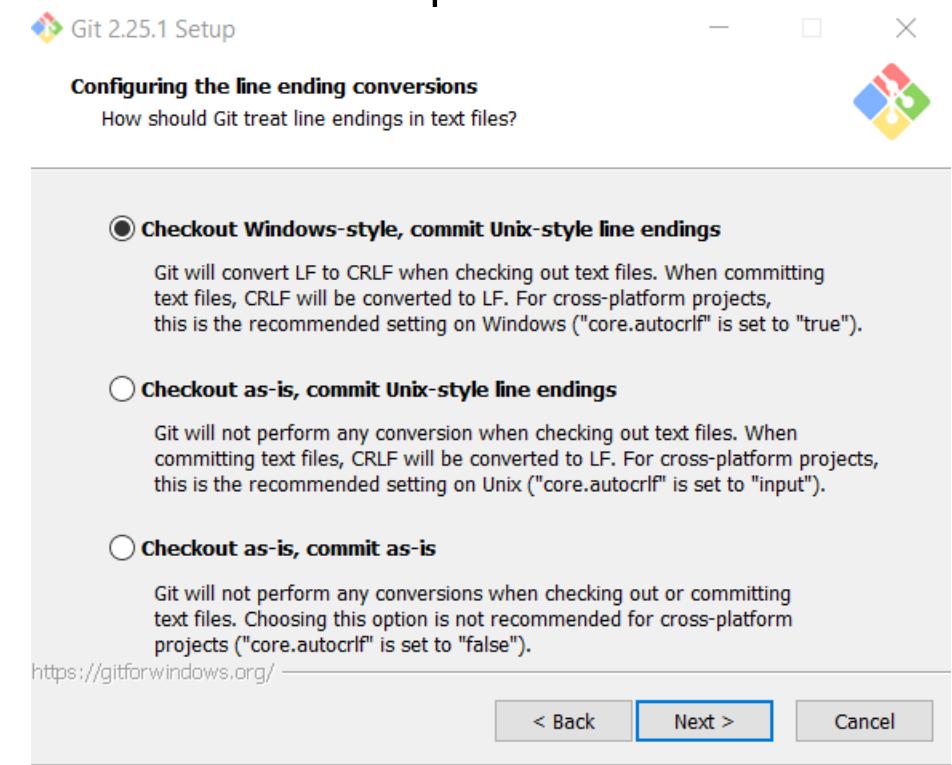
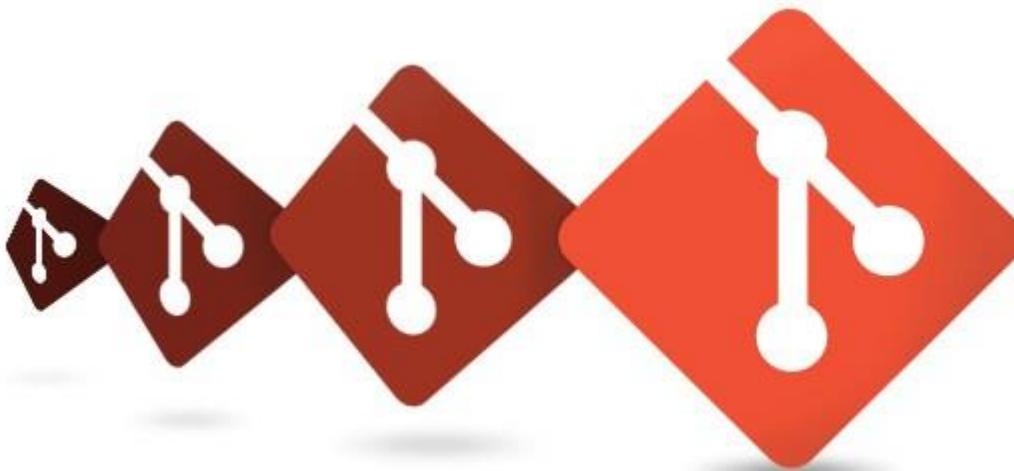
Git & GitHub → Instalando e configurando o GIT

- Passo 8: nada a acrescentar aqui



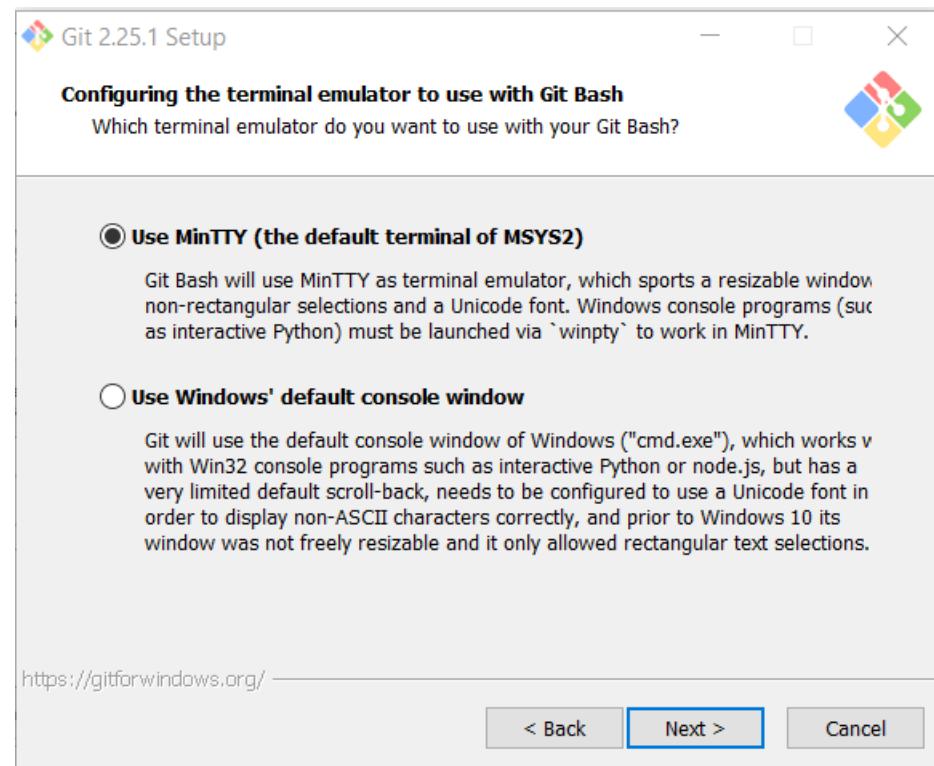
Git & GitHub → Instalando e configurando o GIT

- Passo 9: Aqui é um ponto que o Windows se diferencia do Linux e do MAC → é preciso definir como o Git deverá interpretar final de linha



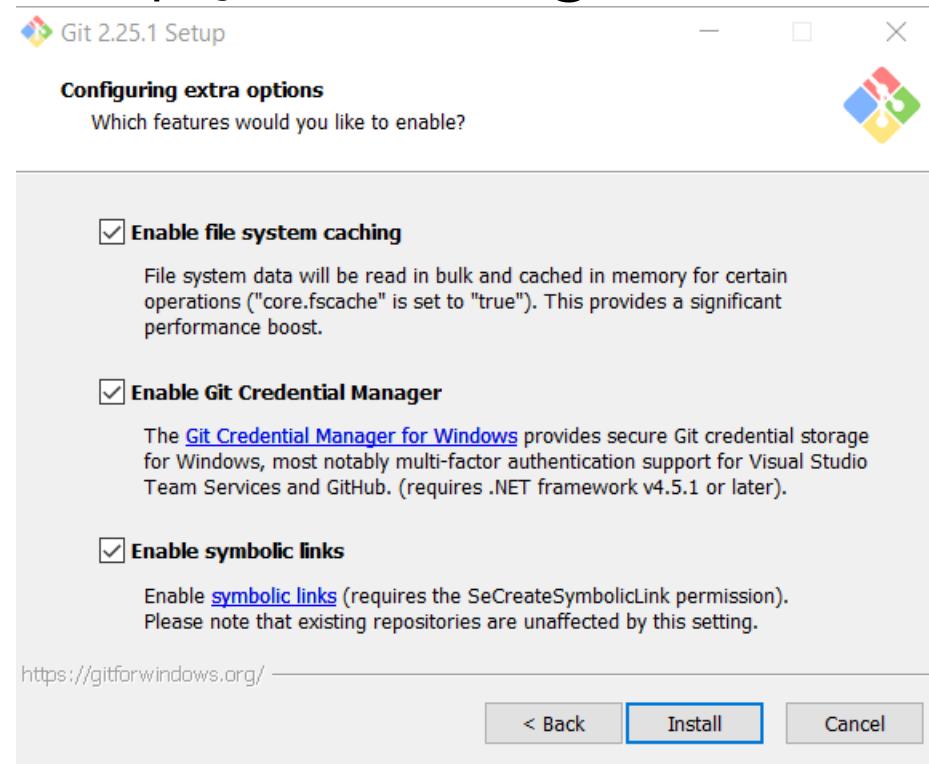
Git & GitHub → Instalando e configurando o GIT

- Passo 10: seguimos no default poderemos usar o git bash como terminal, veremos isso no Visual Studio Code



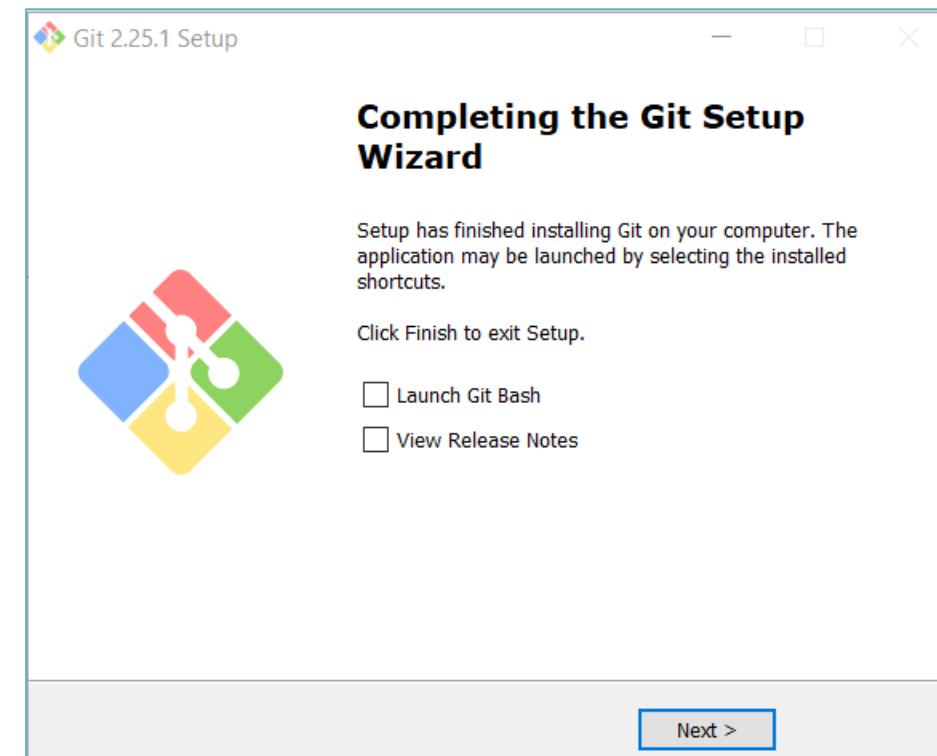
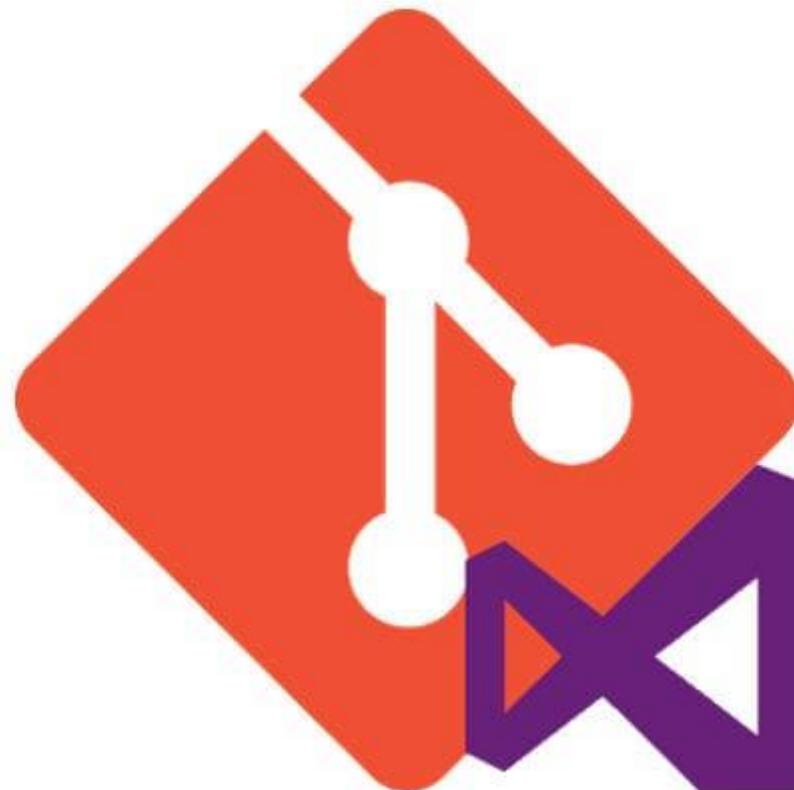
Git & GitHub → Instalando e configurando o GIT

- Passo 11: Selecionamos as três opções a seguir.



Git & GitHub → Instalando e configurando o GIT

- Passo 12: Fim → a seguir configurando o Visual Studio Code



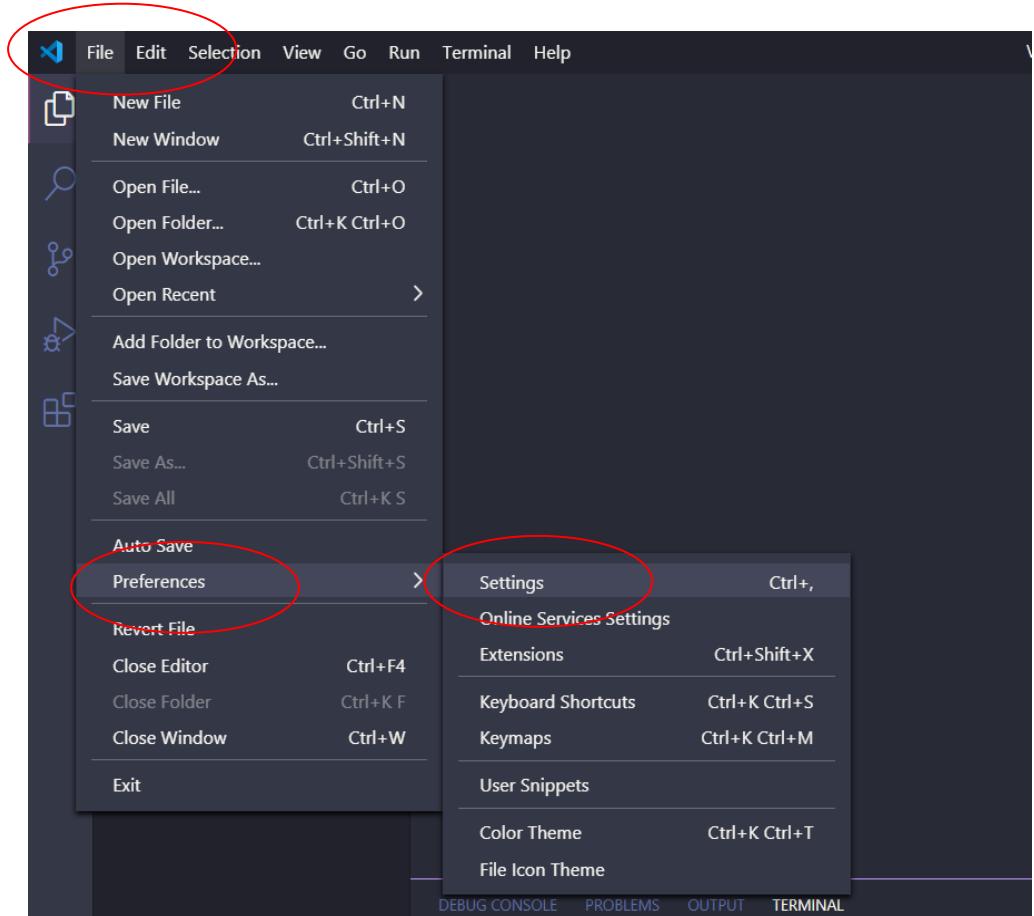
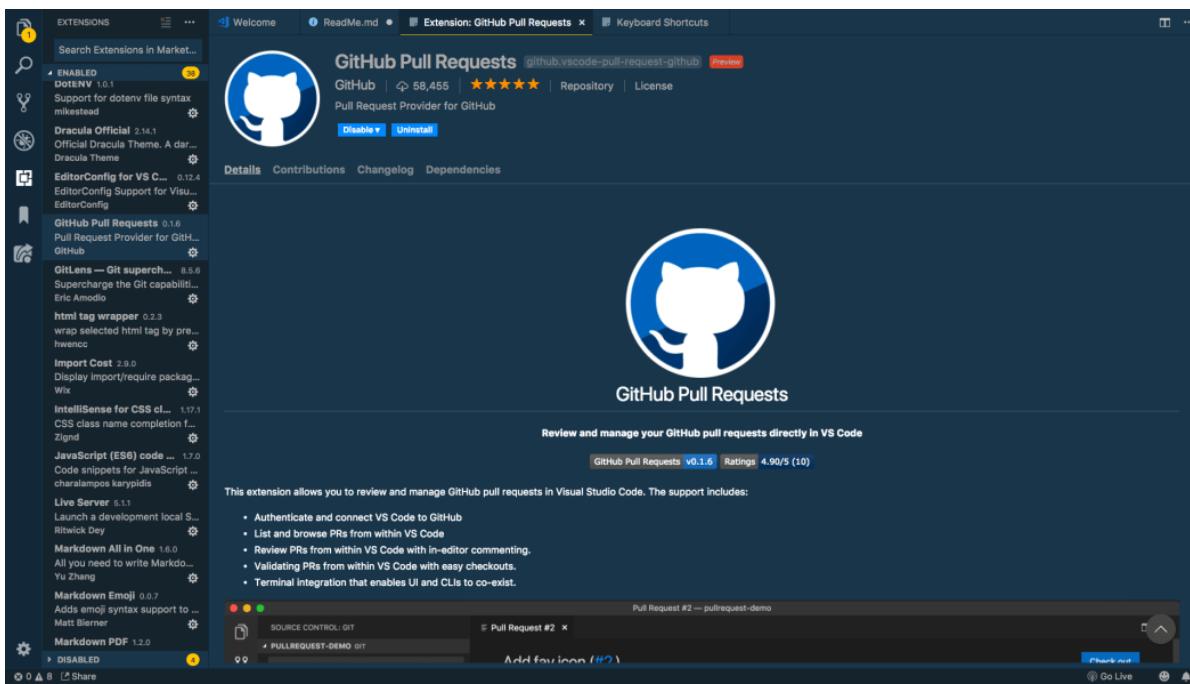
Git & GitHub → Instalando e configurando o GIT

- Baixar e instalar o Visual Studio Code
- Processo de Instalação normal
- 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:



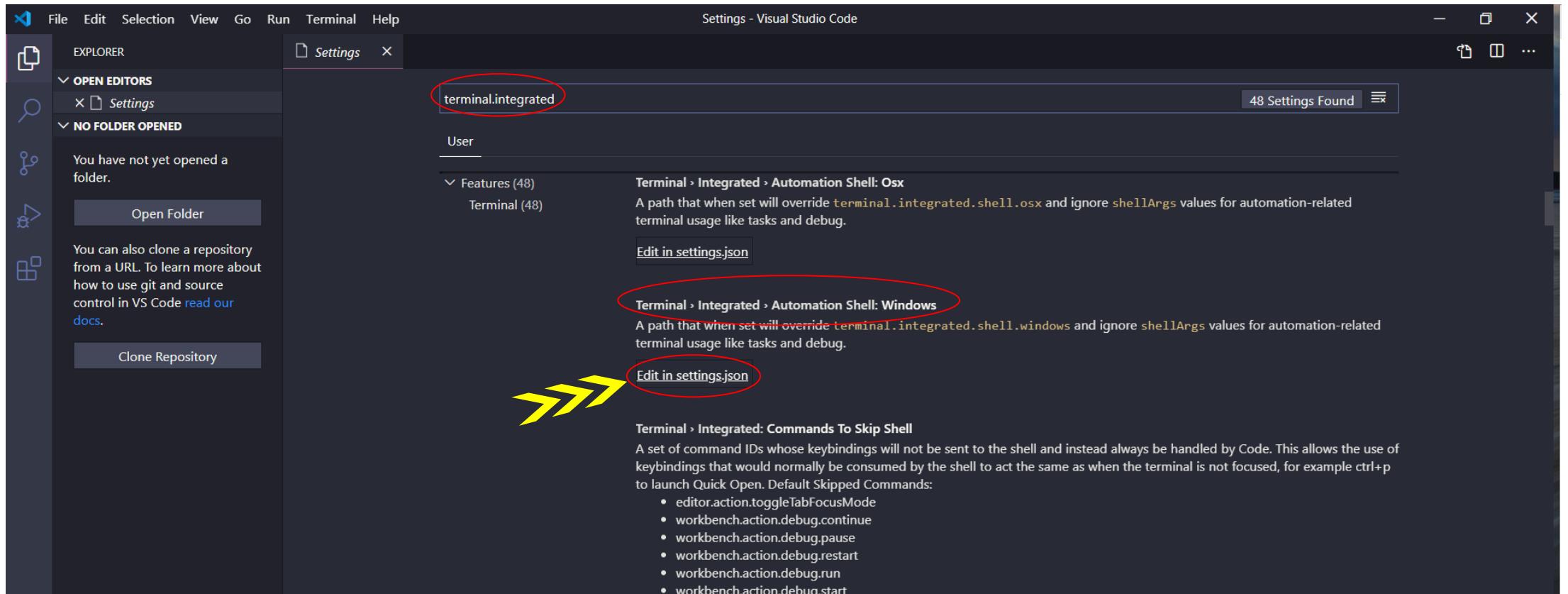
Git & GitHub → Instalando e configurando o GIT

- Passo 13: Primeiro, vá em: File → Preferences → Settings



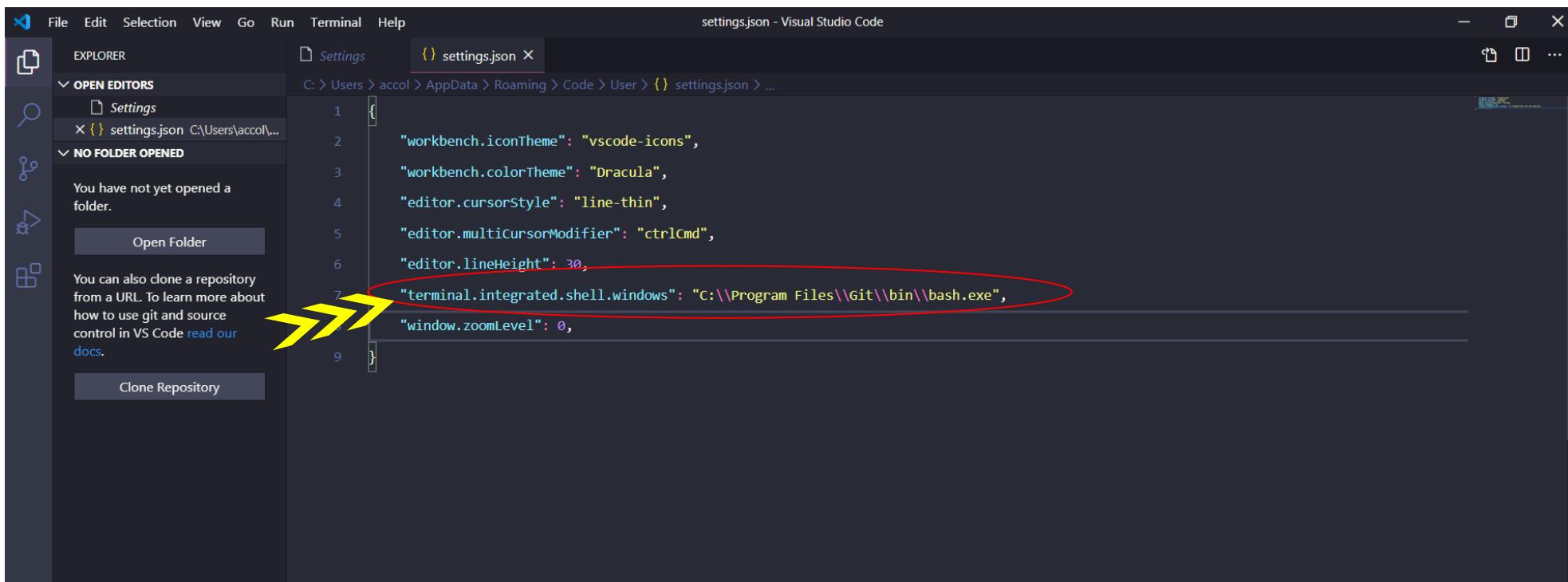
Git & GitHub → Instalando e configurando o GIT

- Passo 14: Digite → terminal.integrated → veja abaixo:
procure por Shell Windows e selecione Edit in settings.json



Git & GitHub → Instalando e configurando o GIT

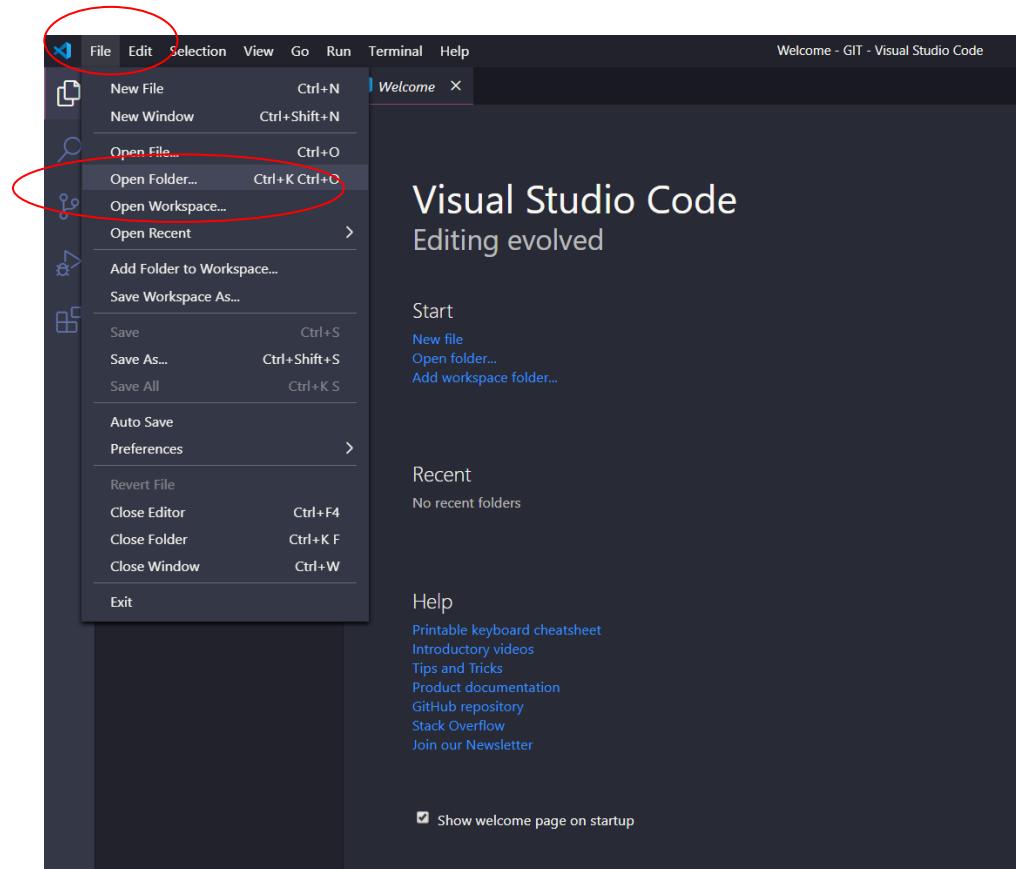
- Passo 15: Na tela de edição acrescente a linha assinalada. Note que, no caso o Git esta instalado nem C:\Program Files\Git → verifique o seu caso. Salve e reinicialize o Visual Studio Code.



```
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": 30,
7   "terminal.integrated.shell.windows": "C:\\Program Files\\Git\\bin\\bash.exe",
8   "window.zoomLevel": 0,
9 }
```

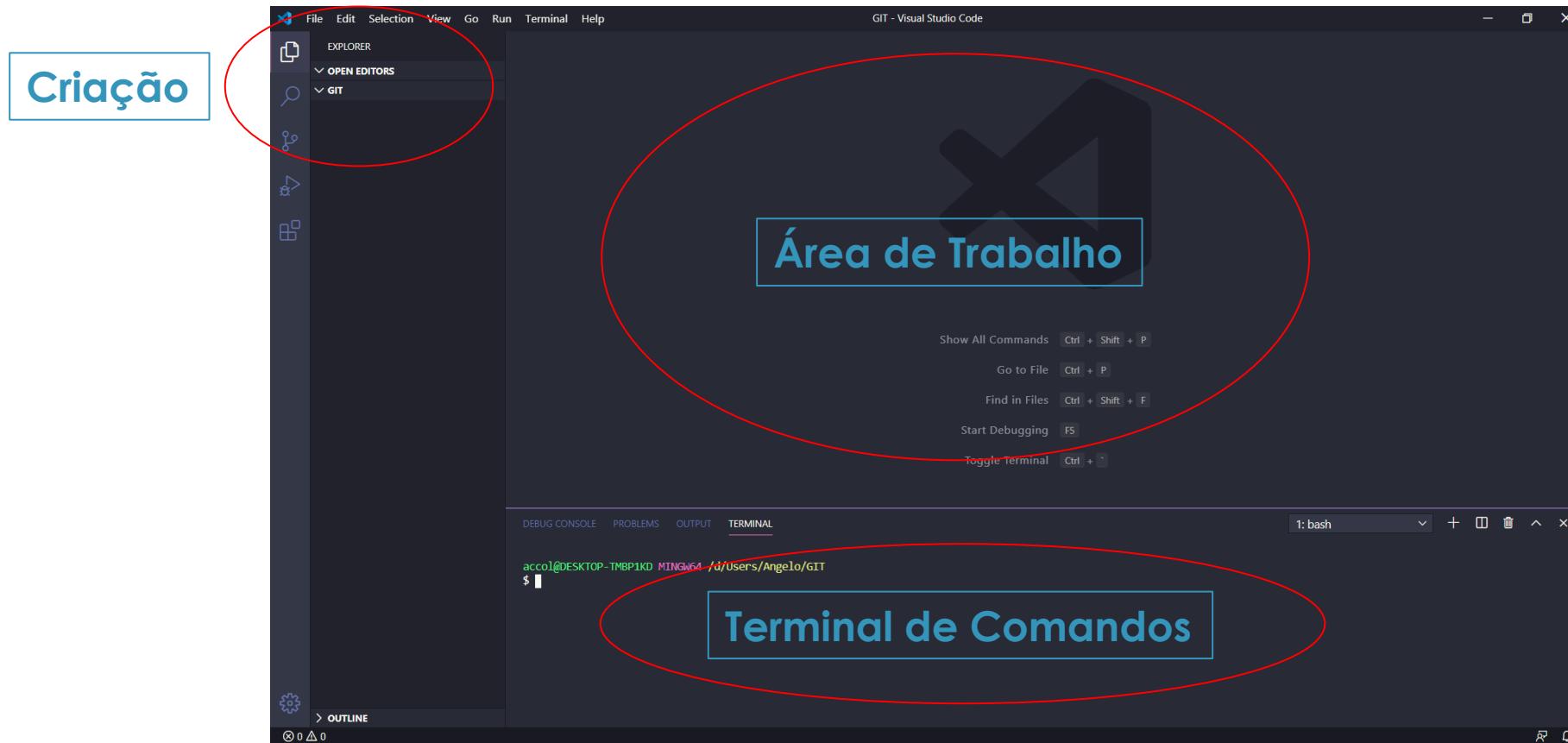
Git & GitHub → Instalando e configurando o GIT

- Passo 16: 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 17: Agora estamos prontos para trabalhar criando novos arquivos e acessando o terminal do Visual Studio Code.

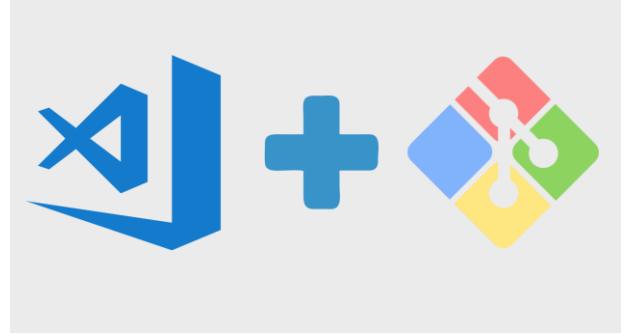


Vamos agora entender como iniciamos o processo de versionamento

Os comandos Git Init e Git Config



GIT INIT & GIT CONFIG



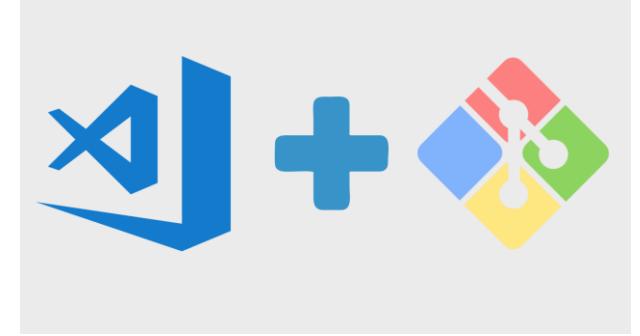
- Para iniciarmos o versionamento de um projeto, tudo o que você precisa é do comando → git init
- Não importa se a pasta está vazia, ou se arquivos de seu projeto já estão salvos, o git irá habilitar seu repositório para o versionamento



```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

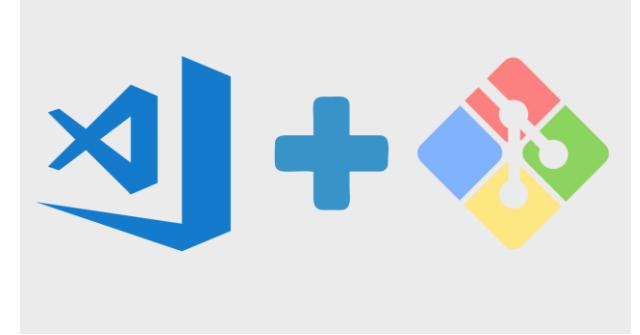
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT
$ git init
Initialized empty Git repository in D:/Users/Angelo/GIT/.git/
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ []
```

GIT INIT & GIT CONFIG



- O Git irá criar uma pasta .git no diretório principal de seu projeto
- 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

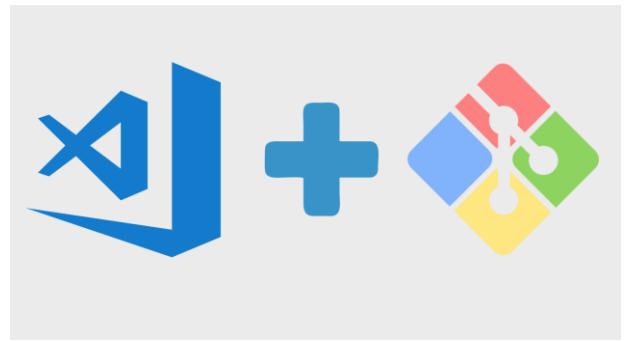
GIT INIT & GIT CONFIG



- Através do git config poderemos controlar quem estará trabalhando no projeto
- Precisamos dizer se esta configuração irá afetar todos os usuários deste computador, neste caso, usamos a **flag --system**. Se é para nós em todos os projetos, usamos a **flag --global**. Agora, se for apenas para esse projeto específico, usamos a **flag --local**

Usaremos nos próximos exemplos a flag -- global

GIT INIT & GIT CONFIG



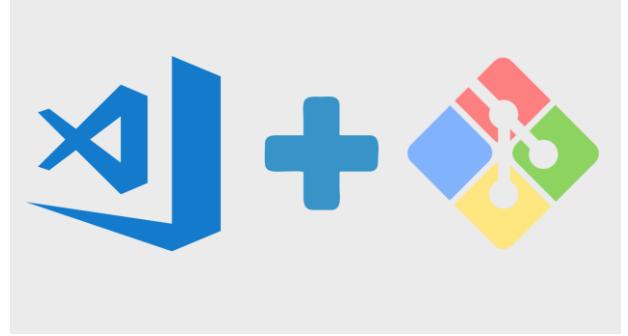
- O interessante da flag `--global` é que sempre que iniciarmos um novo projeto, o Git automaticamente atribuirá essa configuração ao projeto
- Será preciso definir o **git config -- global user.name** e o **git config -- global user.email**

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ git config --global user.name accolombinifake

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ git config --global user.email accolombinifake@hotmail.com

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ []
```

GIT INIT & GIT CONFIG



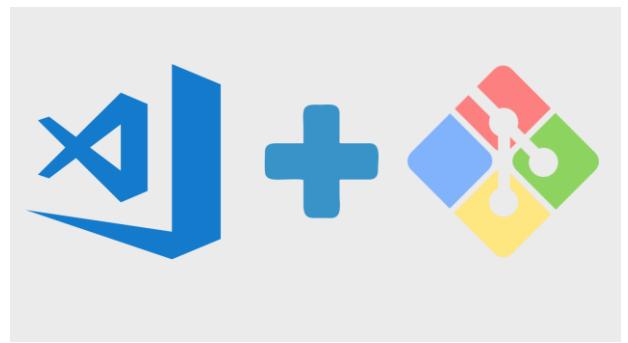
- Caso tenha alguma dúvida e queira conferir as configurações que já estão por padrão definidas e as que você acabou de definir, poderá usar o comando →
git config --global -l (l → list)

```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ git config --global -l
user.name=accolombinifake
user.email=accolombinifake@hotmail.com

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ []
```

GIT INIT & GIT CONFIG



- Também vamos precisar definir um editor para o Git, na instalação, lembra, deixamos no default, agora é a hora de alterarmos para algo mais interessante
- Você poderá definir o editor de sua preferência, no meu caso, vou definir o notepad++ → **git config --global core.editor <seu editor>**



```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ git config --global -l
user.name=accolombinifake
user.email=accolombinifake@hotmail.com

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ git config --global core.editor notepad++

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)
$ git config --global -l
user.name=accolombinifake
user.email=accolombinifake@hotmail.com
core.editor=notepad++
```

**Agora é hora de entendermos
como o Git trabalha**



git

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 que o Git irá 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

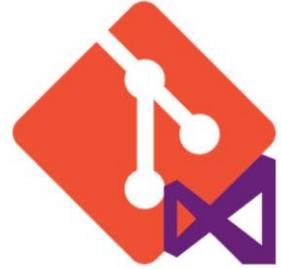


- Mas atenção!!!



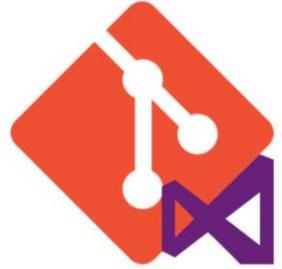
- Sempre que você cria um arquivo ou uma pasta no seu diretório do projeto, o Git irá considera-lo **untracked**, isto é, ele está em um estado não rastreável pelo Git
- É comum você ter arquivos que não deseja rastrear, por exemplo, caso tenha baixado algum texto para completar seu projeto e usado apenas 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

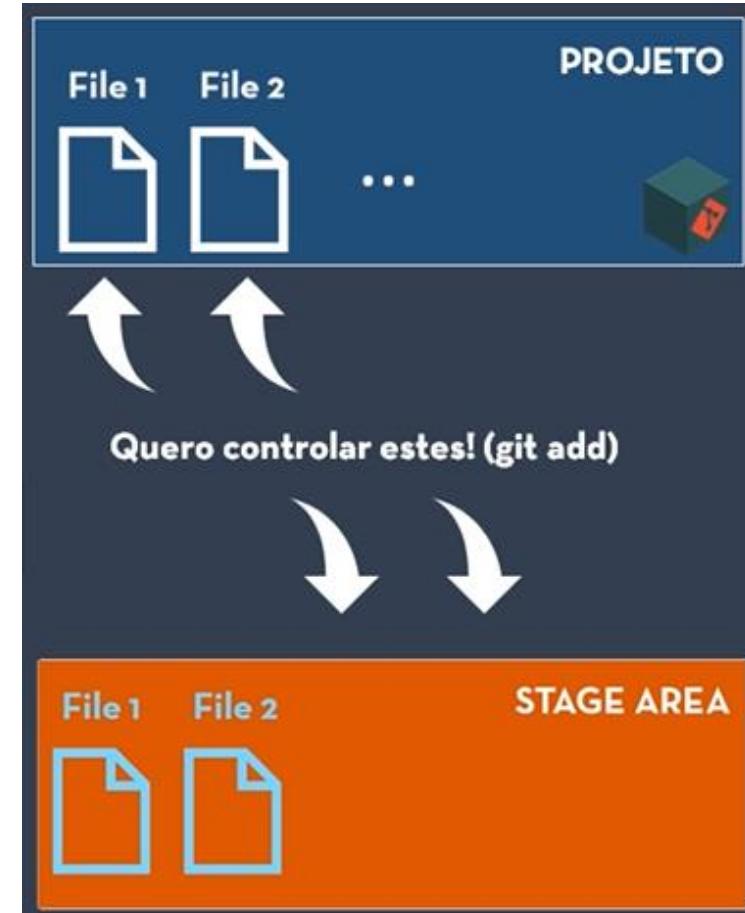


- Ok, mas vamos direcionar nosso foco para os arquivos que queremos rastrear, queremos controlar seu histórico, como dizemos isso para o Git? Como colocamos o Git para trabalhar para nós?
- Para conseguirmos isso, vamos precisar de um comando que colocará nosso arquivo numa área chamada **STAGE**
- O 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

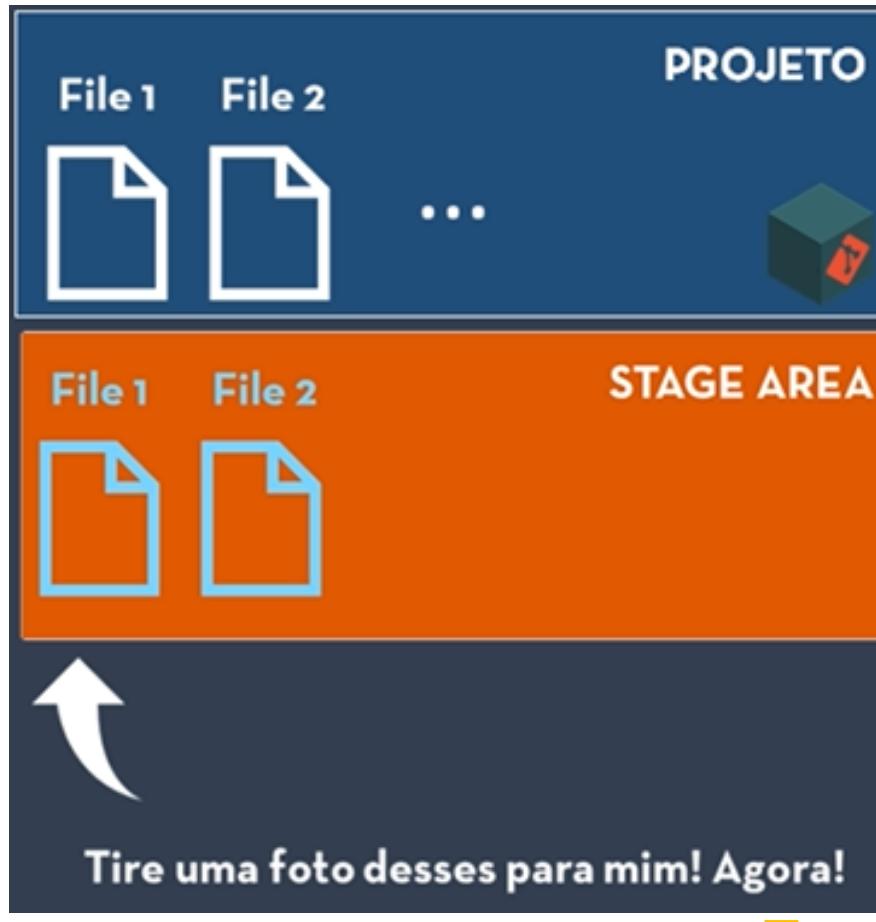


Fluxo de Trabalho do Git



- 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

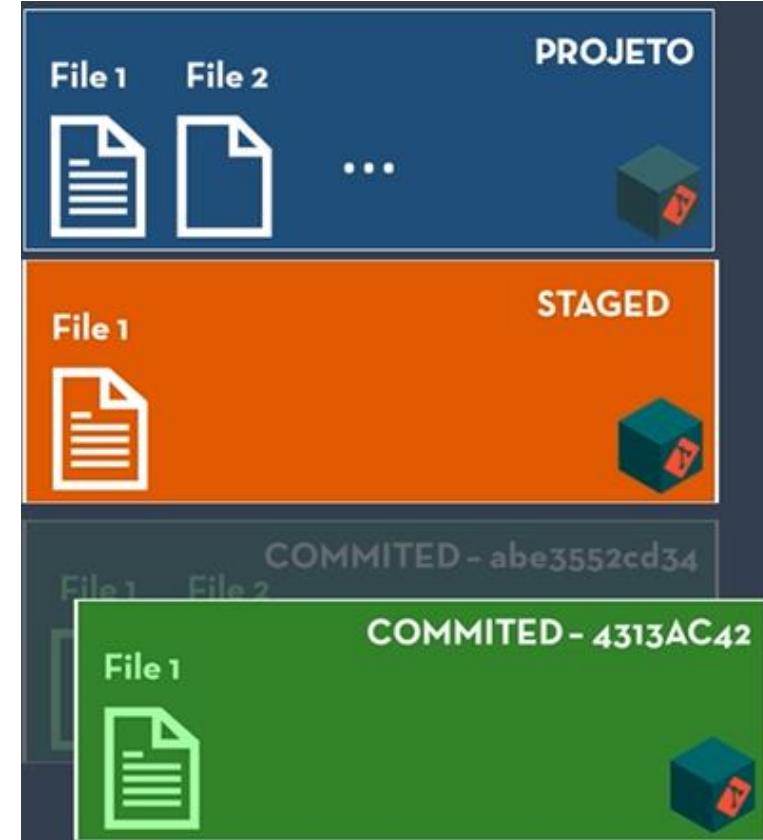


- Este histórico é criado através de commit(commited).
- Sempre que um comitê é realizado é criado um hash code, no caso (abe3552cd34)
- Através dele seu histórico estará assegurado e controlado
- Mas o que achou? Pouco amigável não é mesmo?

Fluxo de Trabalho do Git



- Na medida em você e sua equipe vão trabalhando nos arquivos, novos commits são realizados
- Sempre que um novo commit acontece é como se um novo snapshot do arquivo fosse registrado e gravado

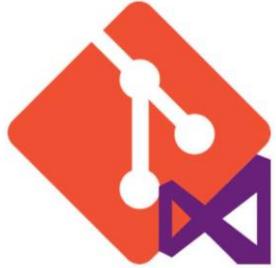


Fluxo de Trabalho do Git



- Quando você precisar ver alguma versão anterior de seus arquivos, basta retornar aos seus snapshots
- Com um simples comando para o Git você fornece o hash code e seu arquivo é trazido no instante exato em que a foto foi criada.
- É como lhe conceder acesso instantâneo ao contexto em que a foto foi criada.
- É algo como “me mostre a foto abe355cd34”

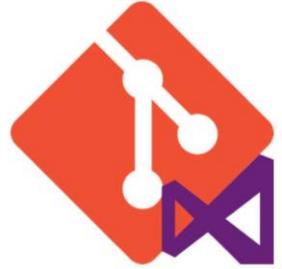
Fluxo de Trabalho do Git



- Como resposta ao seu pedido o Git lhe devolve algo como:

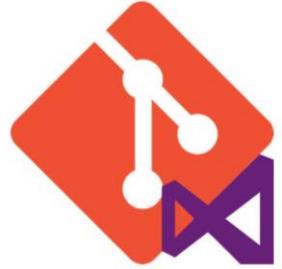


Fluxo de Trabalho do Git



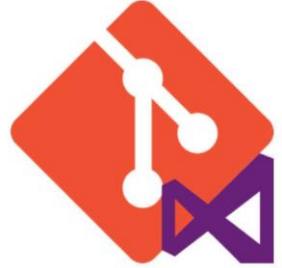
- Quando tiramos uma foto estamos fazendo um commit
- Fique atento, você só faz commits em arquivos que foram adicionados na Stage, não esqueça, este estágio é muito importante
- Podemos resumir tudo isso como: “snapshot (foto) do estado atual do projeto (aquilo que você adicionou na área de Stage)
- 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

Fluxo de Trabalho do Git



- Mas atenção, se novas alterações acontecerem no seu ambiente de projeto, ou seja, nos casos em que algum arquivo rastreado (**TRACKED**) tenha sido alterado e que não passaram por um novo commit, ele passa para um estado chamado de **MODIFIED**
- O Git está te alertando que seu arquivo rastreado foi alterado desde seu último commit, você deverá ter uma estratégia para a realização de novos commits, assegurando um controle eficaz de seu projeto e não tirando Snapshots sem o menor critério

Fluxo de Trabalho do Git

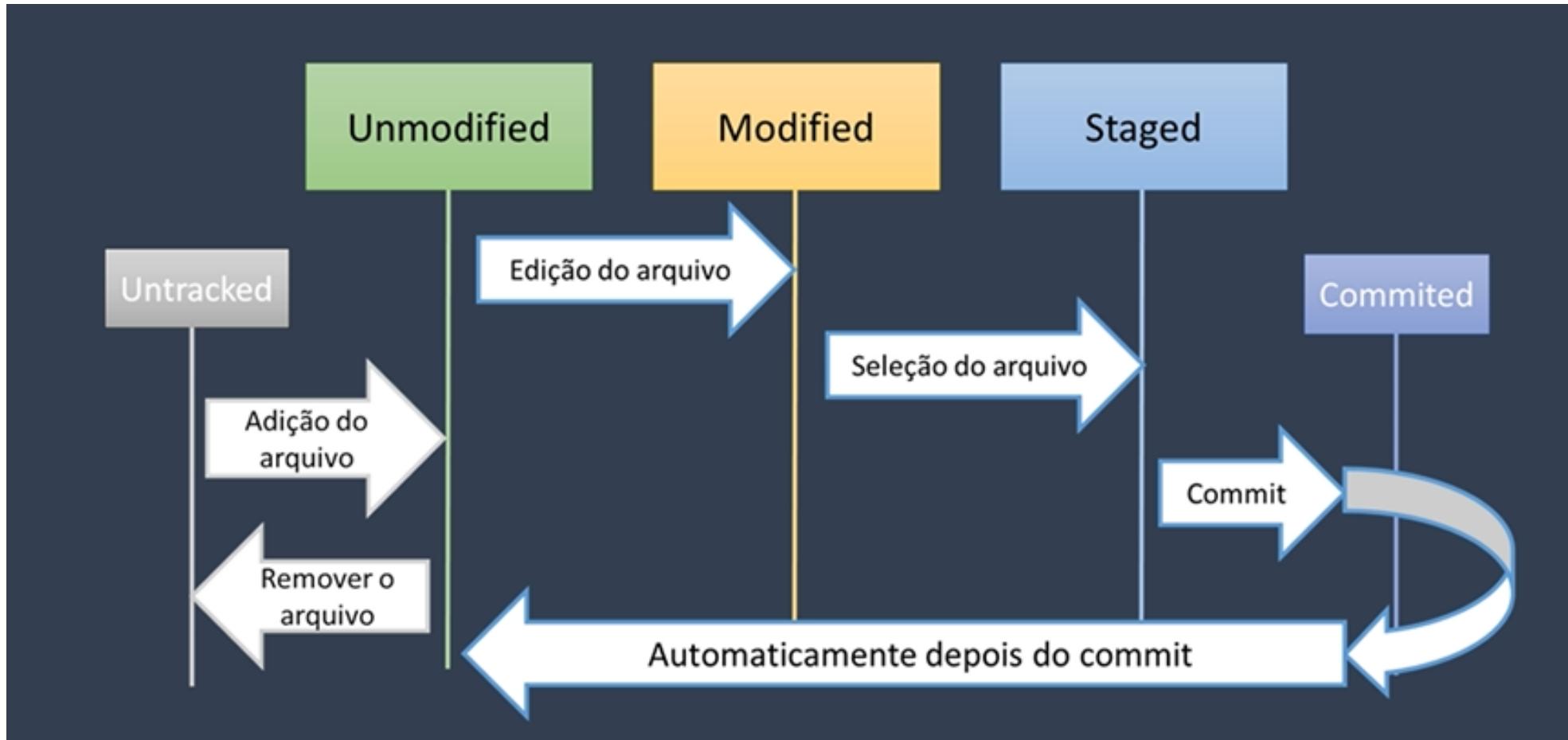


- Quando nosso projeto controlado (Tracked) sofreu alterações (Modified), precisamos repetir o processo, isto é, adicionar à Stage e depois realizar o Commit
- Fique atento, todos os passos são necessários, veja a seguir o resumo de tudo o que foi discutido até agora





Fluxo de Trabalho do Git



Fluxo de Trabalho do Git

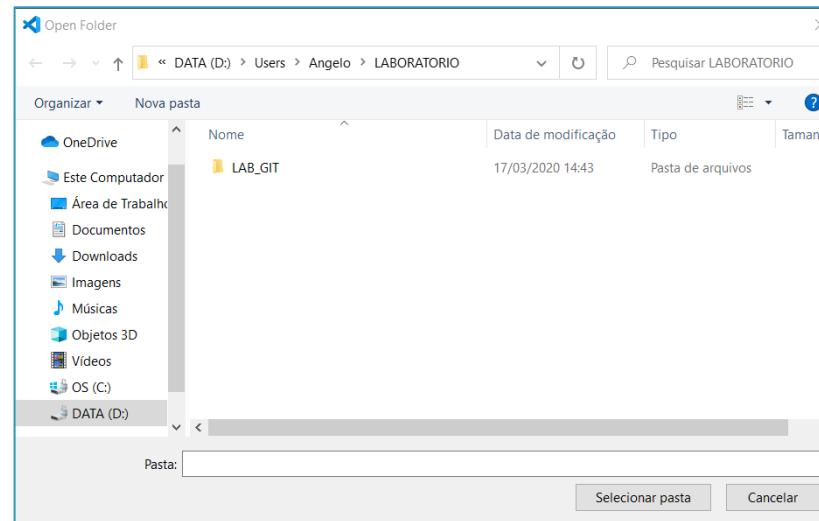
Parte II

Os comandos GIT ADD, GIT COMMIT & GIT STATUS

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Para testar este aprendizado, vamos agora fazer um pequeno laboratório
- Abra o Visual Studio Code
- Abra uma nova pasta, no meu caso, a pasta tem esse caminho



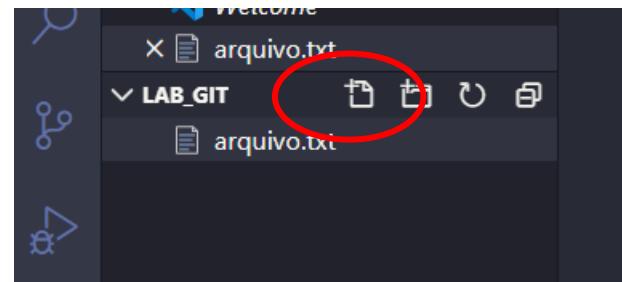
```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT
$ 
```

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Vamos criar um arquivo, editá-lo e salvá-lo, veja os passos:



- Vamos iniciar o Git com o comando git init

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT
$ git init
Initialized empty Git repository in D:/Users/Angelo/LABORATORIO/LAB_GIT/.git/
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ 
```

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Para saber o status do seu arquivo/projeto você deve usar o comando **git status**
- Lembre-se o Git se recorda das configurações feitas anteriormente, logo, seu projeto possui configurações que foram herdadas, e isto, permanecerá até que resolva alterá-las com um novo **git config --global ...**

Observe que seu novo arquivo, no caso "arquivo.txt" esta na modalidade untracked

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git status
On branch master

No commits yet

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

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git config --global -l
user.name=accolombinifake
user.email=accolombinifake@hotmail.com
core.editor=notepad++
```



Os comandos GIT ADD, GIT COMMIT & GIT STATUS

- O próximo passo é colocar este arquivo na área de **Stage**, para que possamos a partir daí, *versioná-lo efetivamente*.
- Para isso vamos precisar do comando: **git add *<nome_do_arquivo>***

Observe agora que
novo arquivo
passou para o
status de: No
commits yet

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git add arquivo.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git status
On branch master
No commits yet

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

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Feito isso, podemos continuar criando novos arquivos para depois adicioná-los na Stage e realizar um única commit mais tarde, ou podemos fazer um commit no nosso arquivo e dar início ao processo de rastreamento
- Para realizarmos o commit vamos precisar do comando **git commit -m** (a opção –m indica que podemos inserir uma mensagem em nosso commit para relatarmos o que foi feito até o momento). Esta mensagem deverá estar entre aspas simples 'mensagem aqui'
- Essa mensagem é importante para tornar o histórico dos arquivos mais amigável. Procure ser claro sucinto e objetivo nesta mensagem



Os comandos GIT ADD, GIT COMMIT & GIT STATUS

- Observe nosso exemplo, veja que após o commit o **git status** nos devolve uma mensagem alegando que não nenhum documento rastreável sem commit até o momento, isso significa que seu projeto está com o controle de versão assegurado

Após o commit o git status nos indica que na nossa branch master não há documentos que precisem de commit

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git commit -m 'adição do arquivo.txt'
[master (root-commit) ca9e81e] adição do arquivo.txt
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Vamos observar o que acontece quando fazemos uma alteração em nosso arquivo
- Observe que nosso arquivo agora estará no **status Modified**, isso é, ainda não foi para a **Stage**

Observe que o Git reconheceu que houve alteração no documento e que o documento ainda não foi para a área de Stage

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_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:   arquivo.txt

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

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Neste momento, a dica é, avalie se vale a pena fazer um commit de seu arquivo, não é recomendado que se façam commits a cada alteração
- A melhor dica aqui é avaliar o contexto, se o contexto demanda uma foto, demandar que registre esse fato, aí então faça um commit, caso não siga em frente atento ao controle do seu projeto, saiba que o Git não poderá recuperar nenhum trecho de seu projeto que não esteja no status de committed
- Uma boa referência é deixar para realizar seus commits quando as alterações realizadas fizerem algum sentido em seu projeto



Os comandos GIT ADD, GIT COMMIT & GIT STATUS

- Como estamos num laboratório vamos realizar mais um commit
- Desta vez vamos usar o comando `(.)` que dirá ao Git o seguinte, adicione à *Stage* todos os arquivos/documentos que sofreram alteração. Veja a seguir todos os passos:

Observe que o Git reconheceu todas as alterações que ocorreram no arquivo, inclusive uma delação de uma palavra

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git commit -m 'Alteração na segunda linha do arquivo.txt'
[master f9e346e] Alteração na segunda linha do arquivo.txt
 1 file changed, 2 insertions(+), 1 deletion(-)

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Os comandos GIT ADD, GIT COMMIT & GIT STATUS



- Continuando nossas simulações, farei mais algumas alterações no arquivo teste
- Como esse processo de adicionar na Stage e depois realizar o commit é meio redundante, vamos fazer agora de uma outra forma, vamos reduzir um pouco
- Atenção → isso só vale para arquivos que já estão sendo rastreados, para os novos arquivos deverá trabalhar com o processo convencional
- O comando para isso é: **git commit -am** 'mm', onde o 'a' é para git add e o 'm' para a mensagem do commit



Os comandos GIT ADD, GIT COMMIT & GIT STATUS

- Não se esqueça só vale para arquivos rastreados!!!

Observe que o Git reconhece que o arquivo já está sendo rastreado e realiza as duas ações com um único comando

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git commit -am 'Últimas alterações'
[master 2AAF4A0] Últimas alterações
 1 file changed, 4 insertions(+)

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git status
On branch master
nothing to commit, working tree clean
```



Tudo bem, mas como vejo as versões anteriores?

Git checkout parte III



Git Checkout

- Vamos entender como podemos visualizar o histórico dos arquivos criados, quem os criou e em que momento foram criados
- O comando responsável por isso é o **git log**, veja a seguir:

Observe o snapshot gerado para o commit. O Git exibe todos os commits que foram realizados, do mais recente para o mais antigo

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git log
commit 2aaaf4a0bf3f19d622a076c96e2f130c65609c974 (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 17 16:02:05 2020 -0300
        Últimas alterações

commit f9e346ef5b1b834163daefb6a0ed8570e81b1c14
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 17 15:52:37 2020 -0300
        Alteração na segunda linha do arquivo.txt

commit ca9e81e1d217ac09c70525f90aa12a07ce6d85a7
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 17 15:25:00 2020 -0300
        adição do arquivo.txt
```

Git Checkout



- O comando `git log` possui uma série de variações que serão apresentadas com o tempo
- Caso seu log seja muito longo seu terminal ficará paralisado. Para sair você deve digitar a letra **q** (quit)
- O nome que se dá ao histórico é **Project History**
- Para visualizar qualquer arquivo de seu Project History, basta copiar seu snapshot e usar o comando **git checkout <snapshot>**, que o Git irá exibir seu documento no instante em que a foto foi tirada. Veja a seguir:

Git Checkout



- Texto atual:

```
arquivo.txt
1 Texto
2 Mais uma linha
3 Texto
4 Mais uma linha
5 Texto
6 Mais uma linha
```

A screenshot of a terminal window titled "arquivo.txt". It contains six lines of text, each starting with a number from 1 to 6 followed by either "Texto" or "Mais uma linha". The lines are color-coded: "1 Texto" is blue, "2 Mais uma linha" is green, "3 Texto" is blue, "4 Mais uma linha" is green, "5 Texto" is blue, and "6 Mais uma linha" is green. The terminal has a dark background.

- Texto no primeiro commit

Texto commit

Checkout

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)
$ git checkout ca9e81e1d217ac09c70525f90aa12a07ce6d85a7
Note: switching to 'ca9e81e1d217ac09c70525f90aa12a07ce6d85a7'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at ca9e81e adição do arquivo.txt
```

A screenshot of a terminal window showing the output of a "git checkout" command. The terminal title is "arquivo.txt X". The output shows the command run, a note about switching to a detached HEAD, and instructions for creating a new branch. It also mentions that the user is in a detached HEAD state and provides examples for switching back. The terminal has a dark background.



Git Checkout

- Se observar mais de perto, verá que o Git está lhe informando que ele está apontando agora para seu arquivo no estado em que ele se encontrava quando você tirou aquela foto
- Se você digitar o comando git log novamente poderá levar um susto, observe:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT ((ca9e81e...))
$ git log
commit ca9e81e1d217ac09c70525f90aa12a07ce6d85a7 (HEAD)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 17 15:25:00 2020 -0300

    adição do arquivo.txt
```

O Git esta apontando para esse head e não mais para a branch master



Git Checkout

- Para você voltar para o status atual ou dar uma verificada em outros commits realizado você deverá usar o comando **git checkout master** (olha a branch master aí)

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT ((ca9e81e...))  
$ git checkout master  
Previous HEAD position was ca9e81e adição do arquivo.txt  
Switched to branch 'master'  
  
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/LABORATORIO/LAB_GIT (master)  
$ 
```

Agora o Git trouxe você novamente para sua branch master, você está de volta ao status atual!

Git Checkout



- Se você digitar **git log** novamente terá todo seu **Project History** à disposição, a partir daí, poderá se deslocar para onde desejar no seu **Project History**
- Observe que trabalhamos apenas com um arquivo. Num projeto trabalhamos com centenas até milhares de documentos, não importa, para o Git, basta que você defina os arquivos a serem rastreados e ele estará para tirar uma foto de quantos forem no momento que você definir

Introdução ao GitHub

GitHub

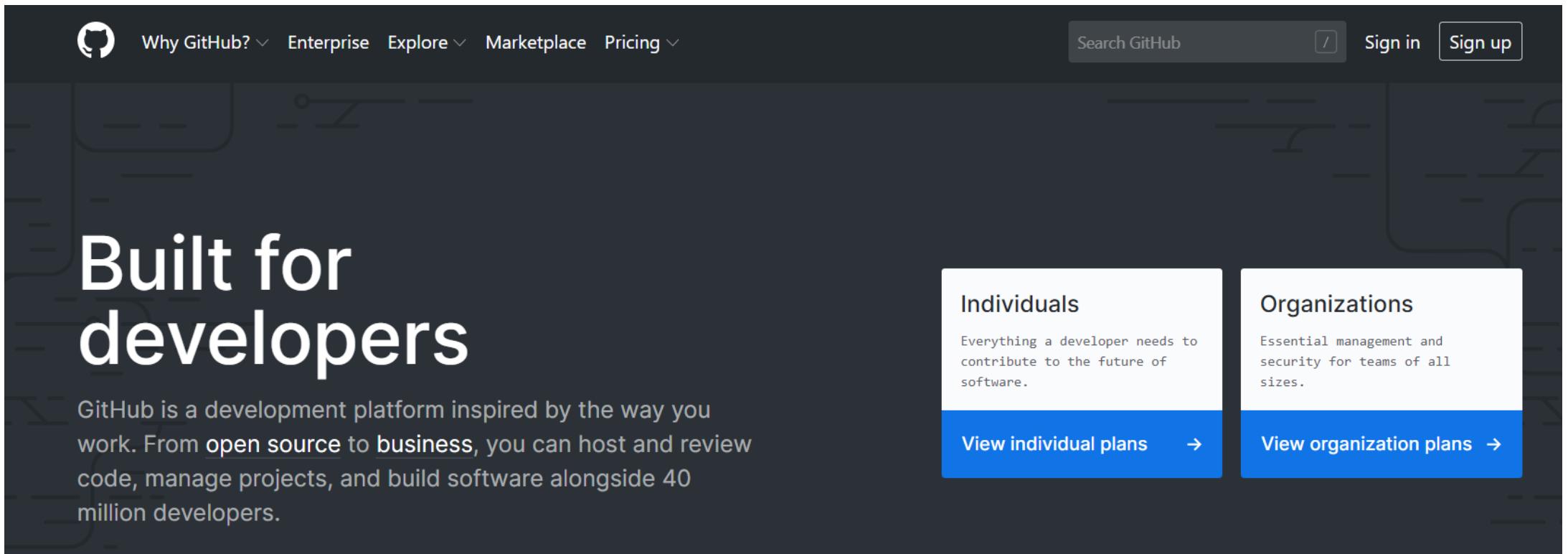
Vamos agora elevar nosso controle para a nuvem



Introdução ao GitHub



- Para iniciar neste tópico vamos ao site
<https://github.com/>



The screenshot shows the GitHub homepage with a dark background featuring a faint circuit board pattern. At the top, there is a navigation bar with links for "Why GitHub?", "Enterprise", "Explore", "Marketplace", and "Pricing". On the right side of the header are "Search GitHub", "Sign in", and "Sign up" buttons. The main headline "Built for developers" is displayed in large white text. Below it, a paragraph describes GitHub as a platform for developers, mentioning its use for open source and business projects, code hosting, project management, and software development. Two prominent blue call-to-action buttons are shown: "View individual plans" with an arrow pointing right, and "View organization plans" with an arrow pointing right. To the left of these buttons, there are two sections: "Individuals" (described as "Everything a developer needs to contribute to the future of software") and "Organizations" (described as "Essential management and security for teams of all sizes").

Introdução ao GitHub



- Git → ferramenta de versionamento – VCS
- Característica do Git → trabalha em repositório Local
- O significa isso → para termos acesso a todo processo de versionamento de nosso projeto precisamos estar na máquina em que ele foi criado, se não possuir acesso remoto a essa máquina não poderá acessar seu repositório de projeto
- É aí que entra o GitHub → que é uma plataforma remota que nos permite sincronizar nosso repositório local com a nuvem

Introdução ao GitHub



- O GitHub é uma plataforma remota de hospedagem de projetos
- É baseado no Git permitindo que continuemos a trabalhar normalmente como se estivéssemos localmente
- Há outras ferramentas com esse propósito como o BitBucket <https://bitbucket.org/> e outros ...
- Mas o GitHub é o Melhor!!!



Introdução ao GitHub



- Poder do GitHub → além de integrar todas as ferramentas do Git, incorpora outras ferramentas que potencializam ainda mais as ações de projeto em times
- Você também irá encontrar um gigantesca comunidade trabalhando com GitHub, e mais um número imenso de projetos compartilhados só esperando por você, em outras palavras, você nem sempre precisará partir do zero, terá a seu alcance muito do trabalho que já foi desenvolvido e que poderá lhe ser útil

Introdução ao GitHub



- Para trabalhar com o GitHub você antes de mais nada deverá criar uma conta, o processo é bem simples, basta seguir os passos:



Create your account

Join GitHub

Username *

Email address *

Password *

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

Email preferences

Send me occasional product updates, announcements, and offers.

Verify your account

Por favor, resolva esse desafio para que possamos saber que você é uma pessoa de verdade.

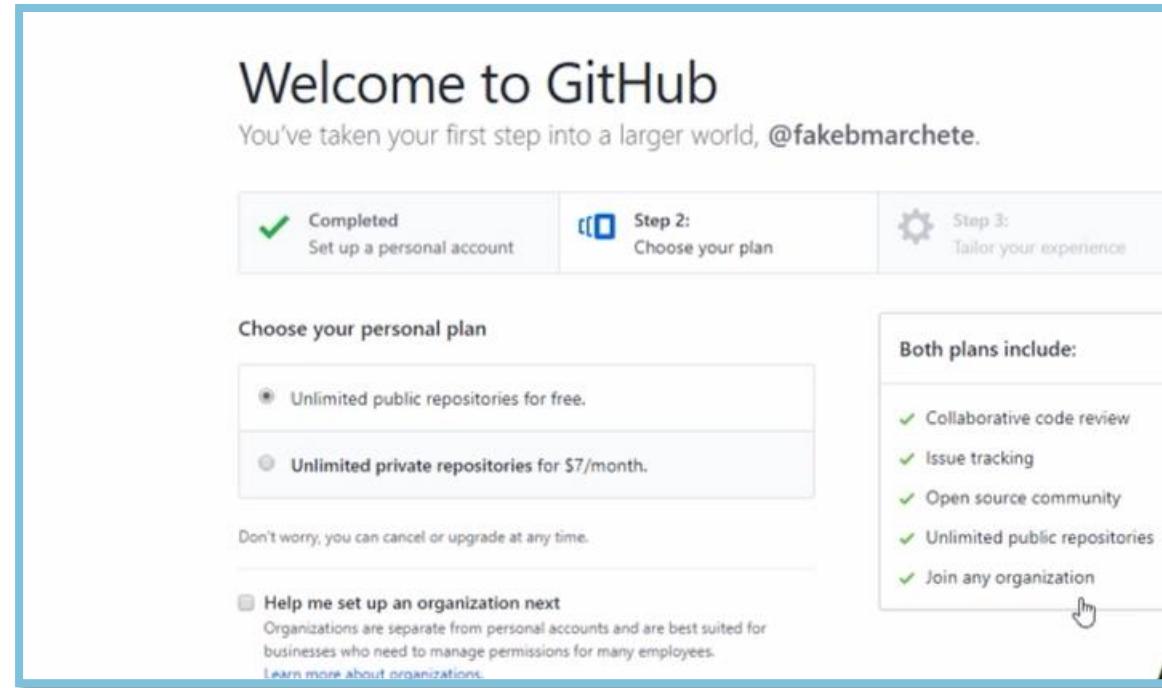
[Verifique](#)

(Speaker icon)

Introdução ao GitHub



- Na sequência seremos questionados a cerca de qual plano queremos?
- No caso, vamos escolher o plano Unlimited public repositório for free



The image shows the "Welcome to GitHub" screen. At the top, it says "Welcome to GitHub" and "You've taken your first step into a larger world, @fakebmarchete." Below this, there are three tabs: "Completed" (Set up a personal account), "Step 2: Choose your plan" (highlighted in blue), and "Step 3: Tailor your experience".

The "Choose your personal plan" section contains two radio buttons:

- Unlimited public repositories for free.
- Unlimited private repositories for \$7/month.

A note below says "Don't worry, you can cancel or upgrade at any time."

At the bottom, there is a checkbox for "Help me set up an organization next" with a descriptive text about organizations and a link to "Learn more about organizations".

To the right, a sidebar lists "Both plans include:" with several checked items:

- Collaborative code review
- Issue tracking
- Open source community
- Unlimited public repositories
- Join any organization

A hand cursor icon is positioned over the "Join any organization" item.

Introdução ao GitHub



- Por último apresenta-se umas questões, que poderemos deixar de lado e submeter a criação de nossa conta



What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

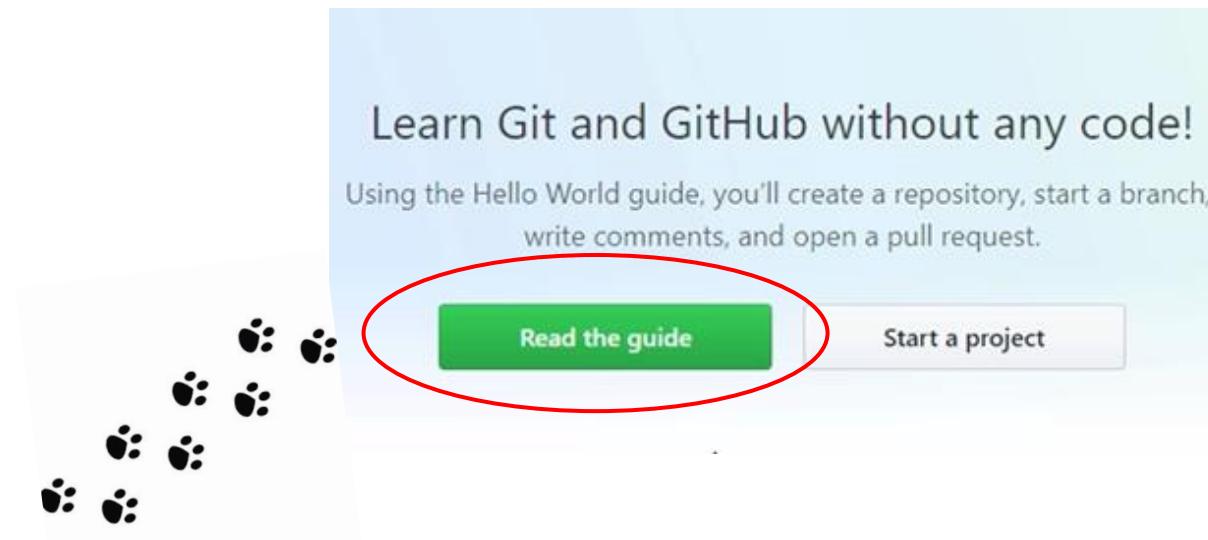
[Submit](#) [skip this step](#)

A screenshot of a GitHub account creation step. It asks for interests in a text input field, with examples like 'tutorials, android, ruby, web-development, machine-learning, open-source'. Below the input is a green 'Submit' button with a cursor icon hovering over it, and a blue 'skip this step' link.

Introdução ao GitHub



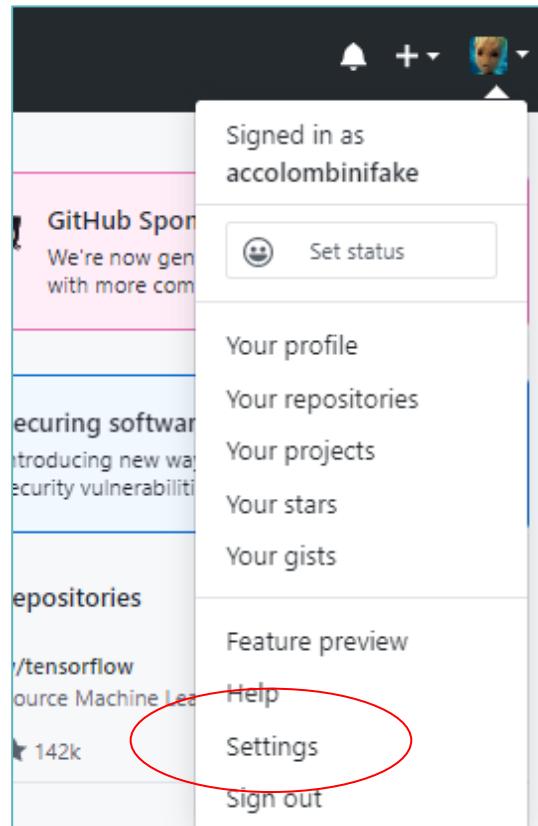
- Há muitos tutoriais que ensinam a trabalhar com o GitHub, observe que o primeiro está à sua disposição ao finalizar sua conta
- Há também um guia para você iniciar seu projeto
- Antes disso, vamos fazer algumas configurações básicas



Introdução ao GitHub



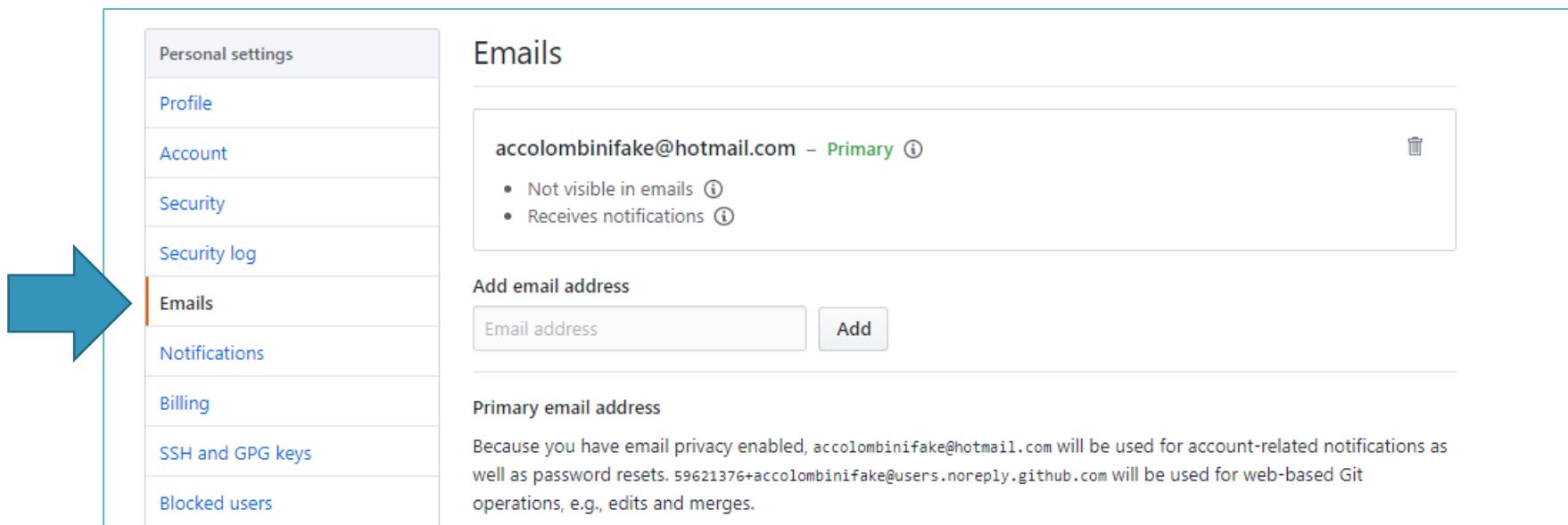
- Em profiles vamos em settings, veja como:



Introdução ao GitHub



- Neste ponto há muito o que preencher, mas o essencial é confirmar seu e-mail. Faça isso e estará pronto para trabalhar



A screenshot of the GitHub Personal settings page, specifically the 'Emails' section. A large blue arrow points from the left towards the 'Emails' tab in the sidebar. The sidebar also lists 'Profile', 'Account', 'Security', 'Security log', 'Notifications', 'Billing', 'SSH and GPG keys', and 'Blocked users'. The main area shows an email account 'accolombinifake@hotmail.com – Primary' with two bullet points: 'Not visible in emails' and 'Receives notifications'. Below this is a 'Add email address' form with an 'Email address' input field and an 'Add' button. At the bottom, there's a note about primary email addresses and privacy.

Personal settings

Profile

Account

Security

Security log

Emails

Notifications

Billing

SSH and GPG keys

Blocked users

Emails

accolombinifake@hotmail.com – Primary ⓘ

- Not visible in emails ⓘ
- Receives notifications ⓘ

Add email address

Email address

Add

Primary email address

Because you have email privacy enabled, [accolombinifake@hotmail.com](#) will be used for account-related notifications as well as password resets. [59621376+accolombinifake@users.noreply.github.com](#) will be used for web-based Git operations, e.g., edits and merges.

GitHub → criando e entendendo repositórios

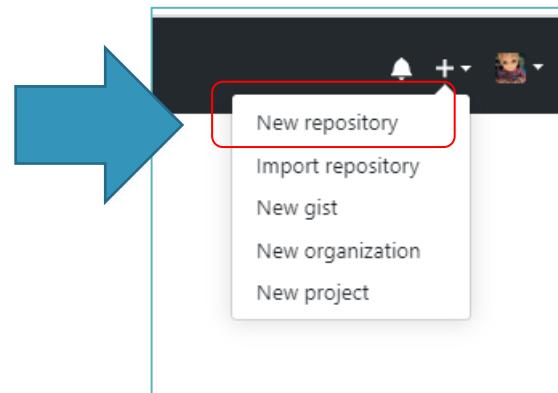


- Todo projeto em GitHub demanda a criação de um repositório
- Logo, você deverá criar tantos repositórios quanto projetos que esteja trabalhando
- Vamos começar com um, já é suficiente para nosso aprendizado
- Existem diversas maneiras de criarmos esses repositórios e fazermos o sincronismo com nosso computador, nosso **ambiente Git, lembra dele!!!**

GitHub → criando e entendendo repositórios



- Criação de repositórios
 - Técnica simplificada → criar o repositório a partir de sua página no GitHub
 - Basta clicar na opção de + e em seguida em novo repositório



GitHub → criando e entendendo repositórios



- Criando um repositório com o nome inicial
- Este repositório será público, privado só para contas especiais
- Demais opções serão exploradas nas próximas aulas

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner Repository name

fakebmarchete /

Great repository names are short and memorable. Need inspiration? How about [super-duper-giggle](#).

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

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

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: Add a license:

GitHub → criando e entendendo repositórios



- A próxima tela mostra como ligamos nosso diretório no GitHub com nosso diretório local

The screenshot shows a GitHub repository creation page for a repository named 'accolombinifake / inicial'. The top navigation bar includes 'Code', 'Issues 0', 'Pull requests 0', 'Actions', 'Projects 0', 'Wiki', 'Security', 'Insights', and 'Settings'. The main content area has a heading 'Quick setup — if you've done this kind of thing before' with a link to 'Set up in Desktop' or 'HTTPS' (selected) or 'SSH' with the URL 'https://github.com/accolombinifake/inicial.git'. It also includes instructions to 'Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.' Below this, there are three sections: 1) '...or create a new repository on the command line' with the following git commands:

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

2) '...or push an existing repository from the command line' with the command:

```
git remote add origin https://github.com/accolombinifake/inicial.git
git push -u origin master
```

3) '...or import code from another repository' with the instruction 'You can initialize this repository with code from a Subversion, Mercurial, or TFS project.' and a 'Import code' button. At the bottom, a note says 'ProTip! Use the URL for this page when adding GitHub as a remote.'

Introdução ao GitHub

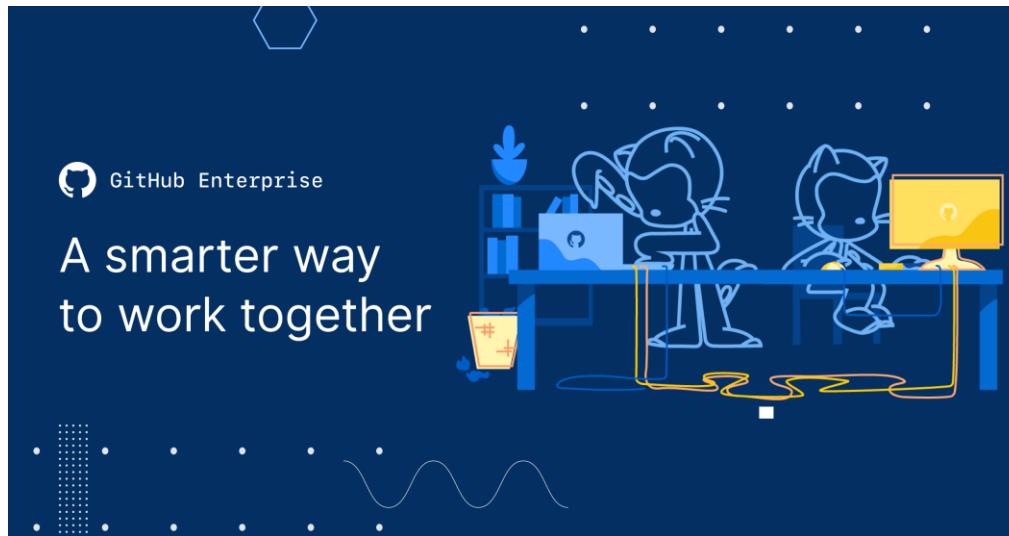


- Vamos fazer uma pausa e falar um pouco sobre o arquivo **Readme.md**
- Este arquivo é muito interessante para você especificar o por quê de seu projeto, você pode colocar instruções de instalação, normas que estão sendo empregadas em seu desenvolvimento, etc
- Você tem a opção de criar o arquivo de **Readme** no momento em que está criando seu Repositório

Introdução ao GitHub



- Veja a tela a seguir:



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: fakebmarchete / Repository name: inicial

Great repository names are short and memorable. Need inspiration? How about super-duper-giggle.

Description (optional): Repertório de teste inicial

Public: Anyone can see this repository. You choose who can commit.

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

Initialize this repository with a README: This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None | ⓘ

Introdução ao GitHub



- Para o GitHub todo arquivo com extensão **.md** possui um esquema de marcação específico criado pelo GitHub
- Para editar esse arquivo usamos diretamente a plataforma do GitHub, clicando sobre o arquivo

A screenshot of a GitHub repository page. At the top, there's a summary bar with metrics: 1 commit, 1 branch, 0 packages, 0 releases, and 1 contributor. Below this are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The main area shows a list of files. The first file, 'README.md', is highlighted with a red rectangular box around its thumbnail and name. To the right of the file list, there's a note about the latest commit: 'Latest commit 1e7dc8e 3 minutes ago'. Below the file list, there's a text input field containing the word 'Teste'.

Introdução ao GitHub



- Em seguida clique no **ícone para edição** e pronto, é só digitar tudo o que considera relevante para uma primeira leitura a cerca o que se trata seu repositório

A screenshot of a GitHub commit page for a repository named 'accolobminifake'. The commit details are as follows:

- Author: accolobminifake
- Commit message: Initial commit
- Date: 1e7dc8e 15 seconds ago
- Contributors: 1 contributor
- File statistics: 1 lines (1 sloc), 7 Bytes
- Actions: Raw, Blame, History, Copy, Edit (highlighted with a red box)

The commit content is a single line of text: "Teste".

Introdução ao GitHub



- Ao clicar em editar, o editor do GitHub é aberto e seu arquivo já recebe uma marcação gerada automaticamente com o título de seu repositório, não delete essa marcação, nas linhas seguintes insira suas informações. **Mas atenção este arquivo não se limita apenas a essa descrição**, falaremos mais nas próximas aulas

The screenshot shows a GitHub repository page for 'accolombinifake / Teste'. The 'Code' tab is selected. A modal window is open over the repository header, showing the file 'Teste / README.md'. The modal has tabs for 'Edit file' and 'Preview changes'. The 'Edit file' tab is active, displaying the contents of the README.md file:

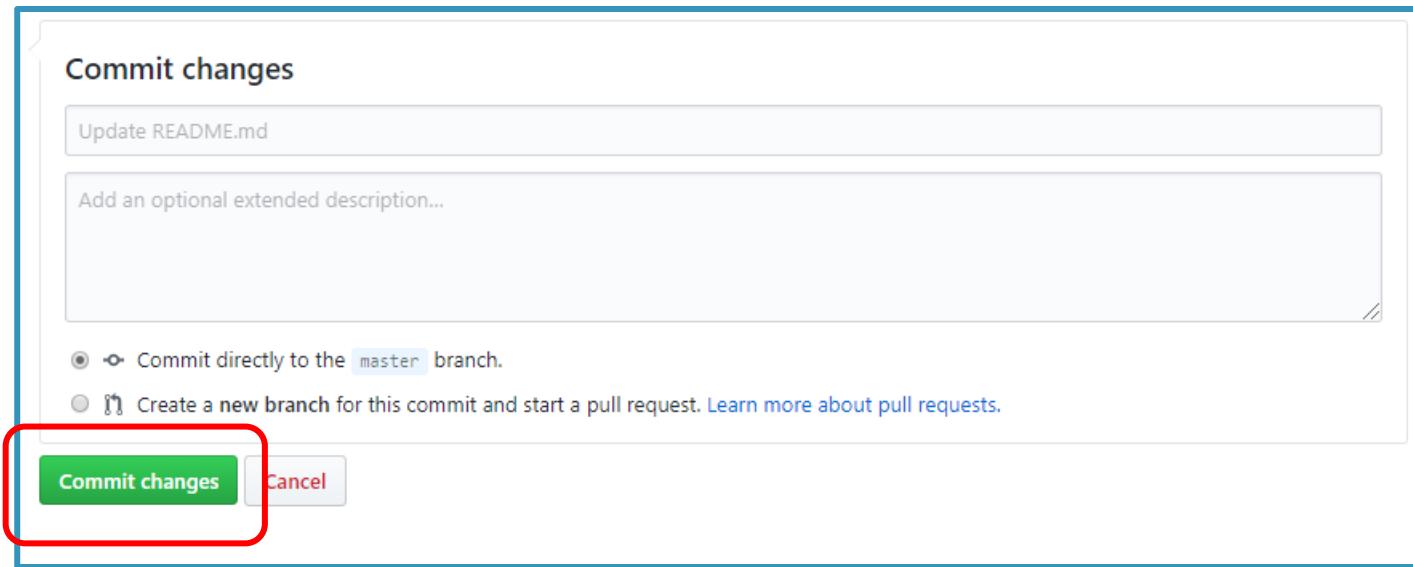
```
1 # Teste
2
3 Este projeto .....
```

A red rectangular box highlights the first line of the file, '# Teste', which is the automatically generated repository title.

Introdução ao GitHub



- Quando concluir sua digitação você deverá fazer o commit do seu trabalho, sim é o mesmo processo que viu com o Git. O GitHub já vai começar a controlar seu projeto, falaremos mais sobre isso, aguarde



Introdução ao GitHub



- Vamos agora conhecer outro arquivo importante, o arquivo **.gitignore**, lembra que comentamos que em algum momento podemos optar por não controlar algum arquivo que usamos em nosso projeto, pois é chegou a hora.
- Esse arquivo pode ser criado diretamente no nosso diretório local, ou a partir da criação de um novo repositório na plataforma do GitHub, para exemplificar, faremos isso na plataforma, acompanhe

Introdução ao GitHub



- Quando criamos um novo repositório temos a opção além de criar o arquivo de README.md, criar o gitignore, observe



A screenshot of the GitHub repository creation interface. The form includes fields for Owner (set to 'accolombinifake'), Repository name ('gitignore'), Description (empty), and two radio button options for visibility: 'Public' (selected) and 'Private'. Below these are sections for initializing the repository with a README and adding a .gitignore file. A red box highlights the '.gitignore: None' dropdown menu. At the bottom is a green 'Create repository' button.

Owner: accolombinifake / Repository name: gitignore

Great repository names are short and memorable. Need inspiration? How about shiny-waddle?

Description (optional):

Public Anyone 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:

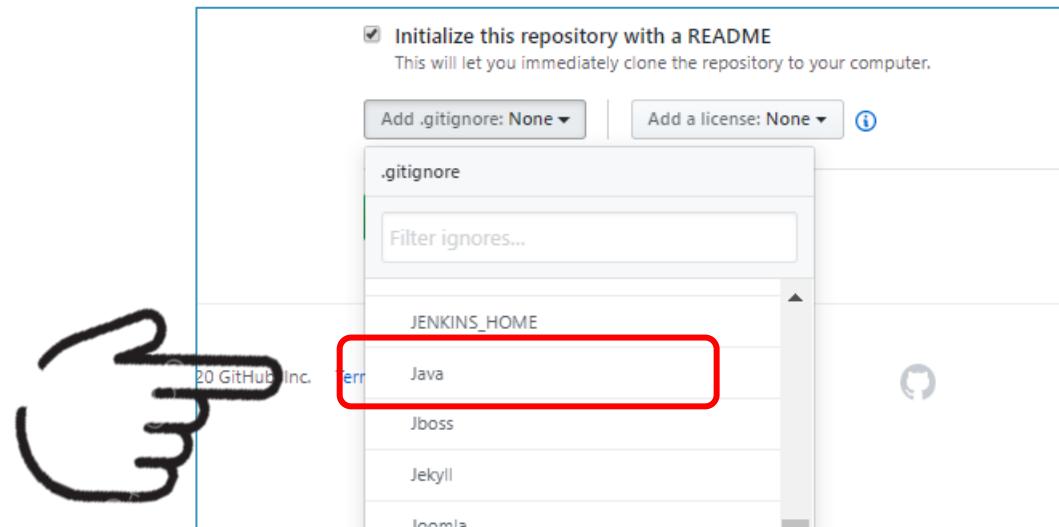
Add a license:

Create repository

Introdução ao GitHub



- Ao clicar na opção Add .gitignore o GitHub trás uma série de opções pré-configuradas que podem apoiá-lo nessa tarefa (trata-se de arquivos comuns de serem ignorados), mas sim é claro, você poderá incluir as suas configurações
- Como exemplo, vamos admitir que estamos num projeto Java



Introdução ao GitHub



- Clicando sobre o nome do arquivo `.gitignore` o editor se abre com uma série de opções previamente estabelecidas pelo GitHub, pense nisso como um suporte a seu trabalho, cabe você analisar se concorda ou não, acrescentando ou tirando arquivos



Branch: master [gitignore](#) / `.gitignore`

acolombinifake Initial commit
1 contributor

23 lines (18 sloc) | 278 Bytes

```
1 # Compiled class file
2 *.class
3
4 # Log file
5 *.log
6
7 # BlueJ files
8 *.ctxt
9
10 # Mobile Tools for Java (J2ME)
11 .mtj.tmp/
12
13 # Package Files #
14 *.jar
15 *.war
16 *.nar
17 *.ear
18 *.zip
19 *.tar.gz
20 *.rar
21
22 # virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
23 hs_err_pid*
```



Todos esses arquivos serão automaticamente ignorados pelo GitHub sempre que um sincronismo for realizado. Portanto, tenha sempre um arquivo `.ignore` em seu projeto



Introdução ao GitHub

Vamos praticar - Laboratório

Sincronizando seu projeto



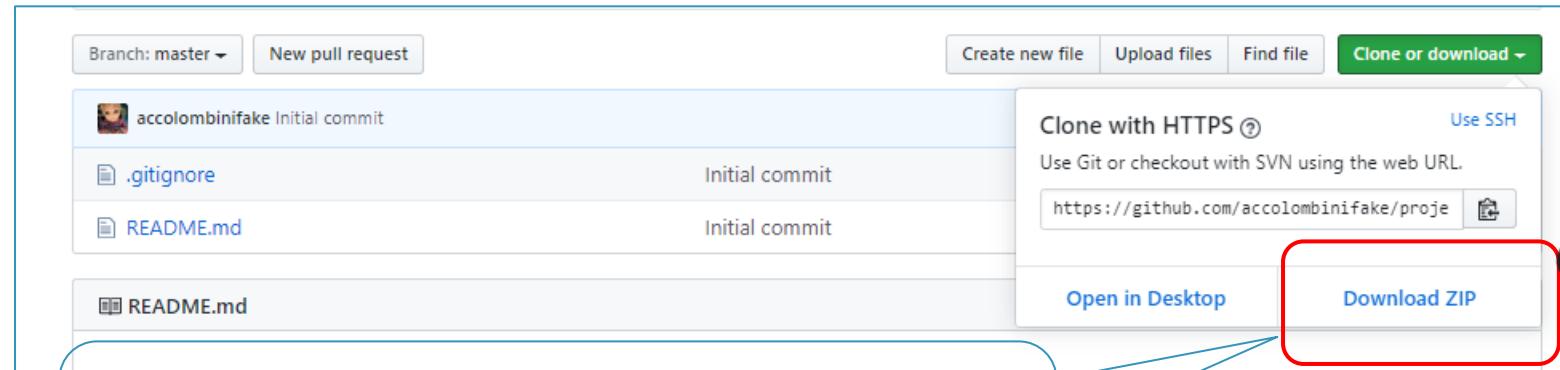
- Dando início ao nosso laboratório, vamos a partir da plataforma GitHub criar um novo repositório
- Vamos chamar esse repositório de `projetolab`, inicializar o `readme.md` e adicionar o `.gitignore` como se fosse um projeto Java, você já sabe fazer isso, mãos à obra. Teremos algo como:

A screenshot of a GitHub repository page. The repository name is `accolombinifake/projetolab`. The page shows basic statistics: 1 commit, 1 branch, 0 packages, 0 releases, and 1 contributor. Below the stats, there's a list of files: `.gitignore` (Initial commit, now), `README.md` (Initial commit, now), and `README.md` (another entry, now). The `README.md` file content is displayed as "projetolab" with a note below it: "Projeto para ações práticas em laboratório, sincronizando Git com GitHub.".

Sincronizando seu projeto



- Precisamos agora, de alguma forma, descarregar esse projeto em nossa máquina, mas como?
- A primeira coisa que nos vem à mente é ...

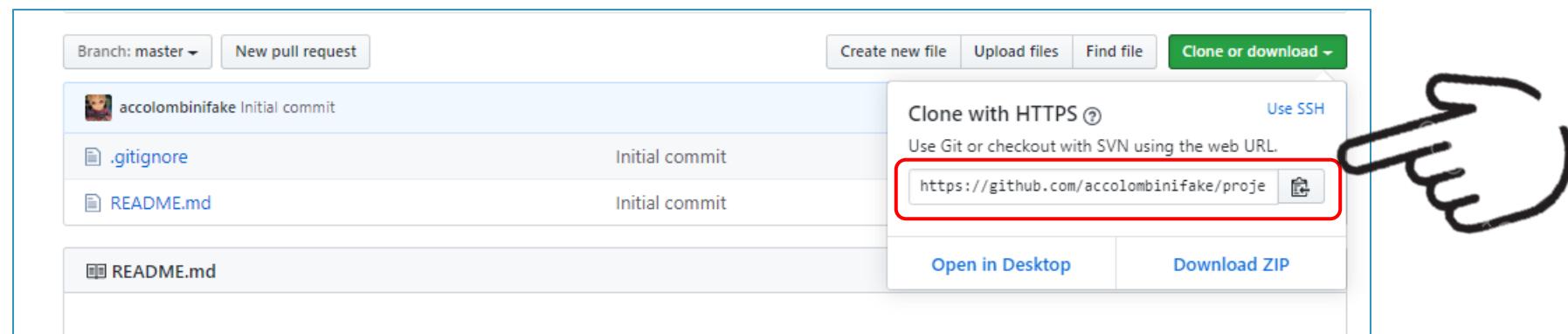


Resista a tentação e não faça isso,
você perderá as ferramentas de gestão
e sincronismo

Sincronizando seu projeto



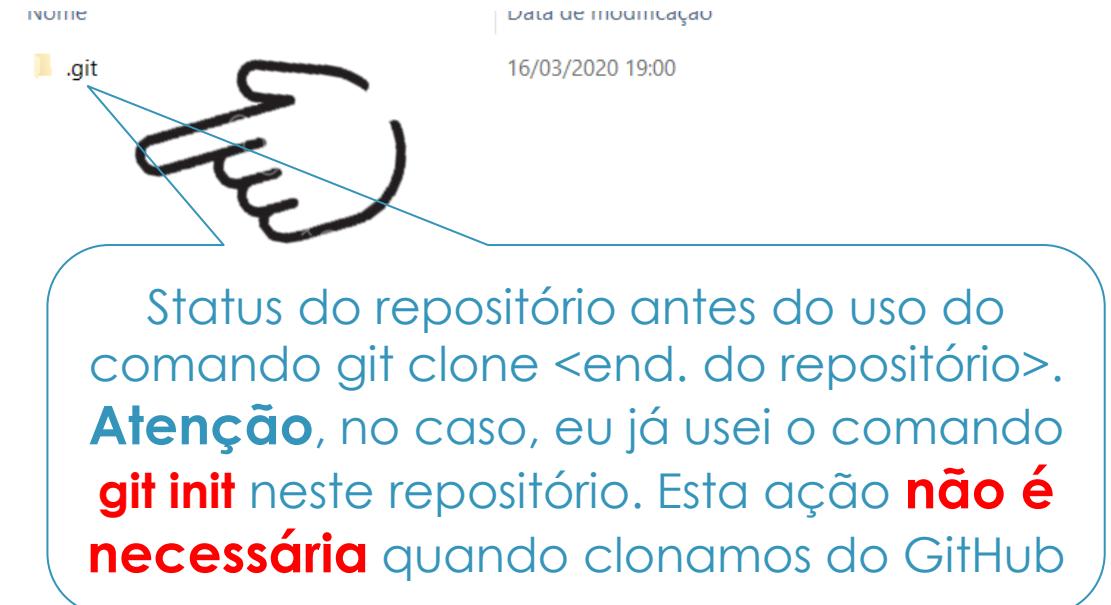
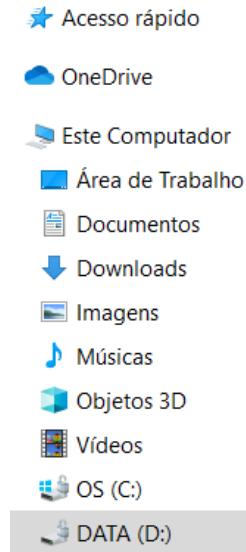
- Para garantir o pleno uso dos recursos e ferramentas do Git e GitHub, você deverá copiar o endereço do seu repositório no GitHub



Sincronizando seu projeto



- Uma vez copiado, volte para seu Visual Studio Code, abra o terminal, vá para a pasta criada anteriormente, no caso, criamos a pasta GIT
- Uma vez na pasta criada usamos o comando **git clone <endereço do repositório no GitHub>**
- Com essa ação, todo repositório criado no GitHub será baixado para sua máquina



Status do repositório antes do uso do comando `git clone <end. do repositório>`.
Atenção, no caso, eu já usei o comando `git init` neste repositório. Esta ação **não é necessária** quando clonamos do GitHub

Sincronizando seu projeto



- Posicionando no diretório destino e usando o comando
git clone <endereço do repositório do GitHub>

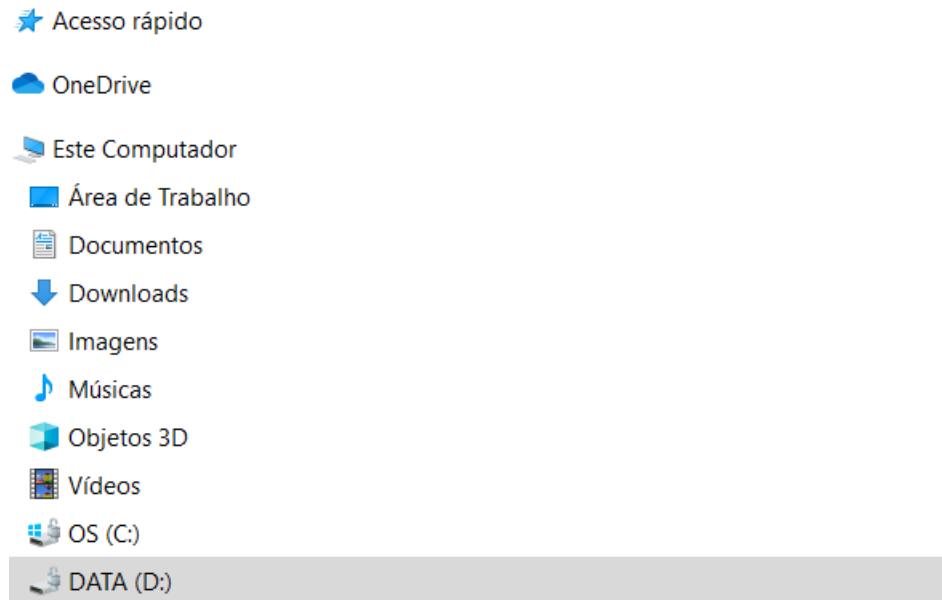
```
acco1@DESKTOP-TMBP1KD MINGW64 /  
$ cd D:/Users/Angelo  
  
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo  
$ cd GIT  
  
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT (master)  
$ git clone https://github.com/acco1ombinifake/projetolab.git  
Cloning into 'projetolab'...  
remote: Enumerating objects: 4, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (4/4), done.  
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (4/4), 917 bytes | 2.00 KiB/s, done.
```

Tudo certo,
podemos
prosseguir

Sincronizando seu projeto



- Diretório após o uso do comando **git clone <end do repositório GitHub>**



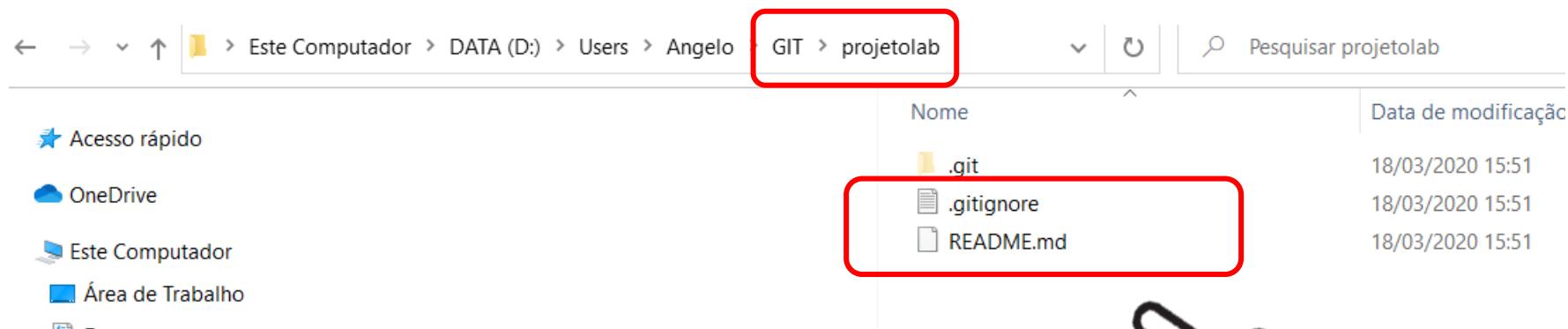
16/03/2020 19:00
18/03/2020 15:51

Repositório clonado, agora diretório local e repositório remoto estão sincronizados. Observe que os nomes foram mantidos

Sincronizando seu projeto



- Abrindo o diretório local (pasta GIT) poderemos visualizar todos os arquivos criados no contexto do GitHub, veja ...



Sincronizando seu projeto



- Até aqui, moleza, mas o que queremos de fato no nosso dia a dia é criar novos arquivos em nosso projeto, realizar alterações, versionarmos toda movimentação no cenário de nosso projeto através do comando **git commit** (se estivermos localmente) ou ainda com o comando **commit** se estivermos no GitHub e sincronizamos com nosso repositório no GitHub
- Então vamos ao trabalho, para isso, vamos ao Visual Studio Code, selecione a pasta do seu projeto e vamos adicionar um novo arquivo, veja como ...

Sincronizando seu projeto



- Uma vez selecionada a pasta de trabalho criemos um novo arquivo

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with sections for 'OPEN EDITORS' (containing 'Welcome') and 'PROJETOLAB' (containing '.gitignore' and 'README.md'). The center is the 'OPEN EDITORS' view, which lists 'Welcome' and 'index.htm' (the current file being edited). The right pane displays the code for 'index.htm':

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9
10 </body>
11 </html>
```

Sincronizando seu projeto



- A partir daí podemos trabalhar normalmente, veja o fluxo padrão a seguir ...

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

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

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git add index.htm

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   index.htm
```

Sincronizando seu projeto



- Arquivo adicionado na Stage, estamos prontos para realizar um commit no nosso repositório local
- Vamos agora realizar um **git commit -m 'mensagem'**

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git commit -m 'arquivo index.htm criado'
[master 81f3a2b] arquivo index.htm criado
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.htm

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Sincronizando seu projeto



- Observe no, entanto, que nossa ação ficou restrita à nossa máquina, veja nosso repositório no GitHub
- Apenas nosso primeiro commit realizado ainda no ambiente GitHub estará presente

A screenshot of a GitHub repository page. At the top, there are summary statistics: 1 commit, 1 branch, 0 packages, 0 releases, and 1 contributor. Below this, a button for 'Clone or download' is visible. A large hand-drawn style cursor points to the '1 commit' button, which is highlighted with a red rectangle. The main content area shows a single commit by 'accolombinifake' with the message 'Initial commit'. This commit was made 2 hours ago and includes files '.gitignore' and 'README.md'. A status bar at the bottom indicates the file 'README.md' is selected.

Sincronizando seu projeto



- Agora no meu ambiente local, lembra, nós clonamos nosso repositório do GitHub, veja o que acontece quando usamos o comando **git log**
- Ele trará dois commits, um realizado no GitHub (referenciado como origin/master, origin/HEAD) e o outro realizado em nosso ambiente local, observe(referenciado como HEAD → master) ...

```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git log
commit 81f3a2b64beb1ccc0c5ac5394827f0566cce00fd (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Wed Mar 18 17:03:20 2020 -0300

    arquivo index.htm criado

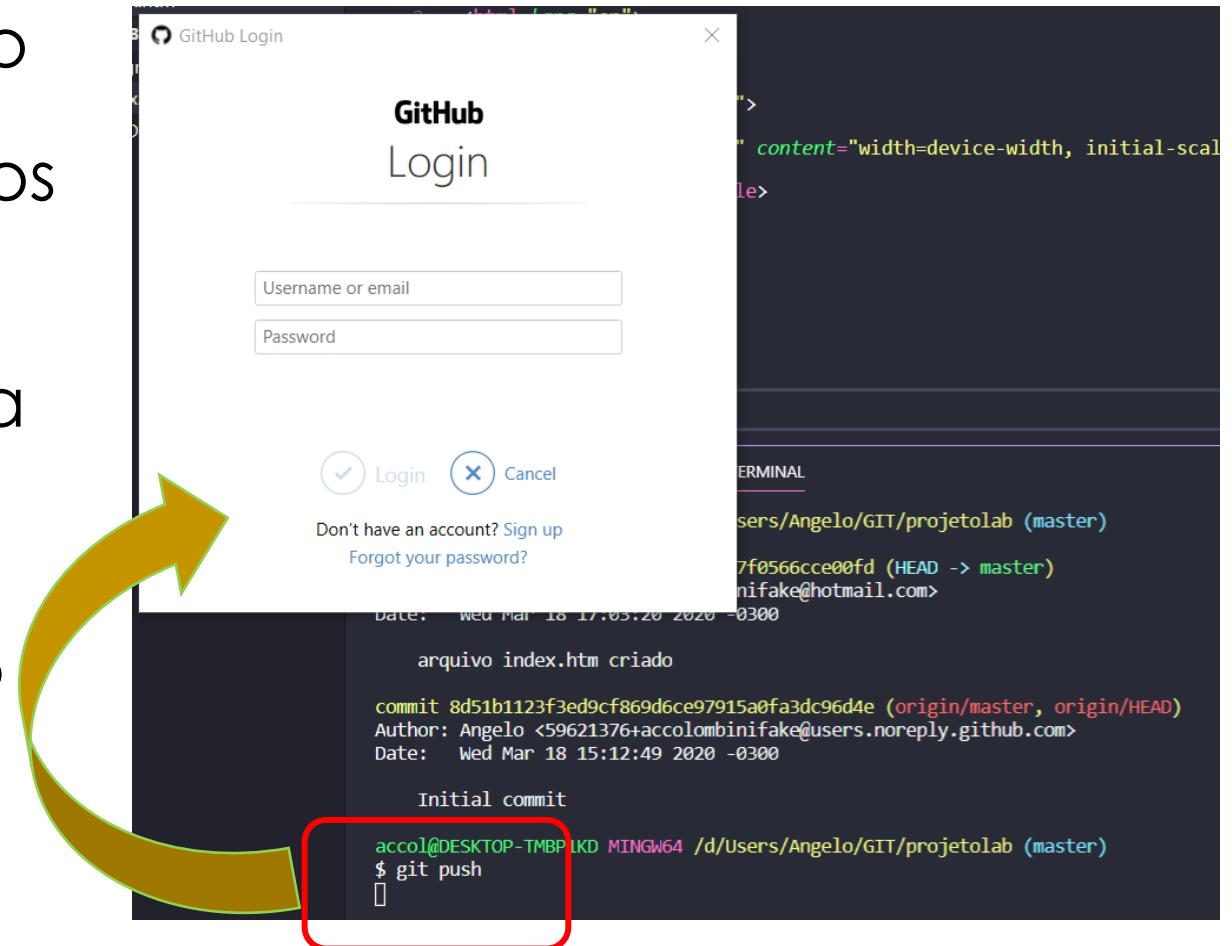
commit 8d51b1123f3ed9cf869d6ce97915a0fa3dc96d4e (origin/master, origin/HEAD)
Author: Angelo <59621376+accolombinifake@users.noreply.github.com>
Date:   Wed Mar 18 15:12:49 2020 -0300

    Initial commit
```

Sincronizando seu projeto



- Para fazermos o sincronismo dessas alterações com o repositório do GitHub, vamos precisar de mais um comando, o **git push**
 - Ao usar esse comando pela primeira vez, será aberta uma caixa de diálogo, é preciso que você tenha permissão para publicar no repositório. Preencha com seus dados



Sincronizando seu projeto



- Podemos confirmar digitando o comando **git config --global -l**

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git config --global -l
user.name=accolombinifake
user.email=accolombinifake@hotmail.com
core.editor=notepad++
```

Sincronizando seu projeto



Nota: qualquer pessoa pode clonar um repositório do GitHub público, mas para você publicar no diretório é preciso que tenha permissão, falaremos mais sobre isso nas próximas aulas

- Uma vez tendo a permissão, observe o que acontece...

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 318 bytes | 318.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/acco1ombinifake/projetolab.git
  8d51b11..81f3a2b  master -> master
```

Sincronizando seu projeto



- Para conferir se tudo está ok, vamos agora ao nosso repositório remoto no GitHub, observe ...
- Temos agora dois commits e nosso novo arquivo se encontra no repositório

The screenshot shows a GitHub repository page for the user 'acolombinifake' named 'projetolab'. The repository description is 'Projeto para ações práticas em laboratório, sincronizando Git com GitHub.' The main statistics section shows 2 commits, 1 branch, 0 packages, 0 releases, and 1 contributor. A red box highlights the '2 commits' link. Below this, a list of commits is shown:

File	Type	Message	Time
.gitignore	Initial commit		2 hours ago
README.md	Initial commit		2 hours ago
index.htm	arquivo index.htm criado	arquivo index.htm criado	26 minutes ago
README.md			

The last commit for 'index.htm' is highlighted with a red box.

Sincronizando seu projeto



- Vamos agora fazer o caminho inverso, queremos a partir de alterações realizadas no nosso repositório remoto, atualizar nosso diretório local, para isso, vamos precisar do comando **git pull**

Nota: antes de fazer o git push você precisa ter a certeza de que seu diretório está atualizado com as últimas modificações do seu repositório remoto GitHub, daí a necessidade de usar o **git pull** para assegurar o sincronismo entre o diretório local e o repositório remoto

Sincronizando seu projeto



- Para nosso laboratório, vamos simular esse evento promovendo alterações em nosso repositório na nuvem GitHub
- Vamos criar um novo arquivo, para isso clicamos em criar novo arquivo, veja ...

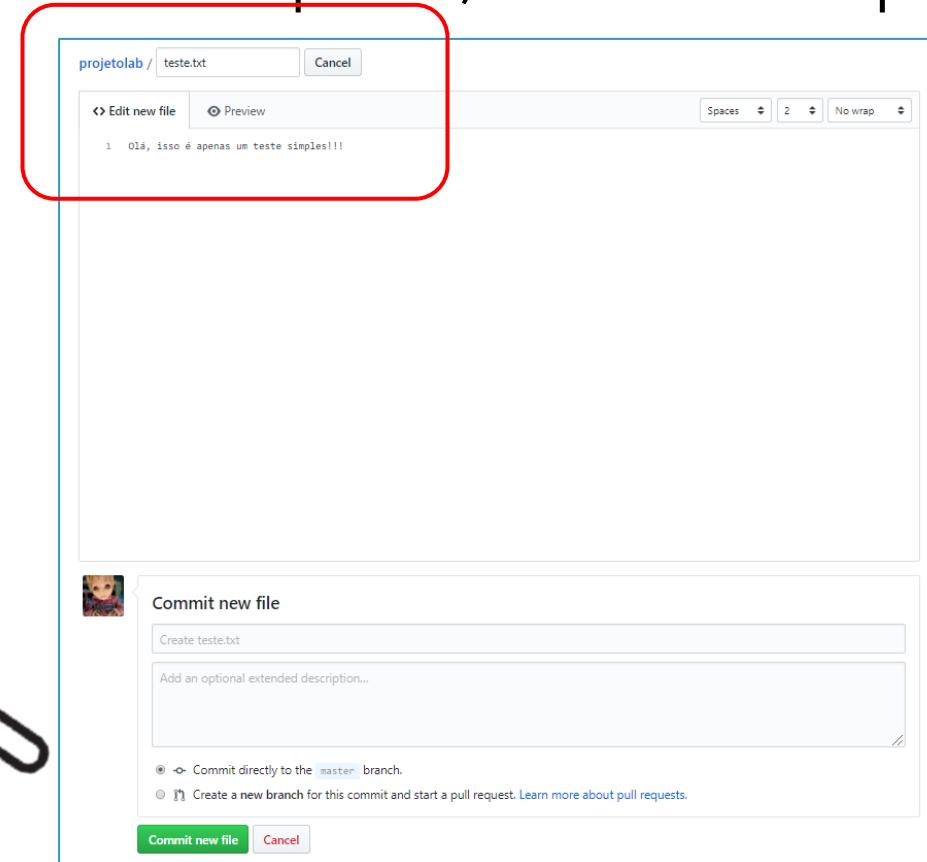
The screenshot shows a GitHub repository page for 'acolombinifake / projetolab'. The page displays basic repository statistics: 2 commits, 1 branch, 0 packages, 0 releases, and 1 contributor. Below these stats, there are buttons for 'Branch: master', 'New pull request', 'Create new file' (which is highlighted with a red box), 'Upload files', 'Find file', and 'Clone or download'. A large hand cursor icon is positioned over the 'Create new file' button. The repository's history is listed below, showing commits for 'acolombinifake' and 'index.htm'. The most recent commit is 'arquivo index.htm criado' made 2 hours ago.

File	Commit Message	Time Ago
.gitignore	Initial commit	3 hours ago
README.md	Initial commit	3 hours ago
index.htm	arquivo index.htm criado	2 hours ago
README.md		

Sincronizando seu projeto



- Uma vez criado o arquivo, não se esqueça, faça o **commit**



Sincronizando seu projeto



- Observe que este novo arquivo está no GitHub, mas não se encontra ainda no repositório local

The screenshot shows a GitHub repository page for 'acolombinifake / projetolab'. The repository has 3 commits, 1 branch, 0 packages, 0 releases, and 1 contributor. A red box highlights the commit 'Create teste.txt' which includes the file 'teste.txt' in the commit message. The commit was made 26 seconds ago.

- Observe que este arquivo não se encontra em nosso diretório local

The screenshot shows a Windows File Explorer window with the path 'GIT > projetolab'. The directory contains the following files:

Nome	Data de modificação	Tipo
.git	18/03/2020 17:03	Pasta de arquivos
.gitignore	18/03/2020 15:51	Documento de Texto
index.htm	18/03/2020 16:13	Chrome HTML Documento
README.md	18/03/2020 15:51	Arquivo MD



Sincronizando seu projeto



- Para sincronizarmos nosso diretório local, basta acessarmos nosso terminal e usar o comando **git pull**

Observe que ele trouxe um arquivo, sendo este adicionado ao diretório local

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 691 bytes | 4.00 KiB/s, done.
From https://github.com/accolombinifake/projetolab
  81f3a2b..891579b  master      -> origin/master
Updating 81f3a2b..891579b
Fast-forward
  teste.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 teste.txt
```

Sincronizando seu projeto



- Observe agora quando fazemos uso do comando **git log** como está nosso diretório atual

Temos em nosso diretório local os três commits realizado e estamos exatamente como nosso repositório no GitHub

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ git log
commit 891579bfaff5811ea7df824afd284d72e36bcfeb (HEAD -> master, origin/master, origin/HEAD)
Author: Angelo <59621376+accolombinifake@users.noreply.github.com>
Date:   Wed Mar 18 18:42:36 2020 -0300

    Create teste.txt

commit 81f3a2b64beb1ccc0c5ac5394827f0566cce00fd
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Wed Mar 18 17:03:20 2020 -0300

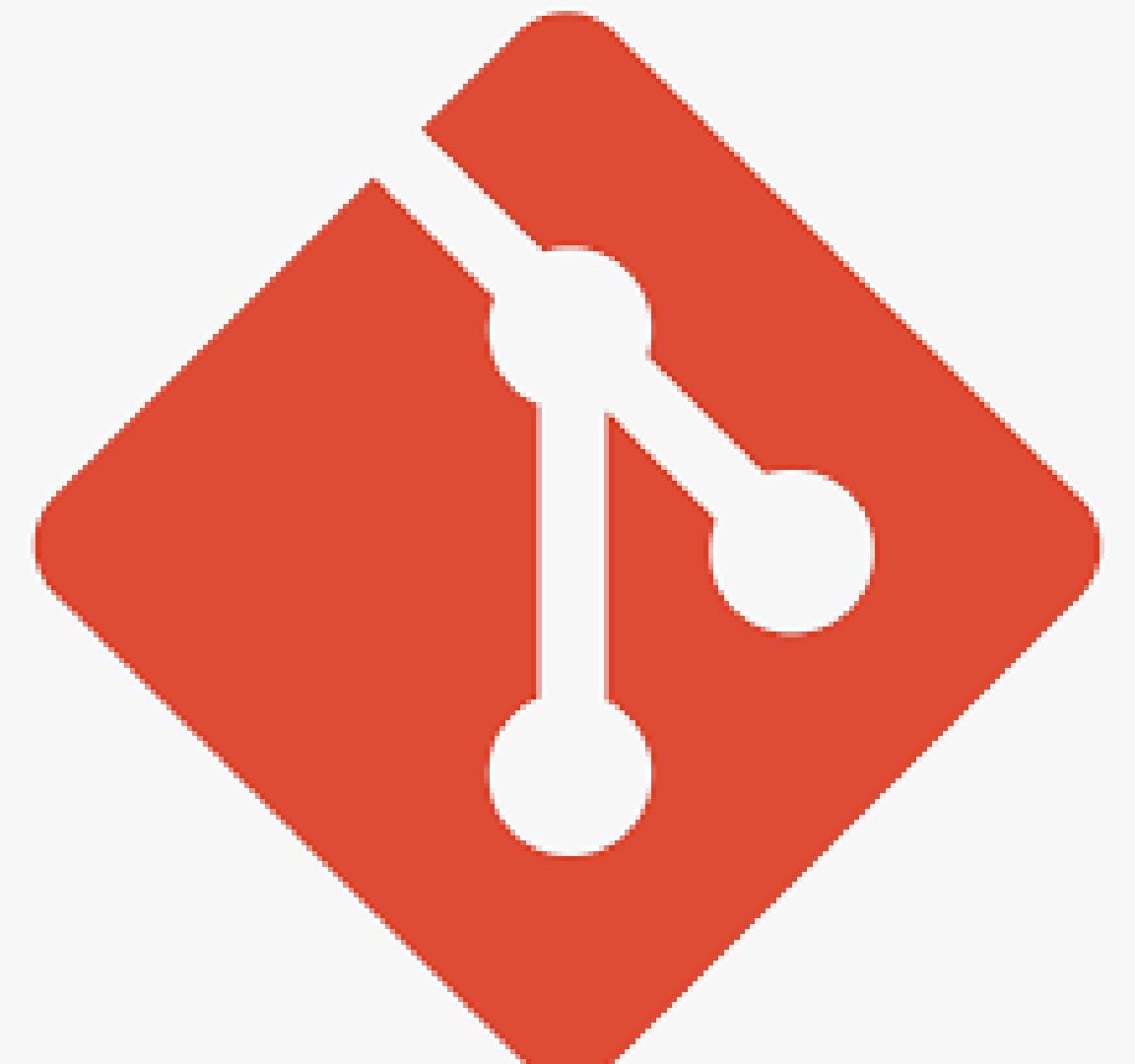
    arquivo index.htm criado

commit 8d51b1123f3ed9cf869d6ce97915a0fa3dc96d4e
Author: Angelo <59621376+accolombinifake@users.noreply.github.com>
Date:   Wed Mar 18 15:12:49 2020 -0300

    Initial commit
```

O que são as branches?

Uma das ferramentas mais
importantes da plataforma
GitHub



Branches - introdução



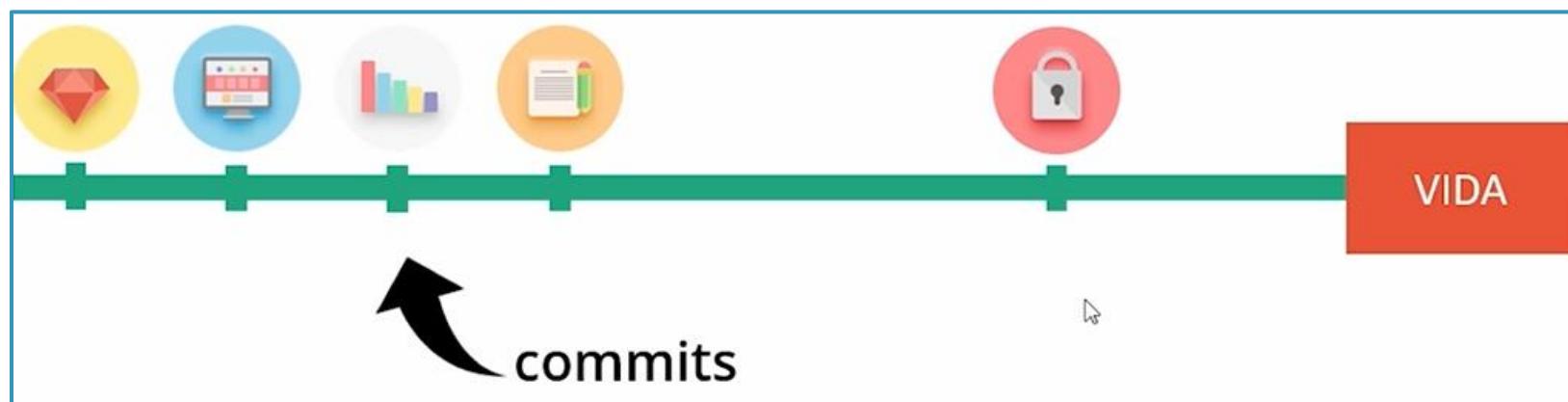
- Para facilitar podemos pensar em Branches como sendo linhas do tempo, isso mesmo no plural, podemos seguir por caminhos diferentes → são as Branches. Até agora, trabalhamos apenas com a nossa **Branch master**
- Em algum momento devemos convergir todas as Branches para a linha do tempo de nosso projeto, sim esse deve ser única
- As Branches são um recurso poderoso que nos ajuda a tentarmos alternativas ou alterações em nosso projeto num ambiente separado (uma branch separada), isso nos permite manter um controle mais assertivo e minimizar os riscos de nosso projeto



Branches - introdução



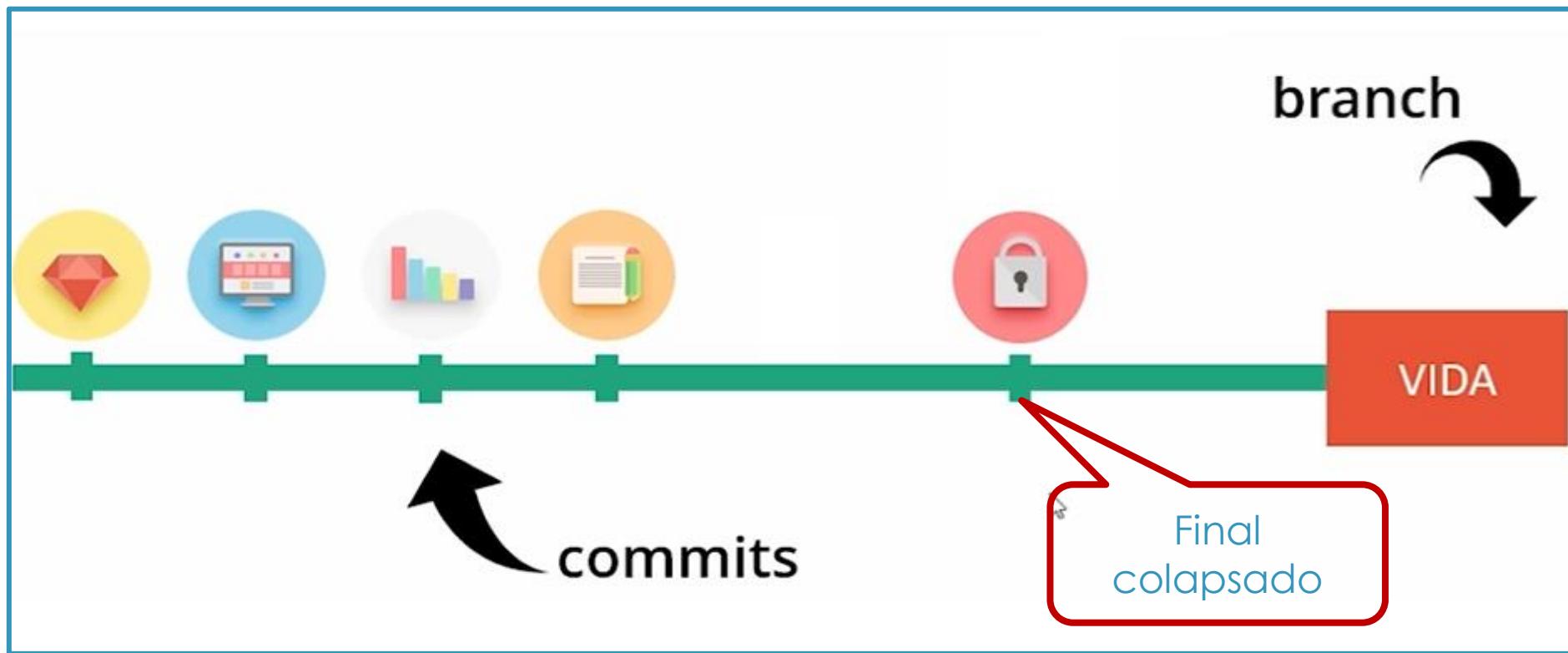
- Observe a linha do tempo a seguir, podemos imaginar que para cada evento nela assinalado temos um commit realizado



Branches - introdução



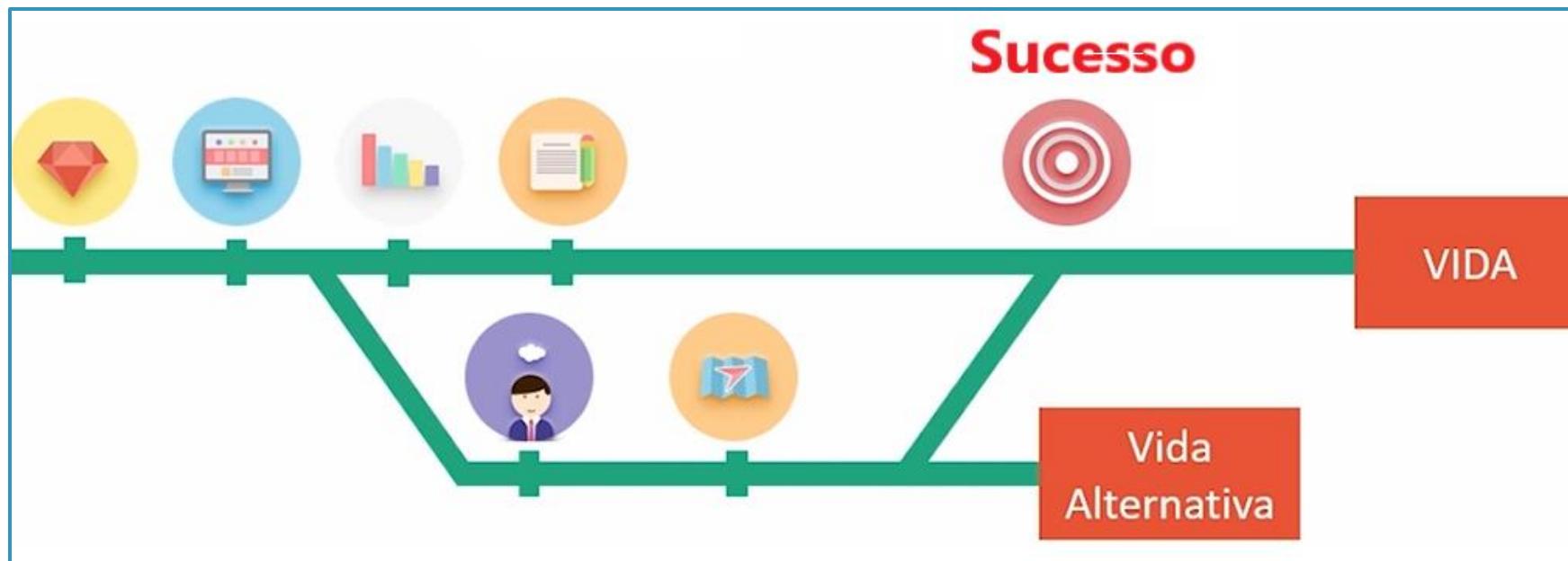
- A **linha do tempo** para o Git recebe o nome de **Branch**



Branches - introdução



- Ao que parece nossa linha do tempo não está convergindo para um bom fim (final colapsado)
- Felizmente para o Git podemos alterar esse final, criando Branches alternativas



Branches - introdução



- Ao realizar os ajustes usando uma Branch alternativa (vida alternativa), o Git permite que você integre todos os commits realizados na Branch alternativa à sua branch master (vida). No final das contas, tudo irá parecer que seu projeto transcorreu de forma linear



Branch master restabelecida e todo histórico (commits) assegurados

Branches - introdução



- Vamos falar um pouco sobre a **Branch Master**, ela nos acompanhou até aqui, mas pouco falamos a seu respeito
- Chegou o momento de conhecê-la um pouco mais de perto



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab (master)
$ []
```

A screenshot of a terminal window showing a command-line interface. The prompt shows the user's name, computer name, and current directory as 'accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projetolab'. To the right of the directory, the text '(master)' is displayed in blue, indicating the current active branch. A large red circle highlights this '(master)' text.

Sim, isso mesmo, é dela que estamos falando

Branches - introdução

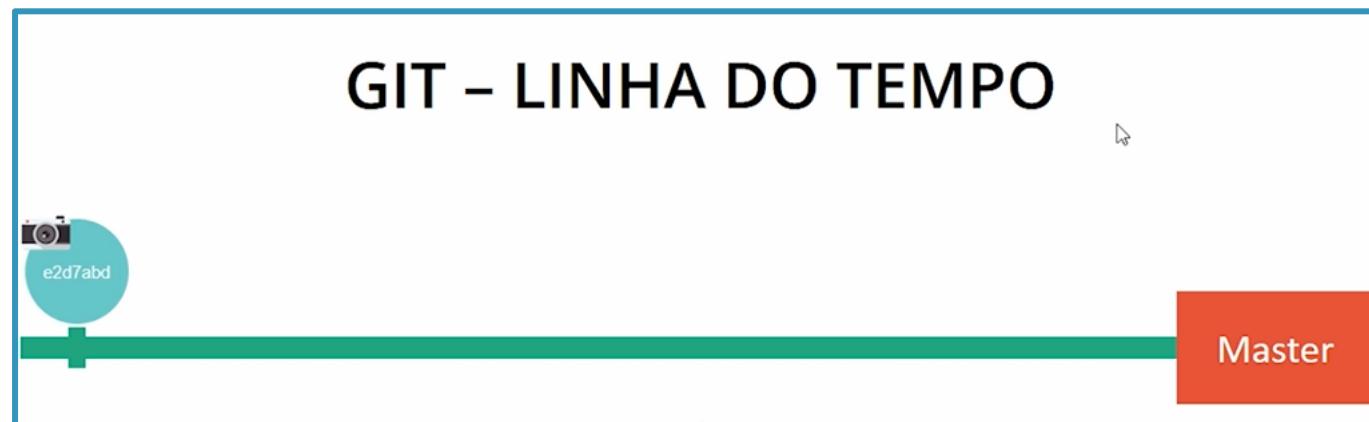


- Como já perceberam sempre que iniciamos nosso Git (ela representa a linha do tempo do nosso projeto) com o comando **git init** é criada a **Branch Master**
- É na Branch Master que todos os nossos commits serão registrados, a menos que, em algum momento você crie uma Branch alternativa
- Cada vez que efetuamos um commit é como se estivéssemos tirando uma foto do status da linha do tempo do nosso projeto naquele instante

Branches - introdução



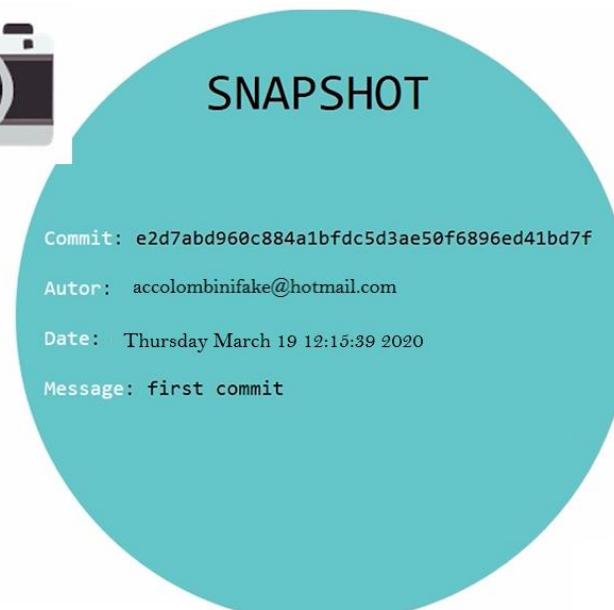
- Para o Git a linha do tempo do nosso projeto é fator crítico de sucesso, podemos navegar por ela revisitando os commits realizados sempre que houver uma demanda
- Ainda é uma forma de assegurar que todos os eventos relevantes (marcos) que geraram commits não se percam



Branches - introdução



- Falamos muito dos commits, mas afinal, o que são esses commits?
- O que são os códigos nele registrados? Olhando um commit por dentro



O nosso Snapshot nos traz:

1. Commit id → e2d7a...
2. Autor → accolombinifake
3. Data → Thursday March 19 ...
4. Message → 'first commit'

Branches - introdução



Nota: A partir deste commit inicial, o Git cria uma árvore que rastreia todos os nossos commits, o que lhe permitirá acesso a qualquer desses ramos no instante em que desejar

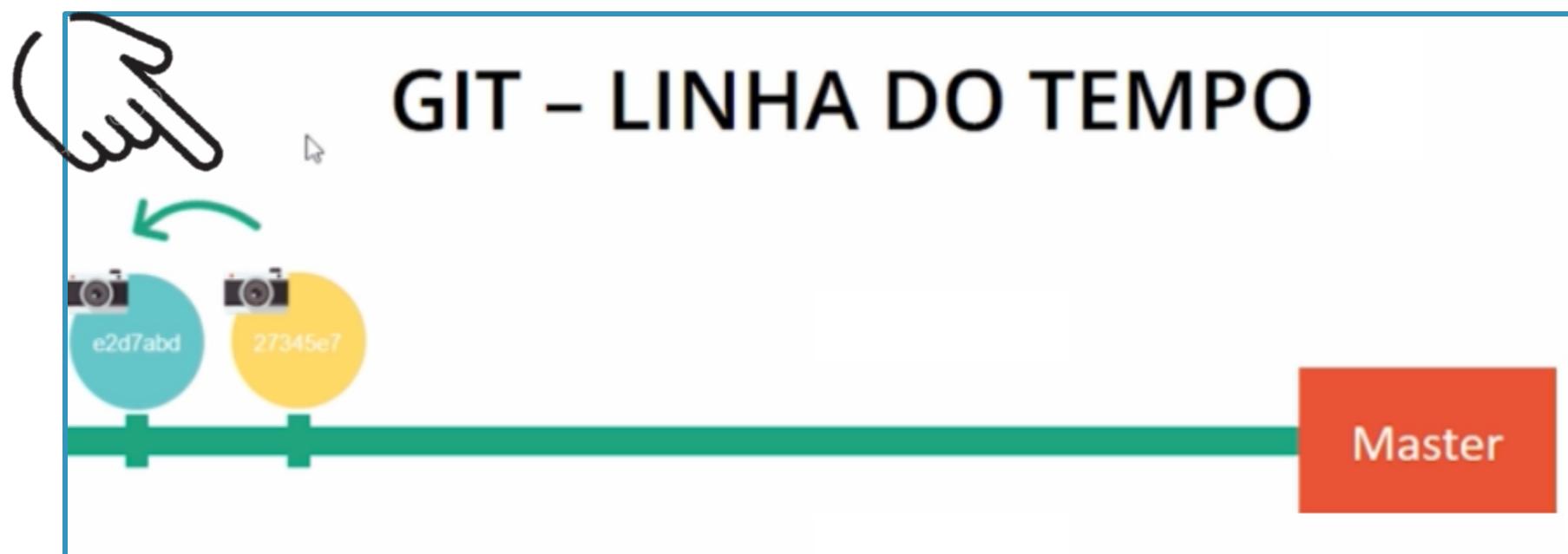
- Podemos executar vários **commits**, sempre que o **Git** adiciona um novo commit ele cria um código de rastreio que nos diz que o commit que estamos criando possui um pai, e aponta para seu antecessor



Branches - introdução



- Observe na figura a seguir que um segundo commit foi gerado. Em seguida vamos abrir esse commit (snapshot) para observá-lo mais de perto



Branches - introdução



- Observe na imagem a seguir que um novo campo surge. Esse campo, **parent**, aponta para o commit pai, ou seja, para o commit imediatamente anterior na árvore de rastreio do Git

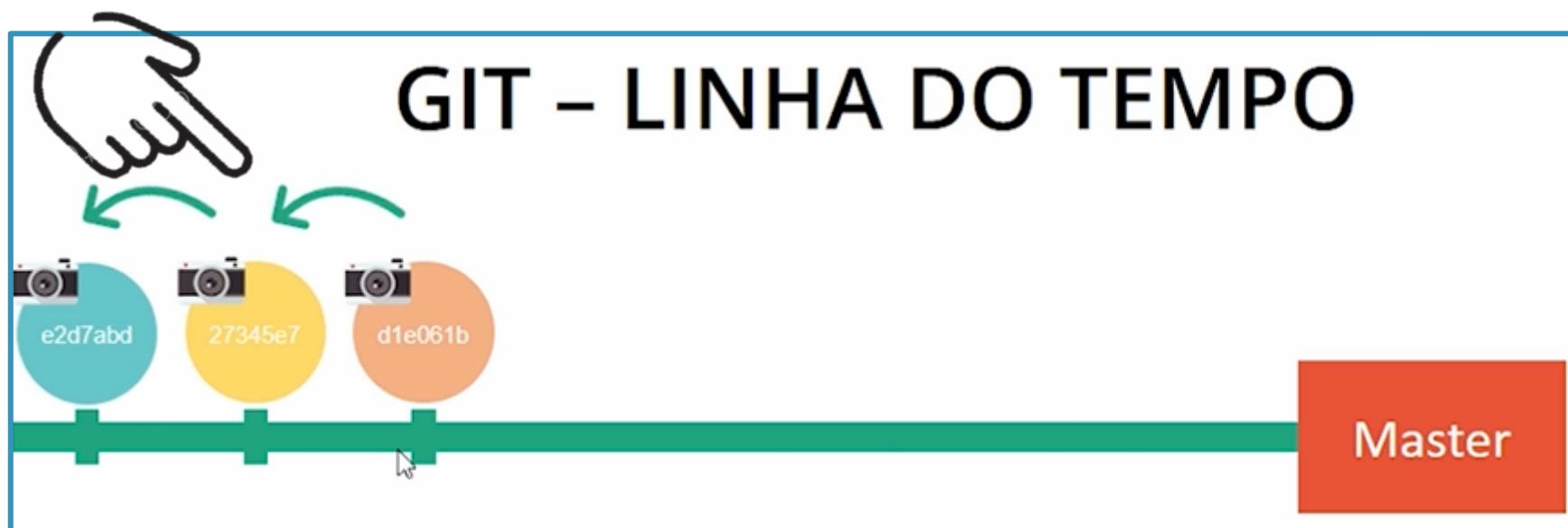


É desta forma que o Git consegue caminhar por seu projeto, rastreando cada etapa registrada em um snapshot

Branches - introdução



- Observe que esse processo se repete, a linha do tempo (nossa Branch Master) é contínua e outros commits serão adicionados, sempre com a mesma dinâmica, ou seja, apontando para o commit anterior (pai)



Branches – FAST FORWARD



- Até aqui trabalhamos apenas com uma Branch, a Branch Master, vamos agora subir um pouco nosso nível de trabalho, vamos criar novas Branches para garantir um controle mais preciso de nosso projeto
- Vamos fazer a seguinte simulação: criamos três commits em nossa Branch Master, e em seguida resolvemos criar uma nova Branch com o propósito de desenvolver novas alterações no projeto sem comprometer nossa Branch Master (imagine um caso em que esteja testando uma nova hipótese, algo que poderá contribuir muito com seu projeto) são muitas as possibilidades, ok

Branches – FAST FORWARD



- Uma grande vantagem em usar esse recurso é o seguinte:
- Imagine um projeto sendo desenvolvido por um time remoto. Manter a Branch Master intacta garante que todos que estão trabalhando remotamente na Branch Master não sejam afetados por um possível teste, ou alteração que não deu certo. Estes testes estão na sua nova Branch, que ainda não foi incorporada na Branch Master
- Em outras palavras, a Branch Master não muda, veja a figura a seguir

Branches – FAST FORWARD



Esta linha não muda!



Criar nova branch neste ponto

Branches – FAST FORWARD



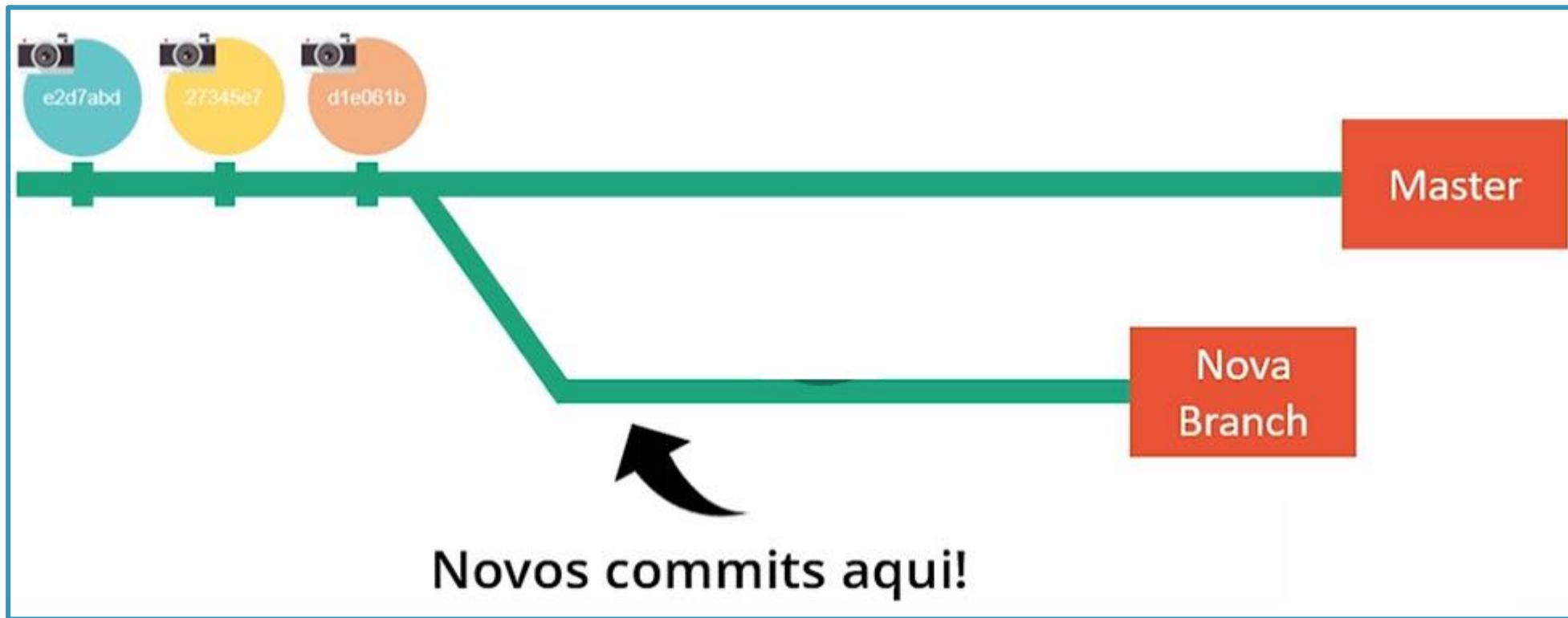
- Como fica nosso projeto então, veja na figura a seguir. Note que você irá trabalhar na nova Branch criando commits tudo como se estivesse na Branch Master.
- Você está apenas resguardando a Branch Master para que ela não receba nenhum commit desnecessário, ou até mesmo, lá na frente você poderá concluir que essas alterações não devam fazer parte de seu projeto, devendo ser até mesmo descartadas



Branches – FAST FORWARD



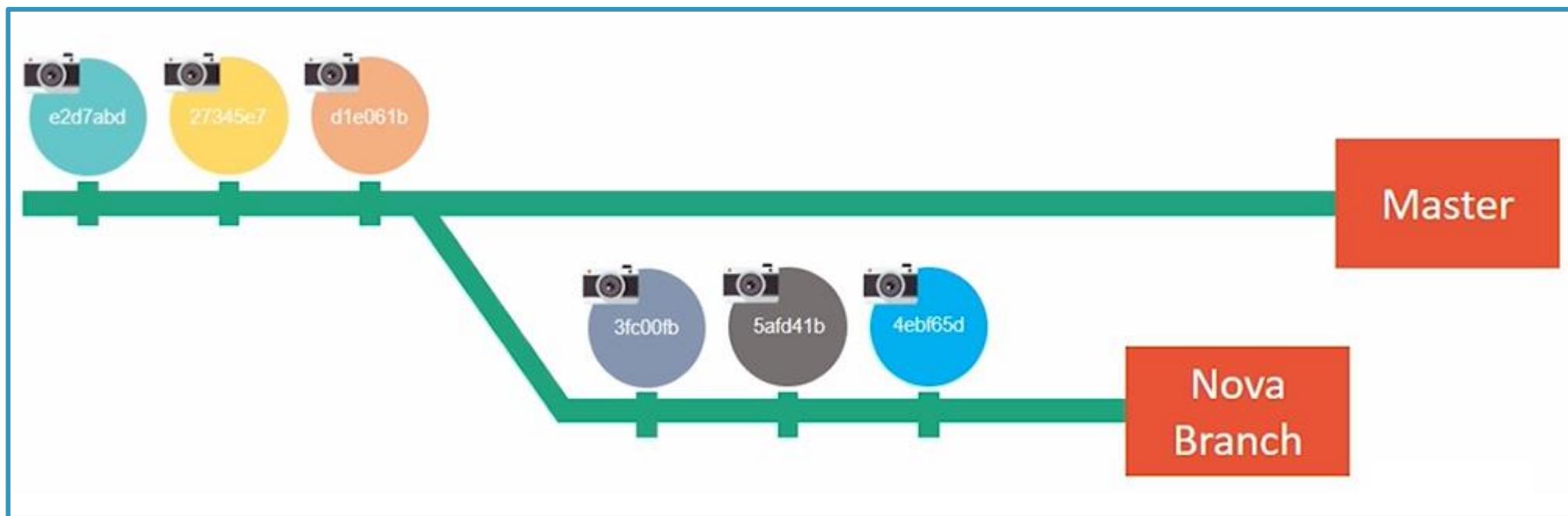
- Observe:



Branches – FAST FORWARD



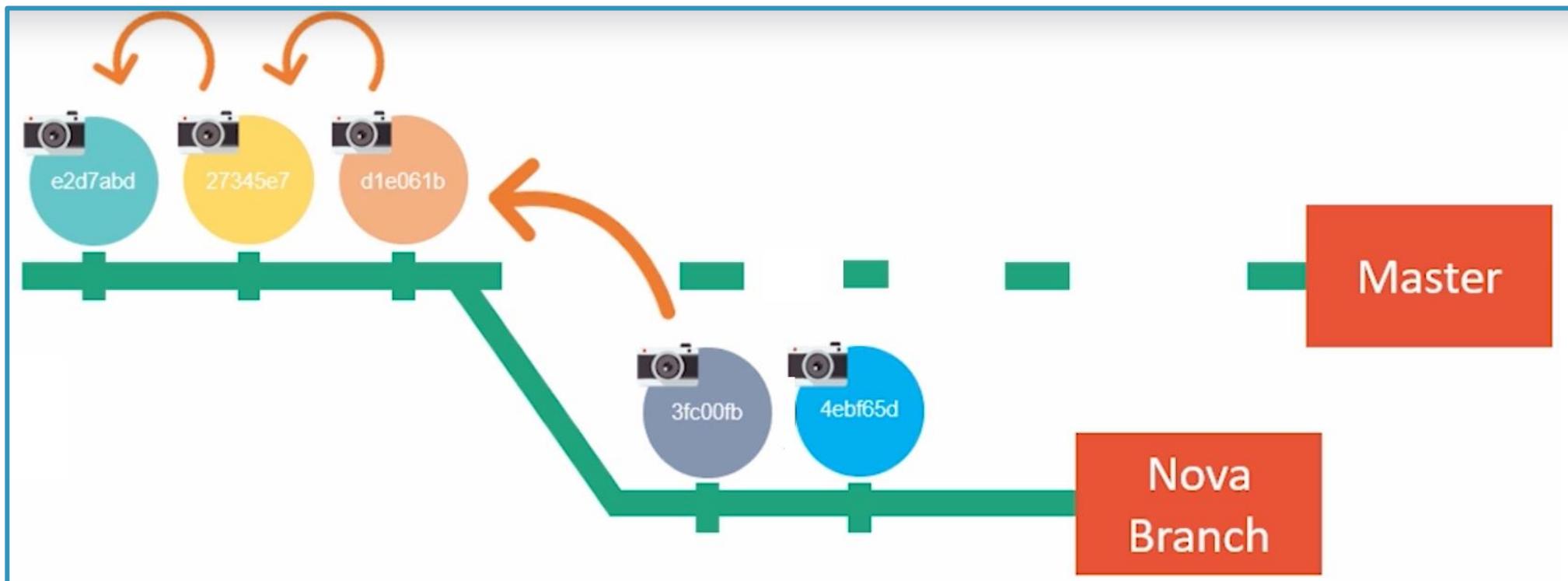
- Novos commits serão adicionados, e a lógica continua a mesma, ou seja, cada commit aponta para o anterior



Branches – FAST FORWARD



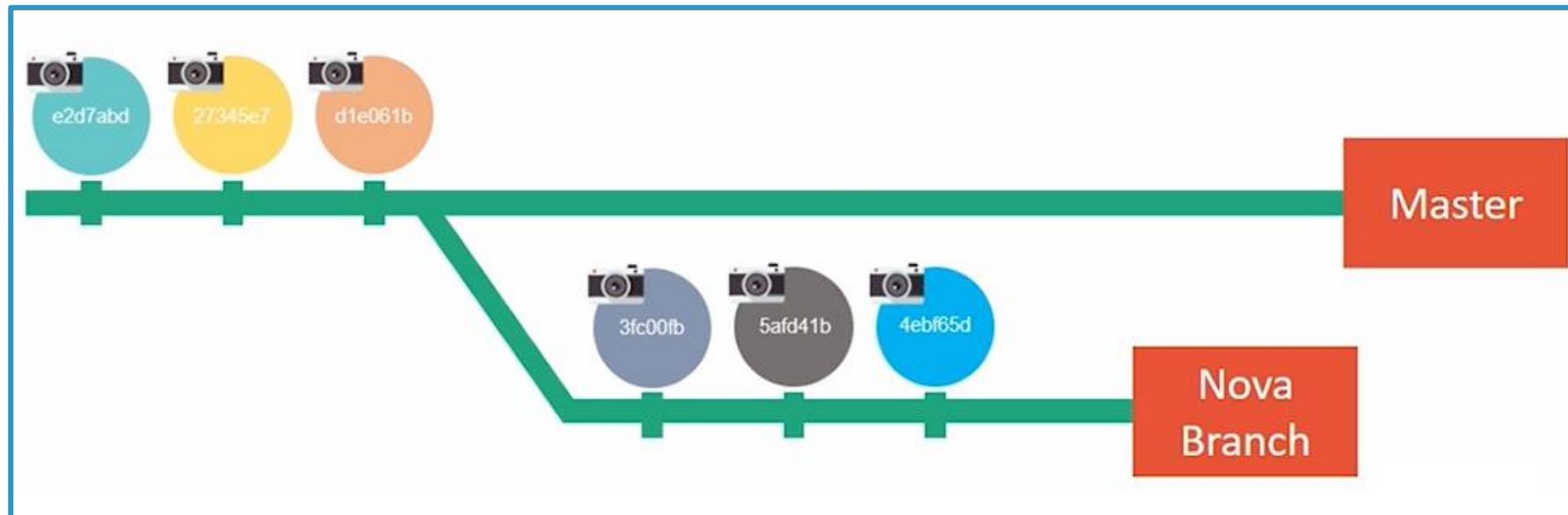
Importante: a nova Branch é composta dos três commits anteriores e os dois commits na Nova Branch, assim, poderemos trabalhar na Nova Branch como se estivéssemos na Branch Master



Branches – FAST FORWARD



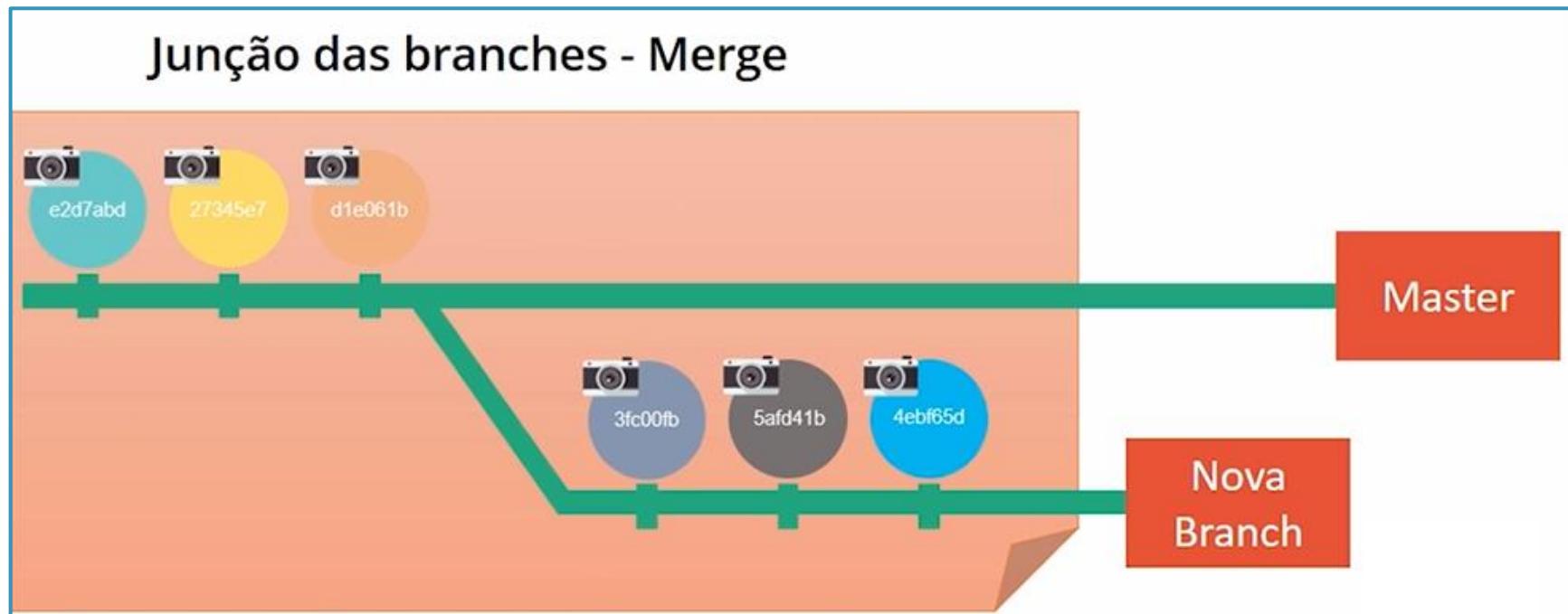
- Vamos avaliar algumas possibilidades:
 - No primeiro caso, vamos imaginar que três commits foram realizados na Branch Master
 - Criamos uma Nova Branch e realizamos novos commits
 - Neste caso, nenhum commit aconteceu na Branch Master enquanto os trabalhos aconteceram na Nova Branch



Branches – FAST FORWARD



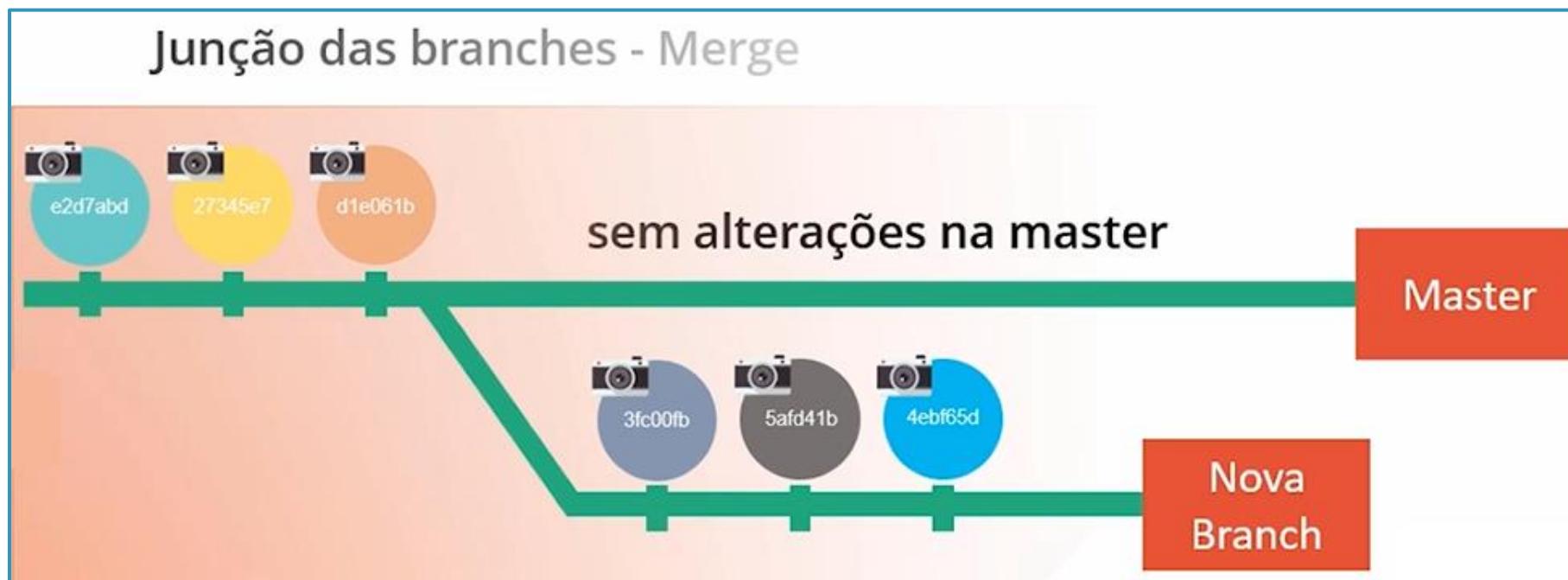
- Imagine que os testes na Nova Branch foram ok e nós queremos agora voltar todo histórico (commits) para a Branch Master, a isso, dá se o nome de Junção das Branches ou Merge



Branches – FAST FORWARD



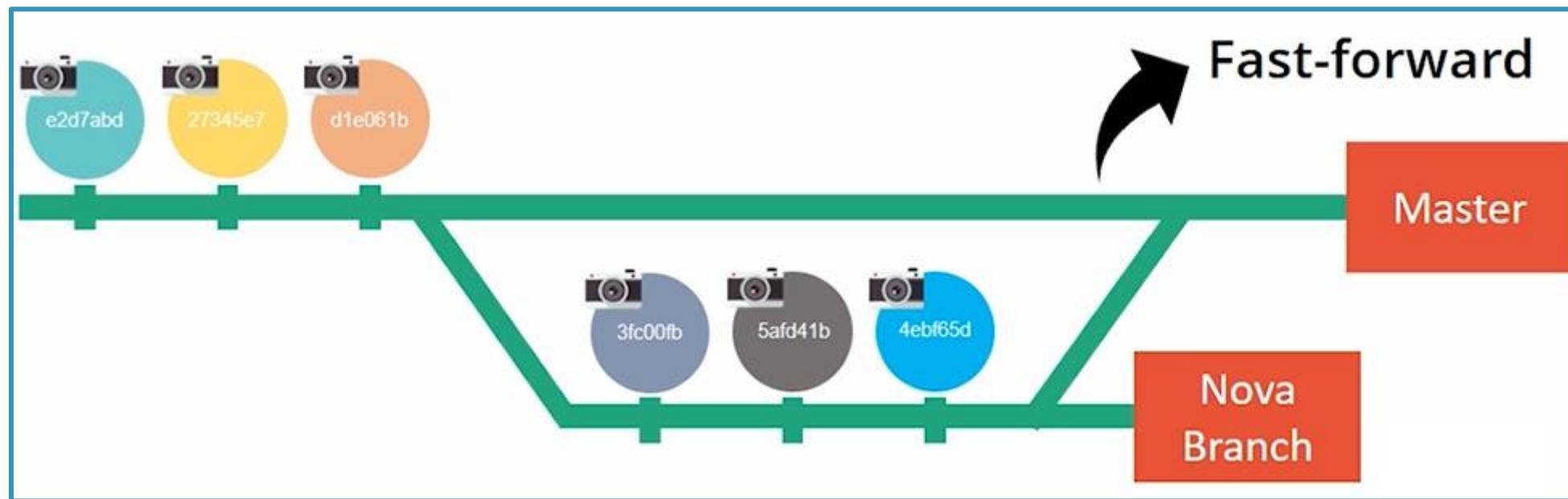
- Em outras palavras, o que queremos é restabelecer a Branch Master, assegurando a plenitude do histórico do seu projeto e neste caso, note que não houveram novos commits na Branch Master



Branches – FAST FORWARD



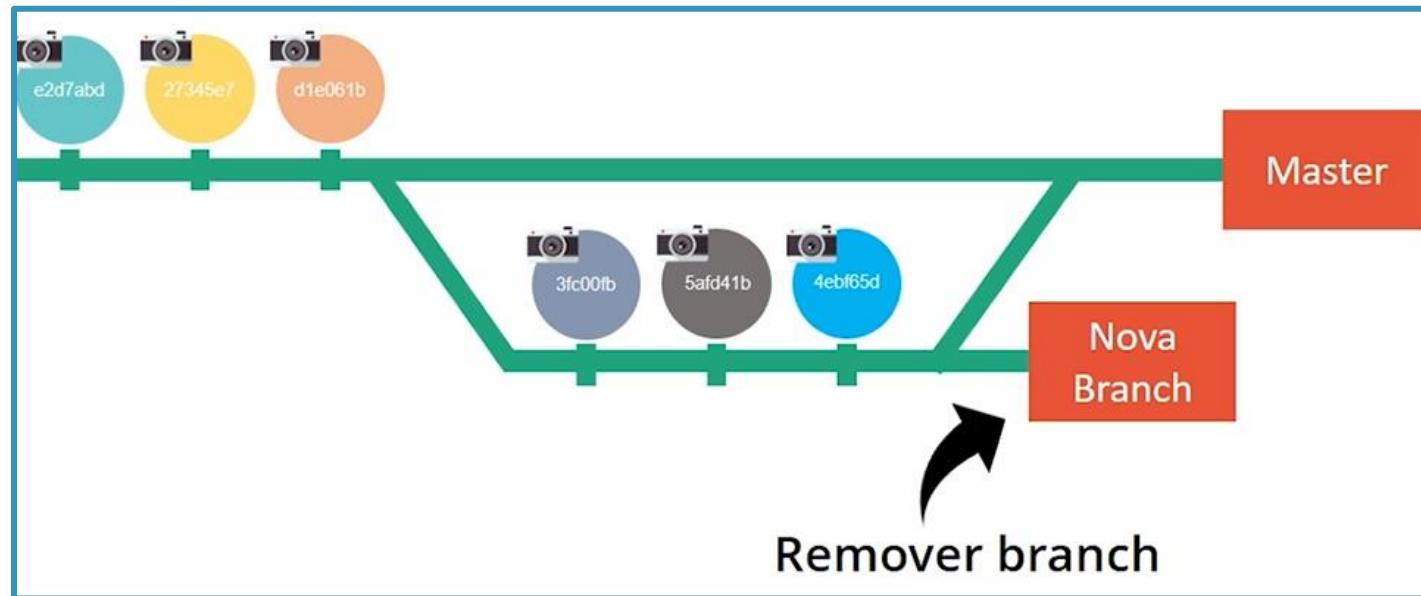
- A este tipo de Merge, dá se o nome de Fast-forward, veja na figura
- Nesta Merge os commits da Nova Branch serão inseridos na linha do tempo da Branch Master na sequência correta



Branches – FAST FORWARD



- Nesta opção de Merge o Git reunirá os novos commits na Branch Master sem qualquer problema, sem precisar fazer maiores alterações, sem lançar mão de qualquer recurso adicional, no caso, a criação de novos commits
- Uma vez feito o Merge, seu histórico de projeto estará restabelecido e você, caso queira (recomendo) poderá excluir a Nova Branch



Branches – FAST FORWARD



- Uma vez feito o Merge teremos a impressão que todos os commits foram realizados na Branch Master, veja a figura, note que a linha do tempo do projeto se mantém intacta



Branches – Recursive Strategy / Conflicts



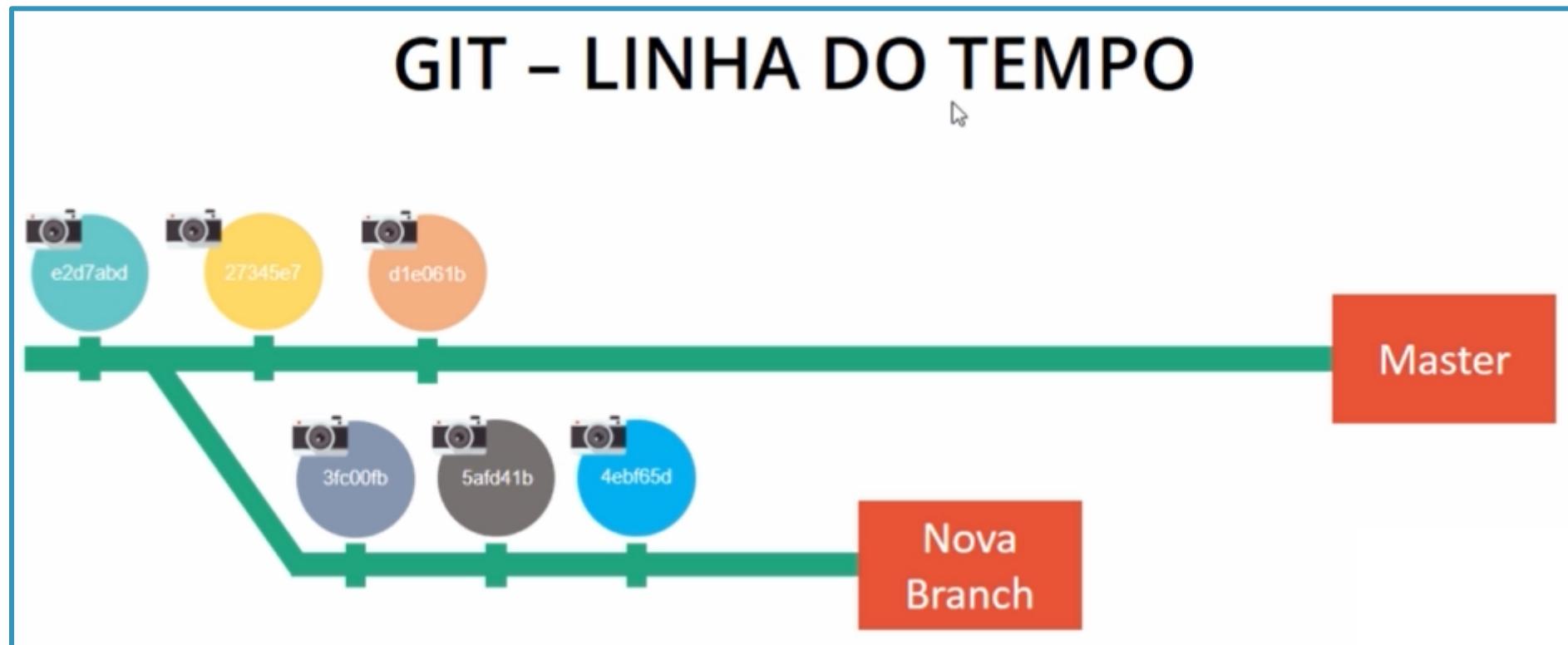
- Imagine um segundo caso de uso de Branches, nós iniciamos nosso projeto fazemos nosso primeiro commit e logo em seguida criamos uma nova Branch



Branches – Recursive Strategy / Conflicts



- O projeto segue e novas Branches são criadas na Branch Master e na Nova Branch



Branches – Recursive Strategy / Conflicts

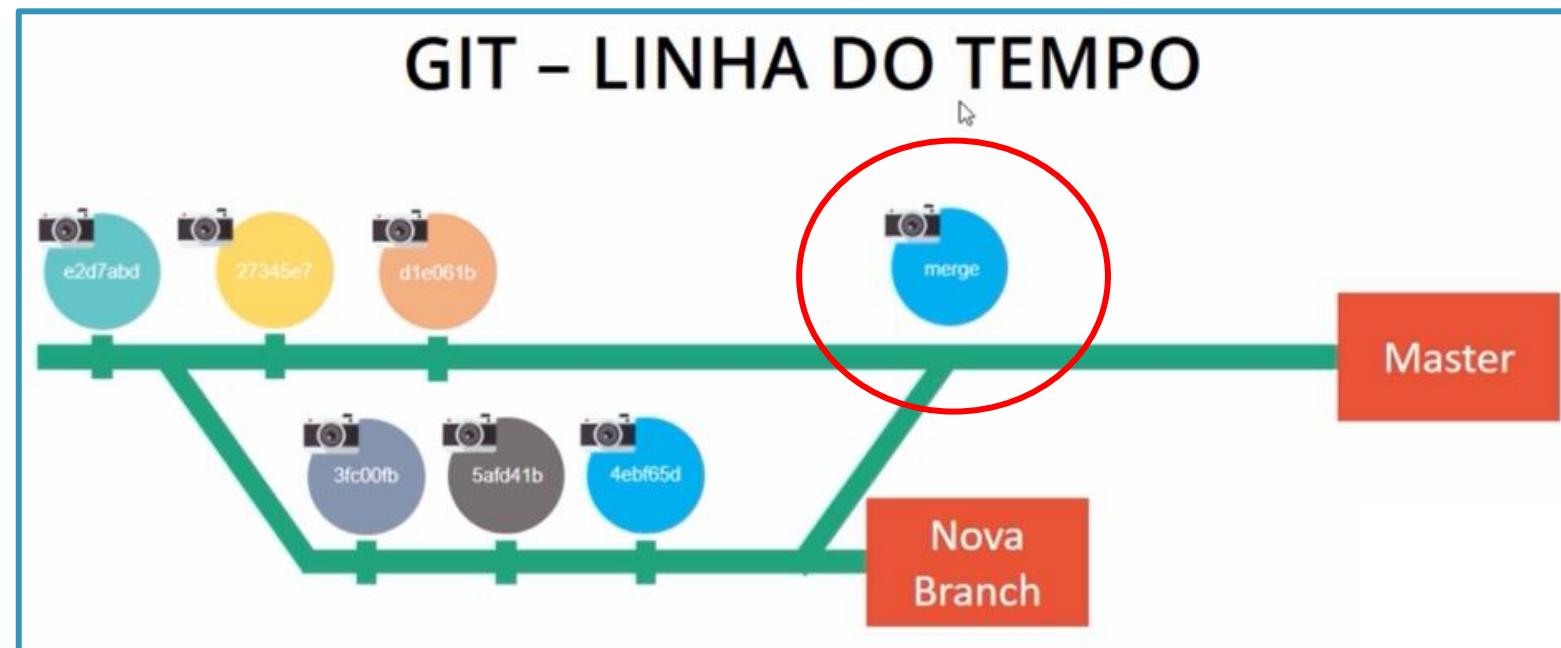


- Essa situação é muito comum, principalmente em ambiente onde o time do projeto trabalha em partes definidas, podemos ter membros da equipe trabalhando em features diferentes (recursos/ferramentas)
- A Branch Master poderá sofrer commits importantes ao mesmo tempo em que as Nova Branch (porque não Novas Branches) também recebem novos commits
- Como a linha do tempo deve ser preservada, ao fazer o Merge destas Nova(s) Branch(es) o Git irá criar um novo commit, commit esse que deverá controlar o histórico do projeto

Branches – Recursive Strategy / Conflicts



- Para o Git resolver essa situação, onde commits foram realizados na Master e na Nova Branch de uma forma bem tranquila
- Para solucionar esse caso, o **Git cria um novo commit**, que recebe o nome de **commit do tipo merge**



Branches – Recursive Strategy / Conflicts

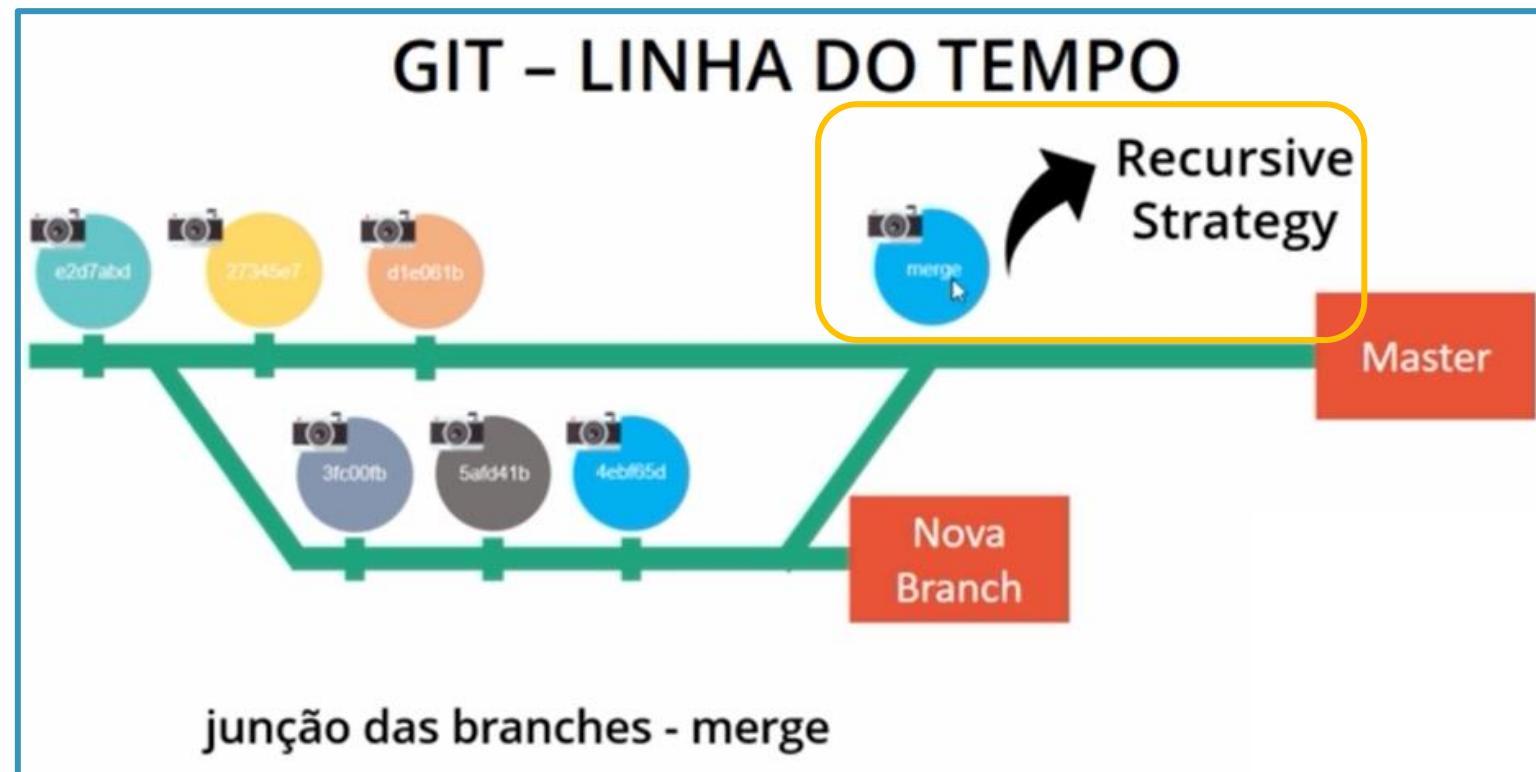


- É no commit do tipo merge que o Git registrará que houveram alterações nas duas linhas de tempo e esse commit do tipo merge passa a ter **dois pais**
- Simples assim, sendo um pai que vem da Nova Branch e o outro da própria Branch Master
- Nesta situação podem acontecer duas coisas, a primeira quando não há conflitos (não há uma sobreposição de dados ou códigos do projeto), neste caso, o Git cria uma junção das Branches – Merge que ele chama de **Recursive Strategy**

Branches – Recursive Strategy / Conflicts



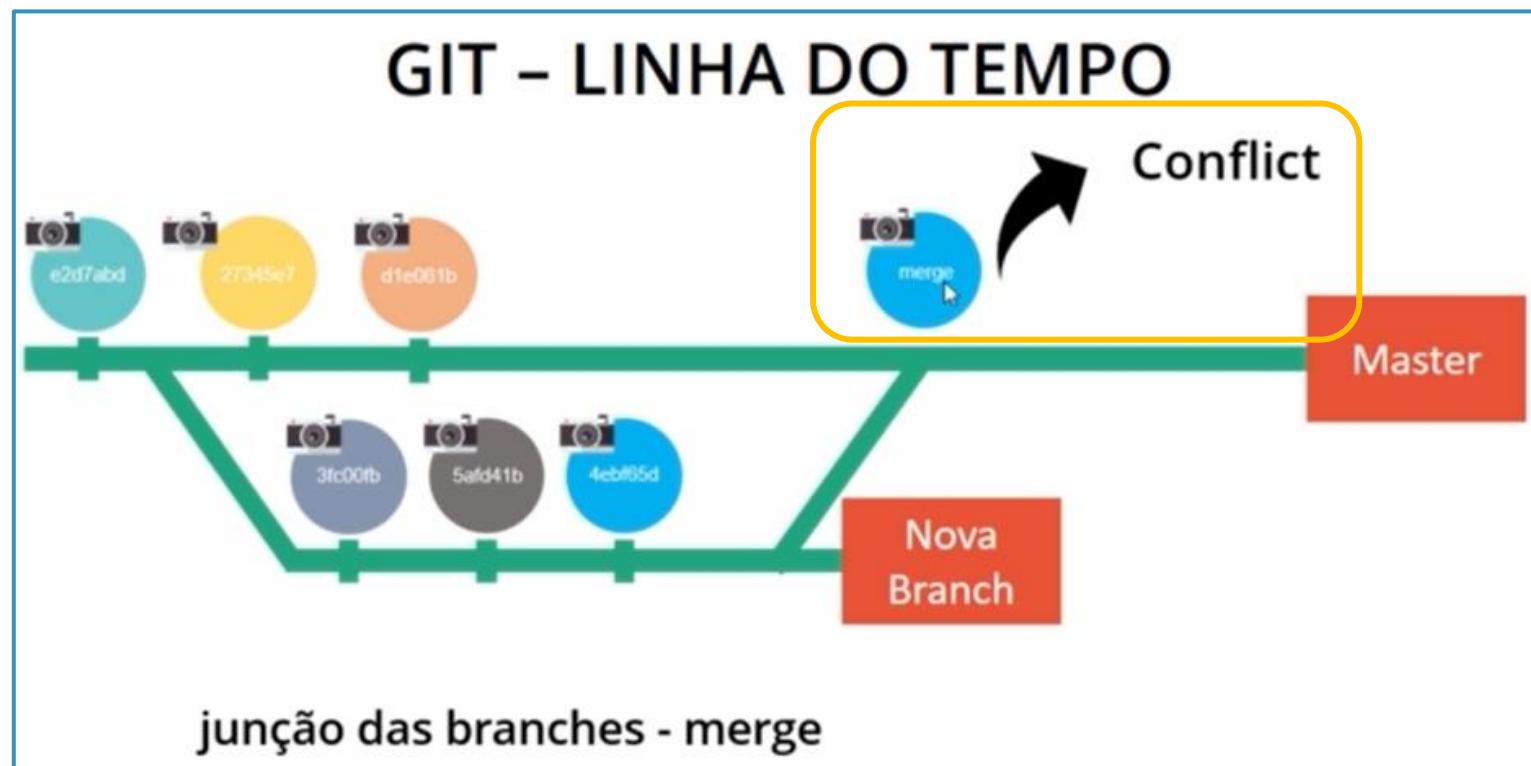
- A figura a seguir ilustra o caso onde tempos um commit do tipo Recursive Strategy



Branches – Recursive Strategy / Conflicts



- Uma outra situação que pode acontecer nestes casos é o Git tentar fazer uma merge e acontecerem alguns conflitos, observe a figura a seguir:



Branches – Recursive Strategy / Conflicts

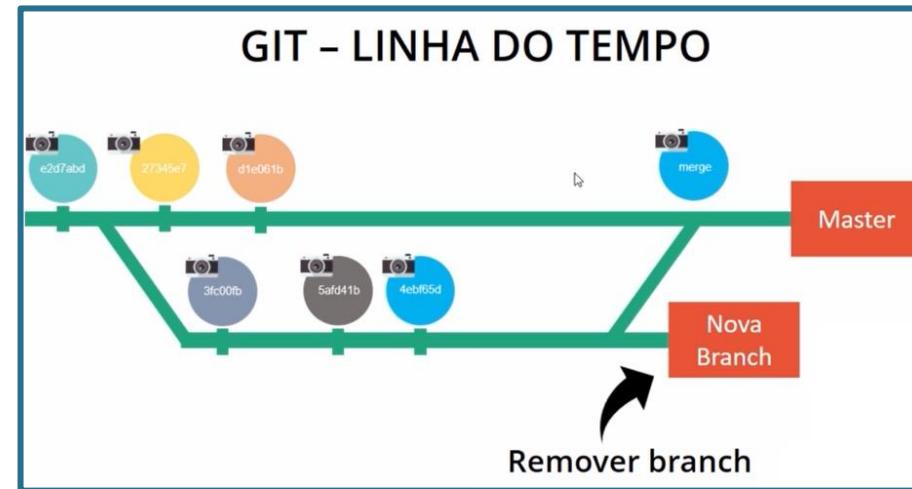


- Estes conflitos vão ocorrer porque provavelmente alguém da Master fez alguma alteração em um arquivo que também está sendo trabalhado na Nova Branch
- Imagine o caso em que você tem um arquivo A e tanto na Nova Branch como na Branch Master este arquivo A está sendo trabalhado nas duas Branches e recebendo commits em ambas as Branches, isso acarretará um conflito. Neste momento, o Git informa que houve um conflito e que você precisará resolver antes de realizar a Merge

Branches – Recursive Strategy / Conflicts



- Quando recebemos a informação do Git, temos a opção de abrir o arquivo de conflitos e decidir qual alteração fica e qual deve ser descartada, feito os ajustes fazemos um novo commit
- Só após a solução do(s) conflito(s) e a realização do novo commit agora sem conflitos irá garantir que o merge possa ser realizado com sucesso e poderemos remover a Nova Branch





Branches – Recursive Strategy / Conflicts

Vamos praticar - Laboratório

Recursive Strategy / Conflicts → Laboratório



- Vamos agora para nosso laboratório e praticar tudo o que aprendemos
- Criação e acesso à Branches
- Para esse laboratório crie uma nova pasta com o nome GIT-BRANCHES (claro é uma sugestão apenas)
- Feito isso, no Visual Studio Code e como o comando **git init** vamos dar início aos trabalhos.
- Estes passos são todos conhecidos de vocês, então ao trabalho



Recursive Strategy / Conflicts → Laboratório



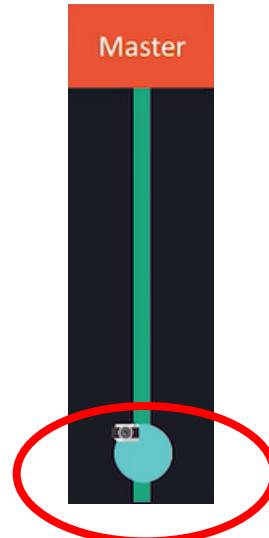
- Feito isso vamos criar um arquivo para dar início aos trabalhos
- Crie um **arquivo txt** dê o nome de **arquivo1.txt** e insira uma linha de texto, salve seu arquivo para começarmos a rastreá-lo e fazer o primeiro commit deste projeto. Você conhece os passos, caso não se recorde volte às aulas anteriores e faça uma breve revisão



Recursive Strategy / Conflicts → Laboratório



- Relembrou, ok então vamos lá:
 1. Adicionando na Stage → **git add .**
 2. Vamos ao primeiro commit → **git commit -m 'commit 1'**
 3. Observe que nossa Master possui um commit que podemos observar através do comando → **git log**



Recursive Strategy / Conflicts → Laboratório



- Veja a seguir a sequência dos trabalhos no Visual Studio Code:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file: arquivo1.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git commit -m 'commit 1'
[master (root-commit) 3a30dd5] commit 1
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo1.txt

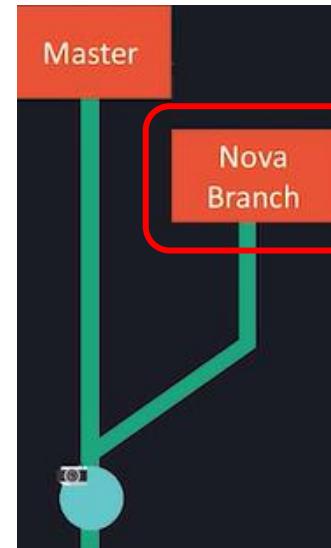
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log
commit 3a30dd54ef2903746090c742de1ce09551061459 (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

  commit 1
```

Recursive Strategy / Conflicts → Laboratório



- Imagine agora que queremos criar uma nova Branch para realizarmos os próximos commits, para isso:
 - Com o comando **git branch <nome-da-branch>** criamos a nova Branch, para manter coerência com os exemplos vamos chama-la de nova-branch
 - A partir daí temos duas Branches, mas ainda nos encontramos na Branch master, observe:



Recursive Strategy / Conflicts → Laboratório



3. Para comprovar nossa tese, vamos usar o comando **git branch**, com este comando o Git nos retorna o status das Branches de nosso projeto, e sinaliza em destaque a branch que estamos trabalhando (asterisco *), observe:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch nova-branch

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch
* master
  nova-branch

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ []
```



Recursive Strategy / Conflicts → Laboratório



4. Nosso próximo passo é acessar a nova branch, para isso, usamos o comando **git checkout <nome-da-branch>**
5. Observe que ao acessar a nova branch seu nome aparecerá no terminal no local antes ocupado pela branch master, observe a seguir
6. Note que, a partir de agora, todas as alterações que eu realizar estarão no contexto da nova-branch, não afetando a Branch Master diretamente

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout nova-branch
Switched to branch 'nova-branch'

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ []
```



Recursive Strategy / Conflicts → Laboratório



7. Para retornar para a Branch Master, basta digitar o comando
git checkout master
8. Dominado estes princípios você está pronto para decidir em qual Branch deseja trabalhar

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch
* master
  nova-branch

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout nova-branch
Switched to branch 'nova-branch'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ git checkout master
Switched to branch 'master'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ []
```

- Vamos praticar agora commits em diferentes Branches
 1. Vamos para a nossa nova branch e nela criaremos um novo arquivo
 2. Vamos chamar este arquivo de **arquivo2.txt**, insira uma linha de texto e salve seu arquivo
 3. **Note que:** este novo arquivo ainda não existe no contexto da branch master e poderá nunca existir, mas sigamos com o nosso laboratório
 4. Na figura a seguir observe a sequência de comandos, note que ao usar o comando **git log** no contexto de nova-branch temos dois commits realizados o que se alinha com a teoria apresentada, onde se diz que os commits são repassados para nova Branch por histórico, confira

Recursive Strategy / Conflicts → Laboratório



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout nova-branch
Switched to branch 'nova-branch'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ git commit -m 'comit2'
[nova-branch 6b4c9ce] comit2
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo2.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ git log
commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3 (HEAD -> nova-branch)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 12:27:09 2020 -0300

    comit2

commit 3a30dd54ef2903746090c742de1ce09551061459 (master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

    commit 1

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ 
```

Recursive Strategy / Conflicts → Laboratório



5. Para comprovar a teoria voltemos à Branch Master e com o comando **git log** poderemos observar que apenas o commit inicial estará no escopo de visão do Git, ou seja, na Branch Master

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ git checkout master
Switched to branch 'master'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log
commit 3a30dd54ef2903746090c742de1ce09551061459 (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

    commit 1

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ 
```

Recursive Strategy / Conflicts → Laboratório



6. Mas não é só isso, se você observar mais de perto, ao voltar para a Branch Master, o **arquivo2.txt** não estará presente na lista de arquivos, vejas as figuras a seguir:

```
OPEN EDITORS
Welcome
arquivo1.txt

GIT-BRANCHES
arquivo1.txt
```

Na Branch Master

```
OPEN EDITORS
Welcome

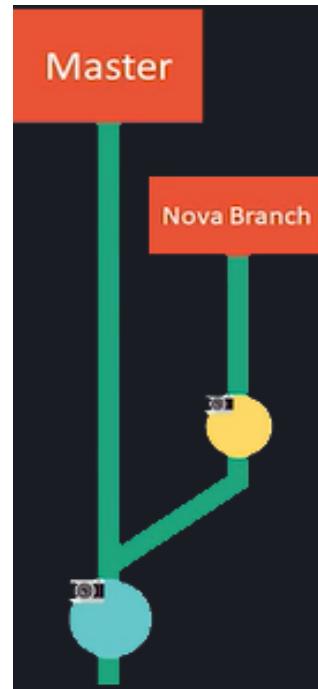
GIT-BRANCHES
arquivo1.txt
arquivo2.txt
```

Na Nova-Branch

Recursive Strategy / Conflicts → Laboratório



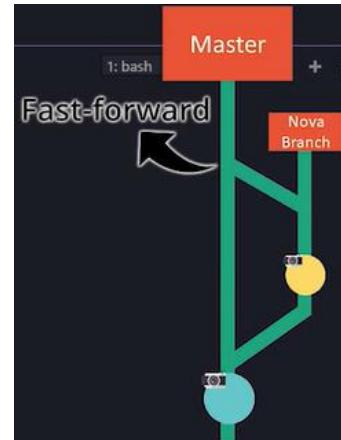
- Observe a seguir como está nosso projeto:



Recursive Strategy / Conflicts → Laboratório



- Continuando com nosso laboratório, vamos agora realizar o Merge, no caso faremos o Fast-Forward Merge, acompanhe:
 - Observe que não há conflitos em nosso projeto, o Git irá identificar isso e realizará um Git Fast-Forward Merge
 - O fluxo será todo direcionado para a Branch Master
 - Para realizarmos essa Merge precisaremos estar dentro de nossa branch Master e usar o comando **git merge <nome da branch>**. Acompanhe na figura a seguir:



Recursive Strategy / Conflicts → Laboratório



```
✓ OPEN EDITORS
  ✘ Welcome
  ✓ GIT-BRANCHES
    arquiv01.txt
    arquiv02.txt
```

Após o Merge →
nossa Branch
Master agora
com os dois
arquivos. Observe
que os dois
commits também
podem ser
visualizados na
Master

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (nova-branch)
$ git checkout master
Switched to branch 'master'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git merge nova-branch
Updating 3a30dd5..6b4c9ce
Fast-forward
  arquiv02.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 arquiv02.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch
* master
  nova-branch

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log
commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3 (HEAD -> master, nova-branch)
Author: accolombini fake <accolombini fake@hotmail.com>
Date:   Sat Mar 21 12:27:09 2020 -0300

  commit2

commit 3a30dd54ef2903746090c742de1ce09551061459
Author: accolombini fake <accolombini fake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

  commit 1
```

Observe a
mensagem
Fast-forward.
Essa junção
ocorre como se
nunca houvesse
existido um
outra Branch

Observe que as
duas Branches
continuam
existindo

Recursive Strategy / Conflicts → Laboratório



- Para melhorar nossa visualização do status do projeto, vamos usar uma nova flag para o comando git log, a flag (--graph), **git log --graph**
- Este comando **graph** nos fornece uma visão gráfica da linha do tempo de commits em formato de mini gráfico, veja a seguir:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log --graph
* commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3 (HEAD -> master, nova-branch)
| Author: accolombinifake <accolombinifake@hotmail.com>
| Date:   Sat Mar 21 12:27:09 2020 -0300
|
|       commit2
|
* commit 3a30dd54ef2903746090c742de1ce09551061459
| Author: accolombinifake <accolombinifake@hotmail.com>
| Date:   Sat Mar 21 11:48:44 2020 -0300
|
|       commit 1
|
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ 
```

Observe que o gráfico nos diz que houve um primeiro commit e em seguida o segundo commit. Não temos a percepção de que houve uma segunda Branch, isto porque nosso Merge foi do tipo Fast-Forward

Recursive Strategy / Conflicts → Laboratório



- Como nossa Branch nova-branch já serviu a seu propósito podemos agora exclui-la
- Para excluir uma Branch usamos o comando **git branch -d <nome-da-branch>**
- Você deverá estar na Branch Master para realizar a exclusão, vamos também usar o comando **git branch** para confirmar a exclusão, veja a seguir:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch -d nova-branch
Deleted branch nova-branch (was 6b4c9ce).

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch
* master

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ []
```

- Vamos agora nesse laboratório simular uma **situação conflituosa** e realizar uma merge do tipo Recursive Strategy Conflicts, pratiquemos:
 1. Nesta situação criaremos uma nova Branch
 2. Faremos alterações na nova branch
 3. E faremos alterações na branch master
 4. Já aprendemos a criar uma nova branch com o comando **git branch <nome-da-branch>**
 5. Vamos agora conhecer uma variação desse comando que nos permitirá criar uma nova branch e já entrarmos na nova branch, o comando é **git checkout -b <nome-da-branch>**

Recursive Strategy / Conflicts → Laboratório



6. Vamos chamar essa nova branch de branch2
7. Observe que estamos no ambiente da branch master e somos imediatamente direcionados para a nova branch, a branch2, veja a seguir:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout -b branch2
Switched to a new branch 'branch2'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ []
```

Recursive Strategy / Conflicts → Laboratório



8. Para descobrir em que branch está, basta usar o comando **git branch**, o **asterisco (*)** assinalará o local de trabalho, é claro que basta você olhar para seu terminal e estará lá o nome da branch de trabalho, mas como estamos num laboratório, sigamos o protocolo

A screenshot of a terminal window on a Windows system (evidenced by the MINGW64 prompt). The user has run the command \$ git branch. The output shows two branches: 'branch2' with an asterisk (*) indicating it is the current branch, and 'master'. A yellow horizontal bar highlights the '(branch2)' part of the output. A dashed yellow rectangle highlights the entire first line of the output, from the prompt to the '(branch2)' part. The terminal window has a dark background and light-colored text.

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git branch
* branch2
  master
```

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$
```

Recursive Strategy / Conflicts → Laboratório



9. Vamos criar um novo arquivo dentro dessa nova branch, vamos chama-lo de arquivo3.txt, digite uma linha de texto, salve, e realize seu “**primeiro commit**” desta branch
10. As aspas servem para lembrarmos de que a nova branch herda os commits da branch master por histórico do projeto, relembrando:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git branch
* branch2
  master

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git commit -m 'commit 3'
[branch2 0ad1bad] commit 3
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo3.txt
```

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git log
commit 0ad1bad59dc07bd09927a54a7352c0dbfca6707 (HEAD -> branch2)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 15:57:32 2020 -0300

    commit 3

commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3 (master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 12:27:09 2020 -0300

    comit2

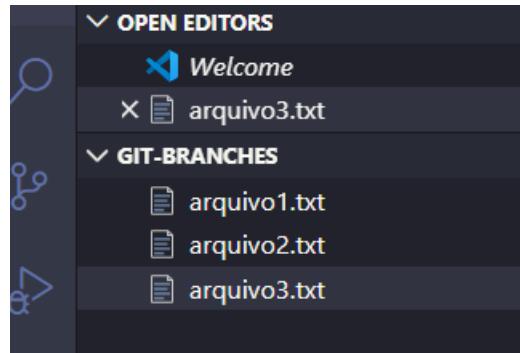
commit 3a30dd54ef2903746090c742de1ce09551061459
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

    commit 1
```

Recursive Strategy / Conflicts → Laboratório



11. Vamos voltar agora para a branch master e realizar algumas alterações, observe que estaremos trabalhando nas duas Branches ao mesmo tempo
12. Fique atento, a branch2 herdou por histórico os arquivos arquivo1.txt, arquivo2.txt, veja a figura a seguir:

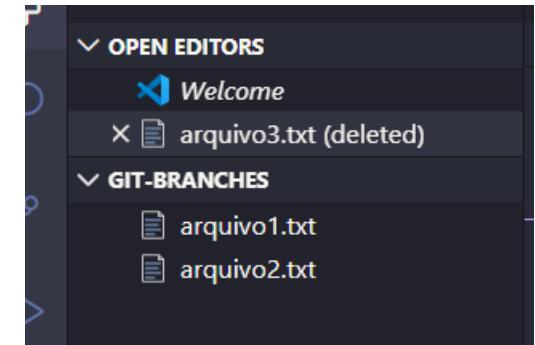


Na branch 2

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git checkout master
Switched to branch 'master'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ 
```

Mudando para
Branch Master



Na branch master

Recursive Strategy / Conflicts → Laboratório



13. Já na Branch Master vamos criar um novo arquivo, o arquivo4.txt e realizar os mesmos passos anteriores.
14. Algo interessante ocorre aqui, veja quando digito o comando **git log**. Note que, o commit 3 não existe para a Branch Master

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  arquivo4.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git commit -m 'commit 4'
[master 2fdb1e9] commit 4
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo4.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log
commit 2fdb1e9d584fa52a3e1fd85821e291b1c4955389 (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 16:11:57 2020 -0300

    commit 4

commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 12:27:09 2020 -0300

    comit2

commit 3a30dd54ef2903746090c742de1ce09551061459
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

    commit 1
```



Recursive Strategy / Conflicts → Laboratório



15. Para confirmar, vamos para a branch2 realizarmos o comando git log e observar o que temos lá:

Para a Branch2 não existe o commit 4 como era de se esperar!

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout branch2
Switched to branch 'branch2'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git log
commit 0ad1bad59dc07bd09927a54a7352cc0dbfca6707 (HEAD -> branch2)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 15:57:32 2020 -0300

    commit 3

commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 12:27:09 2020 -0300

    comit2

commit 3a30dd54ef2903746090c742de1ce09551061459
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

    commit 1
```

Recursive Strategy / Conflicts → Laboratório



16. Vamos agora fazer a nossa Merge, o processo já é nosso conhecido, vamos para a Branch Master e realizamos o Merge
17. Neste caso, o Git nos avisará que realizou um Merge do tipo Recursive Strategy e que foi tudo bem, nenhuma ação extra foi necessária acompanhe:

Observe a mensagem!!

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (branch2)
$ git checkout master
Switched to branch 'master'

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git merge branch2
Merge made by the 'recursive' strategy.
 arquivo3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 arquivo3.txt

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ []
```



Recursive Strategy / Conflicts → Laboratório



18. Podemos observar agora que os quatro arquivos pertencem a nossa branch master, observe com o **git log** e as figuras a seguir

```
✓ OPEN EDITORS
  Welcome
  arquivo3.txt
  X arquivo4.txt
GIT-BRANCHES
  arquivo1.txt
  arquivo2.txt
  arquivo3.txt
  arquivo4.txt
```

Os quatros arquivos
no contexto da
master



```
accol@DESKTOP-TMPD1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log
commit 0ef183bfb49d10a033feb594c1f15a9c61940a56 (HEAD -> master)
Merge: 2fdb1e9 0ad1bad
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 16:47:10 2020 -0300

    Merge branch 'branch2'

commit 2fdb1e9d584fa52a3e1fd85821e291b1c4955389
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 16:11:57 2020 -0300

    commit 4

commit 0ad1bad59dc07bd09927a54a7352cc0dbfcfa6707 (branch2)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 15:57:32 2020 -0300

    commit 3

commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 12:27:09 2020 -0300

    comit2

commit 3a30dd54ef2903746090c742de1ce09551061459
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 11:48:44 2020 -0300

    commit 1
```

Recursive Strategy / Conflicts → Laboratório



19. Observe que além dos 4 commits esperados, um quinto commit pode ser visualizado, este commit é um commit de Merge que tem por propósito controlar de forma mais apurado o histórico de nosso projeto
20. É no commit de Merge que o Git indica que ele realizou um merge de uma branch para a master



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log
commit 0ef183bfb49d10a933feb594c1f15a9c61940a56 (HEAD -> master)
Merge: 2fdb1e9 0ad1bad
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 16:47:10 2020 -0300

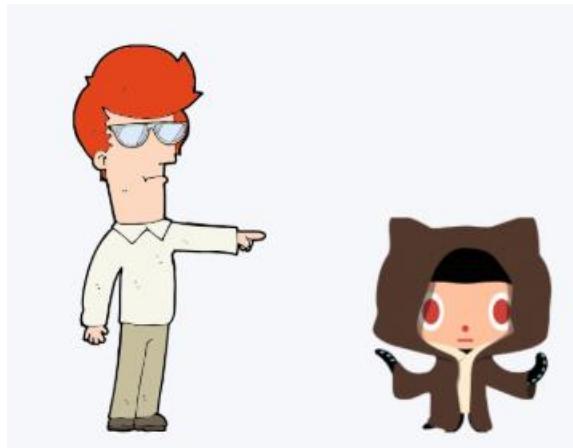
    Merge branch 'branch2'

commit 2fdb1e0d584fa52a3e1fd85821e291b1c4955389
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Sat Mar 21 16:11:57 2020 -0300
```

Recursive Strategy / Conflicts → Laboratório



21. Agora que já estamos mais avançados, vamos dar uma olhada em nosso projeto usando mais uma vez o comando **git log --graph**



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log --graph
* commit ee183bf49d10a933feb594c1f15a9c61940a56 (HEAD -> master)
| \ Merge: 2fdb1e9 0ad1bad
| | Author: accolombinifake <accolombinifake@hotmail.com>
| | Date:  Sat Mar 21 16:47:10 2020 -0300
|
|   Merge branch 'branch2'
|
* commit 0ad1bad39dc07bd09927a54a7352cc0dbfcfa6707 (branch2)
| | Author: accolombinifake <accolombinifake@hotmail.com>
| | Date:  Sat Mar 21 15:57:32 2020 -0300
|
|   commit 3
|
* commit 2fdb1e9d584fa52a3e1fd85821e291b1c4955389
| | Author: accolombinifake <accolombinifake@hotmail.com>
| | Date:  Sat Mar 21 16:11:57 2020 -0300
|
|   commit 4
|
* commit 6b4c9ce1b97c42fff3ba959a3126a47867defca3
| | Author: accolombinifake <accolombinifake@hotmail.com>
| | Date:  Sat Mar 21 12:27:09 2020 -0300
|
|   comit2
|
* commit 3a30dd54ef2903746090c742de1ce09551061459
| | Author: accolombinifake <accolombinifake@hotmail.com>
| | Date:  Sat Mar 21 11:48:44 2020 -0300
|
|   commit 1
```

Observe que o commit de merge possui dois pais, um referente à branch master e o outro referente à branch2

Recursive Strategy / Conflicts → Laboratório



22. Vamos agora realizar o mesmo comando adicionando uma nova flag, esta nova flag diz ao Git, resuma, não me mostre tudo, o comando fica git log --graph --oneline



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log --graph --oneline
* 0ef183b (HEAD -> master) Merge branch 'branch2' <----- dashed line
|\ 
* 0ad1bad (branch2) commit 3
* | 2fdb1e9 commit 4
|/
* 6b4c9ce comit2
* 3a30dd5 commit 1

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ []
```

Observe que o commit de merge na versão reduzida não exibe os pais, fique atento!!!

Recursive Strategy / Conflicts → Laboratório



23. Como tudo deu certo, vamos agora excluir a branch2 com o já conhecido comando **git branch -d branch2**
24. Podemos observar que mesmo após a deleção da branch2 o histórico do projeto permanece intacto
25. E a nossa branch master está preservada e pronta para prosseguir com o projeto, observe na figura a seguir:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch -d branch2
Deleted branch branch2 (was 0ad1bad).

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log --graph --oneline
* 0ef183b (HEAD -> master) Merge branch 'branch2'
|\ 
* 0ad1bad commit 3
* | 2fdb1e9 commit 4
|/
* 6b4c9ce comit2
* 3a30dd5 commit 1

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch
* master

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ []
```

Recursive Strategy / Conflicts → Laboratório



- Continuando com nosso laboratório, vamos para nossa última prática, vamos simular uma situação em que teremos que tratar de conflitos de Branch durante Merge
- Os conflitos vão ocorrer quando fizermos alterações na branch Master que conflitam com alterações que estão acontecendo na ou nas novas Branches
 1. Vamos pra tica? Criemos uma nova Branch chamada de Branch final vamos usar o comando de criação de branch na forma reduzida: **git checkout -b final**

Recursive Strategy / Conflicts → Laboratório



2. Observe os passos a seguir:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git branch
* master

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout -b final
Switched to a new branch 'final'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (final)
$ git branch
* final
  master
```

3. Vamos neste nova branch alterar o arquivo1.txt, insira linha aleatórias
4. Vamos neste nova branch alterar o arquivo2.txt, insira linha aleatórias
5. Vamos adicionar à Stage e realizar o commit ao mesmo tempo

Recursive Strategy / Conflicts → Laboratório



6. Observe os comandos a seguir:

Observe esse commit. Já utilizamos ele, mas vale a revisão

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git checkout -b final
Switched to a new branch 'final'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (final)
$ git branch
* final
  master

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (final)
$ git commit -am 'alterações nos arquivos 1 e 2'
[final db55be3] alterações nos arquivos 1 e 2
  2 files changed, 6 insertions(+), 2 deletions(-)

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (final)
$ git status
On branch final
nothing to commit, working tree clean
```

Recursive Strategy / Conflicts → Laboratório



- Vamos agora retornar à branch master e observar que estas alterações não existem na master, observe:

The screenshot shows the VS Code interface. In the Explorer pane, there are two open editors: 'arquivo1.txt' and 'arquivo2.txt'. The 'GIT-BRANCHES' section lists four branches: 'arquivo1.txt', 'arquivo2.txt', 'arquivo3.txt', and 'arquivo4.txt'. In the Editor pane, 'arquivo1.txt' is open, displaying the text 'Este é o primeiro arquivo'. In the Terminal pane, the command '\$ git checkout master' is being run, followed by the output 'Switched to branch 'master''. The status bar at the bottom indicates the current path: accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (final).

- Observe que o arquivo1.txt na master possui apenas a primeira linha, o mesmo acontece com o arquivo2.txt, confira você mesmo, pratique

Recursive Strategy / Conflicts → Laboratório



9. Vamos agora realizar alterações nestes dois arquivos no contexto da Branch master e também alteremos o arquivo3.txt.
10. Uma vez realizada as alterações aleatórias faça o commit, siga os passos:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (final)
$ git checkout master
Switched to branch 'master'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git commit -am 'alterações nos arquivos 1 2 3'
[master c6dd4fe] alterações nos arquivos 1 2 3
 3 files changed, 9 insertions(+), 3 deletions(-)
```

Recursive Strategy / Conflicts → Laboratório



11. Vamos agora realizar o merge, observe que o **Git não tem como saber qual(is) alterações devem ser mantidas** e **ele irá gerar um conflito**, observe as mensagens e seu próprio terminal define que você está em um situação **master | MERGING**



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git merge final
Auto-merging arquivo2.txt
CONFLICT (content): Merge conflict in arquivo2.txt
Auto-merging arquivo1.txt
CONFLICT (content): Merge conflict in arquivo1.txt
Automatic merge failed; fix conflicts and then commit the result.

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master|MERGING)
$ []
```

12. Observe que as mensagens apontam conflitos nos arquivos1 e arquivos2, o arquivo3, no entanto, não apresenta como esperado, conflito algum

Recursive Strategy / Conflicts → Laboratório



13. Vamos realizar o comando git log e observar os commits que lá estão
14. Você observará que o commit de Merge esperado não foi realizado, significando que algo deu errado e você se encontra num estado de **merging**
15. Para termos certeza vamos realizar o comando git status, observe as mensagens

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

    Unmerged paths:
      (use "git add <file>..." to mark resolution)
        both modified: arquivo1.txt
        both modified: arquivo2.txt

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

Recursive Strategy / Conflicts → Laboratório



16. Para resolver esses problemas basta abrir os arquivos que estão em conflito que o próprio Visual Studio Code lhe oferecerá as opções de solução, veja a seguir:

```
1 Este é o primeiro arquivo
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2 <<<<< HEAD (Current Change)
3 Estou na master e altero o arquivo
4 As linhas alteradas irão conflitar, que ótimo!!!
5 =====
6 Alterando arquivo 1 na branch final
7 Criando mais uma linha no arquivo1
8 >>>>> final (Incoming Change)
9
```

Arquivo1.txt

```
1 Este é o segundo arquivo
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2 <<<<< HEAD (Current Change)
3 Alterando o arquivo2 no contexto da master
4 Isso é muito interessante, o conflito terá que ser avaliado e resolvido.
5 =====
6 Inserindo uma nova linha no arquivo2
7 Estamos agora no contexto da branch final
8 >>>>> final (Incoming Change)
9
```

Arquivo2.txt

17. Basta sua intervenção manual selecionando qual a versão a permanecer, ou se desejar poderá manter as duas.
18. Para saber o que está em conflito, caso esteja, num ambiente diferente do Visual Studio Code, observe o campo **HEAD** e **Final** *são eles que exibem as situações de conflito*
19. Você deverá remover todos os trechos de código que são indicação do Git e em seguida definir o que deseja manter ou alterar nos seus documentos
20. Faça as alterações que julgar necessário e salve os arquivos
21. Uma **outra alternativa** é selecionar uma das opções que o Git lhe oferece, reveja a figura com mais atenção

Recursive Strategy / Conflicts → Laboratório



Menu de opções do Git, teste cada uma delas



```
arquivo1.txt
1 Este é o primeiro arquivo
2 <<<<< HEAD (Current Change)
3 Estou na master e altero o arquivo
4 As linhas alteradas irão conflitar, que ótimo!!!
5 =====
6 Alterando arquivo 1 na branch final
7 Criando mais uma linha no arquivo1
8 >>>>> final (Incoming Change)
9
```

Nota: uma vez que o realizador das mudanças para eliminar conflitos, sua decisão é que irá prevalecer!

22. Tudo feito, precisamos agora realizar os mesmos passos de sempre, adicionar para a Stage e realizar um novo commit, observe que a situação de merging foi removida e o commit realizado com sucesso:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master|MERGING)
$ git commit -am 'merge final'
[master 3e844cf] merge final

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ 
```

Recursive Strategy / Conflicts → Laboratório



- Para avaliarmos os resultados podemos usar o comando **git log --graph --oneline**, observe o resultado:

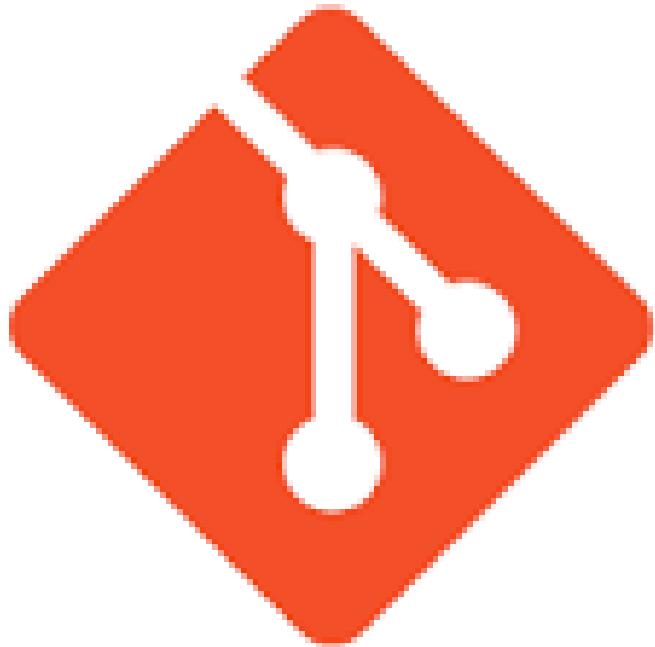
```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/GIT-BRANCHES (master)
$ git log --graph --oneline
* 3e844cf (HEAD -> master) merge final
|\ \
| * db55be3 (final) alterações nos arquivos 1 e 2
| * c6dd4fe alterações nos arquivos 1 2 3
|/
| * 0ef183b Merge branch 'branch2'
|/
| * 0ad1bad commit 3
| * 2fdb1e9 commit 4
|/
* 6b4c9ce comit2
* 3a30dd5 commit 1
```

Brnach Final

Brnach Branch2



- Observe que seu histórico trás seus trabalhos realizados na branch2 e adicionou seu último laboratório onde foram realizadas alterações nos arquivos 1 2 na branch Final e alterações nos arquivos 1 2 3 na branch master



git

Issues e branches remotas

Issues o que fazer com
isso?

GitHub ISSUES



- Trata-se de uma das mais importantes ferramentas do GitHub
- Ela pode ser utilizada por diversos motivos, sendo dois deles os mais relevantes para nosso propósito
 1. Primeiro para fazer o rastreio de Bugs → criamos novas ISSUES no projeto quando queremos informar Bugs no projeto e acompanhar todo processo de solução
 2. A segunda aplicação é para quando queremos sugerir novas Features ou Ferramentas para o projeto

Nota: tanto no primeiro quanto no segundo caso, a partir das ISSUES conseguimos estabelecer um histórico de todas as alterações que realizamos na solução de Bugs ou na criação de uma nova ferramenta para o projeto

GitHub ISSUES



- Através das ISSUES conseguimos:
 - **Histórico de alterações** → de todo processo passo-a-passo. Com isso será possível documentar todos os passos realizados para a solução do(s) Bug(s) e construção da(s) ferramenta(s)
 - **Documentação passo-a-passo** → de todo processo realizado, assegurando com isso, um controle mais efetivo e um aprendizado consistente para todo time envolvido no projeto

Nota: sendo esse um tema importante, vamos partir direto para nosso laboratório, ao longo dos experimentos apresentaremos conceitos relevantes



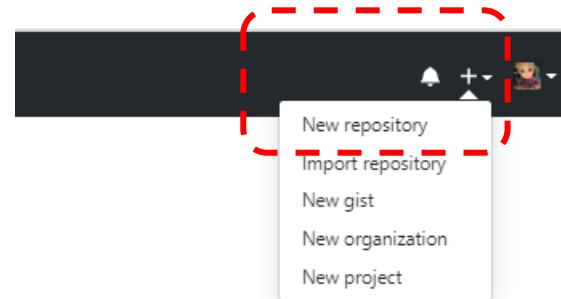
ISSUES – Controlando alterações e sugerindo mudanças

Vamos praticar - Laboratório

GitHub ISSUES



- Vamos para nosso ambiente no GitHub e para essa prática vamos criar um novo Projeto
 - Só para recordar, para isso, vamos em (+) clicamos em adicionar um New repository, veja a seguir:



- Vamos usar como nome do repositório github-master, adicione também o arquivo de README.md e clique em criar novo repositório

GitHub ISSUES



- Observe o processo:

Create a new repository

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

Owner /

Great repository names are short and memorable. Need inspiration? How about [fictional-adventure?](#)

Description (optional)

Public
Anyone 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.

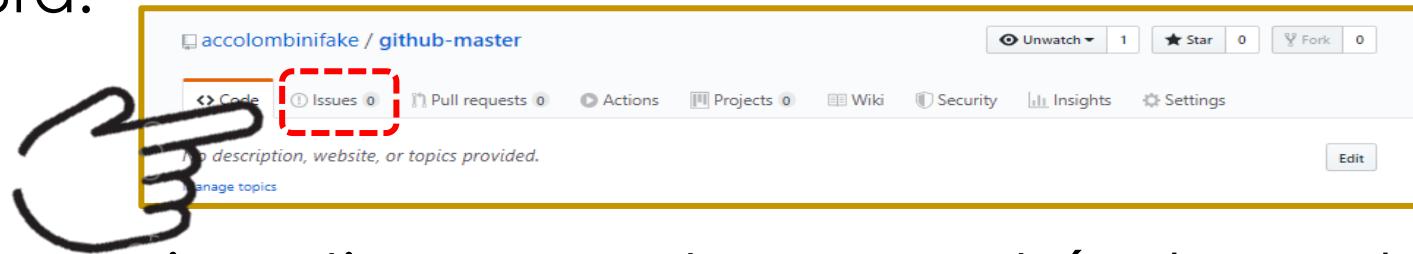
Add .gitignore: | Add a license: ⓘ

Create repository

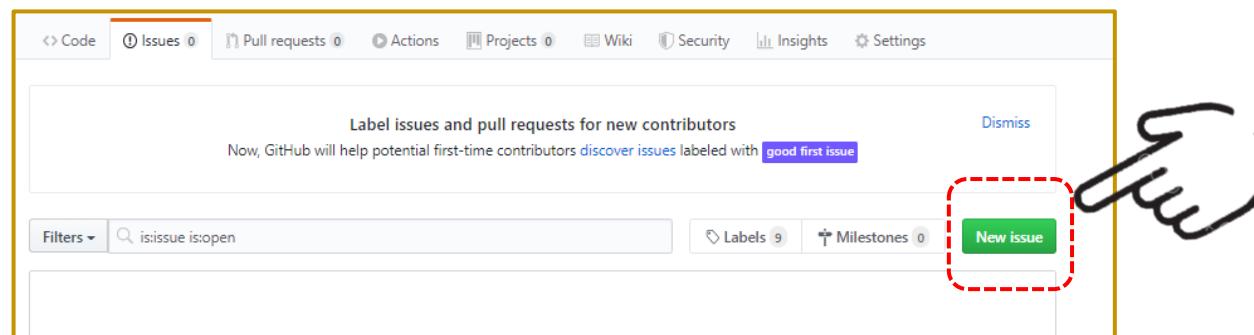
GitHub ISSUES



- Observe que ao criar seu novo repositório ele vem com uma aba chamada **ISSUES** (sinalizando que não há ISSUES até o momento), veja a figura:



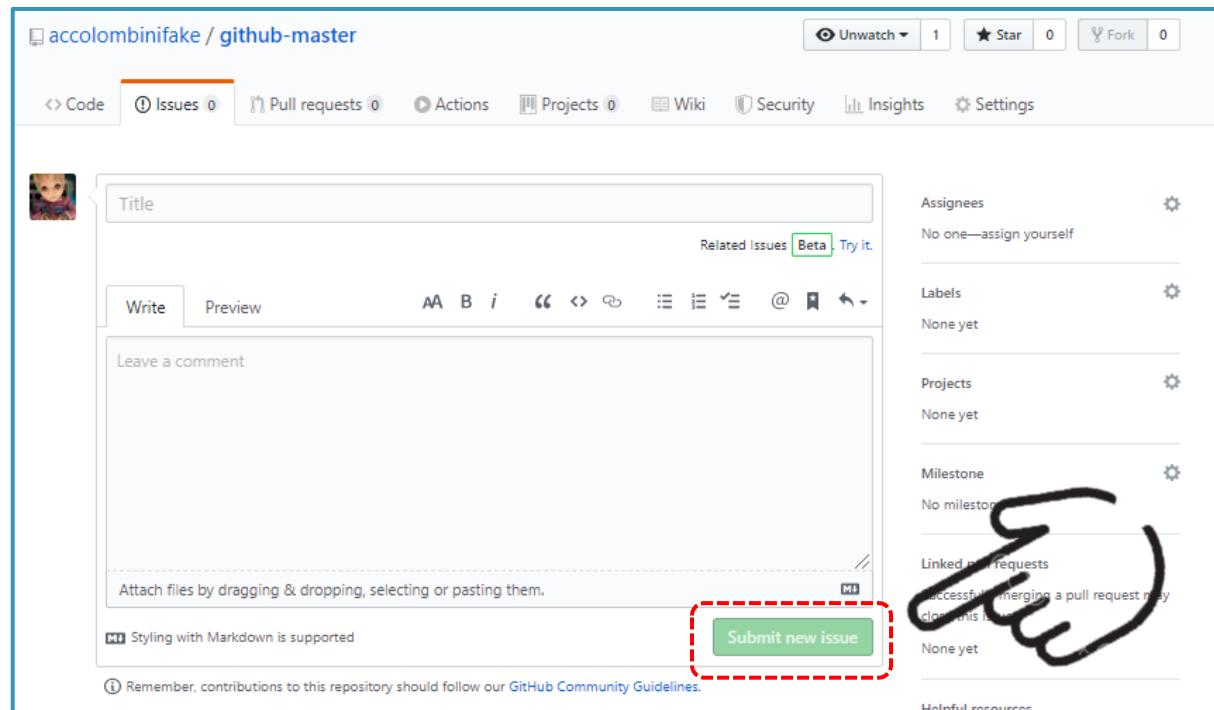
- Vamos clicar e investigar o que temos por trás dessa aba, observe a opção **New issue**



GitHub ISSUES



- Ao clicar em New issue é aberta uma caixa de diálogo solicitando um título para nossa issue e uma descrição
- Após inserir o dados, basta clicar em **Submit new issue**, observe:



GitHub ISSUES



- Para nosso laboratório, vamos simular a criação de um site simples
- Não se preocupe, não vamos desenvolver nada, será apenas a título de exemplo de projeto
- Este site deverá contar três páginas:
 - A página inicial
 - A página de contato
 - A página sobre nós



Nota: não estamos preocupados com conhecimentos de aplicações web, html etc., simplesmente vamos testar Issues

GitHub ISSUES



1. A primeira Issue que vamos criar é o pedido de criação de uma nova ferramenta que é a página inicial do projeto, veja a seguir o exemplo e clique em Submit new issue

A screenshot of the GitHub 'New Issue' form. The title field contains 'Pagina inicial'. The description text area contains the text 'O Projeto precisa de uma página inicial para integração de todos os membros do time.' Below the text area is a file upload placeholder: 'Attach files by dragging & dropping, selecting or pasting them.' At the bottom left is a note: 'Styling with Markdown is supported'. At the bottom right is a green 'Submit new issue' button.

GitHub ISSUES



2. Observe que agora temos uma alteração no nosso Status, em Issues temos assinalado o número 1 e uma descrição da Issue, veja:

Observe o título da Issue e o responsável por sua abertura

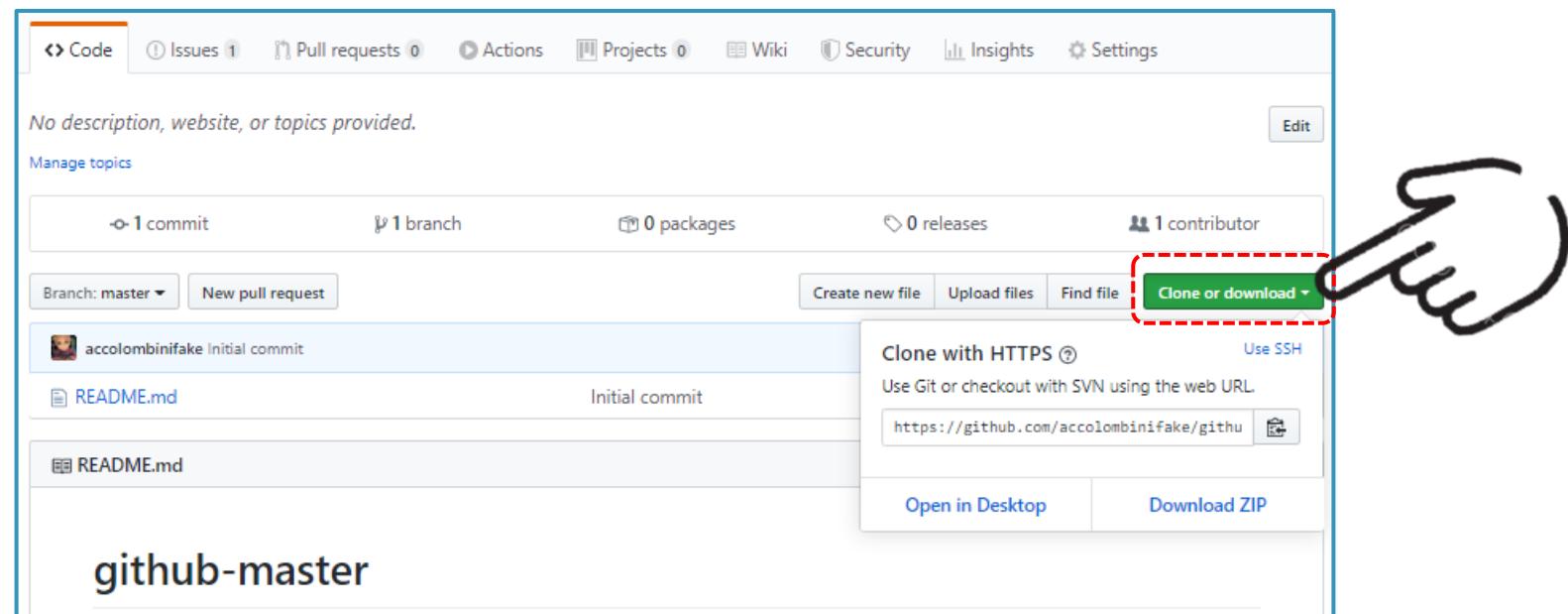
A screenshot of a GitHub repository page for 'acolombinifake / github-master'. The top navigation bar shows 'Issues 1' (highlighted with a red box), 'Pull requests 0', 'Actions', 'Projects 0', 'Wiki', 'Security', 'Insights', and 'Settings'. Below the bar, the repository name 'acolombinifake / github-master' is displayed. A large green button labeled 'New issue' is visible on the right. In the center, there is a card for an issue titled 'Pagina inicial #1' with a status of 'Open'. A note below the title says 'acolombinifake opened this issue now · 0 comments'. There are also 'Edit' and 'New issue' buttons at the bottom of the card.

3. Fique atento, neste caso, o autor do projeto é também o responsável pela criação da Issue, é claro que isto nem sempre será o caso

GitHub ISSUES



4. Feito isso, vamos agora clonar esse repositório para realizarmos nosso trabalho localmente. Para isso, vá para a aba Code e clique em Clone or download, veja a seguir:



GitHub ISSUES



- Vamos agora para o Vistual Studio Code, e na pasta escolhida para o projeto vamos clonar nosso Repositório, faremos isso utilizando o já conhecido comando **git clone <end-do-repositório>** veja a seguir:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/Git (master)
$ git clone https://github.com/accolombinifake/github-master.git
Cloning into 'github-master'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 602 bytes | 2.00 KiB/s, done.

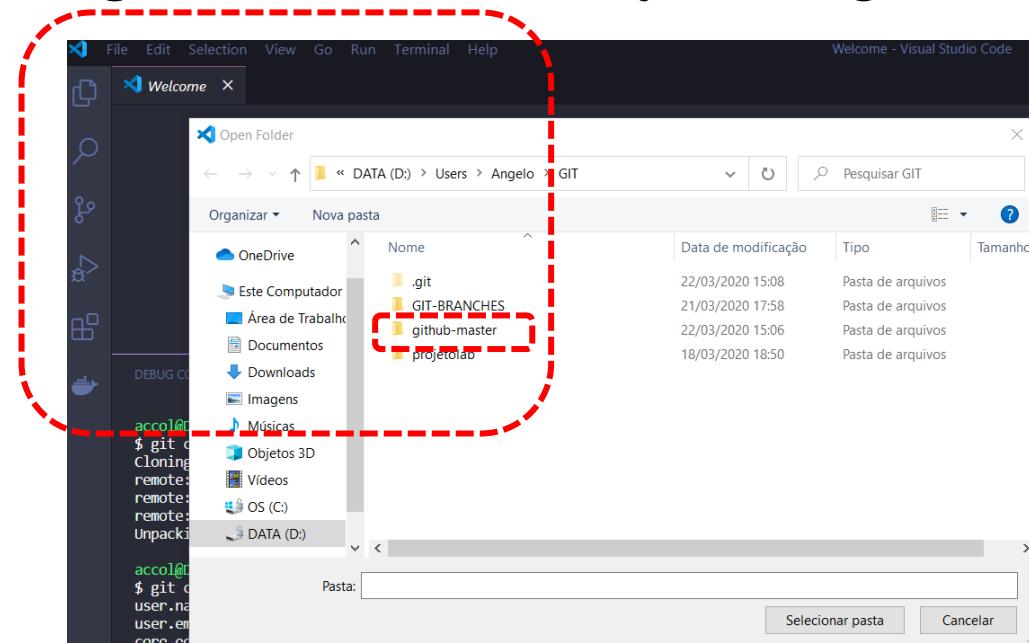
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/Git (master)
$ 
```



GitHub ISSUES



6. Uma vez clonado, vamos agora acessar a nova pasta criada para o projeto
7. Para isso, no Visual Studio Code acesse **File → Open Folder** → e escolha o folder **github-master**, veja a seguir:



GitHub ISSUES



8. Feito isso, ao abrir nosso terminal no Visual Studio Code estaremos no ambiente do Projeto, observe alguns comandos para se situar no Projeto:

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git config --global -l
user.name=acco1ombinifake
user.email=acco1ombinifake@hotmail.com
core.editor=notepad++

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ []
```

GitHub ISSUES



9. Uma vez na pasta do Projeto, vamos criar nosso primeiro arquivo, vamos chamá-lo de **index.html**, e inserir um texto “**Página Inicial**”, salve o arquivo, e faça todos os passos → adicione na Stage, realize o commit e envie para seu repositório no GitHub



```
index.html x
index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <p>Página Inicial</p>
10 </body>
11 </html>
```

GitHub ISSUES



10. Acompanhe:

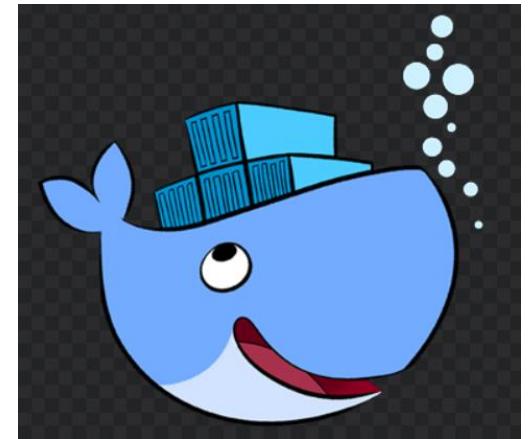


```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git commit -m 'criação da página inicial'
[master 20673cd] criação da página inicial
 1 file changed, 11 insertions(+)
 create mode 100644 index.html

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 463 bytes | 154.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/accolombinifake/github-master.git
 e6d17fb..20673cd master -> master

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ []
```



GitHub ISSUES



11. Voltemos agora para nosso repositório na nuvem, digo no GitHub
12. Se observarmos com cuidado veremos a existência de dois commits, opa dois, não foi um? O primeiro commit foi realizado ao criarmos nosso repositório
13. Além disso, temos uma Issue aberta, lembra dela?
14. Observe também que nosso arquivo html, página inicial já se encontra em nosso repositório remoto, acompanhe na figura a seguir:

GitHub ISSUES



15. Confira seu ambiente, talvez você precise recarregar sua página para que todas as alterações sejam vistas:

The screenshot shows a GitHub repository page for a project named "accolombinifake". A yellow box highlights the main content area. Three red arrows point to specific parts of the interface:

- An arrow points to the "Issues" tab in the top navigation bar.
- An arrow points to the commit history section, showing two commits: "accolombinifake criação da página inicial" (Initial commit) and "criação da página inicial".
- An arrow points to the list of files: "README.md", "index.html", and another "README.md".

The repository statistics shown are: 2 commits, 1 branch, 0 packages, 0 releases, and 1 contributor. The latest commit was made 6 minutes ago.

GitHub ISSUES



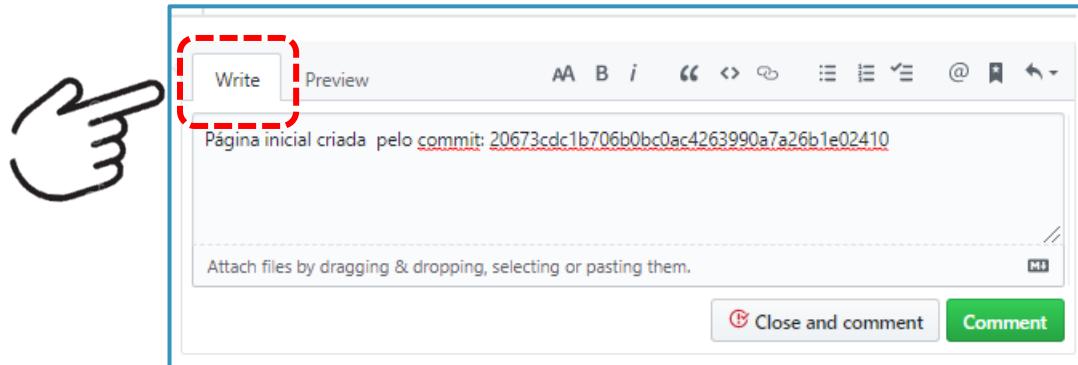
16. Vamos voltar para nossa **Issue**, e finalizá-la, uma vez que nossa página inicial foi criada, é boa prática de Projeto apontar qual o commit que resultou no encerramento da Issue, para isso vá em Code e copie o hash do commit, vejamos:

A screenshot of a GitHub repository page for the user 'accolombinifake' with the repository name 'github-master'. A large yellow box highlights the commit history. A red arrow points from the left towards the 'Code' button at the top left of the commit list. Another red arrow points downwards from the top right towards the commit details for the first commit. The commit list shows two entries: 'criação da página inicial' (commit hash 28673cd) and 'Initial commit' (commit hash e6d17fb). Both commits were made by 'accolombinifake' on March 22, 2020, with the initial commit being 3 hours ago and the other being 1 hour ago. The commit details for '28673cd' are shown in a callout box, including the commit message, author, date, and three action buttons: 'Verified', 'Copy', and 'Compare'.

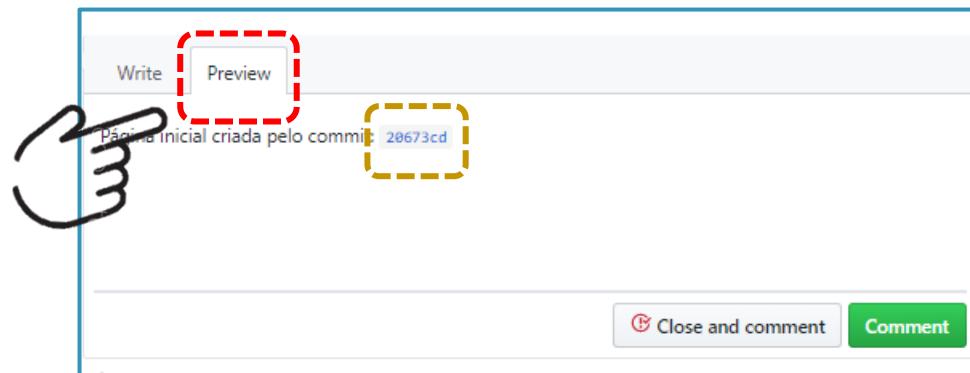
GitHub ISSUES



17. Copiamos o hash do commit e finalizamos nosso comentário



18. Temos a opção de pré-visualização que nos mostra que nosso comentário virou um link que poderá ser visitado sempre que necessário, observe:



GitHub ISSUES



19. Feito nosso comentário podemos abrir o link e analisar o status antes e depois do commit, veja a seguir:

The screenshot shows a GitHub commit history for a file named 'index.html'. The commit message is 'criação da página inicial' and it was made by 'accolumbinfake' 1 hour ago. The commit hash is 'e6d17fb'. The commit details show 11 additions and 0 deletions. The code diff is displayed in a 'Unified' view. Two boxes highlight specific parts of the code:

- A blue box highlights the first few lines of the code: '+ <!DOCTYPE html>', '+ <html lang="en">', '+ <head>', '+ <meta charset="UTF-8">', '+ <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no" />', '+ <title>Document</title>'. This box is labeled 'Estado anterior do projeto, antes do commit'.
- A green box highlights the code after the commit: '+ <head>', '+ <body>', '+ <p>Página Inicial</p>'. This box is labeled 'Estado atual, após executar o commit'.

```
+ <!DOCTYPE html>
+ <html lang="en">
+ <head>
+   <meta charset="UTF-8">
+   <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no" />
+   <title>Document</title>
+
+ <head>
+ <body>
+ <p>Página Inicial</p>
+ </body>
+ </html>
```

20. Dominado estes passos, vamos criar uma página e seguir o mesmo fluxo, crie uma issue para a página de contato. É com você, pratique → **dica** [crie uma nova Branch](#) para realizar os trabalhos sem afetar a Branch Master:

GitHub ISSUES



21. Observe a criação da Issue no ambiente do GitHub e a criação da nova branch no Visual Studio Code, observe:

The screenshot shows a GitHub repository interface. At the top, there are navigation tabs: Code, Issues (with a red box around it), Pull requests, Actions, Projects, Wiki, Security, and Insights. Below the tabs, a list of issues is displayed. The first issue is titled "Criação de página de contatos #2". A green "Open" button is next to the title. Below the title, it says "accolombinifake opened this issue 1 minute ago · 0 comments". A comment from "accolombinifake" is shown, stating "Aguardamos a criação desta página para uma maior integração com as partes interessadas do Projeto." There are "Owner", "Smile", and "More" buttons at the end of the comment.



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git checkout -b pagina-contato
Switched to a new branch 'pagina-contato'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-contato)
$ 
```



GitHub ISSUES



22. Observe que nosso commit tem algo diferente, o **#2**, isso indica para o GitHub que esse commit foi realizado para atender a uma demanda gerada pela **Issue #2**
23. Com a inserção do rótulo #2 no commit, automaticamente o GitHub irá inserir esse commit no histórico de rastreio da Issue #2



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-contato)
$ git commit -m 'criação de página de contato #2'
[pagina-contato 5f99639] criação de página de contato #2
 1 file changed, 11 insertions(+)
 create mode 100644 contato.html
```

GitHub ISSUES



24. Agora vamos realizar o push, observe o resultado a seguir:



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-contato)
$ git push
fatal: The current branch pagina-contato has no upstream branch.
To push the current branch and set the remote as upstream, use
git push --set-upstream origin pagina-contato
```



Estamos na
Branch pagina-
contato

25. Observe que o comando `git push` está configurado para sempre enviar para o **GitHub** a nossa **Branch Master**

26. Para enviarmos uma Branch diferente da Master para o GitHub é preciso antes definir para o push qual a Branch que está sendo enviada, o comando é **git push origin <nome-da-branch>**

Atenção: “origin” é por **default** o nome do nosso repositório no **GitHub**, assim, estamos dizendo ao **push** que queremos enviar para o “origin” a nossa Branch de nome **<nome-da-branch>**

GitHub ISSUES



27. Observe o resultado do **push** a seguir:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-contato)
$ git push origin pagina-contato
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 496 bytes | 496.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'pagina-contato' on GitHub by visiting:
remote:     https://github.com/accolombinifake/github-master/pull/new/pagina-contato
remote:
To https://github.com/accolombinifake/github-master.git
 * [new branch]      pagina-contato -> pagina-contato
```



28. Voltando ao **GitHub** poderemos observar que haverá um commit referenciando a **Issue #2**, observe na figura a seguir:

GitHub ISSUES



29. Observe o commit associado à ISSUE #2

The screenshot shows a GitHub issue page for issue #2. The title of the issue is "Criação de página de contatos #2". A blue arrow points from the title to the commit message. The commit message is "Aguardamos a criação desta página para uma maior integração com as partes interessadas do Projeto." Below the commit message, another commit is listed: "acolombinifake added a commit that referenced this issue 6 minutes ago" with the commit hash "5f99639". The commit message is "criação de página de contato #2". A red dashed box highlights the commit message and the commit itself, while a blue dashed box highlights the issue title and the first comment.

30. Clicando no commit poderemos observar as alterações que foram feitas e identificar que estas **ocorreram em uma Branch diferente da Branch Master**

GitHub ISSUES



31. Observe a presença da nova Branch:

32. Se clicarmos em **Code** poderemos navegar por todas as Branches criadas para o Projeto, observe na figura a seguir:

A screenshot of a GitHub commit page. A hand icon with a yellow outline points to the commit message 'criação de página de contato #2'. A red dashed box highlights this message. Below it, another commit message 'pagina-contato' is shown, also highlighted by a red dashed box. The commit was made by 'accolombinifake' 39 minutes ago. The commit details show 'Showing 1 changed file with 11 additions and 0 deletions.' and a diff view for 'contato.html'.

Clicando no commit, observamos

A screenshot of a GitHub repository's code navigation page. A hand icon with a yellow outline points to the 'Code' tab at the top. A yellow callout bubble says 'Clicando em Code, observamos'. Below the tabs, there is a summary: 'No description, website, or topics provided.', 'Manage topics', '2 commits', '2 branches', and 'Your recently pushed branches: pagina-contato (19 minutes ago)'. The 'Branch: master' dropdown is selected. A red dashed box highlights the 'Branches' tab in the dropdown menu. The dropdown also shows 'Tags' and 'master' (which is checked). Other branches listed are 'pagina-contato' and 'default'. At the bottom, there is an 'Initial commit' and a commit message 'criação da página'.

GitHub ISSUES



33. Alterando para a Branch pagina-contato, teremos acesso ao arquivo contato.html
34. Desta forma conseguimos levar nossas Branches criadas no **repositório local para o GitHub**

A screenshot of a GitHub repository page. A red dashed box highlights the 'Branch: pagina-contato' dropdown menu at the top left. Another red dashed box highlights the file list below, showing 'README.md', 'contato.html', and 'index.html'. The commit history shows three commits: 'Initial commit' (4 hours ago), 'criação de página de contato #2' (1 hour ago), and 'criação da página inicial' (3 hours ago).

File	Commit Message	Time Ago
README.md	Initial commit	4 hours ago
contato.html	criação de página de contato #2	1 hour ago
index.html	criação da página inicial	3 hours ago





ISSUES – Como Ferramenta Colaborativa

Vamos praticar - Laboratório

GitHub ISSUES como Ferramentas Colaborativas



- Até agora usamos as ISSUES como se apenas um único usuário estivesse trabalhando no projeto. A partir daqui, vamos iniciar nossas ações como se seu um time estivesse em atividades
- Em um time podemos ter muitas pessoas trabalhando numa mesma ferramenta ou numa mesma parte do projeto
- Neste cenário multiusuário todos podem gerar Issues e é disso que falaremos neste laboratório

GitHub ISSUES como Ferramentas Colaborativas



1. Para simular esse cenário crie outro usuário no GitHub para que ele possa interagir com seu usuário padrão
2. No meu caso, criei o usuário mfake0323
3. A ideia é que mfake0323 faça pedidos para accolombinifake, vamos acompanhar → pratique

Importante: qualquer pessoa em um projeto público poderá fazer pedidos através das ISSUES, desde que, elas estejam ativadas. Por padrão o GitHub trás as ISSUES ativas, cabe a você desativá-las, caso queira

GitHub ISSUES como Ferramentas Colaborativas



4. Para remover ou melhor desabilitar as ISSUES, basta você ir em Settings e desabilitar a caixa ISSUES, acompanhe:

accolombinifake / github-master

Code Issues 2 Pull requests 0 Actions Projects 0 Wiki Security Insight

Unwatch 1 Star 0 Fork 0

Options Manage access

Branches Webhooks

Notifications

Settings

Repository name: github-master Rename

Template repository: accolombinifake/github-master can be used as a template for creating other repositories.

Vá em settings e role a tela até o campo
Features

Features

Wikis

Restrict editing to collaborators only

Issues

Issues integrates lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.

Get organized with issue templates

Give contributors issue templates that help you cut through the noise and help them push your project forward.

Sponsorships

Sponsorships help your community know how to financially support this repository.

Display a "Sponsor" button

Add links to GitHub Sponsors or third-party methods your repository accepts for financial contributions to your project.

Projects

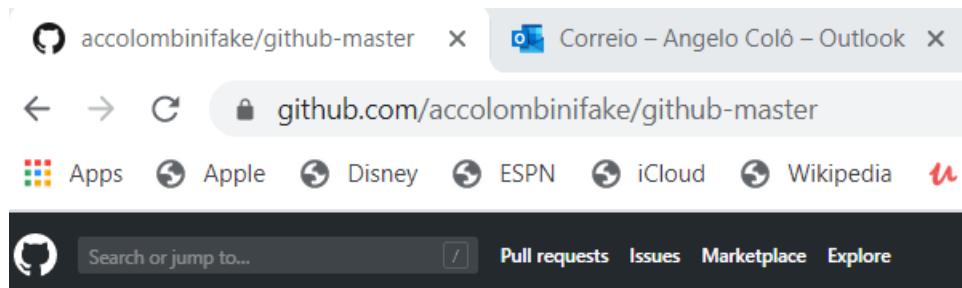
Project boards on GitHub help you organize and prioritize your work. You can create project boards for specific feature work, comprehensive roadmaps, or even release checklists.

Desabilitando a caixa de diálogo ISSUES, elas
não estarão disponíveis

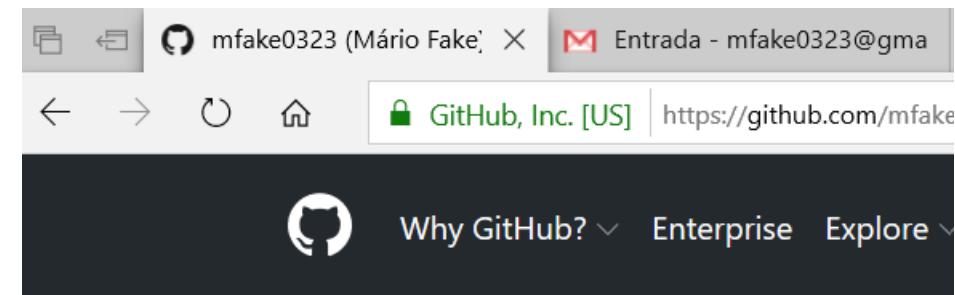
GitHub ISSUES como Ferramentas Colaborativas



5. Para tornar mais realista nosso laboratório, abra dois navegadores, um deles com seu usuário GitHub e e-mail abertos e outro com o usuário criado para testes e o e-mail criado para ele, veja na imagem a seguir:



Usuário accolombinifake e e-mail

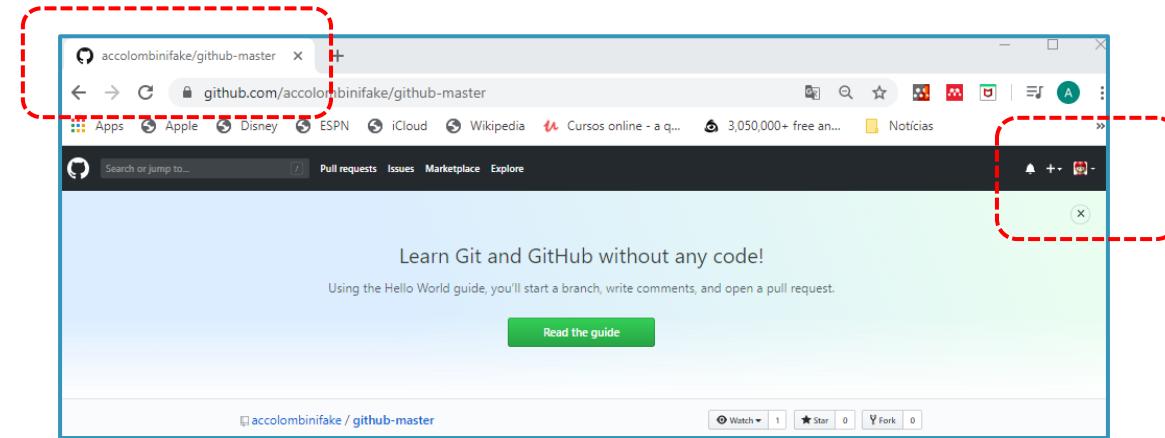


Usuário mfake e e-mail

GitHub ISSUES como Ferramentas Colaborativas



6. Agora vá para seu novo usuário e acesse o seu usuário GitHub (accoleombinifake no meu caso), repositório de trabalho github-master, ficará assim:



7. Com o repositório github-master acesse a guia ISSUES e crie uma ISSUE solicitando a criação de uma página de Sobre o Projeto

GitHub ISSUES como Ferramentas Colaborativas



8. Observe o que acontece quando você clica na guia ISSUE

Seu usuário acessando a ISSUE do repositório github-master

accolombinifake / github-master

Code Issues Pull requests Actions Projects Wiki Security Insights

Title

Related Issues Beta Try it.

Helpful resources GitHub Community Guidelines

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Styling with Markdown is supported

Submit new issue

accolombinifake / github-master

Code Issues Pull requests Actions Projects Wiki Security Insights

Want to contribute to accolombinifake/github-master?

If you have a bug or an idea, browse the open issues before opening a new one. You can also take a look at the [Open Source Guide](#).

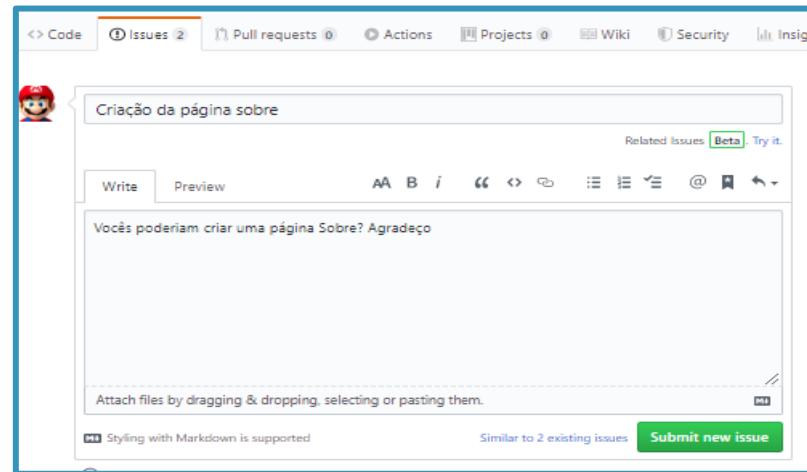
Next

Ao clicar na guia ISSUE você é questionado se deseja contribuir com o projeto de accolombinifake/github-master

GitHub ISSUES como Ferramentas Colaborativas



9. Criando a ISSUE, aqui o processo é o mesmo, siga em frente, feito isso, clique em submeter e sua solicitação através da criação desta ISSUE estará disponível para toda equipe do projeto



10. A parte interessada/responsável pelo projeto poderá ou não atender a demanda de sua ISSUE, normalmente quando se entende que a ISSUE é uma contribuição relevante para o projeto ela é aceita

GitHub ISSUES como Ferramentas Colaborativas



11. A partir da criação de sua ISSUE receberá uma identificação, no caso #3 e ela estará disponível para os desenvolvedores do projeto trabalharem numa resposta para você, observe com cuidado

Identificação #3

Criação da página sobre #3

Open mfake0323 opened this issue now - 0 comments

mfake0323 commented now
Vocês poderiam criar uma página Sobre? Agradeço

Write Preview AA B i Leave a comment Attach files by dragging & dropping, selecting or pasting them.

Close issue Comment

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Linked pull requests
Successfully merging a pull request may close this issue.

GitHub ISSUES como Ferramentas Colaborativas



12. Vamos imaginar que foi encontrado um problema em nossa ISSUE #2 que se encontra na **branch-contato** e queremos resolvê-lo antes de integrá-la à **branch master**. Observe que já estamos com três ISSUES abertas e vamos trabalhar na Issues #2

Vamos corrigir um bug identificado na ISSUE #2

acolombinifake / github-master

Code Issues 3 Pull requests 0 Actions Projects 0 Wiki Security Insights

Filters is:issue is:open Labels 9 Milestones 0 New issue

3 Open 0 Closed Author Label Projects Milestones Assignee Sort

Criação da página sobre #3 opened 8 minutes ago by mfake0323

Criação de página de contatos #2 opened 22 hours ago by accolombinifake

Página inicial #1 opened yesterday by accolombinifake

ProTip! Exclude everything labeled bug with -label:bug.



GitHub ISSUES como Ferramentas Colaborativas



13. Vamos fazer os ajustes, mas antes, vamos anunciar na ISSUE que um problema está sendo tratado, para isso, acessamos a ISSUE e neste caso, escreveremos nossa mensagem no contexto do commit já realizado, com isso, teremos maior transparência no histórico, confiram:

The screenshot shows a GitHub issue page for the repository 'accolombinifake / github-master'. The title of the issue is 'Criação de página de contatos #2'. It is marked as 'Open' and was created by 'accolombinifake' 22 hours ago. There is one comment from 'accolombinifake' stating: 'Aguardamos a criação desta página para uma maior integração com as partes interessadas do Projeto.' A commit has been added to this issue, with the message 'criação de página de contato #2'. The commit ID is SF99639. Below the commit, there is a text input field for leaving a comment and a file attachment area. A red dashed box highlights the commit message and the comment section.



GitHub ISSUES como Ferramentas Colaborativas



14. No contexto do commit poderemos inserir nosso comentário em dois lugares (não é necessário), mas para fins didáticos faremos os dois.
15. No campo Write e no próprio commit, clicando e adicionando um comentário, observe:

Pressiona o símbolo (+) e uma caixa de comentários se abre

The screenshot shows a GitHub commit interface for a file named 'contato.html'. The commit message is: 'Trocando Página de contato para Contato do site'. Two comments are highlighted with red dashed boxes. The first comment is from 'accolombinifake' and says: 'Aqui deveria ser Contato do site'. The second comment is also from 'accolombinifake' and says: 'Trocando Página de contato para Contato do site'. A large black hand icon points to the '+' symbol in the commit message area.



GitHub ISSUES como Ferramentas Colaborativas



16. Se voltarmos para nossas ISSUES e entrando na ISSUE #2 veremos um novo símbolo (observe na figura), esse símbolo significa que há uma discussão aberta sobre essa ISSUE. observe:

Criação de página de contatos #2

Open accolombinifake opened this issue 23 hours ago · 0 comments

accolombinifake commented 23 hours ago

Aguardamos a criação desta página para uma maior integração com as partes interessadas do Projeto.

accolombinifake added a commit that referenced this issue 23 hours ago

criação de página de contato #2



GitHub ISSUES como Ferramentas Colaborativas



17. Nesta sequencia de comandos quero chamar a atenção para três fatos:
- Observe que estamos trabalhando na branch pagina-contato
 - Observe o commit → inserimos a cláusula closes #2, esta cláusula encerra uma ISSUE, no caso a ISSUE #2
 - Por fim, voltamos à branch master, fazemos o merge, e realizamos o push para o GitHub de dentro da master, estamos desta forma atualizando a branch master do GitHub

```
1 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git checkout pagina-contato
Switched to branch 'pagina-contato'

2 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-contato)
$ git commit -m 'alteração da página de contato closes #2'
[pagina-contato c29dd34] alteração da página de contato closes #2
 1 file changed, 1 insertion(+), 1 deletion(-)

3 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-contato)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git merge pagina-contato
Updating 5f99639..c29dd34
Fast-forward
  contato.html | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 398 bytes | 398.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/accolombinifake/github-master.git
  5f99639..c29dd34 master -> master
```

GitHub ISSUES como Ferramentas Colaborativas



18. Só para registrar, vamos ao GitHub conferir se nossa ISSUE foi encerrada, observe:

acolombinifake / github-master

Issues 1 Pull requests 0 Actions Projects 0 Wiki Security Insights

Criação de página de contatos #2

Closed accolombinifake opened this issue yesterday · 1 comment

acolombinifake commented yesterday

Aguardamos a criação desta página para uma maior integração com as partes interessadas do Projeto.

acolombinifake added a commit that referenced this issue 23 hours ago

criação de página de contato #2 5F99639

acolombinifake commented 43 minutes ago · edited

A alteração será realizada

Comment edit history is now public Got it!

To see a comment's previous revisions, click on the edited dropdown and select the revision you want to see.

Need to remove sensitive information? Go to the specific edit where the information was added and click the Options menu to delete. Learn more about edit history.

acolombinifake closed this in c29dd34

Unwatch 1

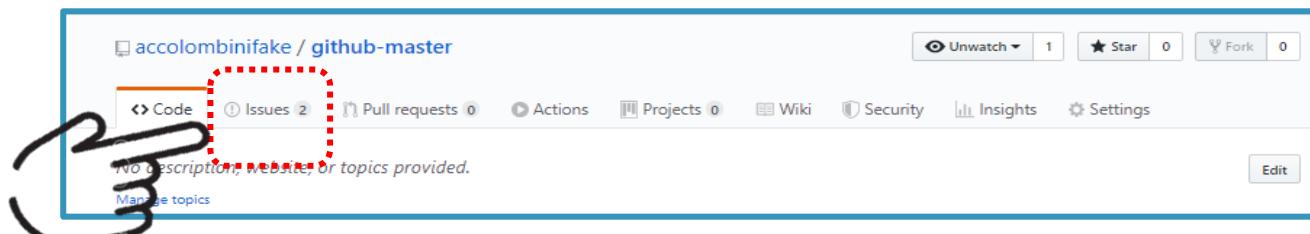
Marcação de
ISSUE encerrada



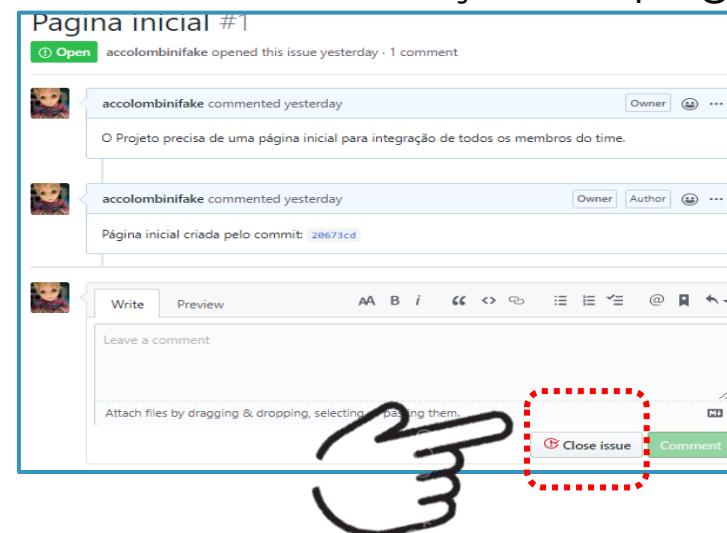
GitHub ISSUES como Ferramentas Colaborativas



19. Voltando para o GitHub poderemos observar que o número de ISSUES abertas agora é 2 e indicando que uma das ISSUES fora encerrada, observe:



20. Nos esquecemos de fechar nossa primeira ISSUE, faremos isso agora de forma manual, uma vez que já realizamos a tarefa de criação da página inicial, observe:





ISSUES – Trabalhando com Labels

Vamos praticar - Laboratório

Labels em ISSUES



- Você deve ter percebido que controlar essas ISSUES não é tão trivial assim
- Imagine que num projeto grande com um time extenso as coisas podem se complicar muito ...
- Uma forma de organizar melhor essas ISSUES é adicionando Label a ela
- Um Label é um rótulo que pode ser adicionado a uma ISSUE identificando que tipo de pedido esta sendo reportado
- O GitHub trás alguns Labels predefinidos, mas você poderá criar os seus. Vamos para o laboratório e entendermos isso na prática

Labels em ISSUES



1. Vamos para nosso GitHub e adicionemos um rótulo a Issue que ainda está aberta. Na figura abaixo observe as Labels preestabelecidas e o botão New Label

A screenshot of the GitHub Labels page. At the top, there's a navigation bar with tabs for Code, Issues (selected), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation is a search bar labeled 'Search all labels'. A red dotted circle highlights a list of nine predefined labels: bug (red), documentation (blue), duplicate (grey), enhancement (teal), good first issue (purple), help wanted (green), invalid (yellow), question (pink), and wontfix (light blue). Each label has a description and 'Edit' and 'Delete' buttons. To the right of this list is a 'Sort' dropdown and a green 'New label' button, which is also highlighted with a red dashed box. A blue callout bubble points to the 'New label' button with the text 'Para você criar suas próprias labels'. In the bottom left corner, there's a small GitHub Octocat icon.

Labels predefinidas pelo GitHub

Para você criar suas próprias labels

Labels em ISSUES



2. Vamos começar criando a nossa Label, essa Label se chamará nova ferramenta e indicará todas as ISSUES que solicititem a criação de uma ferramenta. Podemos escolher uma cor para nossa ISSUE e a descrição que é opcional. Feito isso clique em Create label → Acompanhe:

The screenshot shows a GitHub interface with the 'Issues' tab selected. A modal window titled 'New label' is open, containing the following fields:

- Label name:** nova ferramenta
- Description:** Solicitação de criação de novas ferramentas
- Color:** #95c41d (green)

At the bottom right of the modal are 'Cancel' and 'Create label' buttons. To the left of the modal, there is a cartoon character of a cat wearing glasses and a yellow hoodie, holding a hammer, with a hand-drawn style pointer pointing towards the modal.

Labels em ISSUES



3. Label criada, voltemos para nossa ISSUE para associarmos a LABEL criada.
Acompanhe:



The screenshot shows a GitHub issue page for pull request #3. The page includes a comment from user mfake0323 suggesting to create a 'Sobre' page. The 'Labels' section on the right side of the page lists various labels with their descriptions. A red dashed box highlights the 'nova ferramenta' label, which is checked. A large black hand icon points to this highlighted area. A speech bubble on the right contains the text: 'Selecione Labels e navegue procurando pela Label desejada'.

Label	Description
enhancement	New feature or request
good first issue	Good for newcomers
help wanted	Extra attention is needed
invalid	This doesn't seem right
nova ferramenta	Solicitação de criação de novas ferramentas
question	Further information is requested
wontfix	This will not be worked on

Selecione Labels e
navegue
procurando pela
Label desejada

Labels em ISSUES



4. Observe agora que você passa a ter um **suporte visual poderoso** para o controle de suas ISSUES, isso facilitará nos momentos que tiver que **priorizar** e ainda, você poderá **pesquisar por Labels**



Criação da página sobre #3

Open mfake0323 opened this issue 3 hours ago · 0 comments

mfake0323 commented 3 hours ago

Vocês poderiam criar uma página Sobre? Agradeço

accolombinifake added the nova ferramenta label now

Assignees
No one—assign yourself

Labels
nova ferramenta

Projects
None yet

Milestone

Write Preview AA B i “ ‘ < > @ Leave a comment

A screenshot of a GitHub issue page titled "Criação da página sobre #3". The issue is open and was created by mfake0323 3 hours ago with 0 comments. A comment from mfake0323 says: "Vocês poderiam criar uma página Sobre? Agradeço". Another comment from accolombinifake adds the "nova ferramenta" label. On the right side, there are sections for Assignees (No one—assign yourself), Labels (with "nova ferramenta" highlighted in green), Projects (None yet), and Milestone. A red dashed box highlights the "Labels" section. At the bottom, there are "Write" and "Preview" buttons, and a rich text editor toolbar. Below the toolbar is a text input field with placeholder "Leave a comment".

Labels em ISSUES



5. Para pesquisar por Labels (filtrar) faça como apresentado abaixo, você clica em Label e filtre por Label. Neste caso, só temos uma ISSUE, mas vale o aprendizado, observe:

The screenshot shows a GitHub Issues page with one open issue titled "Criação da página sobre nova ferramenta". The "Label" button in the top navigation bar is highlighted with a red dashed circle. A dropdown menu titled "Filter by label" is open, listing several labels: "duplicate", "enhancement", "good first issue", "help wanted", "invalid", "nova ferramenta" (which is also highlighted with a red dashed circle), and "question". A hand icon points to the "Label" button, and another hand icon points to the "nova ferramenta" label in the dropdown list.

Author - Label - Projects - Milestones - Assignee

Filter by label

Filter by label

- duplicate
- enhancement
- good first issue
- help wanted
- invalid
- nova ferramenta**
- question

Use **alt + click/return** to exclude labels.





E-MAILS E CITAÇÕES EM ISSUES

Vamos praticar - Laboratório



E-MAILS E CITAÇÕES EM ISSUES

1. Vamos retomar os trabalhos acessando a ISSUE pendente e vamos dar ao cliente, no caso mfake0323 um feedback dizendo que a página já está em construção, depois clique em Comment → observe:



Criação da página sobre #3

Open mfake0323 opened this issue 4 hours ago · 0 comments

mfake0323 commented 4 hours ago

Vocês poderiam criar uma página Sobre? Agradeço

acolombinifake added the nova ferramenta label 38 minutes ago

A página já está em produção

Attach files by dragging & dropping, selecting or pasting them.

Close and comment Comment



E-MAILS E CITAÇÕES EM ISSUES

2. Vamos agora para o GitHub de nosso segundo usuário, no caso mfake0323
3. Você observará que ele já recebeu a notificação de que sua ISSUE está em produção
4. Mas, isso não é tudo, como foi ele quem gerou a ISSUE, um e-mail foi enviado automaticamente para sua caixa de e-mail, documentando que o trabalho de sua ISSUE está em produção. Veja nas figuras a seguir:



E-MAILS E CITAÇÕES EM ISSUES

Figuras da mensagem no GitHub e e-mail enviado para mfake0323

No
GitHub

Criação da página sobre #3

Open mfake0323 opened this issue 4 hours ago · 1 comment

mfake0323 commented 4 hours ago

Vocês poderiam criar uma página Sobre? Agradeço

accolombinifake added the nova ferramenta label 1 hour ago

accolombinifake commented 12 minutes ago

A página já está em produção

Edit New issue

Assignees
No one assigned

Labels
nova ferramenta

Projects
None yet

Milestone



No e-mail

Gmail Pesquisar e-mail

Escrever

Caixa de entrada 1

Principal Social Promoções

Com estrela Angelo Re: [accolombinifake/github-master] Criação da página so... 18:44

Adiados GitHub [GitHub] Welcome to GitHub, @mfake0323! - GitHub, Inc. W... 11:32



E-MAILS E CITAÇÕES EM ISSUES

Nota: caso queira você poderá responder a esse e-mail que automaticamente ele será incorporado ao histórico da ISSUE, assim, você se quer precisa entrar no GitHub

5. Vamos fazer um teste usando um e-mail como resposta, observe:



Re: [accolombinifake/github-master] Criação da página sobre (#3) Caixa de entrada

Angelo <notifications@github.com>
para accolombinifake/github-master, mim, Author

18:44 (há 24 minutos)

A página já está em produção

You are receiving this because you authored the thread.
Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).

accolombinifake/github-master (reply+AO5IK66GI2PJBKVVVOJ3OVF4QU...)

Ok estou aguardando

Enviar



E-MAILS E CITAÇÕES EM ISSUES

- Voltando agora ao GitHub, no painel de ISSUES poderemos observar o e-mail que foi enviado, onde mfake0323 diz que está aguardando este e-mail agora faz parte do histórico da ISSUE, observe:



Criação da página sobre #3

Open mfake0323 opened this issue 4 hours ago · 2 comments

mfake0323 commented 4 hours ago

Vocês poderiam criar uma página Sobre? Agradeço

acolombinifake added the nova ferramenta label 1 hour ago

acolombinifake commented 33 minutes ago

A página já está em produção

mfake0323 commented 2 minutes ago

Ok estou aguardando

Em seg., 23 de mar. de 2020 às 18:44, Angelo <notifications@github.com> escreveu:
...

A screenshot of a GitHub issue page titled "Criação da página sobre #3". It shows a list of comments. The bottom comment from "mfake0323" is highlighted with a red dashed box. A large hand cursor icon is positioned over this highlighted comment.





E-MAILS E CITAÇÕES EM ISSUES

7. Vamos agora ao Visual Studio Code implementar as mudanças solicitadas
8. Para isso, siga os mesmos passos já vistos, crie uma nova branch (pagina-sobre) e implemente a ferramenta solicitada (no caso a página sobre)
9. Não se esqueça que o **push** deverá ser da branch criada para o GitHub (**origin**), use **git push origin <nome-da-branch>**

Observe a
sequência de
comandos
utilizada!!!

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git checkout -b pagina-sobre
Switched to a new branch 'pagina-sobre'

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-sobre)
$ git add .

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-sobre)
$ git commit -m 'criação da página sobre #3'
[pagina-sobre 56e4446] criação da página sobre #3
 1 file changed, 11 insertions(+)
 create mode 100644 sobre.html

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-sobre)
$ git push origin pagina-sobre
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 450 bytes | 450.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'pagina-sobre' on GitHub by visiting:
remote:   https://github.com/accolombinifake/github-master/pull/new/pagina-sobre
remote:
To https://github.com/accolombinifake/github-master.git
 * [new branch]      pagina-sobre -> pagina-sobre
```





E-MAILS E CITAÇÕES EM ISSUES

10. Volte ao GitHub na aba ISSUE e observe que um commit foi efetuado para solucionar a ISSUE #3

A screenshot of a GitHub issue comment. The comment is from user 'mfake0323' and was posted 36 minutes ago. The text reads: "Ok estou aguardando" followed by a reply from 'Angelo <notifications@github.com>' at 18:44 on Monday, March 23, 2020, stating "escreveu:". Below this, another comment is shown with a red dashed box around it: "acolombinifake added a commit that referenced this issue 3 minutes ago". This commit is associated with a user profile picture and the text "criação da página sobre #3". The commit hash is 56e4446.

11. Acessando o GitHub com o usuário mfake0323, ele terá essas mesmas informações, acesse os comentários via commit e peça um complemento para a página



E-MAILS E CITAÇÕES EM ISSUES

- Vamos agora subir um pouco o nível, é boa prática de GP direcionar a mensagem para a pessoa correta, para isso usamos a opção (@) e o GitHub nos mostrará todos os recursos alocados no projeto, escolha para quem indicar sua solicitação.

Nota: Uma vez utilizada essa opção o recurso destino receberá um e-mail indicando a solicitação

0 comments on commit 56e4446

Write Preview

Você poderia colocar seu telefone no site? @accoleombinifake

Attach files by dragging & dropping, selecting or pasting them.

Comment on this commit



E-MAILS E CITAÇÕES EM ISSUES

13. Entre no e-mail de accolombinifake e observe a notificação para inserção do nome no site, observe:



Traduzir a mensagem para: Português (Brasil) | Nunca traduzir do: Inglês

Mário Fake <notifications@github.com>
seg, 23/03/2020 16:14
Você; accolombinifake/github-master; Mention

Você poderia colocar seu telefone no site? [@accolombinifake](#)

You are receiving this because you were mentioned.
Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).



ASSINATURAS DE ISSUES

Vamos praticar - Laboratório

Assinaturas de ISSUES



- Um outro controle interessante que o GitHub nos oferece é o controle de assinaturas de ISSUES
- Além das pessoas que estão diretamente envolvidas na ISSUE e consequentemente receberão todas as notificações da ISSUE, pode ser interessante adicionar pessoas chaves, por exemplo, as partes interessadas no seu projeto
- Na parte superior da página do **GitHub** você possui a opção **Assignees**, lá você poderá fazer o cadastro das pessoas que irão acompanhar o desenrolar da **ISSUE**

Assinaturas de ISSUES



1. Vamos lá, acesse a página do GitHub e observe a opção Assignees



The screenshot shows a GitHub issue page for a repository named 'acolombinifake / github-master'. The issue is titled 'Criação da página sobre #3' and is marked as 'Open'. It was created by 'mfake0323' 5 hours ago and has 2 comments. The comments are as follows:

- mfake0323 commented 5 hours ago: Vocês poderiam criar uma página Sobre? Agradeço
- acolombinifake added the **nova ferramenta** label 2 hours ago
- acolombinifake commented 2 hours ago: A página já está em produção

On the right side of the issue page, there is a sidebar with several settings sections:

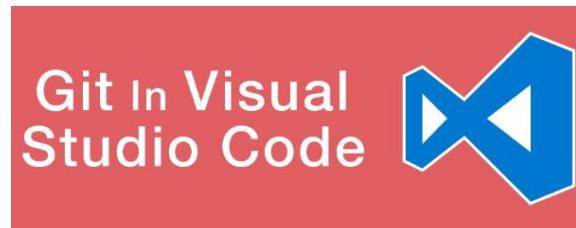
- Assignees**: No one—assign yourself (highlighted with a red dashed box)
- Labels**: nova ferramenta
- Projects**: None yet
- Milestone**: No milestone

2. Quando você clica em Assignees uma aba se abre e você escolhe quem assina a Issues (quem participa da ISSUE)

Assinaturas de ISSUES



3. Neste caso, só temos duas pessoas envolvidas no projeto e no caso, as duas já estão envolvidas na ISSUE, num cenário real temos outras partes interessadas
4. Essa é uma ferramenta bem interessante que ajuda que as partes interessadas acompanhem todo processo de criação e solução de bugs
5. Vamos agora atender a solicitação mfake0323 e encerrar essa ISSUE
6. Para isso, vá ao Visual Studio Code e efetue os passos já conhecidos



Assinaturas de ISSUES



- Além dos passos já vistos anteriormente usarei aqui o comando **git status**, e observe que o **Git** nos dirá que nosso repositório **origin** (**GitHub**) está desatualizado e que devemos fazer um **push**

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-sobre)
$ git commit -am 'finalização da pagina closes #3'
[pagina-sobre b392627] finalização da pagina closes #3
 1 file changed, 1 insertion(+)

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (pagina-sobre)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git merge pagina-sobre
Updating c29dd34..b392627
Fast-forward
  sobre.html | 12 ++++++++-
  1 file changed, 12 insertions(+)
  create mode 100644 sobre.html

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/github-master (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```



Assinaturas de ISSUES



- Observe o fechamento da ISSUE e que o commit que a fechou já foi incorporado à branch master



mfake0323 commented 2 hours ago

Ok estou aguardando

Em seg., 23 de mar. de 2020 às 18:44, Angelo <notifications@github.com> escreveu:

...

accolombinifake added a commit that referenced this issue 1 hour ago

criação da página sobre #3

accolombinifake closed this in b392627 1 minute ago

accolombinifake / github-master

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security

finalização da pagina closes #3

master

accolombinifake committed 11 minutes ago



ISSUES E MILESTONES



Vamos praticar - Laboratório

ISSUES E MILESTONES



- Milestones é outra ferramenta interessante para se trabalhar com ISSUES
- Os Milestones são como agregadores de ISSUES que nos mostram alguma tarefa específica que queremos realizar e que está tarefa é encerrada a partir do momento que algumas ISSUES são resolvidas
- Outra forma é pensar num Milestones como sendo um agrupador de ISSUES que têm característica específica
- Mais fácil fazer do que escrever, vamos para o laboratório

ISSUES E MILESTONES



- Para este laboratório vamos imaginar a situação em que mfake0323 encontrou uma série de bugs no sistema e quer reportá-los através de ISSUES para que os desenvolvedores possam resolver
- Ocorre que muitas vezes os desenvolvedores também encontram alguns bugs e precisam reportá-los também
- Assim, neste laboratório vamos criar uma série de ISSUES para tratar de alguns problemas

ISSUES E MILESTONES



1. Com o usuário mfake0323, vamos acessar o GitHub e criar duas ISSUES
2. Criamos duas ISSUES para mfake0323
3. Criamos duas ISSUES para accolombinifake, veja na relação de ISSUES a seguir:

Das quatro ISSUES criadas, observe que três delas têm haver com Bugs do Sistema



The screenshot shows the GitHub interface for the repository 'accolombinifake / github-master'. The 'Issues' tab is selected, displaying four open issues:

- ① Título da página de contato #7 opened 2 minutes ago by accolombinifake
- ① Fundo de página inicial #6 opened 3 minutes ago by accolombinifake
- ① Contato sem contato #5 opened 5 minutes ago by mfake0323
- ① Bug da página inicial #4 opened 6 minutes ago by mfake0323

A red dashed box highlights the first three issues, and a callout bubble with a pointing hand points to this box. The text in the bubble reads: 'Das quatro ISSUES criadas, observe que três delas têm haver com Bugs do Sistema'.

ISSUES E MILESTONES



4. Vamos agora agrupar essas três ISSUES em um Milestones e dizer que há um prazo específico para que sejam concluídas as alterações/correções
5. Para isso, clique em New milestone e adicione um novo Milestone no seu projeto, confira

The screenshot shows a project management interface with a toolbar at the top. The toolbar includes filters (Filters, search bar), issue counts (Labels 10, Milestones 0), and buttons for Author, Label, Projects, Milestones, Assignee, and Sort. A green button labeled 'New issue' is visible on the far right. A hand-drawn arrow points from the text 'A criação de um Milestone só pode ser feita por um desenvolvedor ou colaborador do Projeto' to the 'Milestones' button in the toolbar.

Issue ID	Title	Author	Status
#7	Título da página de contato	acolombinifake	Open
#6	Fundo de página inicial	acolombinifake	Open
#5	Contato sem contato	mfake0323	Open
#4	Bug da página inicial	mfake0323	Open

A criação de um Milestone só pode ser feita por um desenvolvedor ou colaborador do Projeto

ISSUES E MILESTONES



Observe a figura a seguir. A partir do momento que definimos uma data para término das atividades o GitHub passa a controlar nossa agenda de trabalhos sinalizando nossos prazos. A partir daqui, clique em Create Milestone e seu Milestone estará criado. Note que até o momento nenhuma ISSUE foi associada ao Milestone



accolombinifake / github-master

Code Issues 4 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

New milestone

Create a new milestone to help organize your issues and pull requests. Learn more about [milestones and issues](#).

Title

Ajuste de erros

Due date (optional)

24/03/2020

Description

Ajustes de erros nas páginas do site

Create milestone

Jew

ISSUES E MILESTONES



6. Criado o Milestone, como desenvolvedor do projeto você pode agora clicar em cada ISSUE e dizer que ela faz parte do Milestone criado, observe como e repita o processo para todas as 3 ISSUES:

The image shows two screenshots from GitHub illustrating the process of associating an issue with a milestone.

Left Screenshot (Project Overview): Shows the project page for "acolombinifake / github-master". It displays 4 issues, 0 pull requests, 0 actions, 0 projects, 0 wiki pages, 0 security vulnerabilities, and 0 insights. A "Milestones" tab is selected. A red dashed box highlights the "New milestone" button in the top right corner of the main content area.

Right Screenshot (Issue Detail): Shows the details of issue #7, titled "Título da página de contato #7". The issue is open and was created by "acolombinifake" 32 minutes ago. A comment from "acolombinifake" states: "O título da página de contato está incorreto". Below the comment, another comment from "acolombinifake" says: "acolombinifake added this to the Ajuste de erros milestone now". A red dashed box highlights this comment. A large blue callout bubble with a red border contains the text: "Aqui você têm o registro da atribuição da ISSUE ao Milestone aqui!".

Bottom Right Area (Issue Editor): Shows the "Set milestone" section of the issue editor. It includes a "Filter milestones" dropdown, checkboxes for "Open" and "Closed" milestones, and a "Clear this milestone" button. A specific checkbox for the milestone "Ajuste de erros" is checked and highlighted with a red dashed box. A red dashed box also highlights the "Milestone" section of the sidebar.

ISSUES E MILESTONES



7. Volte agora para a guia ISSUE e observe que há um Milestone criado, e dentro dele temos três ISSUES abertas, observe:

The screenshot shows a GitHub repository page for 'accolombinifake / github-master'. The 'Issues' tab is selected, showing 4 issues in total. A red arrow points to the 'Milestones' tab, which is also selected. Another red arrow points to the milestone titled 'Ajuste de erros', which has a due date of March 24, 2020, and was last updated 3 minutes ago. The milestone details show 0% complete, 3 open issues, and 0 closed issues. There are 'Edit', 'Close', and 'Delete' buttons at the bottom.

ISSUES E MILESTONES



8. Vamos agora realizar todas as ações para realizar as tarefas que as ISSUES estão solicitando
9. Para abrir, basta clicar em open e as ISSUES se abrirão. Aproveite e observe o que acontece com o Milestone à medida que os commits que solucionam as ISSUES vão acontecendo

Atenção: as soluções são simples e você já conhece os passos, eles não serão aqui apresentados, caso tenha dúvidas, retorne para as aulas anteriores e faça uma revisão

acolombinifake / github-master

Issues 3 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

New milestone

1 Open 0 Closed

Ajuste de erros
Due by March 24, 2020 Last updated 1 minute ago
Ajustes de erros nas páginas do site

Sort ▾

33% complete 2 open 1 closed
Edit Close Delete

Observe que o GitHub sinaliza que já cumprimos 33% das tarefas deste Milestone

ISSUES E MILESTONES

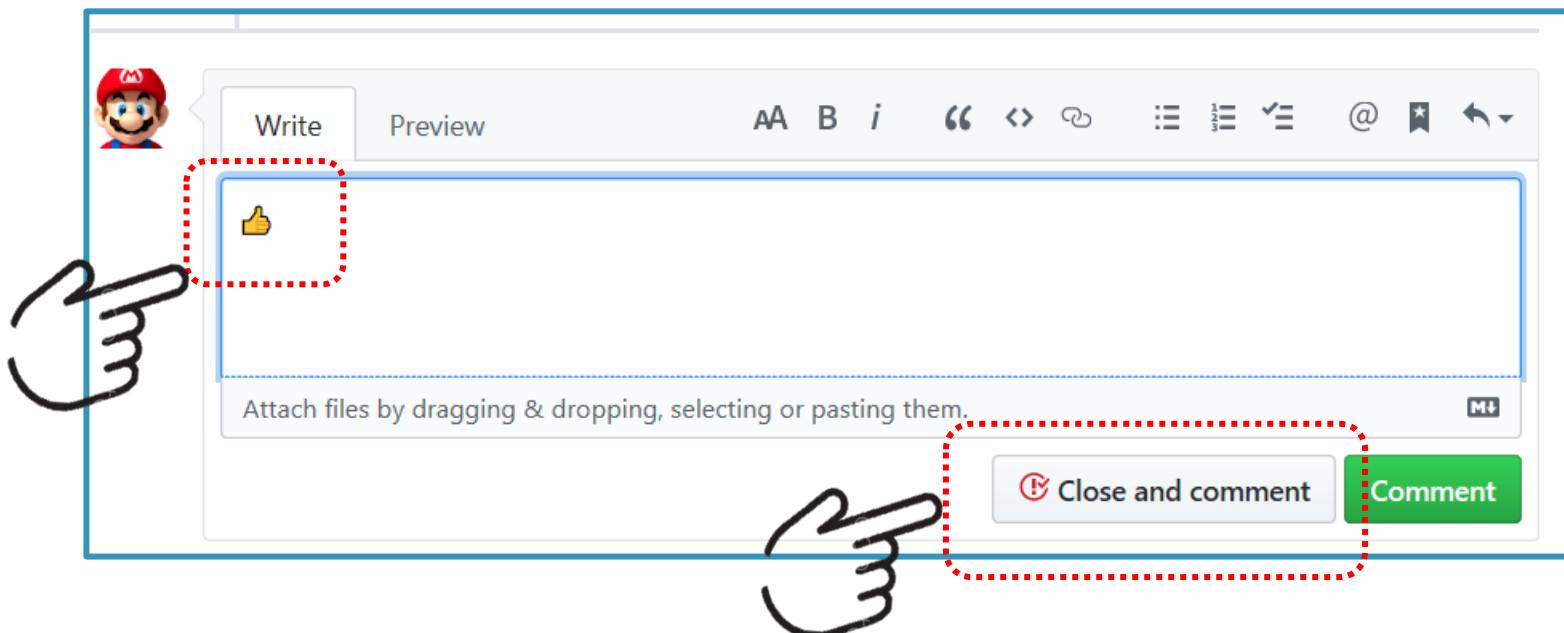


10. Como estamos simulando várias iterações, na ISSUE #5 não fechamos no commit, estamos permitindo que mfake0323 verifique uma vez mais se sua demanda foi atendida.
11. Observe que no comentário ele está começando com (:), isso fará com que o GitHub abra uma palheta com Emojis para que você coloque em seu comentário
12. Por fim, estando satisfeito e sendo fake0323 o criador da ISSUE ele poderá fechá-la, observe

ISSUES E MILESTONES



Observe a figura, feito o comentário basta clicar em **Close and comment**:



ISSUES E MILESTONES



13. Como tarefa, finalize as duas ISSUES restantes, observe o Milestone

Finalizamos aqui – Git GitHub (básico e intermediário)





Git GitHub Avançado

Dominando Git GitHub

Git e GitHub

Indo um além do trivial!

Checkout e reset



- Sempre é possível desfazer tudo tivermos feito no Git
- Podemos por exemplo: desfazer um commit, remover um arquivo que tenhamos enviado para a área de Stage e também deixarmos de rastrear um arquivo
- Tudo isso parece ótimo não é mesmo?
- Sim, sem dúvida é ótimo podemos fazer isso, mas devemos ter muito cuidado, pois, sempre ao desfazermos algo poderemos estar comprometendo o histórico do projeto (Project History), e isso, nem sempre é uma boa prática, principalmente se você já tiver enviado seus arquivos para o repositório do projeto no GitHub

Checkout e reset



Dica: tome sempre cuidado em se certificar de que a branch que está sofrendo alterações não tenha sido publicada no GitHub, em outras palavras, tenha muito cuidado ao enviar suas Branches para a nuvem (GitHub)





CHECKOUT E RESET

Vamos praticar - Laboratório

Checkout e reset



1. Vamos praticar → no Visual Studio Code abra um novo diretório dedicado ao nosso projeto prático, no meu caso chamarei de <check_reset>
2. Realize os passos já conhecidos, crie um novo arquivo, adicione na Stage e realize seu primeiro commit, veja os passos para relembrar acrescentei outros comandos como git status e git log



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git init
Initialized empty Git repository in D:/Users/Angelo/GIT/check_reset/.git/

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git commit -m 'primeiro commit'
[master (root-commit) f3ef5fc] primeiro commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git status
On branch master
nothing to commit, working tree clean

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log
commit f3ef5fce1b1bad33f815df2cf542882005ebaabe (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 15:14:45 2020 -0300

primeiro commit
```

Checkout e reset



3. Faça agora uma alteração no arquivo criado e salve seu arquivo, use o comando **git status** e observe que o Git registrou a mudança, mas o arquivo **ainda não foi para a área de Stage**



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (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:   a.txt

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ []
```

O fato de ter salvo o arquivo não significa para o Git que ele deva rastreá-lo. Isso significa que você pode desfazer essas alterações mesmo já tendo salvo o arquivo!

Checkout e reset



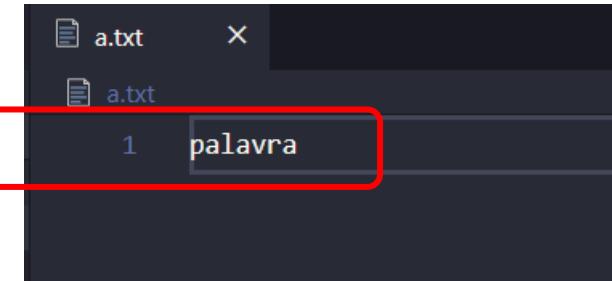
Importante: o fato de ter salvo um arquivo, para o Git, não implica em necessidade de rastreamento, sendo assim, você pode facilmente descartar essas alterações!

- Para restaurar ou descartar todas as alterações realizadas num arquivo salvo que ainda não tenha sido adicionado à **Stage** através do comando **git add .** basta usar esse comando **git checkout <nome-do-arquivo>**, observe no Visual Studio Code que as alterações que realizou serão automaticamente removidas

Checkout e reset

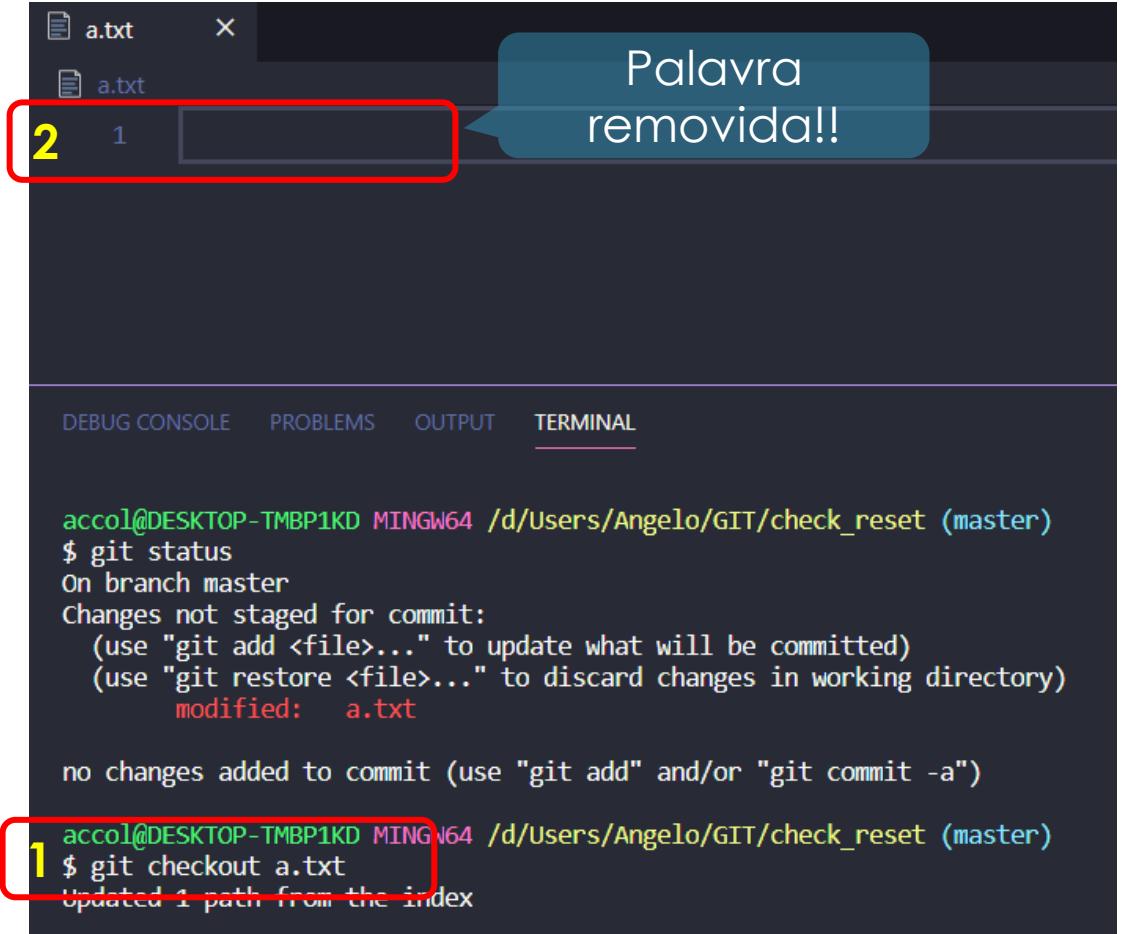


- A seguir veja a alteração realizada, no arquivo em branco foi inserida a palavra 'palavra'



```
a.txt
1 palavra
```

4. Faça uso do comando git checkout <nome-do-arquivo> e observe o resultado ao lado



```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (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:   a.txt

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

1 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git checkout a.txt
Updated 1 path from the index
```

Checkout e reset



Dica: recomendo que se tome cuidado com este comando, pois, como observou ao **utilizá-lo você zera todas as modificações realizadas**, e aí poderá perder seus dados, portanto, tenha cuidado

5. Vamos agora alterar novamente o arquivo e adicioná-lo na Stage, neste caso, estaremos colocando as alterações em modo Rastreio para o Git
6. Podemos observar que as alterações já estão prontas aguardando o commit, mas ainda é possível reverter, vamos lá → use o comando **git reset head <nome-do- arquivo>**

Checkout e reset



Nota: a palavra reservada **head** indica que nós estamos neste ponto do commit, e podemos sabendo disso, remover tudo o que esta na Stage até esse ponto da linha do tempo, é como se pudéssemos voltar no tempo, computacionalmente, estamos movendo nossa **head** do commit



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git reset head a.txt
Unstaged changes after reset:
M      a.txt

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ []
```

Observe que não temos mais arquivos na área de Stage

Checkout e reset



5. Verifique usando o comando **git status**, e em seguida restaure seu arquivo com o comando **git checkout <nome-do-arquivo>**

A screenshot of a terminal window in a dark-themed code editor. The terminal tab is active at the bottom. The window shows a file named 'a.txt' with one line of text. The terminal output is as follows:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (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:   a.txt

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git checkout a.txt
Updated 1 path from the index
```

The terminal output is annotated with three red boxes and numbers:

- Box 1 surrounds the first command and its output: `git status` followed by the status message.
- Box 2 surrounds the second command and its output: `git checkout a.txt` followed by the message "Updated 1 path from the index".
- Box 3 surrounds the file 'a.txt' in the top left corner of the terminal window.



Checkout e reset



6. Ok, moleza até aqui, agora vamos dar um passo a mais, vamos modificar nosso arquivo, adicioná-lo na Stage e realizar o commit → no Visual Studio Code pratique

Observe que para melhorar a rastreabilidade realizei também um git status e um git log, observe os resultados

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git commit -m 'alteração'
[master 98e5a53] alteração
 1 file changed, 1 insertion(+)

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git status
On branch master
nothing to commit, working tree clean

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log
commit 98e5a538b2f58456118cbe6b48afe94486f921fa (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 16:04:53 2020 -0300

        alteração

commit f3ef5fce1b1bad33f815df2cf542882005ebaabe
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 15:14:45 2020 -0300

primeiro commit
```

Checkout e reset



- Vamos agora aprender uma nova forma de usar o comando git log → **git log --decorate --oneline**. A opção oneline você já conhece, vai mostrar tudo em uma linha, já a opção decorate tem por função nos mostrar onde está a nossa HEAD (em que ponto estamos no Project History → em que commit estamos), acompanhe:

Observe a HEAD está apontando para o último **commit na master**, o que vamos fazer agora é remover esse commit

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log --decorate --oneline
98e5a53 (HEAD -> master) alteração
f3ef5fc primeiro commit
```

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ []
```

Checkout e reset



8. Para fazer isso o que faremos na verdade é voltar a HEAD um commit atrás, fazendo isso, estaremos eliminando esse commit da nossa Project History
9. Para fazer isso, você já sabe, o comando que permite a você manipular a HEAD é o comando **git reset head**, em termos de commit, precisamos de algo mais, vamos acrescentar a opção (^), cada (^) utilizado é um commit que voltamos, no nosso caso voltaremos apenas um commit então observe, assim usaremos **git reset head^**:

Observe o
hashCode do
commit
apontado pela
HEAD no caso (1)
e em (3)

```
1 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log --decorate --oneline
98e5a53 (HEAD -> master) alteração
f3ef5fc primeiro commit

2 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git reset head^
Unstaged changes after reset:
M      a.txt

3 accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log --decorate --oneline
f3ef5fc (HEAD -> master) primeiro commit
```



Checkout e reset



10. Agora para avaliar o que foi feito, use os comandos **git status** e em seguida **git log**, você deverá observar que o commit 2 não existe mais e que o arquivo a.txt foi modificado, mas não se encontra na área de Stage

Observe o arquivo foi modificado mas não se encontra na área de Stage

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (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:   a.txt

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log
commit f3ef5fce1b1bad33f815df2cf542882005ebaabe (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 15:14:45 2020 -0300

  primeiro commit
```

Observe que a HEAD está apontando para o primeiro commit na Branch master

Checkout e reset



11. Poderíamos facilmente agora com o comando git checkout <nome-do-arquivo> remover as alterações, mas não faremos isso, adicione novamente na Stage e realize novamente um commit, acompanhe os passos:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git commit -m 'commit de novo'
[master cebabee] commit de novo
 1 file changed, 1 insertion(+)

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log
commit cebabeed6934d8145f637e647899c9cc7881f9b7 (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 16:53:07 2020 -0300

    commit de novo

commit f3ef5fce1b1bad33f815df2cf542882005ebaabe
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 15:14:45 2020 -0300

    primeiro commit
```

Checkout e reset



12. Imagine que você não quer mais que seu arquivo seja rastreado pelo Git, você não tem mais interesse que ele permaneça na Project History, as alterações feitas no arquivo a partir deste momento não serão rastreadas, para isso, basta usar o comando **git rm –cached <nome-do-arquivo>**

Atenção: o Git irá deixar de rastrear seu arquivo a partir do momento em que pediu isso a ele, mas todo seu histórico antes desse ponto continuará registrado. Em algum momento, você poderá também voltar atrás e retornar a rastrear o arquivo

Checkout e reset



13. Em algum momento que deseje poderá voltar seu arquivo ao rastreio do Git utilizando o comando **git restore --staged <nome-do-arquivo>**. Enquanto isso, as alterações nos arquivos não serão rastreadas

Atenção: depois que você usar o comando **git rm --cached <nome-do-arquivo>**, você deverá fazer um commit sem realizar o git add ., só a partir desse momento seu arquivo deixará de ser rastreado



Checkout e reset



14. Observe os passos → não se esqueça o commit é essencial, observe:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git rm --cached a.txt
rm 'a.txt'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:   a.txt

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git commit -m 'descarta alterações'
[master 62bbf3b] descarta alterações
 1 file changed, 1 deletion(-)
 delete mode 100644 a.txt
```

Checkout e reset



15. Use o comando git status e observe que o arquivo não está mais em processo de rastreio pelo Git, observe:



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    a.txt

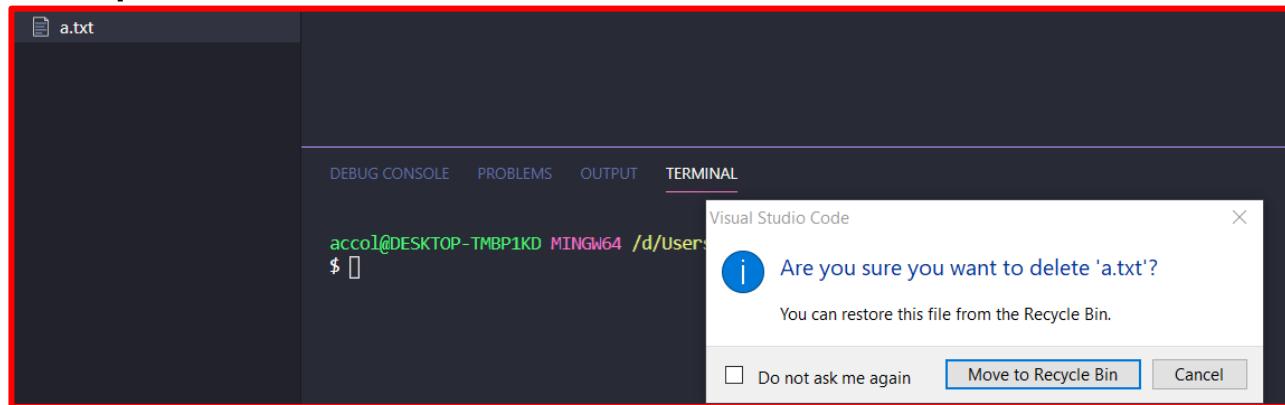
nothing added to commit but untracked files present (use "git add" to track)
```

16. Para simular a possibilidade de voltar ao estágio anterior ao uso do comando **git rm --cached <nome-do- arquivo>** vamos remover esse arquivo de nossa pasta para posteriormente resgatá-lo, acompanhe:

Checkout e reset



17. Para acompanhar:



18. Voltemos para o **commit o primeiro commit** para confirmarmos a tese de que este arquivo pode ser recuperado. Para voltar ao commit desejado, use o comando **git checkout <hash-do-commit>**

Checkout e reset



19. Observe quem voltou:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log
commit 62bbf3bdb0751c4f26da3d4dc5ad7ddebababa5 (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 17:29:43 2020 -0300

    descarta alterações

commit cebabeed6934d8145f637e647899c9cc7881f9b7
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 16:53:07 2020 -0300

    commit de novo

commit f3ef5fce1b1bad33f815df2cf542882005ebaabe
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 15:14:45 2020 -0300

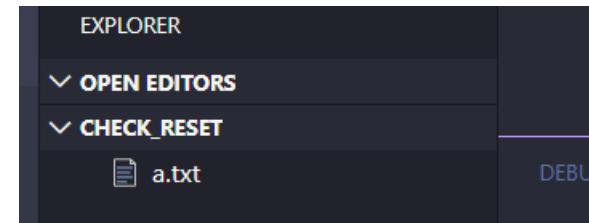
    primeiro commit

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git checkout f3ef5fce1b1bad33f815df2cf542882005ebaabe
Note: switching to 'f3ef5fce1b1bad33f815df2cf542882005ebaabe'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>
```



O arquivo deletado a.txt no status em que se encontrava quando da execução do primeiro commit

Após estes passos veja o resultado ao lado

Checkout e reset



20. Podemos radicalizar, da forma que trabalhamos conseguimos recuperar todas as ações feitas
21. Ao usar o comando **git reset –hard <hashCode-commit-desejado>** com a opção hard estaremos resetando todas as modificações realizadas após o instante que marcamos para a **HEAD retornar** (hashCode-commit-desejado)
22. Vamos retornar ao primeiro commit, observe que ao usar o git log poderemos observar que todos os outros commits foram removidos → acompanhe:

Checkout e reset



23. Acompanhe os passos:



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git reset --hard f3ef5fce1b1bad33f815df2cf542882005ebaabe
HEAD is now at f3ef5fc primeiro commit

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/check_reset (master)
$ git log
commit f3ef5fce1b1bad33f815df2cf542882005ebaabe (HEAD -> master)
Author: accolombinifake <accolombinifake@hotmail.com>
Date:   Tue Mar 24 15:14:45 2020 -0300

    primeiro commit
```



Fork



- Para estas atividades, vamos nesta primeira parte do laboratório construir alguns conceitos chaves, e iniciaremos construindo nossa legenda:
 - Repositório principal → chamado de **upstream**
 - Repositório cópia → chamado de **fork**
 - Repositório local → chamado de **Repositório Local**



Fork



- Mas afinal o que é **Fork**?
- **Fork** é um conceito que envolve as tecnologias **Git** e **GitHub** e que você pode fazer uma cópia um clone de um repositório público qualquer direto para seu repositório
- Isso lhe **permite experimentar alterações** neste projeto, uma vez, que não é o dono do repositório ou ainda não faz parte do Time de desenvolvimento desse projeto
- Ao realizar o **Fork** você passa a acompanhar a vida do projeto como se ele fosse seu

Fork



- O uso de **Fork** está relacionado também com o processo de colaboração com outros projetos. Quando você quiser mandar alterações ou sugerir alterações para projetos de terceiros você poderá usar o ciclo do Fork como parte desse processo de envio de sugestões ou alterações
- **Upstream** é o nome dado ao projeto que queremos fazer um **Fork** (é o repositório principal → normalmente um repositório público que tenha interesse)

Fork



- Quando você clica na opção Fork do GitHub, o GitHub faz uma cópia completa do Repositório Principal (Upstream) para o Repositório Copiado (Fork)
- Essa cópia (Fork) será copiado para sua área do GitHub como se o projeto fosse seu

Atenção: vale lembrar que esses repositórios são Assíncronos, ou seja, alterações feitas no repositório principal (Upstream) não serão refletidas no seu repositório cópia (Fork), sendo as atualizações independentes!

Fork



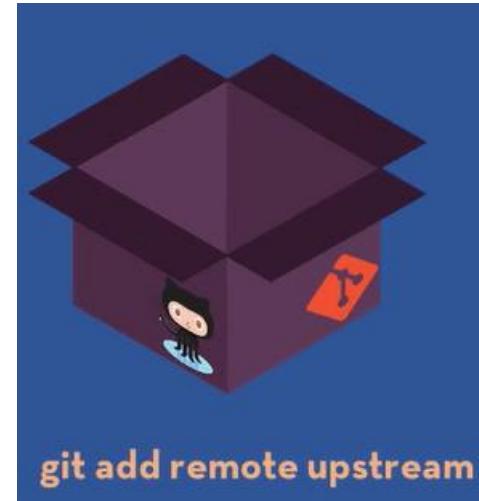
- Mas você pode se manter atualizado com o repositório principal (Upstream), para isso, você precisa realizar um processo de sincronismo
- Para realizar esse processo de sincronismo você precisará acessar seu repositório **Fork no GitHub** e fazer dele um **Clone** para sua máquina local (Repositório Local)



Fork



- Já no seu repositório local você adiciona como um remoto o **nick** para seu repositório principal (Upstream), em outras palavras você faz um **add remote**

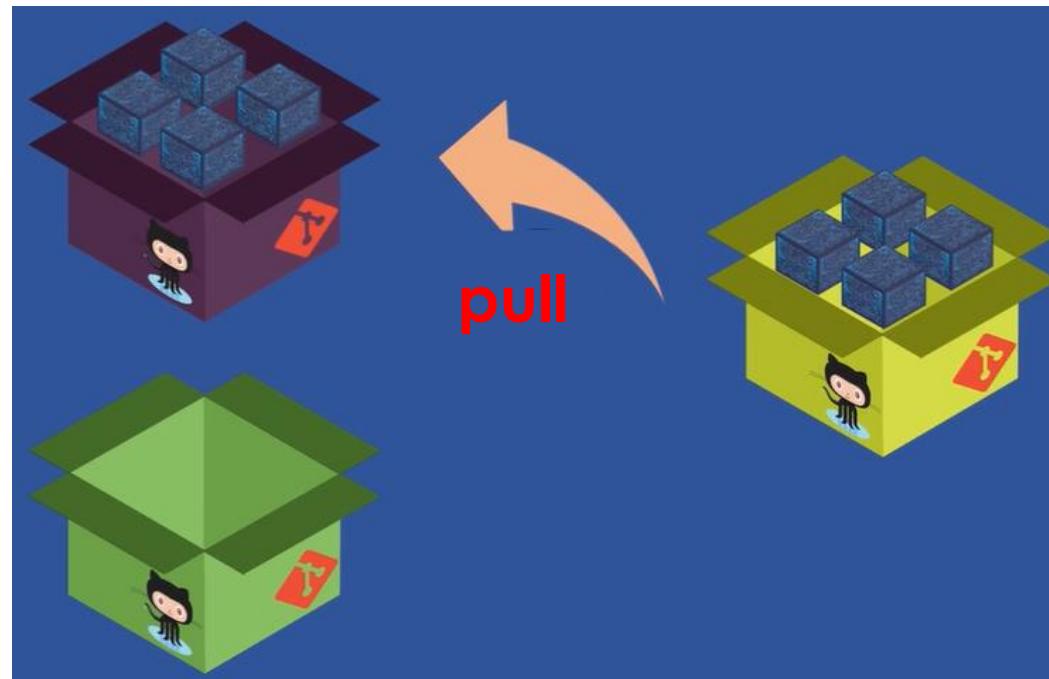


- Com a adição do nick no seu repositório local, você poderá fazer um **pull** do repositório Principal (Upstream) para seu repositório Local

Fork



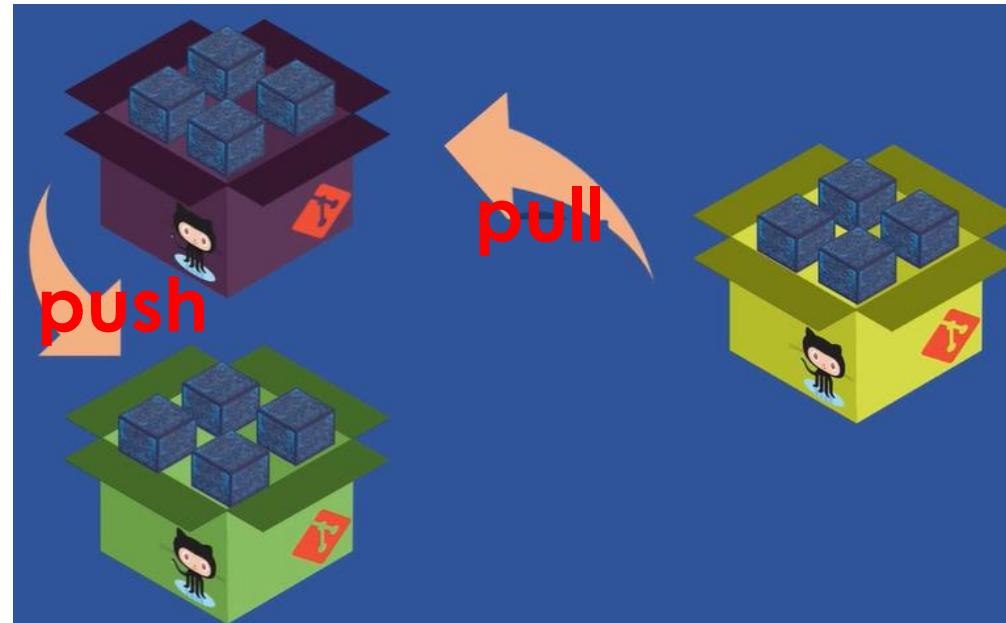
- Fazendo um pull para seu repositório local você baixará para sua máquina todas as atualizadas do repositório Principal ou Upstream sincronizando seu repositório local com o repositório Principal ou Upstream, observe a figura a seguir:



Fork



- Uma vez tendo seu repositório local sincronizado, basta fazer um **push** para seu repositório Fork, estabelecendo assim um sincronismo entre os três repositórios.



Fork



- Resumindo:
 - git add remote upstream
 - git pull upstream (master)
 - git push origin master





Fork



1. Para construir o cenário deste laboratório log que seu usuário fake, no caso usarei accolombinifake
2. Uma vez locado vamos selecionar um repositório para realizarmos nosso Fork, você poderá acessar o repositório de seu interesse. Para fins didáticos recomendo o diretório fakebmarchete/projeto-livro-scifi, didaticamente preparado para esse laboratório



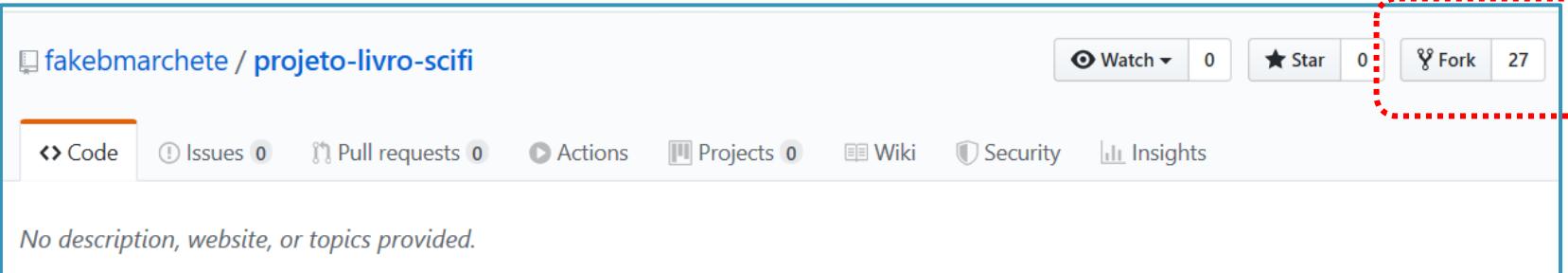
<https://github.com/fakebmarchete/projeto-livro-scifi>



Fork

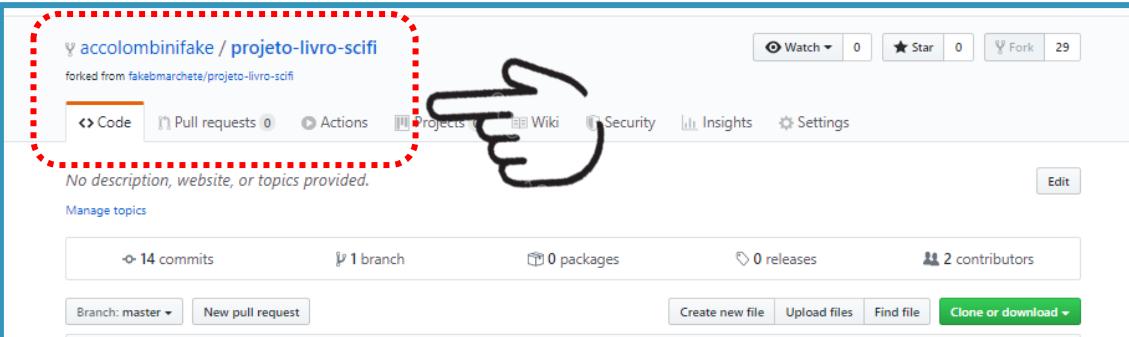


3. Para realizar o Fork desse Repositório clique na opção Fork



A screenshot of a GitHub repository page. The repository name is "fakebmarchete / projeto-livro-scifi". The page shows basic statistics: 0 issues, 0 pull requests, 0 actions, 0 projects, 0 wiki pages, 0 security vulnerabilities, and 0 insights. A large orange diamond icon is on the left. At the top right, there are buttons for "Watch" (0), "Star" (0), and "Fork" (27). A hand icon with a red dotted outline is pointing at the "Fork" button.

4. Ao clicar em Fork o GitHub fará uma cópia do repositório projeto-livro-scifi para seu repositório no GitHub

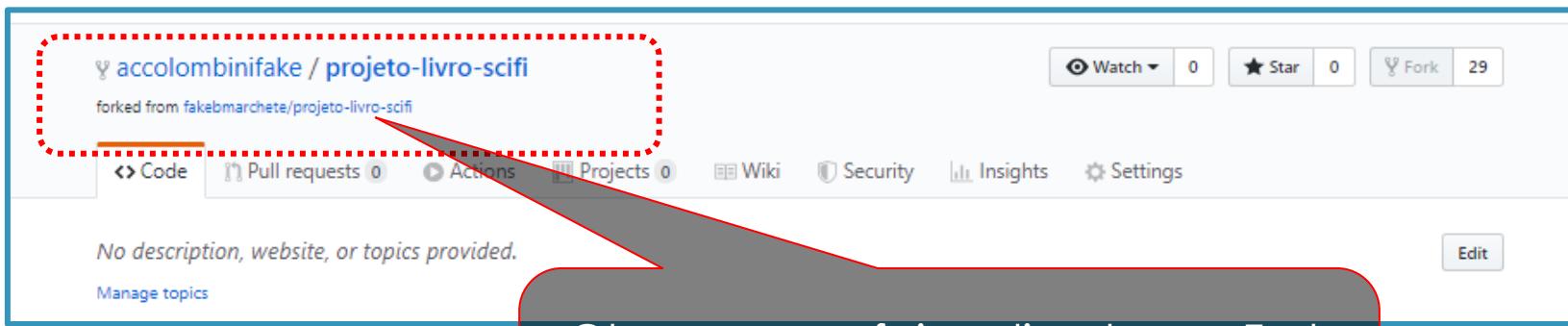


A screenshot of a GitHub repository page for "accolominafake / projeto-livro-scifi", which is a fork of the original repository. The forked repository has 29 forks. The page displays the same basic statistics as the original. A hand icon with a red dotted outline is pointing at the repository name "accolominafake / projeto-livro-scifi".

Fork



5. Olhando no seu repositório Fork observe os detalhes:



A screenshot of a GitHub repository page for 'accolombinifake / projeto-livro-scifi'. The page shows it was forked from 'fakebmarchete/projeto-livro-scifi'. The repository has 0 stars, 29 forks, and 0 pull requests. It includes tabs for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A red dashed box highlights the repository name and the 'forked from' information. A red arrow points from this highlighted area to a callout box below.

Observe que foi realizado um Fork para seu repositório a partir do repositório de fakebmarchete

6. Feito o Fork o próximo passo é fazer o clone do nosso repositório para o nosso repositório local

Fork



7. Este processo já é conhecido, então copie o endereço do repositório e no Visual Studio Code abra um novo folder e realize o clone para esse folder, no meu caso o folder se chama (projeto). O clone será direcionado para dentro desta pasta
8. Não se esqueça, você precisa estar clonando a partir do seu repositório Fork (isso é muito importante)!!!



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto (master)
$ git clone https://github.com/acco1ombinifake/projeto-livro-scifi.git
Cloning into 'projeto-livro-scifi'...
remote: Enumerating objects: 48, done.
remote: Total 48 (delta 0), reused 0 (delta 0), pack-reused 48
Unpacking objects: 100% (48/48), 396.70 KiB | 70.00 KiB/s, done.
```

Fork



9. Para o próximo passo, precisamos retornar ao diretório Upstream e fazer a cópia de seu endereço, fique atento, você deverá estar dentro de seu repositório local, observe:

Certifique-se de estar exatamente no repositório Upstream

No description, website, or topics provided.

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights

14 commits 1 branch 0 packages 0 releases 2 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

Conforme já aprendido faça a cópia como se estivesse se preparando para Clonar o repositório

Fork



10. Volte agora ao Visual Studio Code e execute o próximo passo, que é o **git remote add <nome-do-repositório>**, lembre-se o nome do repositório por padrão chama-se **upstream** observe:



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ git remote add upstream https://github.com/fakebmarchete/projeto-livro-scifi.git

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ 
```

11. Se você quiser saber quantos repositórios remotos fazem parte do nosso projeto você pode usar o comando **git remote -v**

O repositório upstream é o endereço que aponta para fakebmarchete e o Origin que é o nosso Fork

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ git remote -v
origin  https://github.com/fake0323/projeto-livro-scifi.git (fetch)
origin  https://github.com/fake0323/projeto-livro-scifi.git (push)
upstream  https://github.com/fakebmarchete/projeto-livro-scifi.git (fetch)
upstream  https://github.com/fakebmarchete/projeto-livro-scifi.git (push)

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ 
```

Fork



12. Feito isso o próximo passo é atualizarmos, ou melhor, sincronizarmos assegurando que a nossa Branch master esteja atualizada com nosso repositório Upstream, para isso, usamos o comando **git pull upstream master**



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ git pull upstream master
From https://github.com/fakebmarchete/projeto-livro-scifi
 * branch            master      -> FETCH_HEAD
 * [new branch]      master      -> upstream/master
Already up to date.

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ []
```

13. A partir deste ponto nosso repositório local já se encontra sincronizado, precisamos agora fazer um **git push** para nosso repositório Fork e fechamos assim nosso processo de sincronismo

Fork



14. Acompanhe o processo final de sincronização no Visual Studio Code, observe

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/projeto/projeto-livro-scifi (master)
$ git push
Everything up-to-date
```



Segredos do
GitHub
Imagens



Rebase – que nome é esse?

Isso é sério???



REBASE vs MERGE

- A partir do comando **rebase** nós conseguimos alterar o histórico de nosso projeto, o Project History, o que podemos fazer:
 - Modificar Project History
 - Alterar a ordem dos commits
 - Unir commits
 - Modificar mensagens dos commits
- Enfim, o comando rebase nos permite trabalhar e melhorar tudo o que tiver haver com o Project History



REBASE vs MERGE

- Para você que chegou até aqui, deve ter percebido que o **commit** não é para ser usado para fazer **um save**, ou seja, como um **backup** de nosso projeto
 - **Atenção** → commits não são apenas pontos de backup!
- Você deve pensar em **commits** como uma forma de manter o histórico cronológico do seu projeto
- Em outras palavras os commits deverão contar e ou registrar todo ciclo de vida de seu projeto, resumindo, podemos dizer que os commits devem refletir o Project History



REBASE vs MERGE

- E o comando **rebase**, assim como o commit, também é um comando para trabalhar o **Project History**
- Quando comparamos o comando **rebase** com o comando **merge**, nós comumente dizemos que **os dois servem para integrar alterações de uma branch em outra**, mas é bom que se saiba, existem vantagens e desvantagens no uso do **rebase** ou do **merge**
- Quando usamos o **merge** para integrar alguma branch, observamos que **ele não destrói a branch**, assim, podemos dizer que o **merge não é um comando destrutivo**. A linha do tempo de uma branch não é destruída



REBASE vs MERGE

- Sendo assim, usando o merge, podemos ainda acompanhar todas as ramificações e avaliar todos os commits que foram realizados
- **Por outro lado** o uso recorrente de merge acaba poluindo a linha do tempo de uma branch e dificultando a gestão do Project History
- Já o **comando rebase**, quando usado, permite que se limpe esse histórico e essas ramificações deixando muito mais claro seu Project History

REBASE vs MERGE



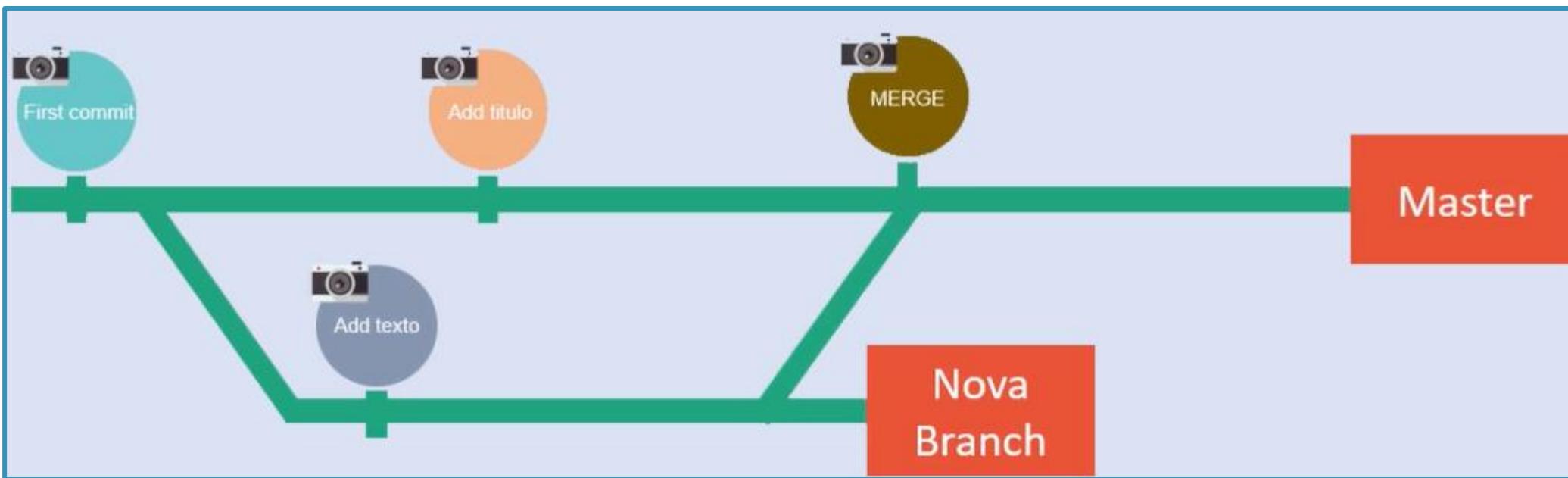
Resumindo podemos dizer que rebase e merge farão a mesma coisa em termos de resultados finais, no entanto, as características de um e de outro impactam no resultado e na gestão de seu projeto, vale ponderar e avaliar o momento certo para se usar um ou outro





CONTROLE DE BRANCH COM REBASE

- Só para relembrar, observe a figura abaixo onde descrevemos o funcionamento típico do uso do comando **merge**



CONTROLE DE BRANCH COM REBASE

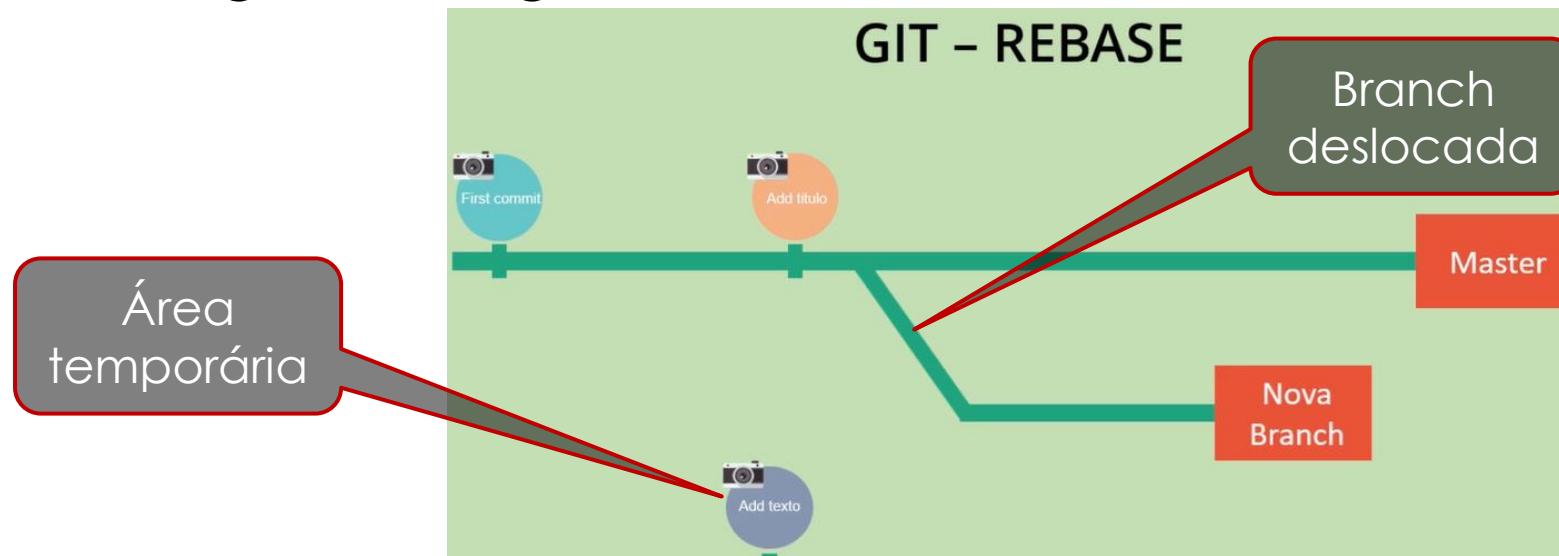


- Observe que o **commit de merge** criado não trás qualquer relevância para seu Project History a menos do fato que ele nos diz que houve um merge neste ponto
- Quando estamos em fases longas do projeto ou no desenvolvimento de ferramentas extensas pode ser interessante manter a rastreabilidade de todas as Branches criadas
- Mas, quando isso não for o caso, ou não houver esse necessidade pode ser mais interessante o uso do comando **rebase**



CONTROLE DE BRANCH COM REBASE

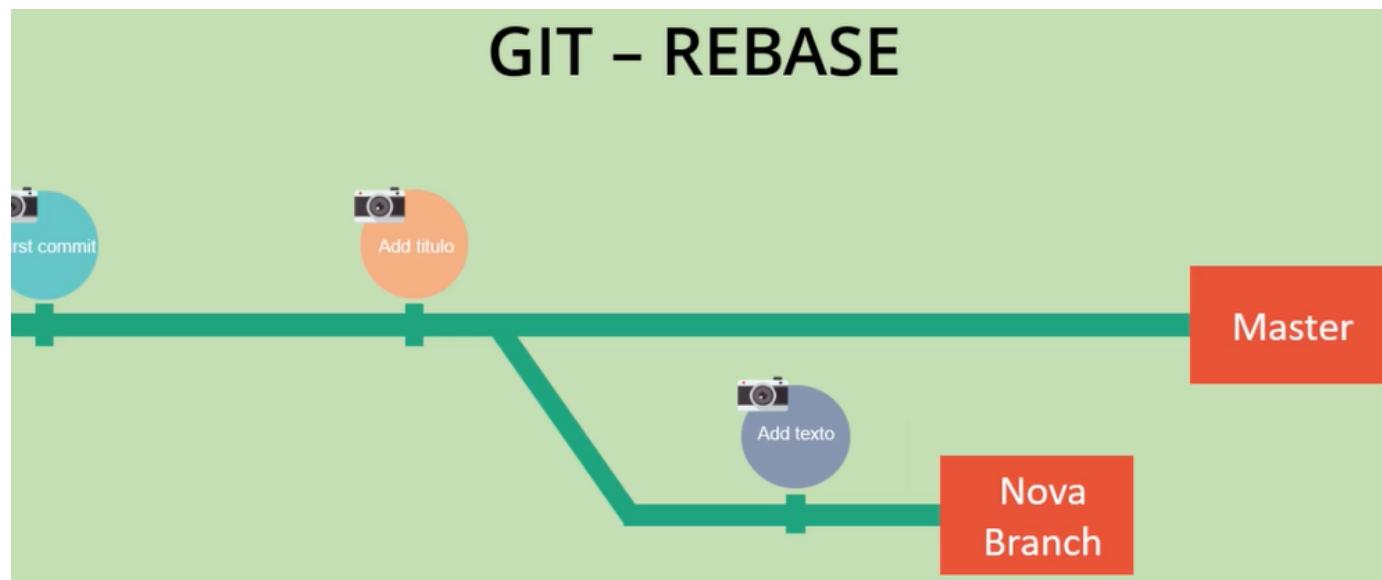
- O comando **rebase** por sua vez tem um conceito diferente, ao usar o rebase o **Git transfere o commit** para **uma área temporária**, **desloca a branch** para um instante após o **último commit registrado na master**
- Realiza o merge, que sempre será do **tipo Fast-Forward**, observe nas imagens a seguir:





CONTROLE DE BRANCH COM REBASE

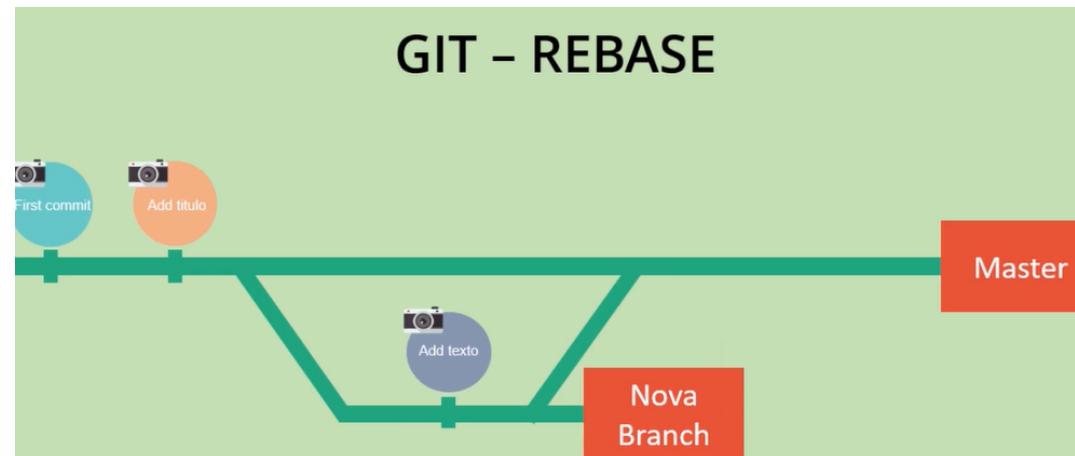
- Após deslocar a branch o **Git** irá reinserir **o commit na branch Nova Branch**, é justamente isso que irá assegurar que nossa junção será do tipo **Fast-Forward** observe:



CONTROLE DE BRANCH COM REBASE



- Ao realizar uma **merge do tipo Fast-Forward** não existirá a inserção do commit de merge, pois agora, a nossa branch Nova Branch aponta para o último commit realizado na Branch Master
- Assim quando realizarmos o merge teremos algo como:



CONTROLE DE BRANCH COM REBASE



- Assim, estando tudo linear nossa merge será do tipo Fast-Forward como visto anteriormente, observe:





CONTROLE DE BRANCH COM REBASE

- Bem, eu sei, parece bom de mais para ser verdade, não é mesmo?
- Pois é, ocorre que, quando o rebase aloca os commits para uma área temporária eles (commits) acabam recebendo **novos hashCode**, o que, altera sua referência na Project History
- Esse é um dos principais motivos para você não fazer uso do **rebase** em Branches que estão compartilhadas ou que já estão no **GitHub**, por exemplo
- Isso acontece porque você está alterando tudo no seu repositório local, mas estas alterações não são refletidas no repositório remoto, portanto, **nunca use rebase nestas condições**



REBASE

Vamos praticar - Laboratório

CONTROLE DE BRANCH COM REBASE



1. Para este laboratório crie um novo folder no Visual Studio Code, vamos chama-lo de rebase
2. Neste primeiro laboratório, vamos focar na diferença entre o uso do rebase e do merge quando estamos trabalhando com Branches, lembre-se inicie seu repositório, você já sabe como:
3. Vamos criar um novo arquivo chamado arquivo1.md, adicioná-lo à Stage e realizar nosso 'first commit'

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/rebase (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/rebase (master)
$ git commit -m 'first commit'
[master 4151878] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 rebase/arquivo1.md
```



CONTROLE DE BRANCH COM REBASE

4. Vamos criar agora uma nova Branch para inserir novos arquivos dentro dela, você já domina isso, então mãos à massa
5. Feito isso, crie um novo arquivo (arquivo2.md), adicione-o na Stage e realize um segundo commit 'adiciona arquivo2 na nova-branch'

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/rebase (master)
$ git checkout -b nova-branch
Switched to a new branch 'nova-branch'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/rebase (nova-branch)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/rebase (nova-branch)
$ git commit -m 'adiciona arquivo2 na nova-branch'
[nova-branch 16f2938] adiciona arquivo2 na nova-branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 rebase/arquivo2.md
```



CONTROLE DE BRANCH COM REBASE

6. Volte agora para sua branch master e faça uma pequena alteração no seu arquivo1.md, repita o processo de salvar, adicionar na Stage e realizar o commit

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (nova-branch)
$ git checkout master
Switched to branch 'master'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git commit -m 'adicionado titulo no arquivo1'
[master a2fd404] adicionado titulo no arquivo1
 1 file changed, 1 insertion(+)
```

7. Com isso, acabamos de criar uma situação em que a branch nova-branch não mais aponta para o último commit da branch master, isso pode ser visualizado utilizando o comando **git log --graph --oneline**

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git log --graph --oneline
* a2fd404 (HEAD -> master) adicionado titulo no arquivo1
* d62110d first commit
```



CONTROLE DE BRANCH COM REBASE

- Vamos neste momento realizar o **merge** destas Branches e verificar o que acontece, observe que o merge foi do tipo “**recursive**” **strategy** e comente:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git merge nova-branch
Merge made by the 'recursive' strategy.
arquivo2.md | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 arquivo2.md

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git log --graph --oneline
*   65593e0 (HEAD -> master) Merge branch 'nova-branch'
|\ 
| * a85c9c5 (nova-branch) adiciona arquivo2 na nova-branch
| * a2fd404 adicionado titulo no arquivo1
|/
* d62110d first commit
```

- Como discutido anteriormente, o legal do comando merge é que ele nos permite rastrear todas as ramificações do nosso projeto, pena que em determinadas situações poderemos poluir mais do que ajudar



CONTROLE DE BRANCH COM REBASE

10. Continuando nosso laboratório, vamos refazer todo processo utilizando o comando **rebase**, para isso, começaremos excluindo o conteúdo de nossa pasta rebase e zerando-a para a próxima fase do laboratório
11. Reinicialize sua pasta e refaça todos os passos anteriores, menos o merge é claro
12. Depois de realizar os passos iniciais, precisamos **acessar novamente a branch nova-branch** para lá realizarmos nosso **rebase**



CONTROLE DE BRANCH COM REBASE

13. Fique atento, pois o merge realizamos de dentro da master, já o rebase precisa ser realizado na branch que queremos mudar o histórico para só então fazer o rebase
14. No nosso caso, estamos alterando o histórico da nova-branch em relação à branch master, por isso, usaremos o comando **git rebase master**
15. Observe que neste momento as alterações que foram realizadas na master (no caso, a adição do título no arquivo1) já aparece no contexto da branch nova-branch, observe o resultado na figura a seguir

CONTROLE DE BRANCH COM REBASE



Mudamos para a nova-branch e realizamos o rebase dela com a branch master, observe

The screenshot shows a terminal window with the following session:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git checkout nova-branch
Switched to branch 'nova-branch'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (nova-branch)
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: adiciona arquivo2 na nova-branch
```

Neste campo você poderá observar que as alterações realizadas na branch master já estão no contexto da nova-branch

Observe o resultado do comando `git log --graph --oneline`

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (nova-branch)
$ git log --graph --oneline
* 9cfdb34c (HEAD -> nova-branch) adiciona arquivo2 na nova-branch
* 916f2b1 (master) altera o título do arquivo1
* 706a02d first commit
```

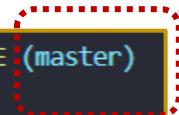


CONTROLE DE BRANCH COM REBASE

16. Retorne agora para a branch master e realize o merge, você deverá observar que o merge agora será do tipo Fast-Forward, observe:



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git merge nova-branch
Updating 916f2b1..9cf34c
Fast-forward
  arquivo2.md | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 arquivo2.md
```



17. Assim, não teremos mais a árvore, observe:



```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git log --oneline --graph
* 9cf34c (HEAD -> master, nova-branch) adiciona arquivo2 na nova-branch
* 916f2b1 altera o título do arquivo1
* 706a02d first commit
```

Observe que tudo se passa como se não existisse a nova-branch



CONTROLE DE BRANCH COM REBASE

18. Mas, não se iluda, é só uma impressão, na verdade, a branch nova-branch continua lá, isto é, ela não foi removida, observe:

Como pode observar a nova-branch continua em seu projeto, para removê-la deverá usar o processo tradicional

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE (master)
$ git branch
* master
  nova-branch
```





REBASE & CONFLITOS

Como no merge podemos encontrar situações de conflitos
com o REBASE → vamos resolver!

Vamos pra tica

REBASE & CONFLITOS



1. Indo direto ao ponto, crie um novo folder e gere um novo arquivo (arquivo.md)
2. Crie a branch nova-branch e altere o arquivo.md adicionando um título a ele
3. Para gerar um conflito, retorne para a branch master e no mesmo arquivo.md altere o título
4. Vamos agora estando na master realizar um merge para tratarmos desse conflito



```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<< HEAD (Current Change)
2 # titulo legal
3 =====
4 # titulo
5 >>>> nova-branch (Incoming Change)
6 |
```

REBASE & CONFLITOS



5. Feitas as alterações vamos realizar agora um novo commit que será o commit que irá apontar a resolução do conflito

Commit realizado
situação de conflito
resolvida

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (master|MERGING)
$ git commit -am 'resolve conflito'
[master 71e44d0] resolve conflito

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (master)
$ []
```

Observe que
precisamos do
commit para
resolvermos o conflito

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (master)
$ git log --graph --oneline
* 71e44d0 (HEAD -> master) resolve conflito
|\ 
* e465cb1 (nova-branch) alteração no título
* | ba68946 alterando título na master
|/
* ca8fb98 first commit
```

REBASE & CONFLITOS



- Vamos agora repetir tudo usando o comando rebase, é com você pratique, não se esqueça que o rebase deverá acontecer dentro da nova-branch:

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch)
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: altera titulo na nova-branch
Using index info to reconstruct a base tree...
M      arquivo.md
Falling back to patching base and 3-way merge...
Auto-merging arquivo.md
CONFLICT (content): Merge conflict in arquivo.md
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001 altera titulo na nova-branch
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
```

- Imaginando que fez tudo correto use o comando **git rebase master**, e observe o conflito gerado. Vamos trabalhar agora o conflito do rebase, observe a figura na página a seguir:

REBASE & CONFLITOS



7. Observe que você terá o mesmo cenário de antes, só que agora estará no rebase e deverá tratar do conflito de forma manual, o mesmo processo observe, **com a ressalva que não precisaremos realizar nenhum commit**, em outras palavras, o processo será automático:

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<< HEAD (Current Change)
2 # titulo legal
3 =====
4 # titulo
5 >>>> altera titulo na nova-branch (Incoming Change)
6
```

REBASE & CONFLITOS



8. Atenção, agora deveremos salvar a alteração e adicioná-la à Stage, **mas não** vamos realizar o commit e **sim** continuaremos com o comando rebase que foi automaticamente interrompido

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch|REBASE 1/1)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch|REBASE 1/1)
$ [ ]
```

Observe
que nosso
rebase esta
aberto

9. Para continuar o rebase usamos o comando **git rebase --continue**; você também poderia usar os comandos **git rebase --skip** (para aceitar tudo o que está vindo da master) ou ainda **git rebase --abort** (aborta o processo de rebase)

REBASE & CONFLITOS



10. Observe que o rebase foi finalizado com sucesso

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch|REBASE 1/1)
$ git rebase --continue
Applying: altera titulo na nova-branch

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch)
$ []
```



```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch)
$ git log
commit 40e467c16ab621055dbcba8ba61d32c3b21e9a (HEAD -> nova-branch)
Author: accolombini <accolombini@gmail.com>
Date:   Wed Mar 25 16:58:35 2020 -0300

    altera titulo na nova-branch

commit 20449eab6a00262bf6fcfc045977ead14d1aa77d4 (master)
Author: accolombini <accolombini@gmail.com>
Date:   Wed Mar 25 17:00:13 2020 -0300

    altera titulo na master

commit 812c7a5437f553fd87c55d11315d12a19efa1528
Author: accolombini <accolombini@gmail.com>
Date:   Wed Mar 25 16:57:24 2020 -0300

    first commit
```

Observe que todo Project History está organizado e não existe o commit de merge

REBASE & CONFLITOS



11. Para finalizar voltemos à master para integrar esta branch, observe que a merge foi do tipo Fast-Forward

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (nova-branch)
$ git checkout master
Switched to branch 'master'

acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (master)
$ git merge nova-branch
Updating 20449ea..40e467c
Fast-forward
  arquivo.md | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
```

11. Para visualizar a diferença, vamos olhar para a árvore, esperamos não encontrar nenhuma ramificação

Observe que tudo se passa como se todas as ações houvessem acontecido na branch master

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_CONFLITOS (master)
$ git log --graph
* commit 40e467c16ab621055dbcba8ba61d32c3b21e9a (HEAD -> master, nova-branch)
| Author: accolombini <accolombini@gmail.com>
| Date:   Wed Mar 25 16:58:35 2020 -0300
|
|     altera titulo na nova-branch
|
* commit 20449eb6a00262bf6fc0e5977ead14d1aa77d4
| Author: accolombini <accolombini@gmail.com>
| Date:   Wed Mar 25 17:00:13 2020 -0300
|
|     altera titulo na master
|
* commit 812c7a5437f553fd87c55d1315d12a19efa1528
| Author: accolombini <accolombini@gmail.com>
| Date:   Wed Mar 25 16:57:24 2020 -0300
|
|     first commit
```



REBASE ITERATIVO

Vamos praticar - Laboratório

REBASE ITERATIVO



- Para acessar o rebase iterativo usamos a opção **rebase -i**
- Essa opção permite:
 - Mover
 - Editar
 - Juntar commits, enfim trabalhar no nosso Project History
- Só relembrando, não use essa opção se sua branch já estiver compartilhada, use apenas nas Branches locais

REBASE ITERATIVO



1. Reproduza o cenário tal como descrito na figura a seguir:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git log --graph --oneline
* e1fdbf0 (HEAD -> master) adiciona arquivo3
*   6d31138 Merge branch 'nova-branch'
|\
| * b68c6ab (nova-branch) adiciona titulo ao arquivo
* | 4b061dc adiciona titulo o arquivo2
|/
* 4e00ef0 adiciona arquivo2 ao projeto
* d583fb0 first commit
```

2. Para este laboratório preciso que se recorde do editor de texto que definiu como padrão, no meu caso, vou usar o Notepad++

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git config --global -l
user.name=accolombini
user.email=accolombini@gmail.com
core.editor=notepad++
credential.helper=wincred
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
filter.lfs.clean=git-lfs clean -- %f
```

REBASE ITERATIVO



3. Vamos iniciar nosso laboratório alterando a mensagem do commit 2. Para fazer isso precisamos acessar o commit pai (commit anterior) → no nosso caso apontamos para o first commit, usaremos o comando **git rebase -i <identificado-commit-pai>**

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git rebase -i d583fb
hint: Waiting for your editor to close the file... unix2dos: converting file D:/Users/Angelo/REBASE_ITERATIVO/.git/rebase-merge/git-rebase-todo to DOS format.
```

3. Neste momento o Git irá abrir o editor que escolheu para que você possa realizar as alterações

```
git rebase -i d583fb
```

The screenshot shows a terminal window with the command `git rebase -i d583fb` entered. A green hand icon points to the terminal window, and a red dashed circle highlights the command line. The terminal output shows the rebase editor with several commits listed:

```
git-rebase-todo - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
pick 4e00ef0 adiciona arquivo2 ao projeto
pick 4b061dc adiciona titulo o arquivo2
pick b68c6ab adiciona titulo ao arquivo
pick e1fdbf0 adiciona arquivo3

# rebase d583fb...e1fdbf0 onto e1fdbf0 (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
$ git rebase -i d583fb
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
```

REBASE ITERATIVO



4. Agora basta realizar a alteração e salvar o arquivo, ao salvar, automaticamente o rebase volta ao status de execução e sempre que encontrar algum ponto de iteratividade o editor será aberto e você será solicitado a prover as alterações. Neste caso substitua apenas **pick** por **r (reword)** e salve o arquivo e feche o Notepad, veja:

```
r 4e00ef0 adiciona arquivo2 ao projeto
pick 4b061dc adiciona titulo o arquivo2
pick b68c6ab adiciona titulo ao arquivo
pick e1fdbf0 adiciona arquivo3

# Rebase d583fb..e1fdbf0 onto e1fdbf0 (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
```

REBASE ITERATIVO



5. Observe que o Notepad se abre novamente solicitando que entre com uma nova mensagem para o commit, observe:

Mensagem alterada

```
*COMMIT_EDITMSG - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
adicionando o arquivo2

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:       Wed Mar 25 18:09:04 2020 -0300
#
# interactive rebase in progress; onto d583fb
# Last command done (1 command done):
"
```

6. Só para confirmar veja na figura a seguir o resultado do rebase com a mensagem alterada

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git log --graph --oneline
* 596d546 (HEAD -> master) adiciona arquivo3
* fe61548 adiciona titulo ao arquivo
* 8132b19 adiciona titulo o arquivo2
* e507c39 adicionando o arquivo2
* d583fb first commit
```

REBASE ITERATIVO

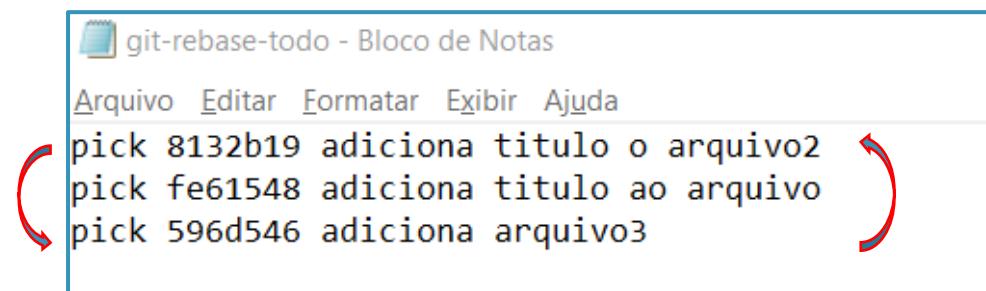


7. Vamos agora usar o comando rebase para alterar a ordem de execução do commit, acompanhe e tente perceber o que está sendo alterado
8. O primeiro passo foi usar o rebase com o hashCode pai que queremos trabalhar, isso irá posicionar o commit para o próximo passo

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git rebase e507c39
Current branch master is up to date.
```

7. Agora o processo se repete

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git rebase -i e507c39
hint: Waiting for your editor to close the file... unix2dos: converting file D:/Users/Angelo/REBASE_ITERATIVO/.git/rebase-merge/git-rebase-todo to DOS format.
..
```



REBASE ITERATIVO



8. Para finalizar vamos fazer um rebase para unir dois commits, para isso, usaremos a opção squash, observe:

Para isso
usaremos a
opção squash

```
git-rebase-todo - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
pick 0082865 adiciona titulo o arquivo2
pick 40ff5a6 adiciona titulo ao arquivo

# Rebase 2407dea..40ff5a6 onto 2407dea (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
```

REBASE ITERATIVO



9. Finalizando uma nova página com seu bloco de notas será aberta, solicitando que insira a nova mensagem, faça isso, salve e feche o seu editor

Substitua tudo
isso pela sua
mensagem
salve e esta
feito

```
COMMIT_EDITMSG - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
# This is a combination of 2 commits.
# This is the 1st commit message:
adiciona titulo o arquivo2

# This is the commit message #2:
adiciona titulo ao arquivo

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Mar 25 18:10:41 2020 -0300
#
```

10. Como resultado, observe que temos agora um commit a menos (4)

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/REBASE_ITERATIVO (master)
$ git log --oneline
dac19c1 (HEAD -> master) adicionando títulos aos arquivos
2407dea adiciona arquivo3
e507c39 adicionando o arquivo2
d583fb1 first commit
```

Fork WorkFlow

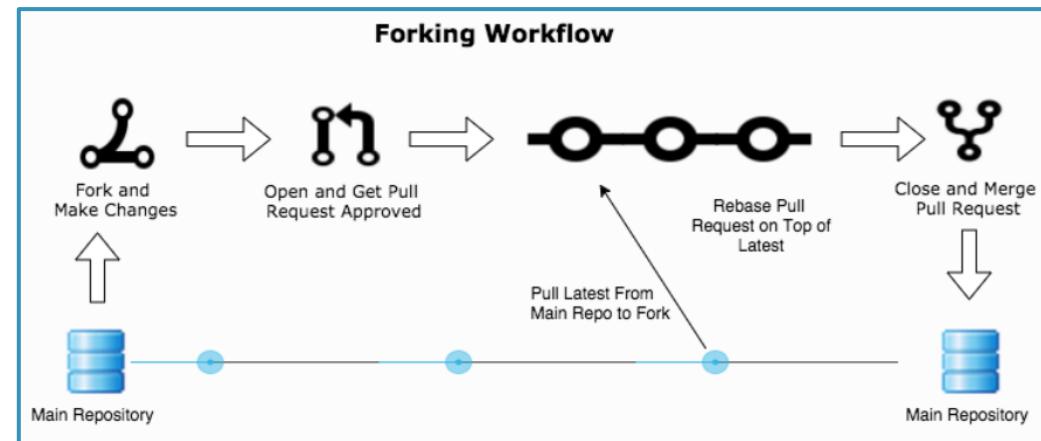
Como podemos
colaborar com
Projetos???





Fork Workflow

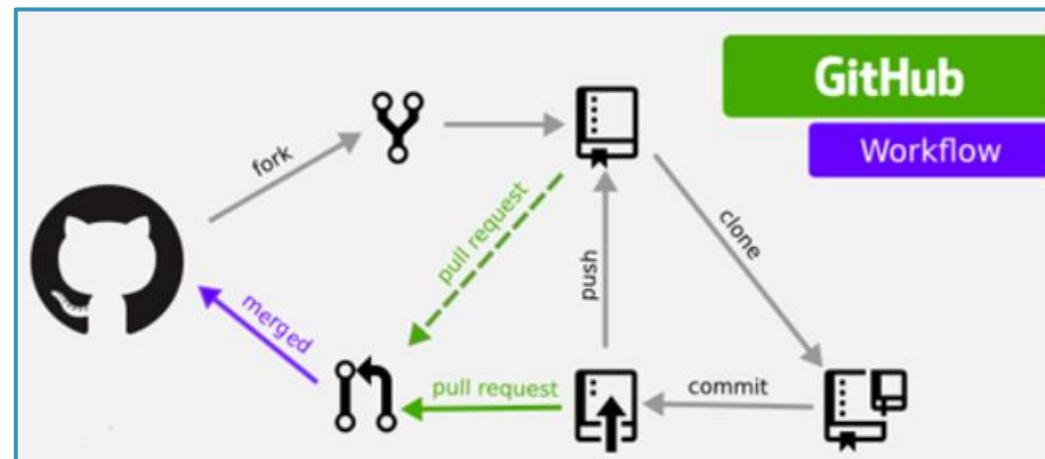
- Trata-se de uma das ferramentas de uso mais frequente do GitHub para trabalhos colaborativos
- Para trabalhar com Fork Workflow você precisará recordar seus conhecimentos de:
 - a) Merge & Conflitos
 - b) Branches
 - c) Rebase
 - d) Issues





Fork Workflow

- Basicamente trabalharemos com Pull Request
 - a) Enviar alterações para serem incorporadas à um projeto
 - b) Discussões sobre código
 - c) Integrar código revisado ao projeto
 - d) Discussões sobre partes específicas do projeto
 - e) Discussões sobre um determinado capítulo de um livro





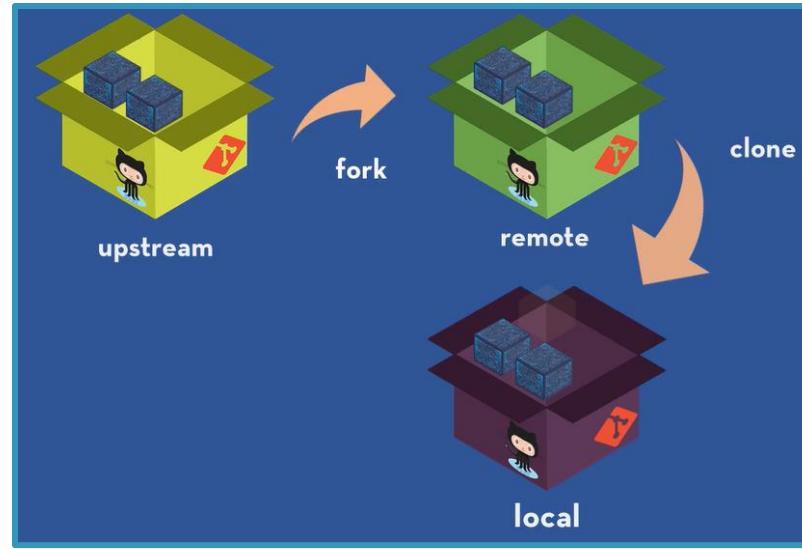
Fork Workflow

- Você faz um Pull Request quando deseja que uma determinada alteração que realizou seja incorporada a um projeto de terceiros
- O dono do repositório poderá avaliar seu pedido, sugestão ou seu código e decidir se vai ou não incorporar ao projeto original
- O dono do repositório poderá também solicitar algumas alterações em sua proposta para que se alinhe melhor com a proposta de seu projeto
- Por fim, cabe ao dono do projeto, caso queira, integrar suas alterações ao projeto dele



Fork Workflow

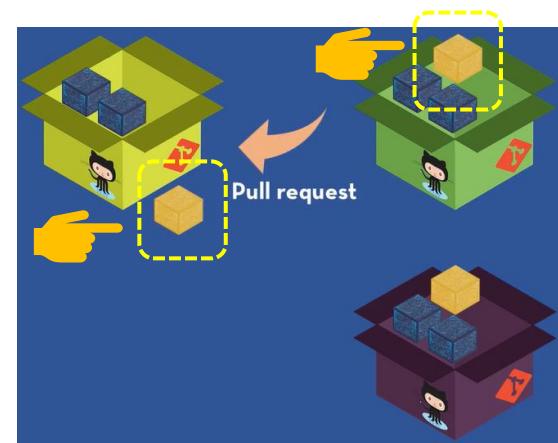
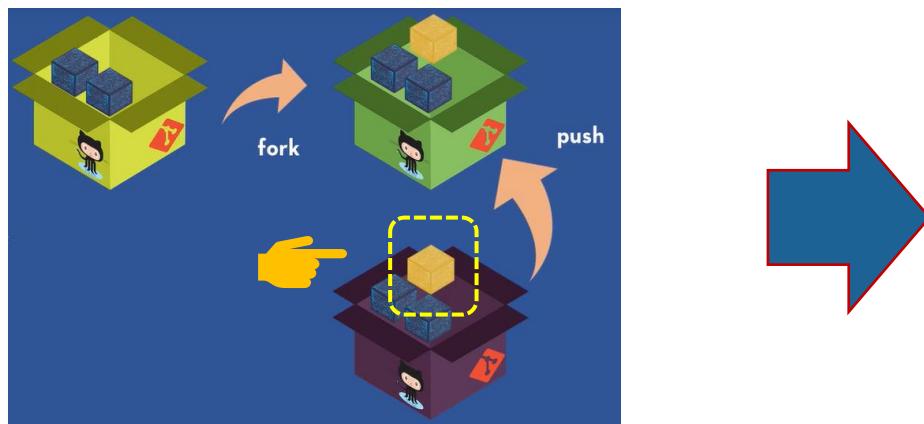
- Ok, tudo certo, mas vamos agora entender como funciona o fluxo básico disso tudo
- Todo começa com nosso processo de Fork tradicional, para recordar:





Fork Workflow

- A partir daí você pode **identificar** a alteração desejada, realizar a proposta/encaminhamento, **fazer um pull para o seu repositório Fork e de lá fazer um Pull Request para o repositório Upstream**, observe o fluxo:





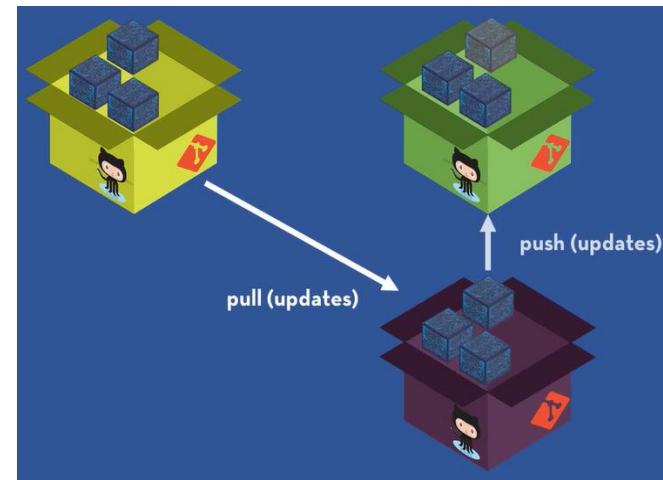
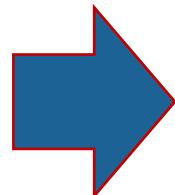
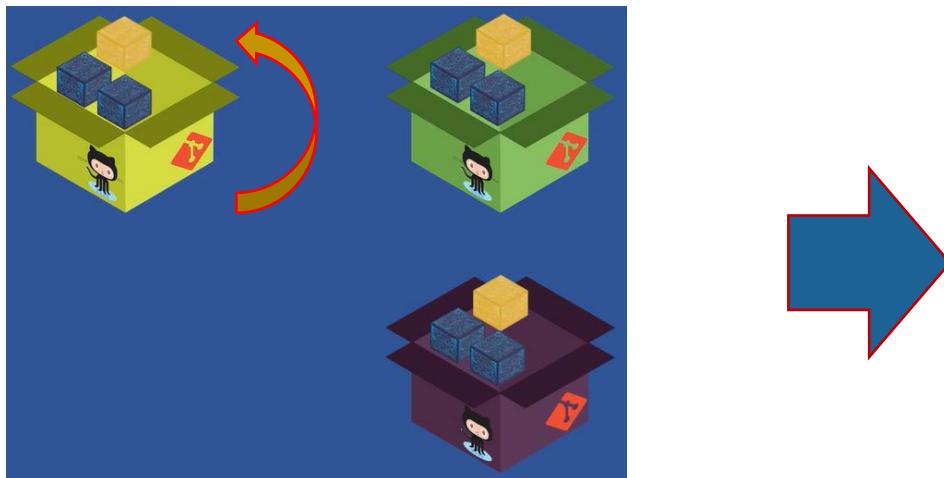
Fork Workflow

- Quando o Pull Request é feito, o dono do projeto Upstream receberá sua solicitação e também a Branch que você utilizou para desenvolver sua proposta
- O interessante disso tudo é que você poderá continuar trabalhando sem problemas na Branch e realizar push de atualização sem problemas, pois o GitHub estará rastreando a Branch e não o Pull Request
- Caso o dono do projeto Upstream decida aceitar sua proposta ela será incorporada ao projeto Upstream, observe



Fork Workflow

- Uma vez integrado ao projeto ela passará a fazer parte do projeto principal Upstream, e caberá a você fazer a sincronia através do seu repositório Fork observe:





Fork Workflow

- Só para revisar e praticar em seguida, vamos lá:
 1. Fork
 2. git clone (para seu repositório local)
 3. git checkout -b <nome-da-branch>
 4. git add .
 5. git commit -m 'mensagem'
 6. git push
 7. Pull Request
 8. git pull upstream (se as alterações forem integradas)
 9. git push (se as alterações forem integradas)





COLABORAÇÃO COM FORK WORKFLOW

Vamos praticar - Laboratório



Fork Workflow

1. Para esse laboratório, vou me logar como accolombinifake e trabalhar com um repositório de terceiros desenvolvido para essa prática, então vamos começar
2. Embora este repositório esteja limpo, uma das formas mais interessantes para participar de um repositório de terceiro é dar uma olhada nas ISSUES, nelas você poderá identificar questões e problemas e encontrar algo que possa contribuir, no caso, não temos ISSUES



Fork Workflow

3. Só para acompanharmos a simulação, vamos no passo-a-passo. Acessando o repositório de terceiro e olhando para as ISSUES

A screenshot of a GitHub repository page. The URL in the address bar is 'github.com/fakebmatchete/git-pratica-v2/issues/10'. The top navigation bar shows several projects like Disney, ESPN, iCloud, Wikipedia, and Cursos online - a quick start guide. Below the bar, there are tabs for Pull requests, Issues, Marketplace, and Explore. A large green button says 'Read the guide'. At the bottom of the page, there's another navigation bar with the repository name 'fakebmatchete / git-pratica-v2' and tabs for Code, Issues (which is highlighted with a red box), Pull requests, Actions, Projects, Wiki, and Security.

A screenshot of the 'Issues' tab for the repository 'fakebmatchete / git-pratica-v2'. The tab is highlighted with a red box. The first issue is titled 'Dica acadêmica #10' and has a yellow hand icon pointing to it. Below the title, it says '(Open)' and 'acolombinifake opened this issue · new · 0 comments'. The rest of the page includes tabs for Code, Pull requests, Actions, Projects, Wiki, and Security.

Na ISSUE escolhida observamos que se trata da #10 e o tema Dica acadêmica nos interessa



Fork Workflow

4. Vamos imaginar que exista uma ISSUE de interesse, por exemplo a ISSUE 1, e vamos desenvolver mais um arquivo com dicas sobre git
5. Passo 1 → criar o Fork com esse repositório (você já sabe como fazer isso)

Fork
realizado!

The screenshot shows a GitHub repository page for 'accolombinifake / git-pratica-v2'. A red dashed box highlights the repository name and the 'forked from fakebmarchete/git-pratica-v2' text. A blue box surrounds the entire header area. Below the header, there's a message: 'No description, website, or topics provided.' and a 'Manage topics' link. Navigation links for 'Code', 'Pull requests 0', 'Actions', and 'Projects' are visible at the top.



Fork Workflow

6. Passo 2 → clonando o Fork para seu repositório local (você já sabe como fazer isso)
7. Feito o clone, crie um novo arquivo, no meu caso criei o arquivo Dica6.md (escreve qualquer coisa) salve o arquivo. Atenção não use add . E nem commit aqui. Lembra, devemos criar uma nova Branch para trabalhar com repositórios de terceiros que queiramos contribuir
8. Passo 3, 4 e 5 → crie uma nova Branch (ISSUE1), add . E commit seu arquivo, veja na próxima figura:



Fork Workflow

- Repita os passos como a figura abaixo:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/FORK_WORKFLOW/git-pratica-v2 (master)
$ git checkout -b issue-1
Switched to a new branch 'issue-1'

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/FORK_WORKFLOW/git-pratica-v2 (issue-1)
$ git status
On branch issue-1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Dica-6.md

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

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/FORK_WORKFLOW/git-pratica-v2 (issue-1)
$ git add .

accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/FORK_WORKFLOW/git-pratica-v2 (issue-1)
$ git commit -m 'dica 6 criada para #10'
[issue-1 9e6645a] dica 6 criada para #10
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Dica-6.md
```



Fork Workflow

9. Passo 6 → precisamos agora fazer um push para nosso repositório Fork. Mas, fique atento, queremos enviar, para nosso repositório Fork a nossa Branch ISSUE1, observe:



```
acc10@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/FORK_WORKFLOW/git-pratica-v2 (issue-1)
$ git push origin issue-1
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 265 bytes | 265.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'issue-1' on GitHub by visiting:
remote:     https://github.com/accolombinifake/git-pratica-v2/pull/new/issue-1
remote:
To https://github.com/accolombinifake/git-pratica-v2.git
 * [new branch]      issue-1 -> issue-1
```



Fork Workflow

10. Executado o push, vamos agora no GitHub e conferir se nossa Branch, observe:

The screenshot shows a GitHub repository page for a forked repository named 'git-pratica-v2'. A yellow hand icon points to the repository name 'git-pratica-v2' on the left. Another yellow hand icon points to the title bar of the main repository page, which displays the forked repository 'accolombinifake / git-pratica-v2'. A large gray callout bubble with a yellow border contains the text: 'Observe que no GitHub eu tenho uma notificação de uma nova BRANCH'. A red arrow points from this callout bubble towards the 'Your branches' section on the right side of the page. This section lists a single branch named 'issue-1' with a status of 'Updated 9 minutes ago by accolombinifake'. A 'New pull request' button is visible at the bottom of this list.



Fork Workflow

11. No repositório Fork na Branch que acabamos de subir vamos realizar nosso pedido de Pull Request, observe:

Active branches

issue-1 Updated 19 minutes ago by accolombinifake 0 | 1

New pull request

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base repository: fakebmarchete/git-pratica-v2 base: master head repository: accolombinifake/git-pratica-v2 compare: issue-1

Able to merge. These branches can be automatically merged.

dica 6 criada para #10

Write Preview AA B i “ “ @

Leave a comment

Helpful resources
GitHub Community Guidelines

Esta é uma mensagem importante, observe que estamos habilitados a fazer um merge com o repositório principal Upstream



Fork Workflow

Importante: procure garantir que sempre estará oferecendo um merge do tipo **Fast Forward**, ou melhor, sem conflitos, pois o dono do repositório dificilmente se habilitará a resolver problemas de conflitos, sendo assim, assegure-se que esteja tudo ok

12. Criado o Pull Request é boa prática deixar uma mensagem:

A screenshot of the GitHub 'Open a pull request' interface. The title bar says 'Open a pull request'. Below it, there's a note: 'Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.' A green checkmark indicates 'Able to merge. These branches can be automatically merged.' The main area shows a message from a user named 'dica 6 criada para #10': 'Criei a dica 6, espero que aprovem'. A yellow hand icon with a red dashed box points to the message body. On the right side, there are 'Helpful resources' and a link to 'GitHub Community Guidelines'.



Fork Workflow

- Ao criar o Pull Request o GitHub pega essa nossa Branch e associa ao Repositório principal Upstream
 - Agora, o dono do Repositório principal Upstream têm acesso à Branch com todos os commits que realizamos
13. Observe que fomos direcionados para o repositório principal Upstream e temos uma Pull Request criada a figura a seguir:

The screenshot shows a GitHub pull request page. At the top, there's a red dashed box around the repository name 'fakebmarchete / git-pratica-v2'. Below it, another red dashed box surrounds the pull request title 'dica 6 criada para #10 #11'. A yellow hand icon points to the repository name. Another yellow hand icon points to the pull request title. The main content area shows a green 'Open' button, a message from 'acolombinifake' wanting to merge '1 commit' from 'acolombinifake:master' into 'fakebmarchete:master', and a comment from 'acolombinifake' saying 'Criei a dica 6, espero que aprovem'. On the right side, there are sections for 'Reviewers' (No reviews) and 'Assignees' (No one assigned). The commit hash '9e6645a' is also visible.



Fork Workflow

14. A partir daí o dono do Repositório Upstream poderá aceitar integrando nossas sugestões ao projeto atendendo à Pull Request, solicitar alguma alteração/implementação ou simplesmente realizar algum comentário
15. Se pudéssemos acessar o Repositório Upstream como o dono do Repositório, observe na simulação, como seria seu ambiente de trabalho

The screenshot shows a GitHub repository page for 'fakebmarchete / git-pratica-v2'. The top navigation bar includes 'Watch 0', 'Star 0', and 'Fork 8'. Below the bar, there are tabs for 'Code', 'Issues 1', 'Pull requests 3' (which is highlighted with a red dashed box), 'Actions', 'Projects 0', 'Wiki', 'Security', and 'Insights'. A search bar contains the filter 'is:pr is:open'. To the right are buttons for 'Labels 8', 'Milestones 0', and a green 'New pull request' button. Below the search bar, it shows '3 Open' and '5 Closed' pull requests. A specific pull request is highlighted with a red dashed box: '#11 opened 17 minutes ago by accolombinifake • Review required'. A tooltip from a callout bubble says 'dica 6 criada para #10'. The callout bubbles contain the following text:

- Ele terá a opção de Pull Request habilitada
- E a Pull Request com o usuário que a gerou



Fork Workflow

16. Observe que acessando como dono do Repositório, do próprio GitHub é possível visualizar a proposta que está sendo feita bem como o dono da proposta de Pull Request, observe como:

The screenshot shows a GitHub pull request interface. The title of the PR is "dica 6 criada para #10 #11". A green "Open" button is visible. Below it, the message "acolombinifake wants to merge 1 commit into fakebmarchete:master from accolombinifake:issue-1" is displayed. The commit details show a commit titled "dica 6 criada para #10" made by "acolombinifake" 3 hours ago. The commit hash is "commit 9e6645a69d86c049a5cb9e88fd6a1f4b62e8c9f4". A yellow hand icon points to the commit message area, which is highlighted with a red dashed circle. The interface includes tabs for Conversation (0), Commits (1), Checks (0), Files changed (1), and a "Review changes" button.



Fork Workflow

17. Observe que o dono do Repositório Upstream poderá fazer algum comentário, aprovar ou solicitar algumas alterações

The screenshot shows a GitHub pull request page for a repository named 'fakebmarchete / git-pratica-v2'. The pull request is titled 'dica 6 criada para #10 #11' and is marked as 'Open'. It shows one commit from 'acolombinifake' with the message 'dica 6 criada para #10'. The commit was made 3 hours ago and includes a file named 'Dica-6.md'. The pull request has 1 file changed. A red dashed circle highlights the 'Review changes' section at the bottom right of the pull request interface. Two yellow arrows point to the 'Comment' and 'Approve' buttons within this section. The 'Comment' button is selected.

dica 6 criada para #10 #11

Open accolombinifake wants to merge 1 commit into fakebmarchete:master from accolombinifake:issue-1

Conversation 0 Commits 1 Checks 0 Files changed 1

Changes from all commits ▾ File filter... ▾ Jump to... ▾ Review changes

dica 6 criada para #10

acolombinifake committed 3 hours ago

0 Dica-6.md

No changes.

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Comment Submit general feedback without explicit approval.

Approve Submit feedback approving these changes.

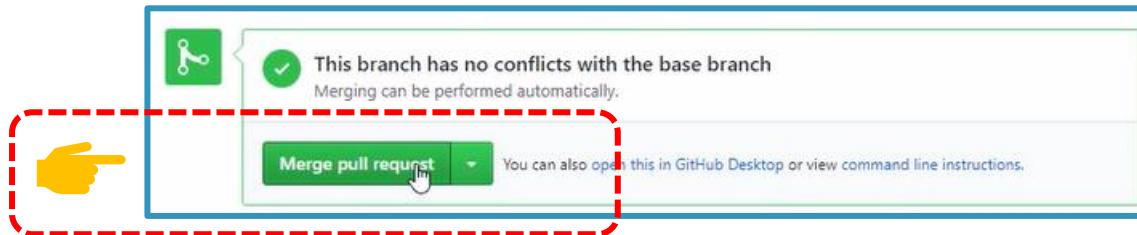
Request changes Submit feedback suggesting changes.

Submit review



Fork Workflow

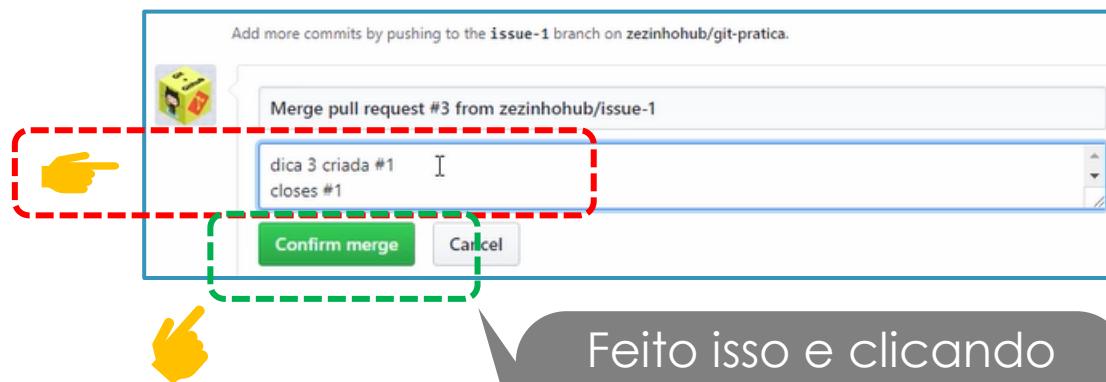
18. No momento em que tudo estiver ok, basta o dono do repositório Upstream realizar o merge. O merge de uma Pull Request é o equivalente a fazer um merge para minha Branch Master, desta forma, toda proposta enviada por sua Branch através do Pull Request, será incorporada à Branch Master do projeto Upstream





Fork Workflow

19. A aceitar a Pull Request, o dono do repositório deverá inserir uma mensagem e neste momento aproveitando da opção **close #<ISSUE>** encerrar a ISSUE que gerou todo esse processo. [Esse é o processo típico de se trabalhar em colaboração utilizando Git GitHub](#)



Feito isso e clicando em **Confirm merge** a ISSUE está encerrada e a Pull Request incorporada

PULL REQUEST MERGE & CONFLITOS

The GitHub logo, which consists of the word "GitHub" in its signature white font, centered within a white octocat icon on a gray background.

Vamos nos aprofundar um pouco mais!!!



Pull Request Merge & Conflitos

- Para este laboratório, vamos criar um cenário iterativo
- Escolham um parceiro, cada um com seu repositório no GitHub
- Definam quem será o dono do Re却itório Upstream, este deverá criar um repositório e nele criar dois arquivos quaisquer contendo dois parágrafos (pode usar algum texto pronto da internet)
- O outro será o dono do Fork ou aquele que fará a solicitação do Pull Request
- O processo é o mesmo conhecido de vocês, a única coisa que irei influenciar criando uma situação de conflito para que resolvam.
- Neste laboratório, não demonstrarei as soluções, é com vocês, agora já são usuários avançados do Git GitHub, então ao trabalho



PULL REQUEST MERGE & CONFLITOS

Vamos praticar - Laboratório



Pull Request Merge & Conflitos

1. No repositório Upstream crie uma ISSUE solicitando que se complemente um dos documentos gerados
2. O responsável pelo Fork deverá acessar o repositório Upstream, ler a ISSUE
3. Faça o FORK, crie o Repositório local e sincronize os repositórios, tudo normal até aqui
4. Como não precisamos sincronizar na prática anterior, vou aqui repetir o comando só como propósito de revisão, dúvidas, consulte as aulas anteriores



Pull Request Merge & Conflitos

5. Para sincronizar seu repositório com o repositório Upstream use o comando **git remote add upstream <end-do-rep-
Upstream>**
6. Não se esqueça de que para realizar o sincronismo você deverá acessar a sua master, pois é lá que queremos estar sincronizados, então **git checkout master**
7. Como queremos a Branch Master que está no Upstream, usamos o comando **git pull upstream master**
8. Como dica, use o comando **git log --graph --oneline** para acompanhar como foram feitas as alterações no Re却itório Upstream



Pull Request Merge & Conflitos

9. A partir daí crie uma nova Branch e siga em frente para tratar da questão sugerida pela ISSUE escolhida
10. Neste ponto iremos criar uma situação de conflito
 - a) O dono do repositório Fork trabalhará no arquivo definido pela ISSUE escolhida adicionando um parágrafo qualquer. Faça isso, realize todos os passos, salve, adicione na Stage realize o commit (lembra de referenciar a #ISSUE) e faça o push para seu repositório FORK → use **git push origin <nome-branch-griada>**
 - b) Feito isso → faça o pedido de Pull Request
 - c) Tudo normal até, mas neste meio tempo, o dono do Re却itório Upstream, ou até mesmo outro colaborador deverá fazer uma alteração no mesmo arquivo, gerando assim o conflito



Pull Request Merge & Conflitos

10. Continuação

- d) Você observará o problema acessando o repositório Upstream e observando se a mensagem de merge está habilitada, observe



- e) Cabe ao dono do Fork a solução do conflito → o dono do Repositório Upstream deverá usar o campo de mensagem notificando que há necessidade de alteração para que se possa seguir em frente com o Merge



Pull Request Merge & Conflitos

11. Para resolver esse problema o dono do Fork deverá retornar ao seu repositório local e faça o sincronismo com o diretório Upstream. **Detalhe importante**, o sincronismo agora deverá ser na **Branch criada para o trabalho e não mais na Master** (fique atento)
12. Para isso vá à Branch de trabalho com o comando **git checkout <nome-da-branch>** e realize o pull para sincronizar com o Upstream **git pull upstream master** observe que uma mensagem de conflito apontará para o problema a ser tratado. **Note que** estou sincronizando com a Branch Master do repositório Upstream



Pull Request Merge & Conflitos

13. Só para ilustrar observe como deverá ser a mensagem que receberá



```
$ git pull upstream master
remote: Counting objects: 3, 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), done.
From https://github.com/fakebmarchete/git-pratica
 * branch           master      -> FETCH_HEAD
   3991493..db61dfc master    -> upstream/master
Auto-merging Dica1.md
CONFLICT (content): Merge conflict in Dica1.md
Automatic merge failed; fix conflicts and then commit the result.
```

14. Imediatamente será aberto o arquivo para resolver o conflito detectado. Resolva-o



```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Cor
3 <<<< HEAD (Current Change)
4 0 git não é nenhum bicho de 7 cabeças
5 =====
6 0 git foi desenvolvido com muito carinho!
7 >>>> db61dfc55d42b76223b05aa7842586d9a459c53a (Incoming
8
```



Pull Request Merge & Conflitos

15. Resolvido o conflito faça os passos → adicione na Stage, realize o commit e faça um push com o origin agora dessa Branch, use **git push origin <nome-da-branch>**
16. Observe que o push foi feito para seu repositório Fork, mas como este repositório está atrelado ao repositório Upstream pelo Pull Request essa alteração automaticamente será refletida no Re却itório Upstream
17. Acompanhem esse processo consultando o Re却itório Upstream

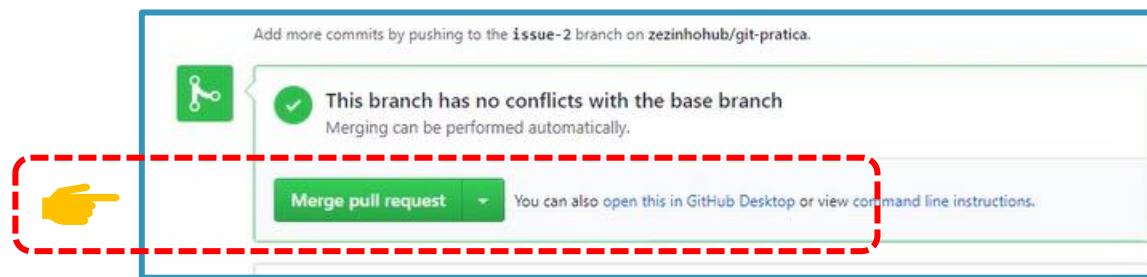


Pull Request Merge & Conflitos

18. Observe agora que a mensagem do Merge estará sinalizando que não existe mais conflito, você busca por algo como:



19. Agora realize o último passo, que o merge, você responsável pelo Upstream poderá aceitar o Merge e finalizar a ISSUE, observe:





REBASE COMO ALTERNATIVA AO MERGE

Vamos praticar - Laboratório



Pull Request Merge & Conflitos

- Para este laboratório, vamos criar um cenário iterativo
- Escolham um parceiro, cada um com seu repositório no GitHub
- Definam quem será o dono do Re却itório Upstream, este deverá criar um repositório e nele criar dois arquivos quaisquer contendo dois parágrafos (pode usar algum texto pronto da internet)
- O outro será o dono do Fork ou aquele que fará a solicitação do Pull Request
- O processo é o mesmo conhecido de vocês, a única coisa que irei influenciar criando uma situação de conflito para que resolvam.
- Vocês devem ter observado que trabalhar com o Merge pode acabar poluindo um pouco seu projeto, sendo assim, vamos repetir o laboratório anterior só que neste caso, faremos uso do REBASE ao invés do MERGE, então ao trabalho.



Pull Request Merge & Conflitos

1. Construa o cenário como no laboratório anterior, agora seria interessante trocar, quem foi dono do Upstream agora deve ser responsabilizar pelo Fork e vice versa
2. Crie um novo projeto e gere pelo menos dois arquivos
3. Crie uma ISSUE → enfim todo passo-a-passo anterior até o sincronismo da sua Branch master, para que a partir da sua master sincronizada você possa criar uma nova branch e realizar as alterações propostas pela ISSUE que escolheu trabalhar

Se você esquecer de ajustar a Master será muito mais difícil seu processo de colaboração, então fique atento!



Pull Request Merge & Conflitos

4. Com sua master sincronizada você estará pronto para iniciar os trabalhos
5. Repita os passos altere um arquivo, uma linha basta e faça todo processo como no laboratório anterior
6. Para simular o que acontece no mundo real, use o comando git log --graph --oneline para visualizar a situação, em projetos reais, você encontrará uma situação densa de informações commits etc

É importante que se conscientize disso para entender como o processo com o Rebase é mais interessante para o dono do repositório



Pull Request Merge & Conflitos

- Este exemplo nem de longe reflete a realidade, mas já dá indícios de onde isso pode chegar

```
$ git log --oneline --graph
*   4ac5dc6 Merge pull request #4 from zezinhohub/issue-2
| *   c10997a ajuste final em dica1 #2
| |
| |
| *
| |   db61dfc Alteração de dica #2
| |   e4c5927 alteração em dica #2
| |
| *
| |   3991493 Merge pull request #3 from zezinhohub/issue-1
| |
| *   a9908cf dica 3 criada #1
| |
| *
| |   3ab074c Segunda dica sobre git
| |   6010b51 Primeira dica sobre git
| |
| *   b7bcd9d Initial commit
```

- Com o Rebase desejamos reduzir esse processo minimizando ao máximo essa “poluição”



Pull Request Merge & Conflitos

9. Depois de feito as alterações faça agora seu pedido de Pull Request
10. Não se esqueça de realizar alterações no Repositório Upstream e de commitá-las antes de nosso Pull Request (queremos um conflito lembra)

Atenção: Agora seria o momento de você fazer o push, mas não devemos fazer isso, lembra estamos trabalhando com Rebase, precisamos ajustar para resolver a questão, não se esqueça que não devemos fazer Rebase em Issues que já foram compartilhadas. Logo, antes de mais nada crie uma nova Branch a partir da Branch que está compartilhando



Pull Request Merge & Conflitos

11. Crie sua nova Branch a partir de sua ISSUE de trabalho → **git checkout -b <nome-da-Issue-v2>** (usei o v2 significando que estamos trabalhando numa ISSUE que foi compartilhada em sua versão anterior)
12. Na Branch criada inicie o processo de **Rebase** para solução dos conflitos, use o comando **git pull upstream master --rebase**. Note que o pedido de Rebase lhe retornará uma mensagem de conflito demandando sua ação

```
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/fakebmarchete/git-pratica
 * branch      master    -> FETCH_HEAD
   4ac5dc6..08ef41c  master    -> upstream/master
First, rewinding head to replay your work on top of it...
Applying: alteração #5
Using index info to reconstruct a base tree...
M       Dica2.md
Falling back to patching base and 3-way merge...
Auto-merging Dica2.md
CONFLICT (content): Merge conflict in Dica2.md
error: Failed to merge in the changes.
Patch failed at 0001 alteração #5
The copy of the patch that failed is found in: .git/rebase-apply/patch

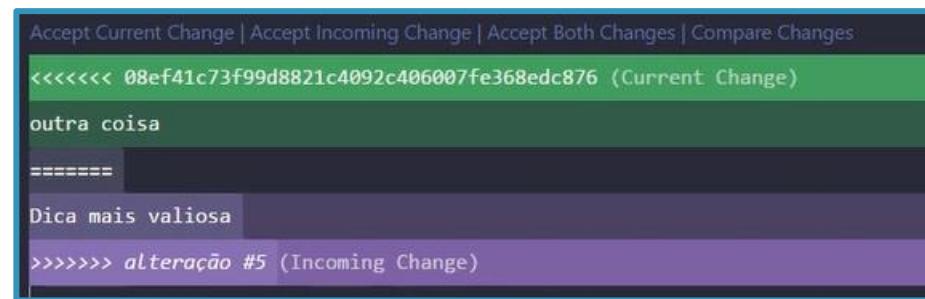
When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".
```





Pull Request Merge & Conflitos

13. Mensagem de conflito gera abertura do solucionador de conflitos do Visual Studio Code. É bom saber que isso pode ocorrer em vários arquivos dependendo da mudança sugerida



14. Feito os ajustes, lembre-se que você está realizando um Rebase, portanto, adicione na Stage, mas não em hipótese alguma realize o commit, apenas siga com o Rebase



Pull Request Merge & Conflitos

15. Agora é só fazer o Rebase continue → **git rabase --continue**

```
$ git rebase --continue  
Applying: alteração #5
```

16. Para avaliar o resultado de se usar o Rebase, novamente faça git log --graph --oneline

Observe agora que sua alteração foi lá para o final da master, assegurando a linearidade para realizar o Merge

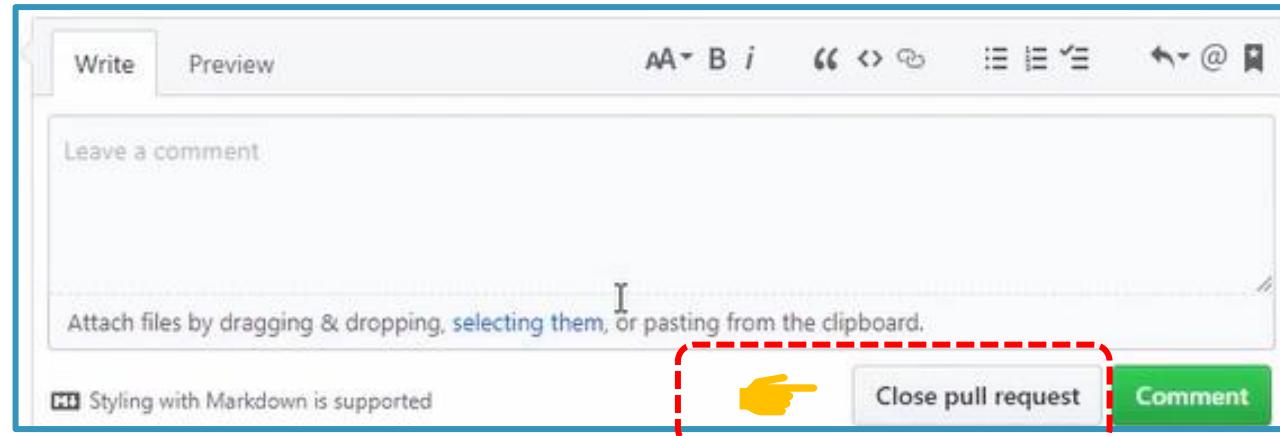


```
$ git log --oneline --graph  
* 67e54df alteração #5  
* 08ef41c Alteração para dica 2 #5  
* 4ac5dc6 Merge pull request #4 from zezinhohub/issue-2  
  \ * c10997a ajuste final em dicai #2  
  /  
* db61dfc Alteração de dica #2  
  * e4c5927 alteração em dica #2  
  /  
* 3991493 Merge pull request #3 from zezinhohub/issue-1  
  /  
* a9908cf dica 3 criada #1  
  /  
* 3ab074c Segunda dica sobre git  
* 6010b51 Primeira dica sobre git  
* b7bcd9d Initial commit
```



Pull Request Merge & Conflitos

17. Agora faça o push dessa nova branch, use **git push origin <nome-da-branche-v2>**
18. Agora o dono do repositório Fork deverá voltar ao pedido de Pull Request que culminou em conflito e fechá-lo acrescentando a mensagem de encerramento por estar enviando em seguida um novo pedido de Pull Request

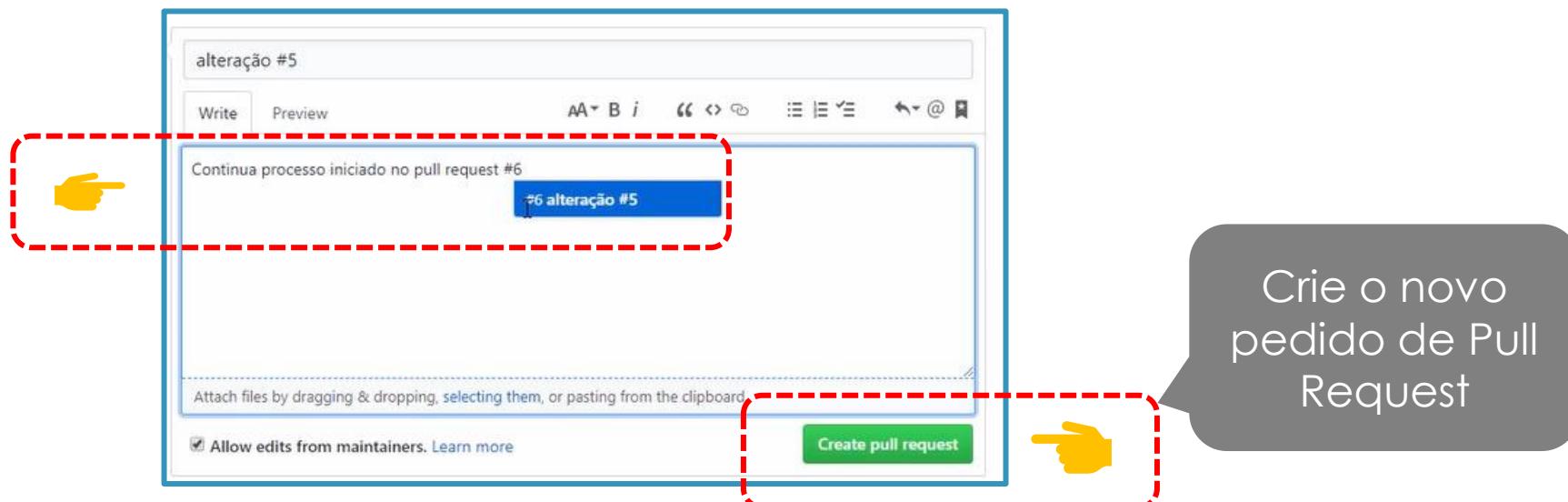




Pull Request Merge & Conflitos

19. Para finalizar o laboratório, volte para o repositório Fork e faça o pedido de uma nova Pull Request por lá.

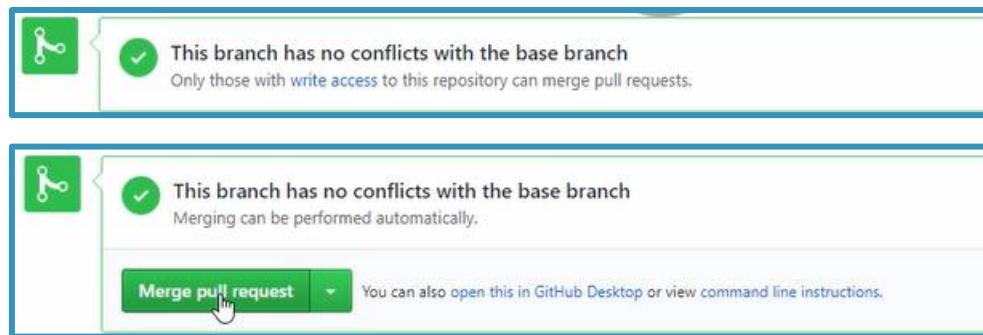
20. Você deverá observar que agora sua solicitação está **Able to merge** adicione uma mensagem, aqui é boa prática dizer que está nova solicitação é uma continuação do Pull Request indicado anteriormente



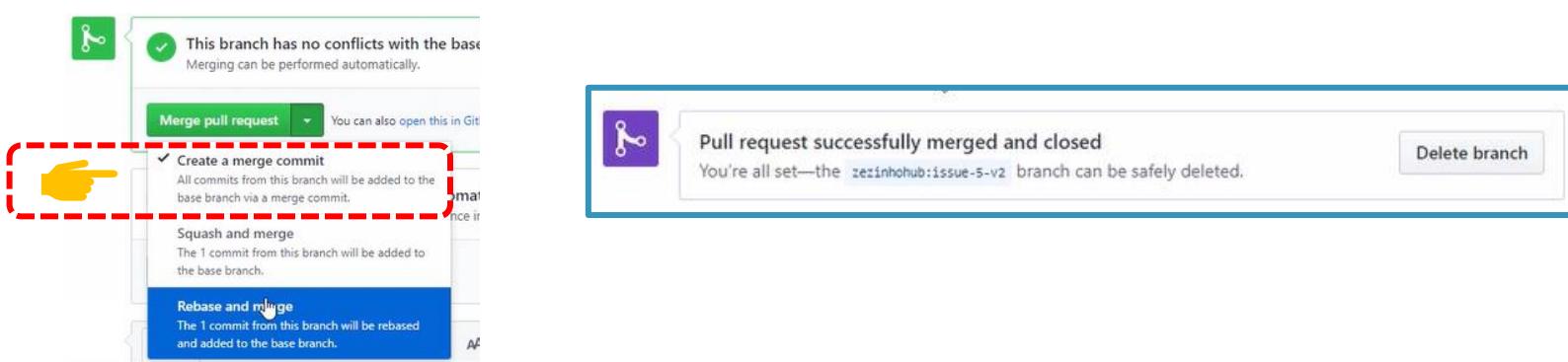


Pull Request Merge & Conflitos

21. Agora sim ele estará pronto para passar pelo estágio de Merge



22. Faça e isso vá ao repositório Upstream e observe que a opção Merge Pull Request estará ativa, escolha a opção desejada (por padrão escolha a primeira) e encerre o laboratório



Como uma última dica deste laboratório, você pode aproveitar o processo, retornar ao Repositório Fork e excluir a Branch



Extra – limpe tudo

Lembre sempre de fazer a limpeza, com o tempo você poderá se esquecer das alterações realizadas e seu ambiente de trabalho rapidamente estará poluído



FORK WORKFLOW - LIMPEZA

1. Atenção, você não pode excluir a Branch em que está, como sugestão faça o checkout para a master e exclua de lá
2. Acesse seu repositório local e verifique quais Branches pertencem ao seu repositório → use o comando **git branch -v**
3. Para excluir Branches locais use o comando **git branch -D <nome-da-branch>**. A opção –D exclui a Branch independente dela pertencer a master ou não



FORK WORKFLOW - LIMPEZA

1. Faxina local feita → lembre-se que lá no GitHub ainda estão as Branches aguardando por sua intervenção
2. O legal é que você localmente poderá excluir a Branch que desejar no GitHub utilizando o comando **git push -d origin <nome-da-branch>**. Observe aqui o uso da opção -d (minúsculo)
3. Com isso encerramos o processo de FORK WORKFLOW





Single Repository Workflow

Está ficando sério
isso!!!
O fluxo de trabalho
do GitHub

SINGLE REPOSITORY WORKFLOW



- Para todos os nossos próximos laboratório o processo será o mesmo
 - Procurem trabalhar em duplas
 - Revisitem os conceitos já aprendidos sempre que sentirem necessidade
 - Me limitarei a chamar à atenção sempre que um conceito novo assim demandar
 - Há muito trabalho, o Git e GitHub apresentam uma série de comandos que não serão aqui apresentados, os que fazem parte deste curso são aqueles que em essência asseguram um domínio avançado para seus trabalhos, então vamos ao trabalho

SINGLE REPOSITORY WORKFLOW



- Nas práticas que envolvem o conceito SINGLE REPOSITORY WORKFLOW, estaremos trabalhando os conceitos mais comumente empregados na prática, no mundo corporativo
- Neste caso, vocês estarão simulando trabalhos em times de desenvolvimento, entenda aqui, todo tipo de projeto que demandar ações multidisciplinares, multitarefas e que promova o uso de recursos de forma ágil e eficiente
- As duplas devem se organizar de forma a se alternarem nas atividades à medida que as práticas forem sendo trabalhadas

SINGLE REPOSITORY WORKFLOW

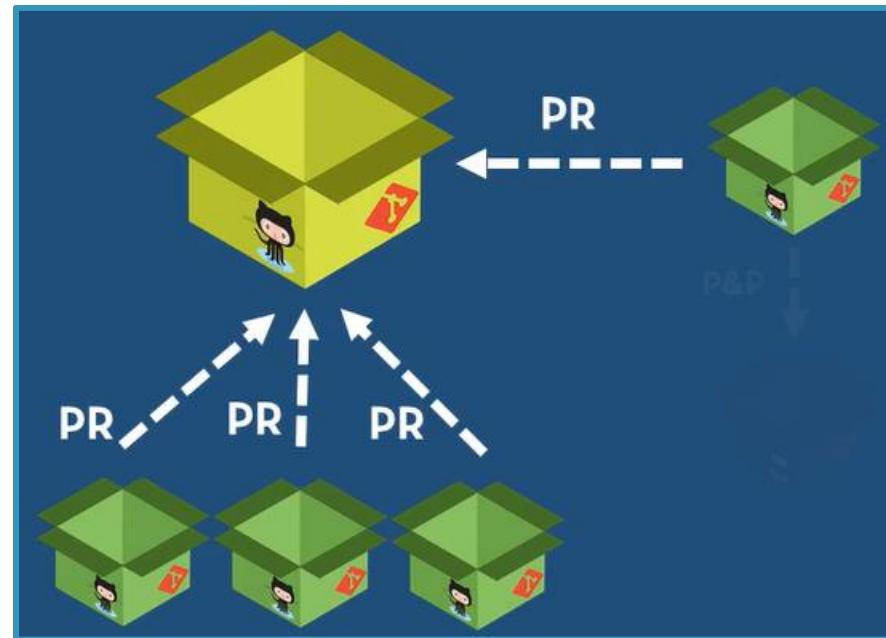


- Mas, antes de iniciarmos, vamos aos pré-requisitos. Certifique-se de que domina os conceitos abaixo apresentados:
 - Merge & Conflitos
 - O uso de Branches
 - O uso do comando Rebase
 - O uso de ISSUES
 - O uso de Pull Request
- Caso não se lembre de algum destes conceitos, volte às aulas e faça uma revisão rápida

SINGLE REPOSITORY WORKFLOW



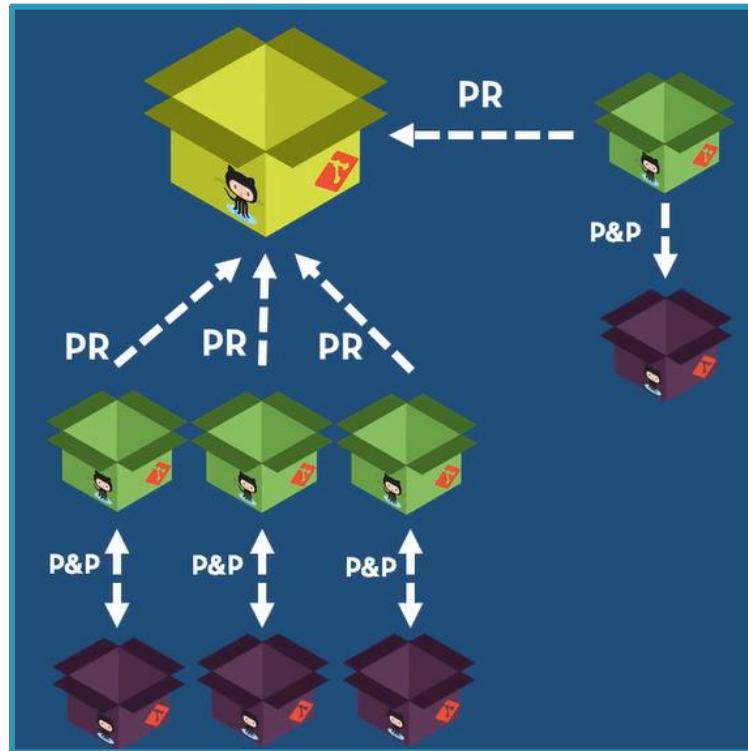
- Antes de iniciarmos vamos entender o conceito de Fork Workflow. Neste processo, há um repositório central (Upstream) a partir do qual os colaboradores deverão realizar o Fork



SINGLE REPOSITORY WORKFLOW



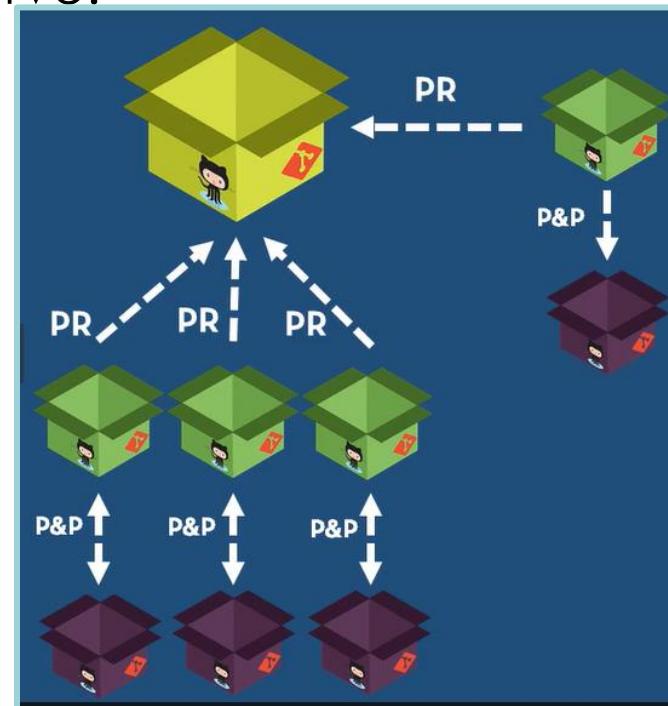
- Uma vez feito o Fork, cada desenvolvedor/engenheiro ou membro ativo do projeto deverá fazer o clone de seu Fork para o repositório local, observe:



SINGLE REPOSITORY WORKFLOW



- A partir de seu repositório cada colaborador deverá desenvolver seu trabalho e submeter via push para seu repositório Fork
- A partir do repositório Fork as ações deverão ser submetidas ao repositório central Upstream via Pull Request que uma vez aprovado será integrado ao repositório Upstream, observe:



SINGLE REPOSITORY WORKFLOW

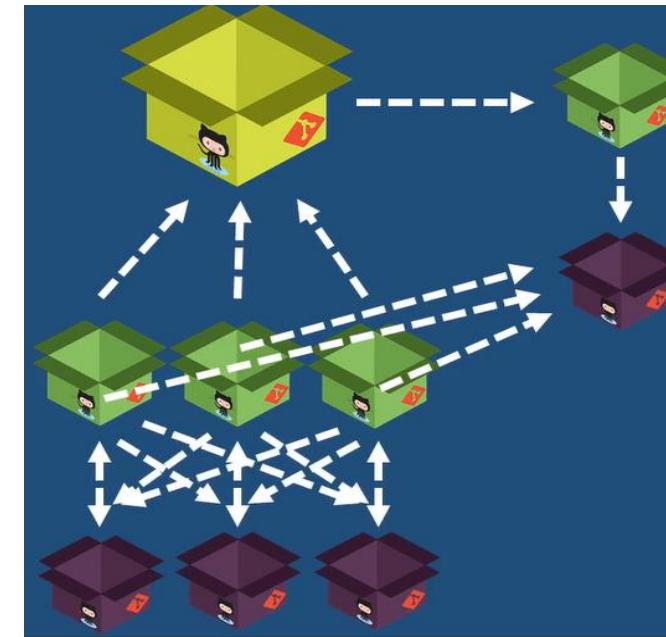
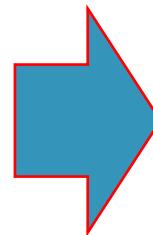
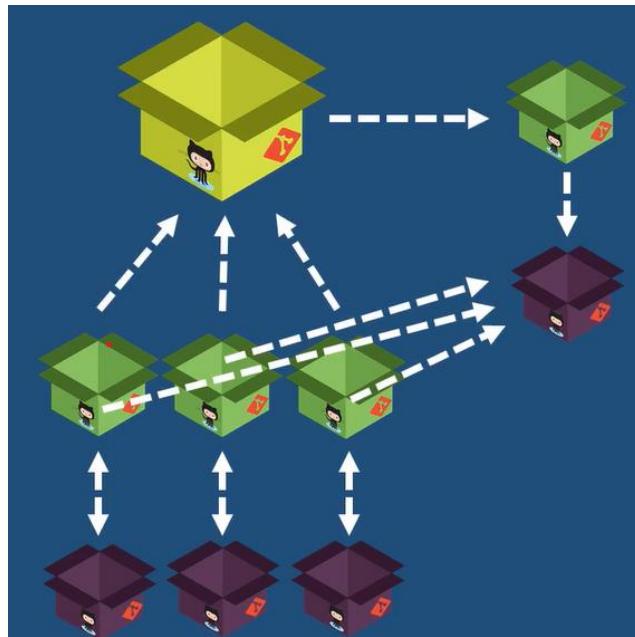


- Lembre-se que tínhamos que manter o sincronismo entre nosso Fork e o repositório Upstream
- Quando existe um fluxo intenso de trabalho entre colaboradores do tipo um colaborador precisa participar do trabalho que outro colaborador está fazendo, isso complica um pouco o fluxo de trabalho desse Workflow
- Cada colaborador que esteja participando do trabalho de outro colaborador precisa rastrear também o Fork deste colaborador, observe a seguir como o fluxo se complica

SINGLE REPOSITORY WORKFLOW



- Observe o aumento do fluxo:



SINGLE REPOSITORY WORKFLOW

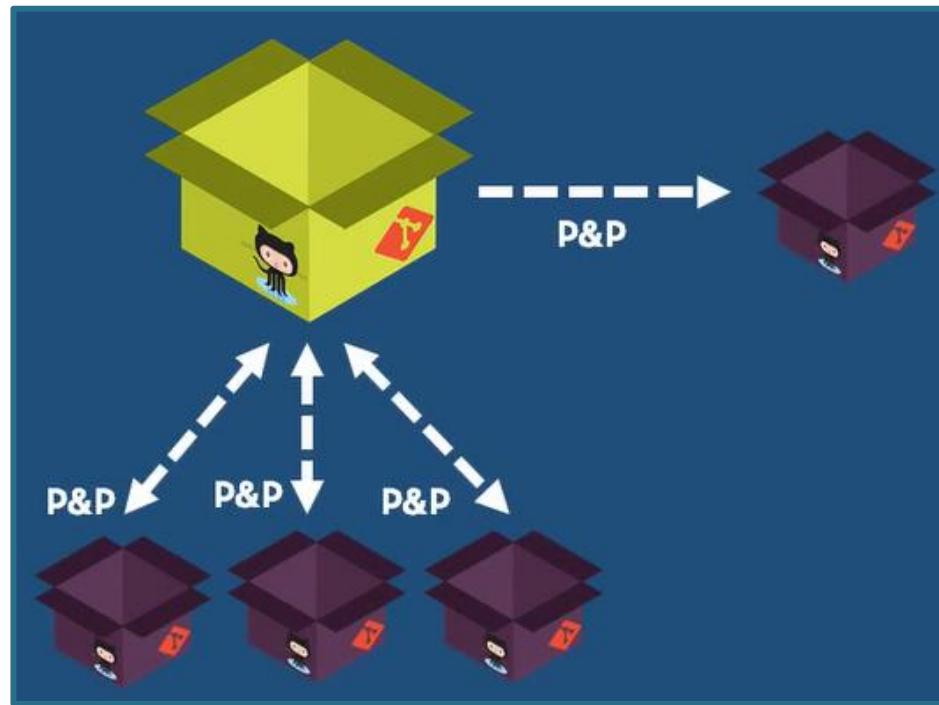


- Embora assustador, lembre-se que este fluxo é muito comum e você com certeza se verá nesta situação, ou até já esteja vivendo isso no mundo corporativo
- O GitHub Workflow é um fluxo de trabalho simplificado que corta alguns passos desse processo todo ajudando os participantes ativos a conhecerem melhor o trabalho que cada um está fazendo
- No GitHub Workflow cada pessoa que colabora com um projeto é um colaborador cadastrado, por isso, cada colaborador pode fazer um clone direto do repositório central Upstream e não precisa do repositório intermediário, em outras palavras não precisa do seu próprio Fork

SINGLE REPOSITORY WORKFLOW



- Dessa forma, qualquer alteração que um colaborador faça e submeta ao Repositório Upstream com um push, todos os outros colaboradores terão acesso às alterações quando fizer um pull, observe:



SINGLE REPOSITORY WORKFLOW

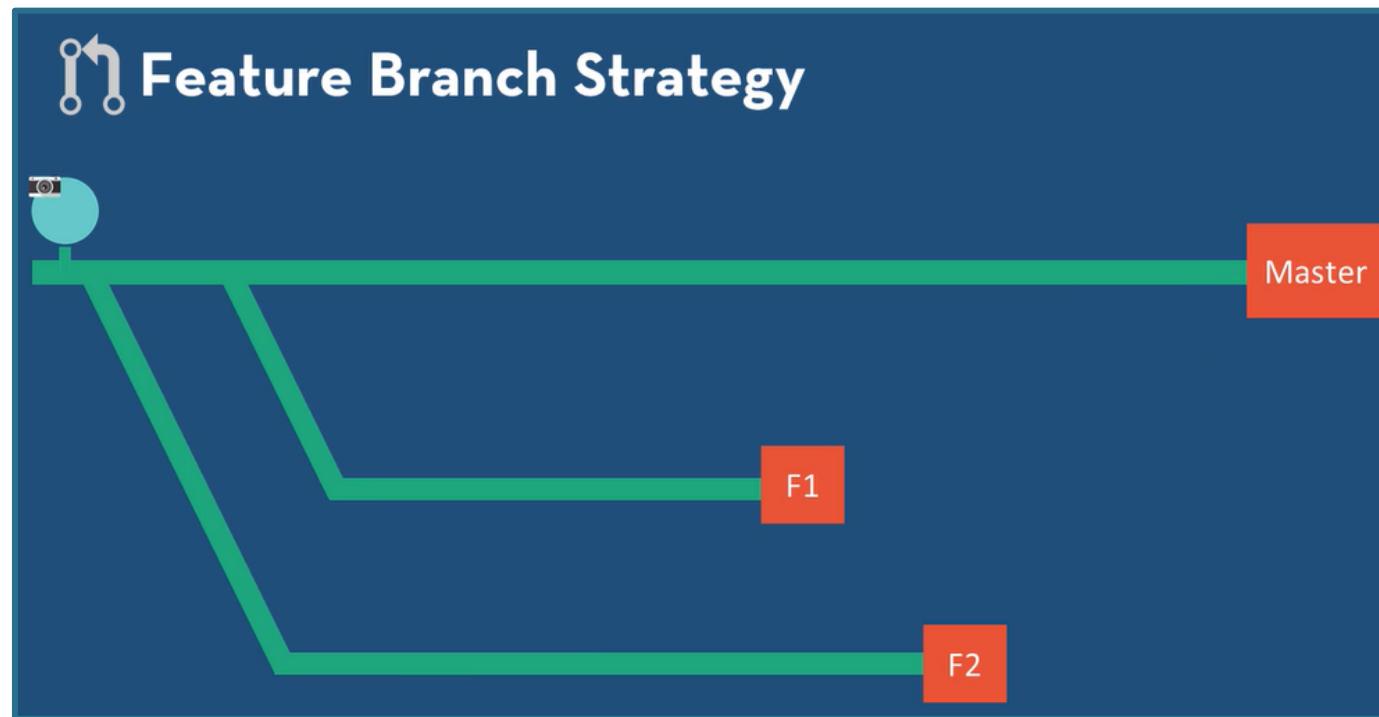


- Como todos os colaboradores terão acesso à branch master e os commits poderão ser feitos diretamente nela, existe uma estratégia para simplificar o trabalho, para organizar esse trabalho
- Essa estratégia recebe o nome de **Feature Branch Strategy**, na Feature Branch Strategy nós temos a nossa Branch master que é a nossa Branch principal e a cada vez formos fazer a inserção de uma nova ferramenta ou uma a criação de uma nova Feature para o projeto, deveremos criar uma nova Branch, observe a seguir

SINGLE REPOSITORY WORKFLOW



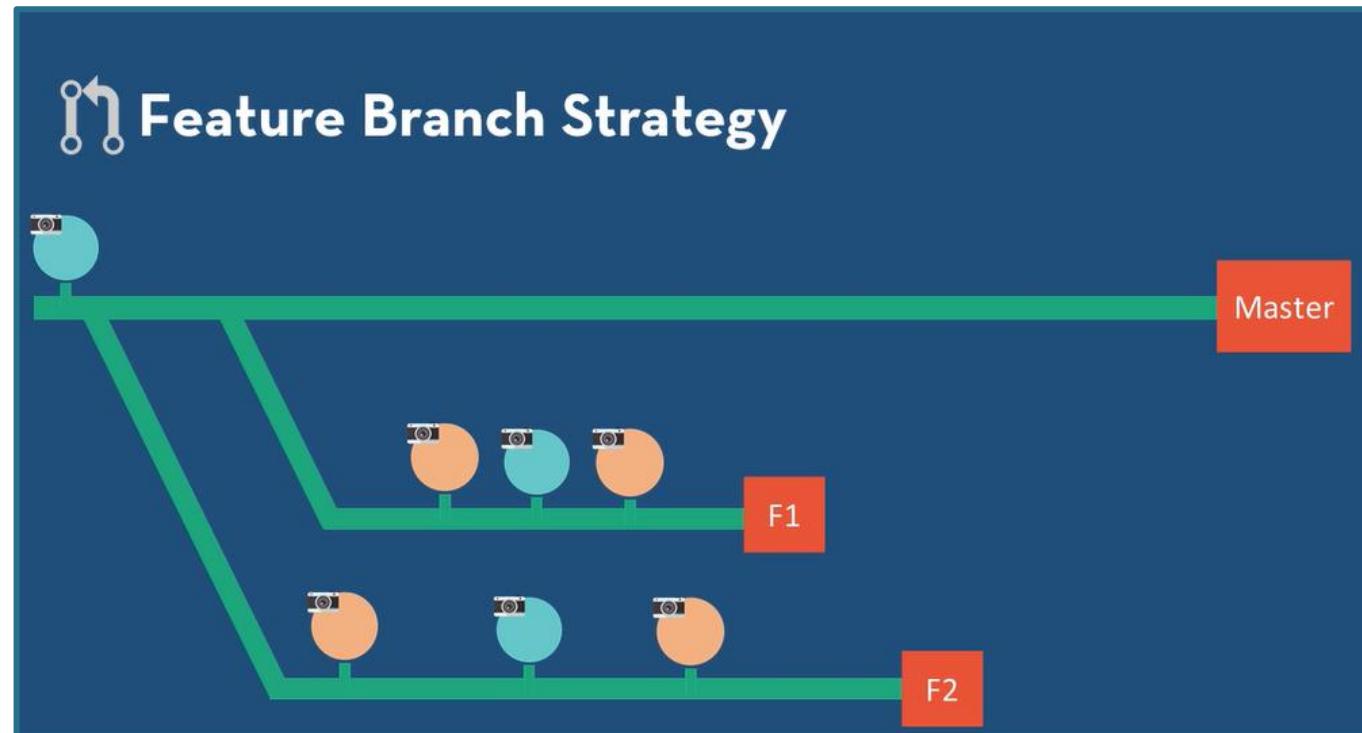
- Assim, teremos N Branches, cada uma cuidando de um pedaço da aplicação/projeto que estivermos desenvolvendo para que fique tudo muito bem organizado



SINGLE REPOSITORY WORKFLOW



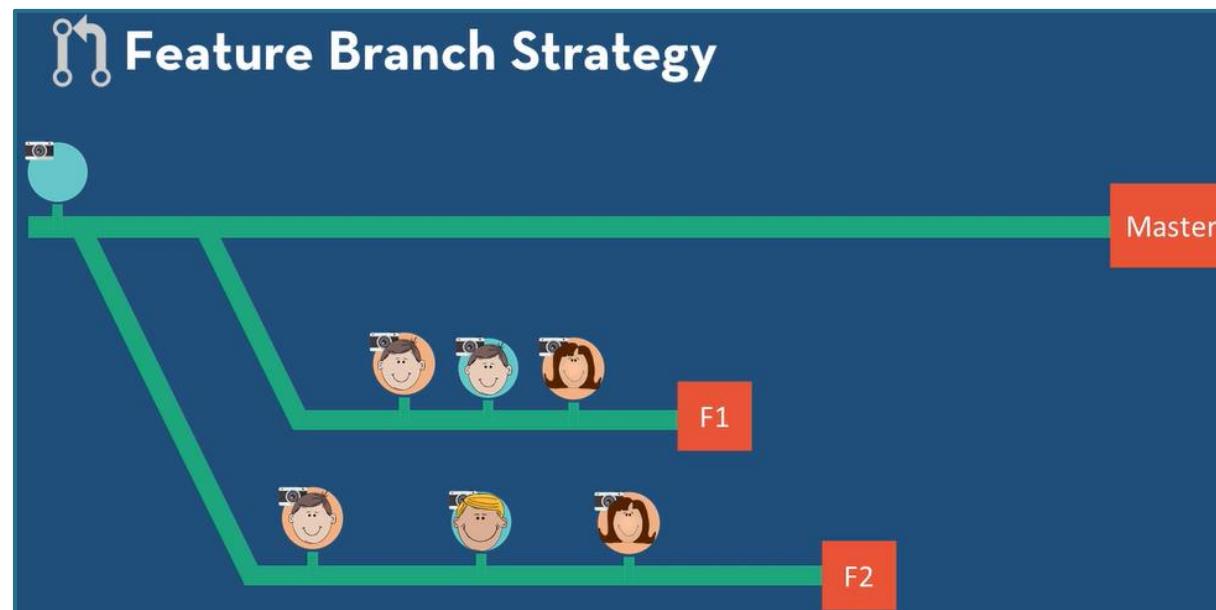
- O legal de se trabalhar com Branches separadas neste Fluxo Workflow e que você poderá realizar os commits independentes do Branch, baseado sempre na ferramenta/Feature que você está desenvolvendo



SINGLE REPOSITORY WORKFLOW



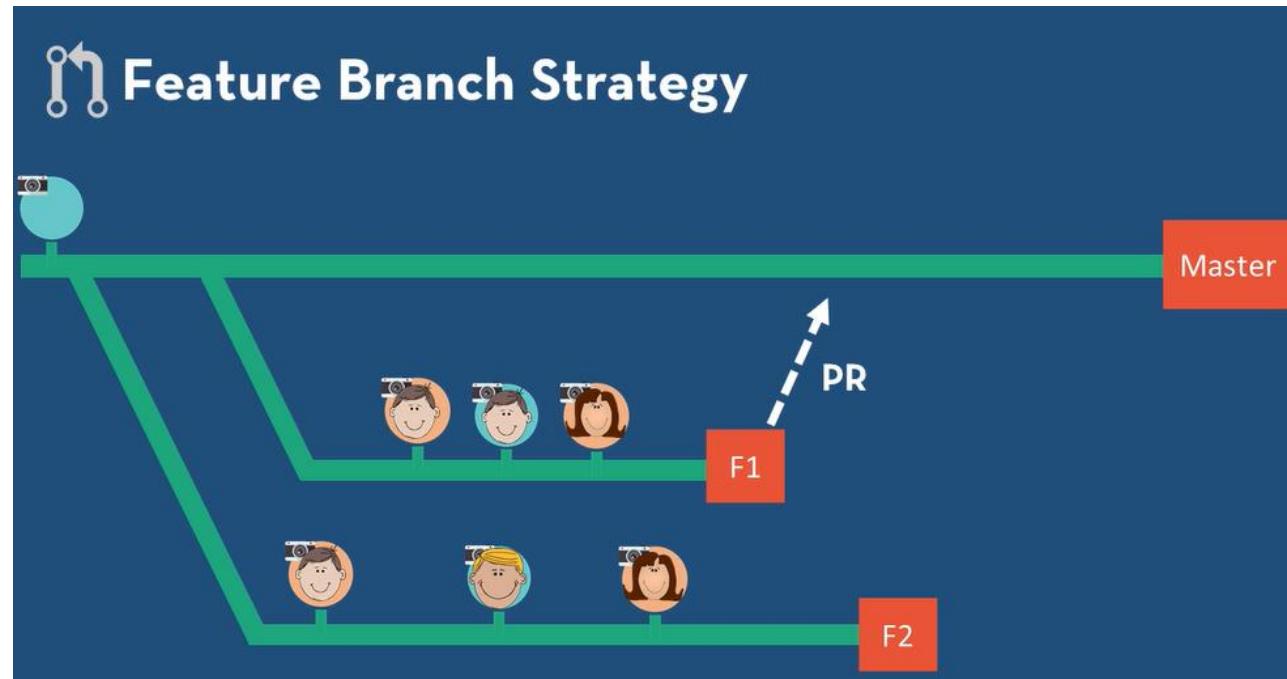
- Outra vantagem é que todos os desenvolvedores que estiverem trabalhando na mesma ferramenta/Feature poderão realizar os commits na mesma Branch já que todos têm acesso ao repositório central



SINGLE REPOSITORY WORKFLOW



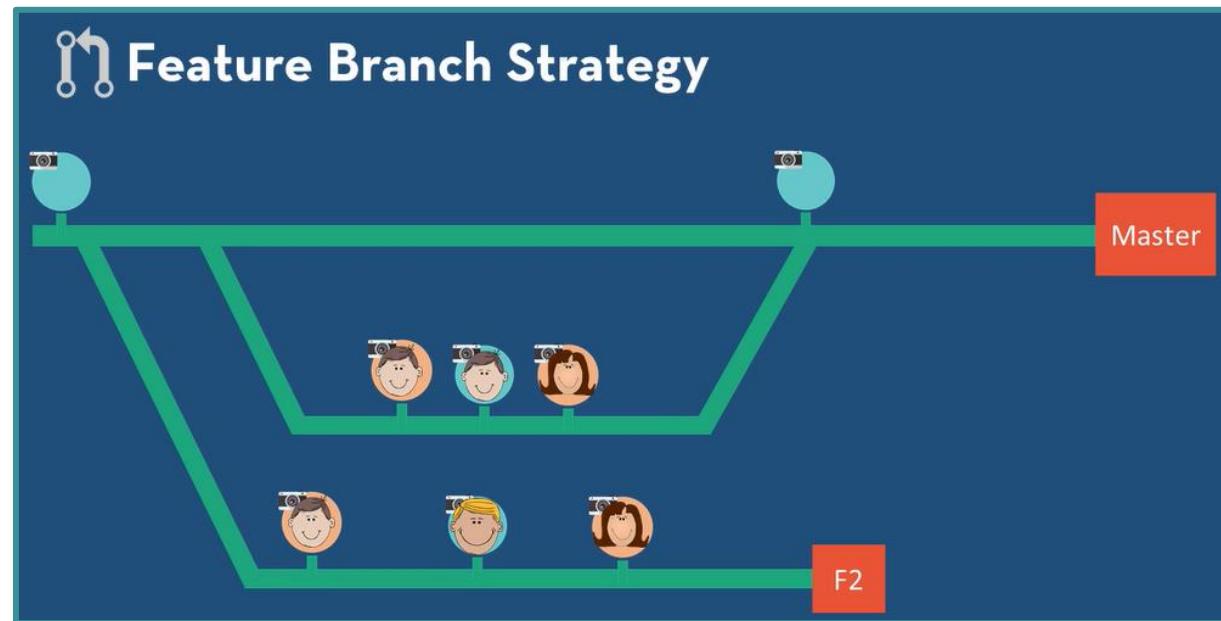
- Eventualmente quando a ferramenta/Feature estiver pronta deverá ser feito um Pull Request para o repositório Upstream



SINGLE REPOSITORY WORKFLOW



- Sendo aceito o Pull Request (falaremos mais dele adiante nas práticas) o trabalho desenvolvido será incorporado à Master do Projeto
- É claro que aqui termos todo trabalho de fazer Merge/Rebase e solucionar conflitos, falaremos mais sobre isso adiante



SINGLE REPOSITORY WORKFLOW



- Revisando o Fluxo GitHub Workflow
 - Passo 1 → realizar o git clone
 - Passo 2 → git checkout –b <nome-da-branch>
 - Passo 3 → git add e git commit
 - Passo 4 → git push
 - Passo 5 → Pull Request (que sendo aceito será integrado à master do projeto)
 - Passo 6 → git pull (deverá ser feito sempre que precisar atualizar qualquer Branch inclusive a master), com o git pull você traz todos os dados do repositório central



SINGLE REPOSITORY WORKFLOW (GITHUB WORKFLOW)

Vamos praticar - Laboratório

SINGLE REPOSITORY WORKFLOW



1. Em duplas ao trabalho. O Usuário Upstream deverá criar um repositório de trabalho (**Workflow-Direto**)
2. O segundo usuário irá participar deste projeto
3. O dono do Upstream deverá criar dois a três arquivos, não precisa conter mais que uma linha, deverá fazer todo processo de salvar adicionar na Stage e realizar os commits
4. O participante deverá fazer o clone direto desse repositório para que possa iniciar seus trabalhos na sua máquina local (defina o nome do repositório de trabalho → **Workflow-Direto**)

SINGLE REPOSITORY WORKFLOW



5. Feito o clone seu próximo passo será criar a branch de trabalho, no meu caso, criarei a branch dica-3
6. Crie um arquivo na nova branch no meu caso Dica-3.md será criado
7. Faça todo processo e realize um push para iniciar o processo de colaboração com esse projeto, só para lembrar use o comando **git push origin <nome-da-branch>**
8. Neste momento você terá algo novo, é aqui que deverá trabalhar um pouco

SINGLE REPOSITORY WORKFLOW



9. Como você inicialmente pode não ter permissão de acesso ao repositório Upstream → deverá solicitar ao dono do Upstream que libere acesso (na prática → os membros da equipe já estão configurados para acesso, mas aqui simularemos todos os passos)

b without any code!

in, write comments, and open a pull request.

Merge guide

Unwatch 1 Star 0 Fork 0

0 Wiki Security Insights Settings

Aqui em Settings
deverá atribuir a
permissão ao
colaborador

O Dono do
Upstream deverá
logar e atribuir
permissão

SINGLE REPOSITORY WORKFLOW



- Continue acompanhando → essa é a parte não estudada nas aulas anteriores, portanto, atenção. Clicando em Settings terá o menu abaixo:

The screenshot shows the 'Settings' page for a GitHub repository named 'Workflow-Direto'. The left sidebar lists various settings options: Options, Manage access (highlighted with a yellow hand icon), Branches, Webhooks, Notifications, Integrations & services, Deploy keys, Secrets, and Actions. The main area displays the 'Repository name' field set to 'Workflow-Direto' with a 'Rename' button. A checkbox for 'Template repository' is present with explanatory text. Below that is a 'Social preview' section with instructions for uploading an image. At the top right, there are buttons for Unwatch, Star, Fork, and Insights, with counts of 1, 0, 0, and 0 respectively. A red dashed box highlights the 'Settings' tab in the header, and another red dashed box highlights the 'Manage access' link in the sidebar.

Em Manage access poderá atribuir as permissões de acesso. Faça isso

SINGLE REPOSITORY WORKFLOW



- Você terá a opção de enviar um convite para um colaborador e observe que na janela que se abre, basta iniciar a digitar o colaborador, que o GitHub já o encontra para você, observe:

The image consists of three screenshots from the GitHub interface, each with a yellow hand icon pointing to a specific action or element, and a red dashed box highlighting that element. The screenshots are arranged vertically.

- Screenshot 1:** Shows the main repository page for "Workflow-Direto". A yellow hand points to the "Invite a collaborator" button at the bottom right of the page. A red dashed box highlights the search bar where "accolombini" has been typed.
- Screenshot 2:** Shows the "Invite a collaborator to Workflow-Direto" modal window. A yellow hand points to the search bar where "accolombini" has been typed. A red dashed box highlights the list of results, showing a card for "Angelo accolombini + Invite collaborator".
- Screenshot 3:** Shows the "Invite a collaborator to Workflow-Direto" modal window with the user "Angelo accolombini" selected. A yellow hand points to the green "Add accolombini to Workflow-Direto" button at the bottom of the modal. A red dashed box highlights this button.

SINGLE REPOSITORY WORKFLOW



- Tudo feito você terá algo como:

The screenshot shows the 'Who has access' section of a GitHub repository settings page. It includes two sections: 'PUBLIC REPOSITORY' (disabled) and 'DIRECT ACCESS'. Under 'DIRECT ACCESS', it says '1 has access to this repository. 1 invitation.' A green button labeled 'Invite a collaborator' is visible. Below this, the 'Manage access' section shows a list with one item: 'Angelo' (Pending Invite). A red callout box points to the 'Manage access' section with the text: 'Um e-mail será enviado para o colaborador que deverá dar o aceite, acompanhe'. Another red callout box points to the 'Pending Invite' status of the collaborator 'Angelo' with the text: 'Neste painel você poderá acompanhar o número de colaboradores em seu projeto'.

Who has access

PUBLIC REPOSITORY

This repository is public and visible to anyone.

DIRECT ACCESS

1 has access to this repository. 1 invitation.

Manage

Manage access

Select all

Find a collaborator...

Type ▾

Angelo
Awaiting accolombini's response

Pending Invite

Invite a collaborator

Um e-mail será enviado para o colaborador que deverá dar o aceite, acompanhe

Neste painel você poderá acompanhar o número de colaboradores em seu projeto

SINGLE REPOSITORY WORKFLOW



- É preciso dar o aceite para que possa iniciar os trabalhos de colaboração. Faça isso e refaça seu push.

A screenshot of a GitHub invitation email. The top part shows a message from 'Angelo <noreply@github.com>' with a yellow hand icon pointing to the 'para mim ▾' button. The bottom part shows an invitation from '@accolombinifake' to collaborate on the 'accolombinifake/Workflow-Direto' repository. It features a yellow hand icon pointing to the accept button, followed by two profile pictures and a plus sign.

GitHub

@accolombinifake has invited you to collaborate
on the
accolombinifake/Workflow-Direto repository

SINGLE REPOSITORY WORKFLOW



- No e-mail dê o aceite e será direcionado para a página

The screenshot shows an email invitation from GitHub. At the top are two profile pictures: Groot on the left and Baby Yoda on the right, separated by a plus sign. Below them is the text: "@accolombinifake has invited you to collaborate on the **accolombinifake/Workflow-Direto** repository". A dashed red box highlights the text "You can accept or decline this invitation. You can also head over to <https://github.com/accolombinifake/Workflow-Direto> to check out the repository or visit [@accolombinifake](https://github.com/accolombinifake) to learn a bit more about them." A yellow hand icon with the number "1" points to the word "Accept".

The screenshot shows the GitHub invitation page. It features the same two profile pictures at the top. Below them is the text: "accolombinifake invited you to collaborate". A dashed red box highlights the "Accept invitation" button, which is green, and the "Decline" button, which is grey. A yellow hand icon with the number "2" points to the "Accept invitation" button.



SINGLE REPOSITORY WORKFLOW



- Acessando o repositório do colaborador você deverá receber a mensagem de que está apto a realizar push no Repositório do projeto Workflow-Direto, observe:

A screenshot of a GitHub repository interface. At the top, there's a navigation bar with 'Pull requests', 'Issues', 'Marketplace', and 'Explore' buttons. On the right side of the header, there's a user icon and a notification bell. Below the header, a blue banner displays the message: 'You now have push access to the accolombinifake/Workflow-Direto repository.' A red dashed box highlights this message, and a yellow hand cursor icon points to it. The number '2' is also present near the bottom left of the box.



SINGLE REPOSITORY WORKFLOW



- Feito o push e acessando o diretório Upstream você poderá visualizar que sua Branch criada já está integrada no projeto e você poderá efetuar o Pull Request, observe:

The screenshot shows a GitHub repository page for 'accolombinifake / Workflow-Direto'. The 'Code' tab is selected. The main area displays the 'Default branch' (master) and 'Your branches' section, which contains a branch named 'dica-3' updated 1 hour ago by 'accolombini'. A yellow hand icon points to the 'dica-3' branch entry. Below this, the 'Active branches' section also lists 'dica-3' with the same update information. To the right of the branches, there is a summary of pull requests: '0 | 1' and a 'New pull request' button. Another yellow hand icon points to the 'New pull request' button. The top right of the page shows statistics: 1 unwatched, 0 stars, 0 forks, and 0 issues/pull requests/actions/projects/wiki/security/insights/settings.

SINGLE REPOSITORY WORKFLOW



10. Agora faça o Pull Request, não se esqueça de deixar uma mensagem

The screenshot shows the GitHub 'Open a pull request' interface. A blue border surrounds the main form area. Three yellow hand icons with red numbers 1, 2, and 3 point to specific elements:

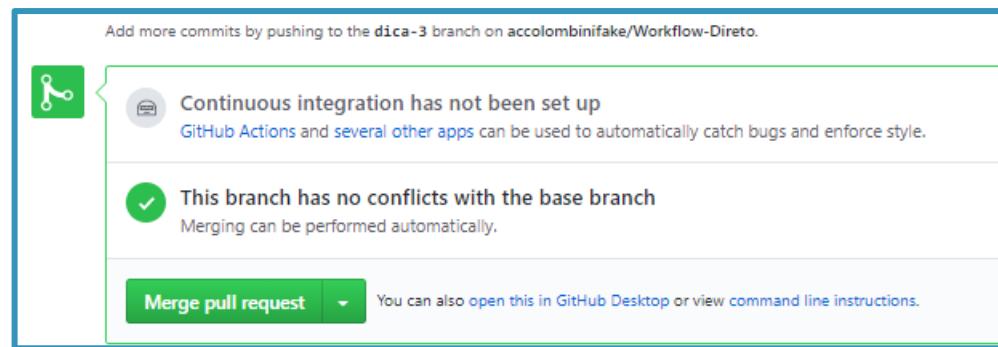
- Step 1:** Points to a grey callout bubble containing the text "Observe que não há conflitos".
- Step 2:** Points to a grey callout bubble containing the text "Deixe a mensagem".
- Step 3:** Points to a grey callout bubble containing the text "Crie o Pull Request".

The interface includes fields for "base: master" and "compare: dica-3". A message box contains the text "criado arquivo Dica-3" and "Acabei de criar uma nova dica a Dica 3!". A green "Create pull request" button is at the bottom right. A red dashed box highlights the "Able to merge" status message: "Able to merge. These branches can be automatically merged."

SINGLE REPOSITORY WORKFLOW



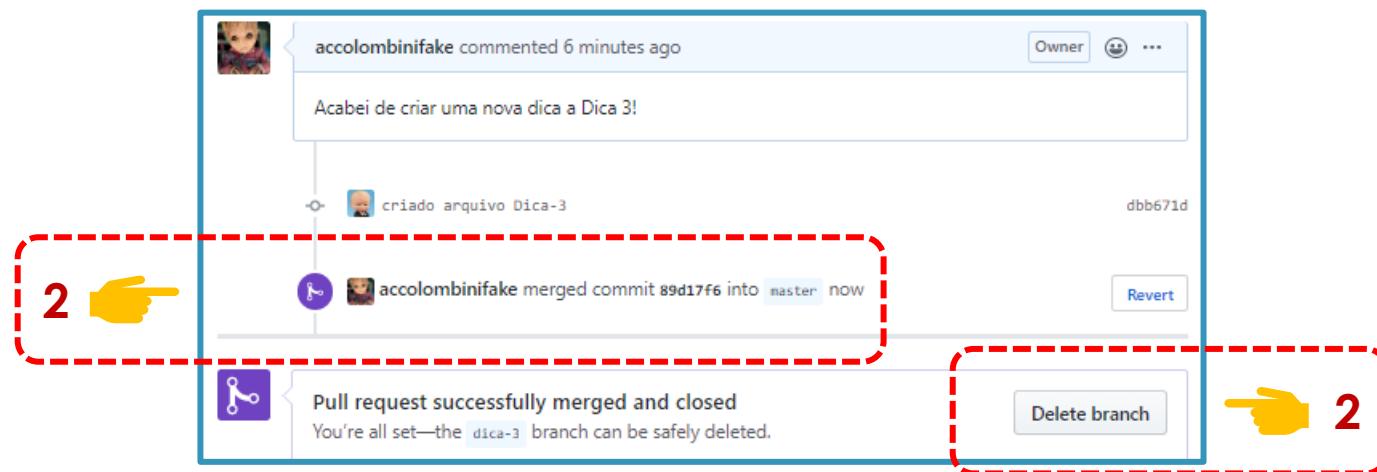
- Observe que embora não sendo o dono do repositório Upstream, ainda você terá a possibilidade de realizar o merge, isso não é uma boa prática de projetos, falaremos disso mais tarde, pois a pessoa que fez as alterações realizar a Merge sem qualquer revisão é algo para ser revisto, por hora siga em frente e realize o merge!



SINGLE REPOSITORY WORKFLOW



- Observe que a merge foi um sucesso e você têm ainda a opção de deletar a Branch criada, que se não houver mais utilidade para ela poderá fazer neste momento sua exclusão



SINGLE REPOSITORY WORKFLOW



11. Para finalizar você poderá avaliar as alterações no GitHub e poderá fazer o sincronismo com o repositório local, note que, lembra que para sincronizar deverá estar em sua Branch master, sincronize no Visual Studio Code, use o comando **git pull**

The screenshot shows a GitHub repository interface. A large red dashed box labeled '1' highlights the commit history on the left, which includes:

- accolombinifake Merge pull request #1 from accolombinifake/dica-3
- Dica-1.md (Create 2 hours ago)
- Dica-2.md (Create 2 hours ago)
- Dica-3.md (criado arquivo Dica-3 2 hours ago)
- README.md (Initial commit 2 hours ago)

A red dashed box labeled '2' highlights the 'Contributors' section at the top right, which shows 2 contributors. A red arrow points from this section to a chart on the right.

A red dashed box labeled '3' highlights the chart titled 'Contributions to master, excluding merge commits'. The chart shows two series: 'accolombinifake' (green) and 'accolombini' (orange). The green series has a sharp peak on March 22, while the orange series shows a more gradual increase over time.



GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO

Vamos praticar - Laboratório

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



- Neste laboratório, não faremos nada de novo, a menos do fato de que, restringiremos o acesso através do **merge**
- Troquem os papéis e refaçam o ambiente de testes, (criem um novo projeto **Workflow_Restricoes**) → repitam todos os passos do laboratório anterior
- Trabalhem por conta até o momento de realizar o push com o Repositório Upstream
- Antes de prosseguir vamos aprender como inserir restrições de acesso, estamos juntos? Vamos a alguns conceitos muito importantes ...



GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



- O objetivo desse laboratório é falar um pouco mais do **Pull Request**
- A finalidade do **Pull Request** é permitir que colaboradores do projeto possam fazer revisões e comentários de tudo o que foi feito no projeto
- Se o **colaborador consegue iniciar um Pull Request e ele mesmo encerrá-lo** não faz sentido algum processo do Pull Request
- Na verdade esse colaborador poderia fazer um merge diretamente na **master** sem passar pelo **Pull Request**
- No laboratório anterior, GitHub Workflow Direto, da forma como foi apresentado, isso é perfeitamente possível, vamos aos testes ...

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



1. Para testar esse conceito, abra o Visual Studio Code e crie uma nova Branch chamada teste, crie um novo arquivo (teste.md), adicione-o na Stage e realize o commit → vamos trabalhar ...
2. A partir deste momento o correto seria realizar nosso Pull Request e passar por todo trâmite
3. Para manter nosso foco em realizar nosso teste, volte para sua branch master, faça um merge e realize um push direto para o repositório Upstream, veja o que acontece ...

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



4. Volte ao repositório no GitHub e observe que a master estará atualizada com o arquivo teste.md

acolombinifake / Workflow-Direto

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki

Repositório criado para simular um ambiente colaborativo na prática

Manage topics

6 commits 2 branches 0 packages

Branch: master New pull request

acolombini criado o arquivo teste.md

File	Commit Message
Dica-1.md	Create Dica-1.md
Dica-2.md	Create Dica-2.md
Dica-3.md	criado arquivo Dica-3
README.md	Initial commit
teste.md	criado o arquivo teste.md

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



- Para evitar esse problema o GitHub fornece ferramentas ao dono do repositório para que ele possa restringir o acesso a algumas Branches
5. O dono do repositório Upstream deverá acessar seu repositório, vá em Settings e em seguida Branches → lá poderá fazer a gestão de acesso, e observe:

The screenshot shows the GitHub repository settings page for 'acolombinifake / Workflow-Direto'. A red dashed box labeled '1' highlights the 'Settings' tab in the top navigation bar. Another red dashed box labeled '2' highlights the 'Branches' link in the left sidebar under the 'Options' section. The main content area displays the 'Settings' page with fields for 'Repository name' (set to 'Workflow-Direto') and 'Rename', and a section for 'Template repository'.

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



6. Acesse a guia Branch e você estará no contexto de gestão de proteção de Branches, escolha a Branch desejada, por Default a Branch master estará selecionada em seguida vou clicar em Add rule, observe como:

The screenshot shows the 'Default branch' settings page in GitHub. The left sidebar has a 'Manage access' section with 'Branches' highlighted by a red dashed box labeled '1'. The main area shows the 'Default branch' configuration with a dropdown set to 'master' and an 'Update' button. Below it is a 'Switch default branch' section with a dropdown set to 'dica-3' and a checkbox for 'master'. A red dashed box labeled '2' encloses this section. To the right is a large red dashed box labeled '3' enclosing the 'Add rule' button.

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Options Manage access Branches Webhooks Notifications Integrations & services Deploy keys Secrets Actions Moderation Interaction limits

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

master Update

Switch default branch

Filter branches

dica-3

✓ master

Add rule

No branch protection rules defined yet.

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



Observe que há muitas proteções a serem aplicadas, cabe a você, dada a natureza do seu projeto trabalhar as proteções mais adequadas. **Atenção**, definida a proteção você deverá salvar a configuração clicando em **create** na base da página, não se esqueça

7. Para esse laboratório vamos exigir que todo merge deva antes passar por um processo de Pull Request para que possa ser avaliado antes que se efetue o merge, observe:

Branch protection rule

Branch name pattern
master

Protect matching branches

Require pull request reviews before merging
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.
Required approving reviews: 1 ▾

Dismiss stale pull request approvals when new commits are pushed
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

Require review from Code Owners
Require an approved review in pull requests including files with a designated code owner.

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



Ao criar a regra você poderá observar no seu repositório que existe uma regra criada e a qual Branch ela se aplica, observe:



8. Vamos agora praticar, refaça todo trabalho anterior, crie uma nova branch, um novo arquivo adicione na Stage, faça o commit e realize o push

GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO



10. Retorne para sua master, realize o merge e tente executar o push exatamente como fez antes. Como resumo dentre outras informações você receberá algo como:

Observe que demanda uma revisão para aprovação



Acesso negado

```
accol@DESKTOP-TMBP1KD MINGW64 /d/users/Angelo/GIT/WORKFLOW/Workflow_Restricoes/WORKFLOW_RESTRICOES (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1) completed with 1 local object.
remote: error: GH006: Protected branch update failed for refs/heads/master.
remote: error: At least 1 approving review is required by reviewers with write access.
To https://github.com/accolombinifake/WORKFLOW_RESTRICOES.git
 ! [remote rejected] master -> master (protected branch hook declined)
error: failed to push some refs to 'https://github.com/accolombinifake/WORKFLOW_RESTRICOES.git'
```



GITHUB WORKFLOW COM RESTRIÇÃO DE ACESSO

Vamos praticar - Laboratório

REVIEW NO PULL REQUEST



- Para resolver o problema anterior, será necessário fazer um reset na branch master para que ela retorne ao estágio anterior
- Organizando tudo para o estado anterior, deveremos proceder com o pedido de Pull Request para atender ao que determina na mensagem do GitHub, só revisando, observe:

Observe que
demanda uma
revisão para
aprovação

Acesso negado

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/workflow_Restricoes/WORKFLOW_RESTRICOES (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1) completed with 1 local object.
remote: error: GH006: Protected branch update failed for refs/heads/master.
remote: error: At least 1 approving review is required by reviewers with write access.
To https://github.com/acco1binifake/WORKFLOW_RESTRICOES.git
 ! [remote rejected] master -> master (protected branch hook declined)
error: failed to push some refs to 'https://github.com/acco1binifake/WORKFLOW_RESTRICOES.git'
```

REVIEW NO PULL REQUEST



1. Fazer um **reset** na branch master retornando a **HEAD** um commit antes (atenção estamos fazendo isso porque sabemos que foi o último commit realizado que gerou as mudanças → poderia haver mais), vamos ao trabalho

Atenção, usaremos o reset com a opção `--hard`, como já sabem é a opção mais drástica do reset e como resultado ela irá zerar tudo o que foi feito após o commit sinalizado não permitindo que sejam recuperado seu Project History desta etapa, tudo se passará como se nada disso houvesse existido

REVIEW NO PULL REQUEST



2. Observe que o arquivo3.md desaparece e você estará no commit anterior a sua criação
3. A área de Stage também estará vazia, significando que o reset --hard deslocou a HEAD para o commit anterior zerando o histórico

```
1 ➤ accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/Workflow_Restricoes/WORKFLOW_RESTRICOES (master)
$ git reset --hard head^
HEAD is now at cb01691 Create arquivo2.md

2 ➤ accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/Workflow_Restricoes/WORKFLOW_RESTRICOES (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'

nothing to commit, working tree clean
```

REVIEW NO PULL REQUEST



4. Feito isso, crie uma nova branch, refaça todo processo, mas agora lembre-se de abrir o **Pull Request** → é com vocês e bom trabalho
5. Observe agora quando o **push** é realizado que a mensagem mudou e um **Pull Request** é necessário, vá ao repositório Upstream e confira, observe

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/Workflow_Restricoes/WORKFLOW_RESTRICOES (dica-3)
$ git push origin dica-3
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 299 bytes | 74.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dica-3' on GitHub by visiting:
remote:   https://github.com/accolombinifake/WORKFLOW_RESTRICOES/pull/new/dica-3
remote:
To https://github.com/accolombinifake/WORKFLOW_RESTRICOES.git
 * [new branch]      dica-3 -> dica-3
```



No GitHub
você já
poderá
fazer o
Pull
Request



The screenshot shows a GitHub repository interface. At the top, it says "Active branches". Below that, there is a list with one item: "dica-3 Updated 7 minutes ago by accolombini". To the right of the list is a "New pull request" button. A yellow hand icon is pointing at the "New pull request" button.

REVIEW NO PULL REQUEST



Caso precise de ajuda para se localizar, observe a seguir os passos que foram dados:

3 commits 2 branches 0 packages 0 releases 1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

accolombinifake Create arquivo2.md Latest commit cb01691 14 hours ago

README.md Initial commit 14 hours ago

arquivo1.md Create arquivo1.md 14 hours ago

arquivo2.md Create arquivo2.md 14 hours ago

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Overview Yours Active Stale All branches Search branches...

Default branch

master Updated 14 hours ago by accolombinifake Default Change default branch

Active branches

dica-3 Updated 12 minutes ago by accolombini 0 | 1 New pull request

Active branches

dica-3 Updated 7 minutes ago by accolombini 0 | 1 New pull request

No GitHub
você já
poderá
fazer o
Pull
Request

REVIEW NO PULL REQUEST



Observe que Pull Request aberto você poderá observar que o Merge estará bloqueado, acompanhe

The screenshot shows the GitHub interface for opening a pull request. It displays two branches: 'base: master' and 'compare: dica-3'. A green checkmark indicates the branches are 'Able to merge'. The main area shows a file named 'criação do arquivo3.md' with the content: 'Solicitação para incluir o arquivo3.md no projeto, avalie'. Below the file content, there's a note: 'Attach files by dragging & dropping, selecting or pasting them.' At the bottom right is a green 'Create pull request' button.

The screenshot shows the GitHub pull request details page for 'criação do arquivo3.md #1'. The top bar indicates the pull request is 'Open'. The main area shows a comment from 'acolombinifake': 'Solicitação para incluir o arquivo3.md no projeto, avalie'. Below the comment, there's a note: 'Add more commits by pushing to the dica-3 branch on acolombinifake/WORKFLOW RESTRIÇÕES'. A red dashed box highlights two error messages: 'Review required' (with a note: 'At least 1 approving review is required by reviewers with write access. Learn more.') and 'Merging is blocked' (with a note: 'Merging can be performed automatically with 1 approving review'). At the bottom, it says 'As an administrator, you may still merge this pull request.' and includes a 'Merge pull request' button.

REVIEW NO PULL REQUEST



6. Vá ao repositório Upstream para trabalhar a revisão e realizar o processo, observe que, você poderá inclusive estabelecer um diálogo a respeito do Pull Request, mas iremos direto ao ponto, o aluno deverá depois com calma explorar as opções disponíveis

The image shows two screenshots of a GitHub pull request interface. The left screenshot shows a pull request from the 'dica-3' branch to the 'master' branch. It has a 'Review required' status and a 'Merging is blocked' status. A yellow hand icon points to the bottom right corner of the comment input area, which contains the text 'Revisarei em breve'. A large red dashed box encloses this entire area. A blue arrow points from the right screenshot towards this box. The right screenshot shows the same pull request after a comment was added. The comment reads 'Revisarei em breve' and is highlighted with a red dashed box. A yellow hand icon points to this comment. The GitHub interface includes standard elements like file lists, commit counts, and check results.

REVIEW NO PULL REQUEST



- Você também poderá clicar diretamente na requisição e fazer seus comentários no próprio arquivo

The image contains two screenshots of a GitHub interface. The left screenshot shows a pull request with two comments from the user 'acolombinifake'. The first comment says 'Solicitação para incluir o arquivo3.md no projeto, avalie' and the second says 'Revisarei em breve'. The right screenshot shows a detailed view of a comment with a reply. The reply says 'Este arquivo ficou incrível, obrigado'. A red dashed box highlights the reply area, and a red number '2' is placed inside it. Another red dashed box highlights the 'Start a review' button at the bottom right, and a red number '3' is placed inside it.

- Seguindo estes passos você estará criando um vínculo e fortalecendo o processo de revisão solicitado pelo Pull Request. Você poderá acompanhar tudo no Repositório do GitHub, confira

REVIEW NO PULL REQUEST



9. Por último poderemos solicitar uma revisão do Projeto ou ferramenta, fazer algum comentário ou aprovar a Pull Request, para isso, basta → clicar em **Add your review**

The screenshot illustrates the GitHub interface for reviewing a pull request. On the left, a modal window is open with two main sections: 'Review required' (with a note about needing at least one approving review) and 'Merging is blocked' (with a note about merging requiring one approving review). A red dashed box highlights the 'Add your review' button in the top right corner of the modal. A large blue arrow points from this box to the second step on the right. Step 2 shows a detailed review interface with a toolbar at the top, a rich text editor, and a comment input field. A red dashed box highlights the three radio button options: 'Comment' (selected), 'Approve', and 'Request changes'. Below these options is a 'Submit review' button.

Review required
At least 1 approving review is required by reviewers with write access. [Learn more.](#)

Merging is blocked
Merging can be performed automatically with 1 approving review.

Merge pull request ▾ You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Add your review

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Comment
Submit general feedback without explicit approval.

Approve
Submit feedback and approve merging these changes.

Request changes
Submit feedback that must be addressed before merging.

Submit review

REVIEW NO PULL REQUEST



10. Aprovada, você estará pronto para fazer a Merge do Pull Request, note que os participantes poderão realizar o merge, escolha um e efetue o merge, e observe que arquivo3 terá sido incorporado na Branch master, como o desejado → observe:

The screenshot shows a GitHub pull request review interface. At the top, there's a green checkmark icon and the text "Changes approved". Below it, "1 approving review by reviewers with write access." and a "Learn more" link. A green button labeled "Merge pull request" is visible. The main message says "This branch has no conflicts with the base branch" and "Merging can be performed automatically." A red dashed box encloses the "Merge pull request" button, with the number "1" inside. Below this, another red dashed box encloses a message: "Pull request successfully merged and closed" and "You're all set—the dica-3 branch can be safely deleted.", with a "Delete branch" button to the right. A red number "2" is placed inside this second red dashed box.



GITHUB WORKFLOW CONFLITOS E REBASE

Vamos praticar - Laboratório

GITHUB WORKFLOW CONFLITOS E REBASE



- Os conflitos continuam independentemente do trabalho desenvolvido, o importante é você sempre ter em mente que eles farão parte de sua rotina de trabalho e que você já possui as ferramentas necessárias para resolvê-los
- Neste breve laboratório vamos apenas nos limitar a conhecer alguns recursos a mais que o Git e GitHub disponibiliza para que tenha ainda mais versatilidade na construção da gestão de seu projeto e na solução dos possíveis conflitos que surgirem
- Sem mais delongas, vamos conhecer o que mais poderá nos ajudar ...



GITHUB WORKFLOW CONFLITOS E REBASE



1. Crie o seguinte cenário → uma nova branch, nela crie um arquivo realize todos os passos e faça um push, veja nada de novo e uma nova branch deverá ter sido criada no repositório remoto, você já sabe isso confira:

```
accol@DESKTOP-TMBP1KD MINGW64 /d/users/Angelo/GIT/WORKFLOW/Workflow_Restricoes/WORKFLOW_RESTRICOES (dica-4)
$ git push origin dica-4
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 339 bytes | 84.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dica-4' on GitHub by visiting:
remote:     https://github.com/accolombinifake/WORKFLOW_RESTRICOES/pull/new/dica-4
remote:
To https://github.com/accolombinifake/WORKFLOW_RESTRICOES.git
 * [new branch]      dica-4 -> dica-4
```

2. Da mesma forma que fez no laboratório passado faça um Pull Request

GITHUB WORKFLOW CONFLITOS E REBASE



3. Pull Request criado, cenário para o laboratório está confluído. Novamente não se esqueçam de trocarem os papéis para que todos possam participar na totalidade da prática, veja a seguir:

The screenshot shows a GitHub Pull Request page for the repository 'accolombinifake / WORKFLOW_RESTRICOES'. The pull request is titled 'criado o arquivo4.md com mais uma dica #2' and is marked as 'Open'. It shows a single commit from 'accolombinifake' with the message 'criado o arquivo4.md com mais uma dica'. A comment from the same user says 'Este arquivo trás dicas incríveis para o projeto, espero que gostem!'. Below the commit, there is a note: 'Add more commits by pushing to the `dica-4` branch on `accolombinifake/WORKFLOW_RESTRICOES`'. At the bottom, there are two error messages: 'Review required' (with a red 'X') and 'Merging is blocked' (with a red 'X'). Both messages indicate that at least one approving review is required. A note below states: 'As an administrator, you may still merge this pull request.' There are 'Merge pull request' and 'View merge details' buttons at the bottom.



GITHUB WORKFLOW CONFLITOS E REBASE



4. Observe o status do laboratório, fique atento aos detalhes, você já está apto a percebê-los, observe:

criado o arquivo4.md com mais uma dica #2

Open accolombini fake wants to merge 1 commit into master from dica-4

Conversation 0 Commits 1 Checks 0 Files changed 1

Changes from all commits ▾ File filter... ▾ Jump to... ▾

criado o arquivo4.md com mais uma dica
dica-4 (v2)
accolombini committed 13 minutes ago

commit a4e589bb41b697fd964e1eeb8b2a84cd1f7f4571

arquivo4.md
+ Criação de mais uma dica através do Arquivo4.md

ProTip! Use [n] and [p] to navigate between commits in a pull request.

5. Voltando ao Visual Studio Code, vamos trabalhar na mesma Branch criada, vamos agora conhecer o comando git branch -a (com a cláusula -a) teremos acesso às Branches locais e às Branches remotas, observe:

Locais

Remotas

```
accol@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/Work_Exp/WORKFLOW_RESTRICOES (master)
$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/dica-3
remotes/origin/dica-4
remotes/origin/master
```

GITHUB WORKFLOW CONFLITOS E REBASE



6. Como deve ter observado o novo colaborador não tem em sua máquina local a branch dica-4, aqui um novo comando para nosso aprendizado.
7. Precisamos criar um track entre o repositório local e o repositório remoto (entre a branch local e a branch dica-4 remota)
8. Para fazer isso, basta criar uma nova branch dando a ela o mesmo nome da branch remota. Ao fazer isso, automaticamente o track será criado, então ao trabalho:

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/Work_Exp/WORKFLOW_RESTRICOES (master)
$ git checkout dica-4
Switched to a new branch 'dica-4'
Branch 'dica-4' set up to track remote branch 'dica-4' from 'origin'.
```

9. Simule agora uma alteração no arquivo4.md → digite uma linha já será suficiente → Salve, coloque na Stage, faça um commit, buscamos a geração de um conflito

GITHUB WORKFLOW CONFLITOS E REBASE



- Feito isso, agora chegou a hora de fazer o push, acompanhe e veja que ele será rejeitado, pois o GitHub identificará que ele não está atualizado

```
acco1@DESKTOP-TMBP1KD MINGW64 /d/Users/Angelo/GIT/WORKFLOW/Workflow_Restricoes/WORKFLOW_RESTRICOES (dica-4)
$ git push -u origin dica-4
To https://github.com/accolombinifake/WORKFLOW_RESTRICOES.git
 ! [rejected]      dica-4 -> dica-4 (fetch first)
error: failed to push some refs to 'https://github.com/accolombinifake/WORKFLOW_RESTRICOES.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- Observe o conflito dos commits

A screenshot of a GitHub pull request interface. The title of the PR is "criado o arquivo4.md com mais uma dica #2". The PR summary says "accolombinifake wants to merge 2 commits into master from dica-4". A yellow hand icon points to the merge conflict message: "Este arquivo trás dicas incríveis para o projeto, espero que gostem!". Below the message, there are two commits: "criado o arquivo4.md com mais uma dica" (commit hash: a4e589b) and "alteração realizado no arquivo4.md" (commit hash: 318cf2c). A red dashed circle highlights the merge conflict message.

GITHUB WORKFLOW CONFLITOS E REBASE



- Neste fluxo de trabalho, recomenda-se tratar dos conflitos utilizando o Rebase, mesmo que você precise fazer um Rebase iterativo e reconstruir o histórico do projeto, as equipes de projeto entendem que está ação é para deixar o Project History organizado
- Sendo assim, ao invés de criar uma nova Branch e refazer todo processo, opta-se por usar o push com a flag -f, ficando assim o comando: **git push -f -u origin <nome-da-branch>**
- A **flag -f (Force)** irá reescrever a Branch que está no Pull Request, mas isso é considerado, ok

Mas, **atenção**, isso dependerá muito de como a equipe de projeto trabalha e de quais normas ela definiu como padrão de segurança para assegurar e preservar o **Project History**

GITHUB WORKFLOW CONFLITOS E REBASE



12. Para continuar, será preciso retornar ao Visual Studio Code, realizar um pull com a opção Rebase, será aberta a tela de conflito, resolva os conflitos manualmente e continue → **git pull --rebase**

A screenshot of a GitHub conflict resolution interface. At the top, there are four buttons: 'Accept Current Change', 'Accept Incoming Change', 'Accept Both Changes', and 'Compare Changes'. Below these buttons, the current change is shown as a green bar containing the text 'O git é o máximo'. To the left of this bar is the commit hash 'c6ce152322166389f39bb0cf2afcbf8afb165d94' and the label '(Current Change)'. Below the green bar is a separator line with five dashes ('=====') and the incoming change as a purple bar containing the text 'O git salvou nossa vida!'. To the right of this bar is the label '(Incoming Change)' and the commit hash 'alteração da dica'. The entire interface has a dark background.

13. Ajuste o conflito, salve e adicione na Stage, use o comando: **git add . && git rebase --continue**
14. Feito os ajustes retorne ao GitHub e faça o merge da forma que realizou no último laboratório

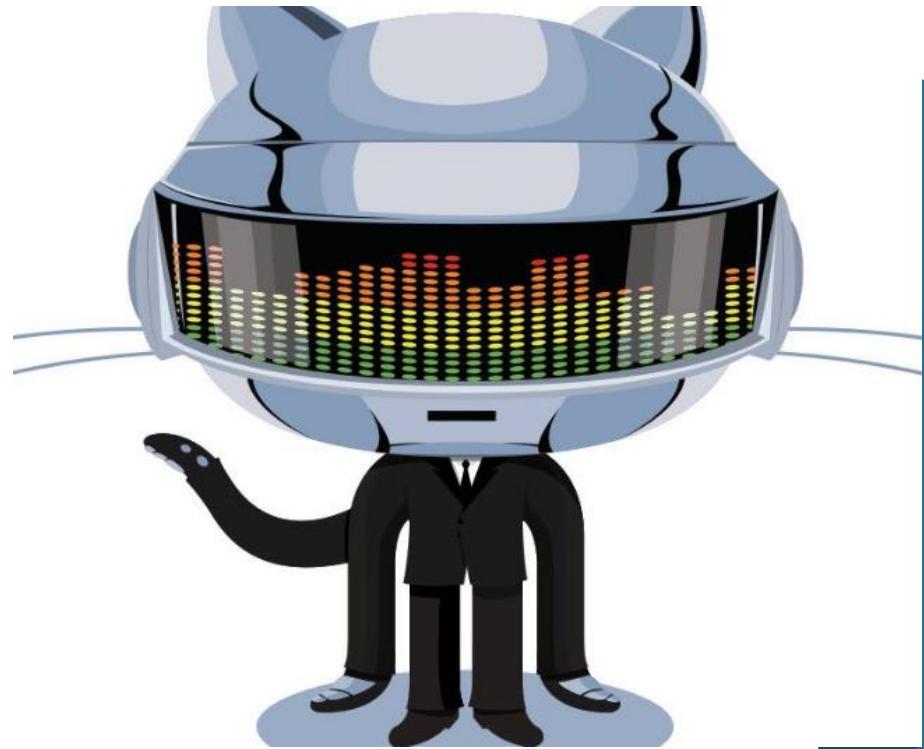
GITHUB WORKFLOW CONFLITOS E REBASE



15. Por fim, você terá algo como:

The screenshot shows a GitHub repository page for 'acolombinifake / WORKFLOW_RESTRICCOES'. The repository has 8 commits, 3 branches, 0 packages, 0 releases, and 3 contributors. A yellow hand icon points to the commit history, which is enclosed in a red dashed circle. The commits are:

File	Message	Time Ago
README.md	Initial commit	19 hours ago
arquivo1.md	Create arquivo1.md	19 hours ago
arquivo2.md	Create arquivo2.md	19 hours ago
arquivo3.md	criação do arquivo3.md	5 hours ago
arquivo4.md	alteração realizada no arquivo4.md	2 hours ago
README.md	(empty)	

A large, solid teal arrow points from left to right, entering a teal-colored rectangular box. Inside the box, the word "Git Stash" is written in a large, white, sans-serif font. Below it, the subtitle "Com esse nome incrível, deve ser importante!!!", also in white, is displayed.

Git Stash

Com esse nome incrível,
deve ser importante!!!



Git STASH



- Para esse nosso laboratório, vamos trabalhar com o último comando do Git para fins deste curso
- Lembre-se há muito mais, e você, dentro das especificidades de seus projetos deverá buscar novas possibilidades destas ferramentas → Git e GitHub
- Nesta prática final, propositadamente reservada para encerrar esse tópico, vamos simular a construção de um TCC, poderia ser qualquer outra coisa que trabalhasse com a criação e gerenciamento de capítulos
- Nesta prática serei menos participativo, você já domina todo conhecimento necessário para realizar o trabalho, apenas intervirei quando algo de novo surgir, então vamos ao trabalho...



GIT STASH



Vamos praticar - Laboratório

Git STASH



1. Crie uma pasta chamada TCC
2. Primeiramente crie o arquivo de Sumario.md e realize seu primeiro commit
3. Nesta pasta crie mais dois arquivos → Capitulo01.md e Capitulo02.md
4. Para cada um destes arquivos adicione algum texto, salve, coloque na Stage e continue trabalhando

Atenção: imagine agora que você precise fazer alguma intervenção em algum de seus arquivos, mas nesta altura, imagine que já tenha trabalhado muito e colocado seu trabalho na Stage, você não quer perder este trabalho, ok!!!

Git STASH



5. Uma das formas de resolver isso, trabalhar nas alterações sem perder o trabalho já realizado é a criação de uma nova Branch chamada **work in progress (wip)**
6. Dentro dessa Branch realize o commit de todas as alterações realizadas
7. Feito isso, volte para a branch master e resolva os problemas que você precisa resolver. Imagine que precise criar um novo tópico no Sumario.md chamado Introdução

Git STASH



8. Imagine que tenha resolvido todos os problemas que precisava, faça o commit das alterações
9. Volte agora para a Branch que criou (wip) e é aqui que teremos novidade, então **fique atento**:
10. Lembra que você fez um commit na (wip), este commit você não quer, então, agora faça um reset voltando para o commit anterior, no caso, o first commit
11. Fazendo um reset do tipo soft use: **git reset head^**
12. Uma vez feito isso, você poderá voltar à Branch Master e continuar trabalhando normalmente

Git STASH



13. Trabalhe nos arquivos, adicione-os na Stage e faça o commit e siga em frente

Atenção: com essa técnica você resolve o problema e segue sem transtornos com seu projeto. O problema é que essa abordagem é um pouco cansativa, para resolver esse problema é que existe o comando **Stash**



Git STASH



- O **stash** nos permite pegar tudo o que foi colocado na Stage e colocar numa área temporária e quando precisar do que foi colocado nesta área temporária, o **stash** permite retirar o que foi adicionado na área temporária e trazer para a área de trabalho normal
- **Para entender o stash pense numa pilha**, o stash pode empilhar vários trabalhos de sua Stage para essa pilha temporária, permitindo que você recupere cada um deles na medida que for precisando e na ordem que lhe for mais conveniente

Git STASH



- A ideia do stash é preservar seu fluxo de trabalho que por algum motivo ainda não estava pronto para virar um commit
- Fique atento, o stash levará para a área temporária todo o trabalho que você tiver adicionado na área de Stage, trabalhos commitados e trabalhos salvos que não foram adicionados na Stage não irão para a área temporária com o comando stash → o comando é simples **git stash** (simples assim)

Git STASH



14. No seu fluxo de trabalho, apenas o Sumario.md foi commitado, logo ele permanecerá na branch master para que você continue trabalhando nele. Faça a alteração desejada, por exemplo, adicione mais uma linha → salve, adicione na Stage e faça mais um commit
15. Feito isso, traga o que foi para a área temporária para sua branch master, use o comando → **git stash apply** (simples assim) e todos os seus arquivos reapareceram na sua branch master, observe no seu Visual Studio Code

Git STASH



16. Finalize os trabalhos, salve os arquivos, acione na Stage e realize o commit



STASH RESOLVENDO CONFLITOS, MENSAGENS

Conflitos, sempre os conflitos

STASH RESOLVENDO CONFLITOS, MENSAGENS



- Nesta prática vamos trabalhar duas ferramentas do Stash
- **Mensagens de Stash** → as mensagens nos ajudam muito a controlar melhor os trabalhos que forem adicionados na pilha de uma forma mais amigável. Atenção, o stash sempre adiciona uma mensagem padrão ao seu stash, agora queremos personalizar os trabalhos
- **Resolução de Conflitos** → quando você tem algum conteúdo na área de stash e você fez alguma alteração na sua área de trabalho, ao tentar trazer de volta esse conteúdo, o Git identificará o conflito e não permitirá sua ação, neste momento, terá que primeiramente resolver a situação conflituosa



STASH RESOLVENDO CONFLITOS E MENSAGENS

Vamos praticar - Laboratório

STASH RESOLVENDO CONFLITOS, MENSAGENS



1. Abra o Capitulo02.md faça um alteração e salve
2. Agora adicione à pilha com o comando stash, veja como → **git stash save 'Alteração no Capitulo02.md'**
3. Um outro comando útil que lhe permite visualizar tudo que foi enviado a pilha de stash é → **git stash list**
4. Um outro comando útil que lhe permite visualizar como as coisas estão é → git stash show –patch, experimente
5. Para simular um conflito continue adicionando alterações no Capitulo02.md

STASH RESOLVENDO CONFLITOS, MENSAGENS



6. Para trazer um conteúdo da stash você poderá usar o comando → **git stash pop**
7. Uma das formas de resolver conflitos é retornar ao passo anterior ao conflito use → **git status** e use o comando → **git checkout <nome-do-arquivo>**
8. Feito isso, realize novamente o comando → **git stash pop** e você trará seu trabalho armazenado na pilha Stash
9. Ok, mas o problema mais comum que iremos encontrar no nosso dia-a-dia é quando já fizemos alterações e nós queremos guardar as que já fizemos e trazer as da área de Stash

STASH RESOLVENDO CONFLITOS, MENSAGENS



10. Para resolver esse problema, basta realizar um commit com o que acabou de trabalhar e aí sim trazer o que estiver na área de Stash normalmente
11. Mesmo que dê algum conflito poderemos tratar os conflitos e realizar um novo commit porque eu tenho o commit anterior salvando as alterações anteriores que eu fiz
12. Para simular isso, vamos levar as alterações realizadas para a área de Stash → **git stash**

STASH RESOLVENDO CONFLITOS, MENSAGENS



13. Faça uma nova alteração no arquivo Capitulo02.md, salvar e tentar trazer da área de Stash o que acabou de enviar para lá → **git stash apply** (isso deverá gerar conflito)
14. Resolva o conflito → faça um commit das alterações que acabou de realizar → **git commit -am 'alterações no Capitulo02.md'**
15. Agora, novamente → **git stash apply** (neste momento será aberta a tela de solução de conflitos, solucione os conflitos e siga em frente)

STASH RESOLVENDO CONFLITOS, MENSAGENS



16. Resolvendo o conflito salve o arquivo e siga o processo, adicione na Stage e faça o commit normalmente
17. Para conferir se está tudo ok, faça um → **git log -oneline** e verifique se todos os commits realizados estão listados





STASH KEEP INDEX



Vamos praticar - Laboratório

STASH KEEP INDEX



- Este é mais um problema que precisaremos administrar no dia-a-dia
- Como visto anteriormente, a área de Stash é uma pilha, sendo possível adicionar várias etapas ou vários status do seu trabalho nesta pilha
- Você deve ter percebido que estando rastreado os documentos irão para a área de Stash
- Acontece, que eu posso querer que nem todos os meus documentos rastreados sejam enviados para a Stash, é disso que se trata essa prática, acompanhe

STASH KEEP INDEX



1. Abra o arquivo Capitulo01.md e insira algumas linhas, salve e adicione na Stage
2. Agora abra o Capitulo02.md e insira algumas linhas também, salve, mas **não** adicione na área de Stage
3. Verifique o status do seu projeto com → **git status**
4. Só para contextualizar, pense nos arquivos que enviou para Stage como sendo aqueles que considera prontos para irem para o próximo commit
5. Por outro lado, existem alterações que realizamos, mas ainda demandam mais trabalho, elas não foram adicionadas na Stage, mas você não quer perder esse trabalho

STASH KEEP INDEX



6. Fique atento, o comando git status nos mostra que todos os seus arquivos estão rastreados, aqueles que você colocou na Stage e aqueles que apenas estão salvos
7. Para entender o que foi dito anteriormente experimente → **git stash** e em seguida → **git status**
8. Você observará que o retorno do → git status será algo como “*On branch master nothing to commit, working tree clean*”

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified: Capítulo01.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified: Capítulo02.md
```

STASH KEEP INDEX



9. Traga seus trabalhos da Stash de volta para sua branch → **git stash apply** e em seguida **git status** novamente

```
$ git stash apply
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Capitulo01.md
        modified:   Capitulo02.md

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

10. Coloque novamente Stage as alterações que realizou no Capitulo01.md → **git add . Capitulo01.md**
11. E agora realize alterações no Capitulo02.md
12. Verifique novamente seu status → **git status**, e preste atenção nas mensagens

STASH KEEP INDEX



13. Vamos agora passar para o comando stash uma **flag diferente** (**--keep-index**). Esta **flag diz** (só leve para Stash aquilo que eu não coloquei na Stage), observe → **git stash --keep-index**
14. Observe que o status de seu projeto deve ser algo parecido com → **git status**

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   Capitulo01.md
```

STASH KEEP INDEX



- 15.Para finalizar essa prática faça o commit da alteração
git commit -m 'alteração no Capítulo01.md'

```
$ git commit -m 'alterações em cap1'  
[master fe2311a] alterações em cap1  
 1 file changed, 4 insertions(+), 1 deletion(-)
```

16. traga de volta da Stash o capítulo que acabou de enviar → **git stash pop**

```
$ git stash pop  
On branch master  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working directory)  
  
        modified:   Capitulo02.md  
  
no changes added to commit (use "git add" and/or "git commit -a")  
Dropped refs/stash@{0} (1677db7e204f721704da1965af6b52343edfb72c)
```

STASH KEEP INDEX



17. Adicione o que acabou de trazer da Stash na Stage e faça o commit → **git commit –am 'alteração no Capítulo02.md'**

```
$ git commit -am 'alteração em cap2'  
[master ccd72bd] alteração em cap2  
 1 file changed, 1 insertion(+)
```





STASH ANTES DO PULL



Vamos praticar - Laboratório

STASH ANTES DO PULL



- Esta tudo certo, mas é só isso mesmo?
- Legal, mas e se eu estiver pensando na pilha do stash contendo vários documentos empilhados, como podemos trazer de volta o trabalho que desejamos?
- Da forma que trabalhamos até aqui, trouxemos sempre o último trabalho adicionado na Stash, em termos de pilha, trouxemos o que estava no topo da pilha. Vamos melhorar isso, queremos trabalhar num ambiente colaborativo, lembra!!
- Neste último laboratório do curso, chamo a atenção para essa complexidade adicional muito bem vinda, é com vocês agora ...

STASH ANTES DO PULL



1. Vamos resgatar um dos antigos projetos trabalhados, por exemplo → git-pratica-v2 → utilizado no fluxo de trabalho do GitHub Workflow
2. Caso não tenha utilizado esse nome, use qualquer outro que tenha trabalhado colaborativamente. Você pode inclusive montar um cenário para essa prática, você já aprendeu como
3. Faça o clone desse repositório e abra seu Visual Studio Code

STASH ANTES DO PULL



4. Faça uma modificação do arquivo Dica-1.md
5. Nesse meio tempo faça uma alteração no mesmo arquivo lá no seu repositório Upstream
6. No seu repositório local, salve suas alterações e tente sincronizar, utilize o comando → **git pull**. Observe que você receberá uma mensagem de erro, dizendo que o repositório local está dessincronizado em relação ao repositório principal Upstream

```
From https://github.com/FabricioMachado/g12-pratica-12
 * [new branch]      master      -> origin/master
error: Your local changes to the following files would be overwritten by merge:
      Dica-1.md
Please commit your changes or stash them before you merge.
Aborting
Updating 8af2bab..e349ccf
```

STASH ANTES DO PULL



7. Você já sabe solucionar este tipo de problema, nosso desafio aqui, é fazer de forma diferente, vamos lá
8. Primeiramente você pode colocar na área de Stash todas as alterações que realizou, fazer o **git pull** normalmente e quando você quiser, ou quando tiver tempo
9. Com tranquilidade poderá trazer de volta suas alterações da Stash e verificar quais foram os conflitos que aconteceram

STASH ANTES DO PULL



7. Passo 1 → **git stash** (suas alterações foram deslocadas para a área de Stash)
8. Passo 2 → **git pull** (será realizado com tranquilidade, pois o repositório se encontra pronto para receber as atualizações do repositório principal Upstream. Observe que no pull já visualizamos que houve alteração no arquivo Dica-1.md, o mesmo que trabalhou ainda a pouco



```
$ git pull
Updating 8af2bab...e349ccf
Fast-forward
  Dica-1.md | 2 ++
  1 file changed, 2 insertions(+)
```

A terminal window showing the output of a git pull command. The command is at the top, followed by 'Updating' and a commit hash. Below that is 'Fast-forward' and a list of files. The first file listed is 'Dica-1.md' with two changes. At the bottom, it says '1 file changed, 2 insertions(+)'.

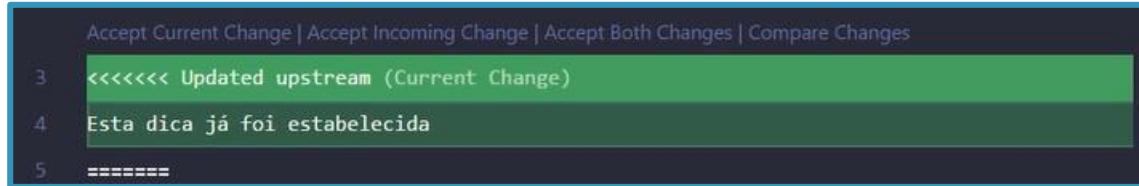
STASH ANTES DO PULL



9. Veja que legal, faça agora → **git stash pop** e o conflito estará a seu dispor para que faça os devidos ajustes

A screenshot of a terminal window showing the command '\$ git stash pop' being run. The output shows 'Auto-merging Dica-1.md' and 'CONFLICT (content): Merge conflict in Dica-1.md'. A yellow hand icon points to the command 'git stash pop'.

10. Com a tela de conflito aberta, passa a ser um trabalho manual, resolva o conflito e siga os passos para finalizar seu trabalho

A screenshot of a merge conflict resolution interface. At the top, there are four buttons: 'Accept Current Change', 'Accept Incoming Change', 'Accept Both Changes', and 'Compare Changes'. Below these buttons, there are three numbered lines of code:
3 <<<<< Updated upstream (Current Change)
4 Esta dica já foi estabelecida
5 =====

STASH ANTES DO PULL



11. Feitas as alterações, salve o arquivo e efetue o commit, em seguida use o comando push para enviar suas alterações para o repositório Upstream



Por hoje é só pessoal!!!

Nos vemos na próxima aula
Sucesso a todos!!!!