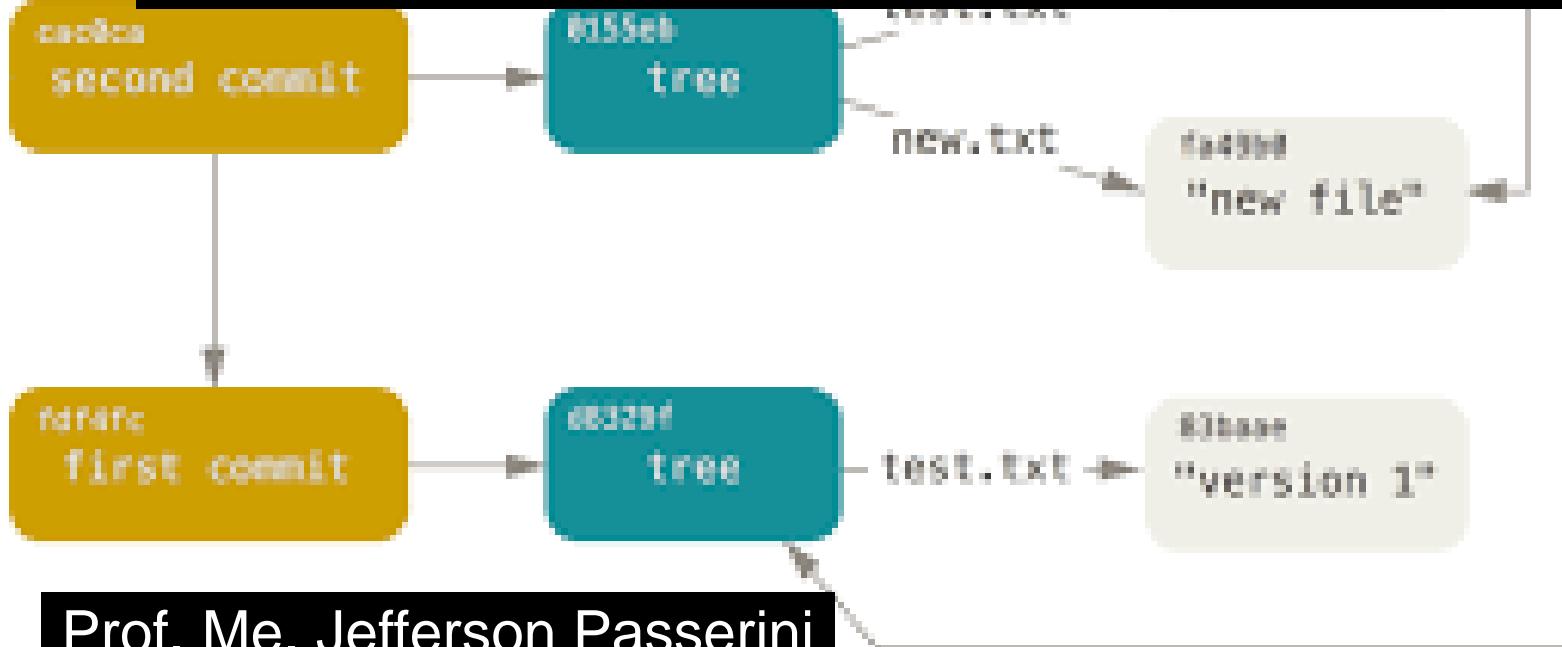


Da-410  
thi

bak

# GIT & Github

## Projeto Integrador (Interdisciplinar)



Prof. Me. Jefferson Passerini

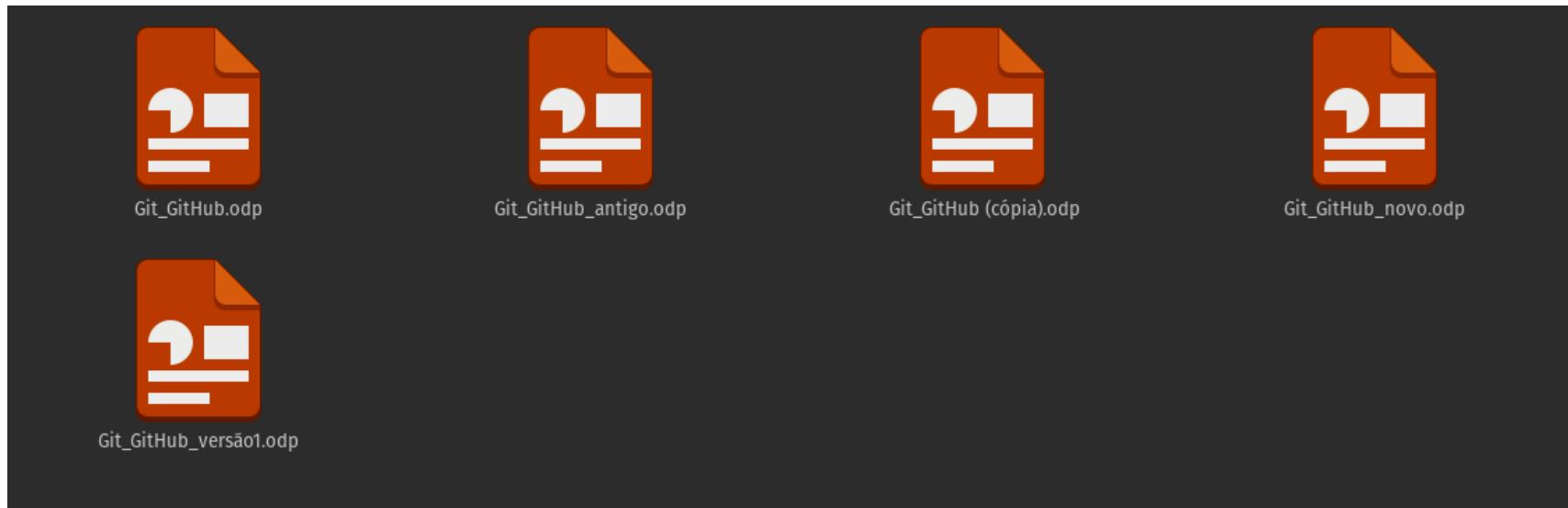


## .Problema

- Você provavelmente deve possuir diversos arquivos em seu computador pessoal;
- Sejam fotos, vídeos, planilhas, documentos do trabalho, faculdade e/ou pessoais;
- E se algo trágico acontecesse com o seu computador como ele queima, der um problema no HD ou até mesmo ser furtado?
- Além do prejuízo financeiro, você perderia todos os seus arquivos.

# .Problema

- Quem nunca fez isso?



## .Problema

- Por conta deste risco, quase todas as pessoas, inclusive pessoas que não trabalham diretamente com a área de tecnologia, já possuem o hábito de realizar cópias de seus arquivos.
- Seja utilizando algum serviço, como Dropbox, Onedrive (Microsoft) ou Google Drive ou ainda utilizando algum disco externo.
- **E no trabalho de desenvolvimento de software ?**

## .Problema

- No processo de desenvolver um software temos a mesma necessidade, pois diversos arquivos são criados, modificados e excluídos ao longo do tempo;
- Além disso, é raro trabalharmos de maneira isolada durante o projeto.
- Geralmente trabalhamos simultaneamente com outras pessoas, que juntas formam um time, e com isso surge um problema: **como fazer para sincronizar o trabalho de todo mundo sem que uma pessoa sobrescreva o trabalho da outra.**

## .O que é controle de versionamento ?

- Para resolver esses problemas e facilitar a vida de quem trabalha com desenvolvimento de software, diversas ferramentas foram criadas ao longo do tempo
- Esse tipo de ferramenta é conhecido como VCS (*Version Control Systems*) ou também como SCM (*Source Code Management*), sendo hoje considerado como obrigatório para qualquer tipo de projeto de software.

## .O que é controle de versionamento ?

- Ajudam o time a gerenciar as alterações no código;
- Permite entender a evolução do código do projeto;
- Ramificação (Branches): criar versões a partir de outras, preservando sua origem e junção (merge) de versões;
- Facilita a reversão quando necessário de algum trecho de código;
- Permite a rastreabilidade das alterações;
- Favorece a colaboração no time de projeto;

.O que é controle de versionamento ?

- Exemplos:

- CVS
- Subversion (SVN)
- SourceSafe
- Git
- Mercurial

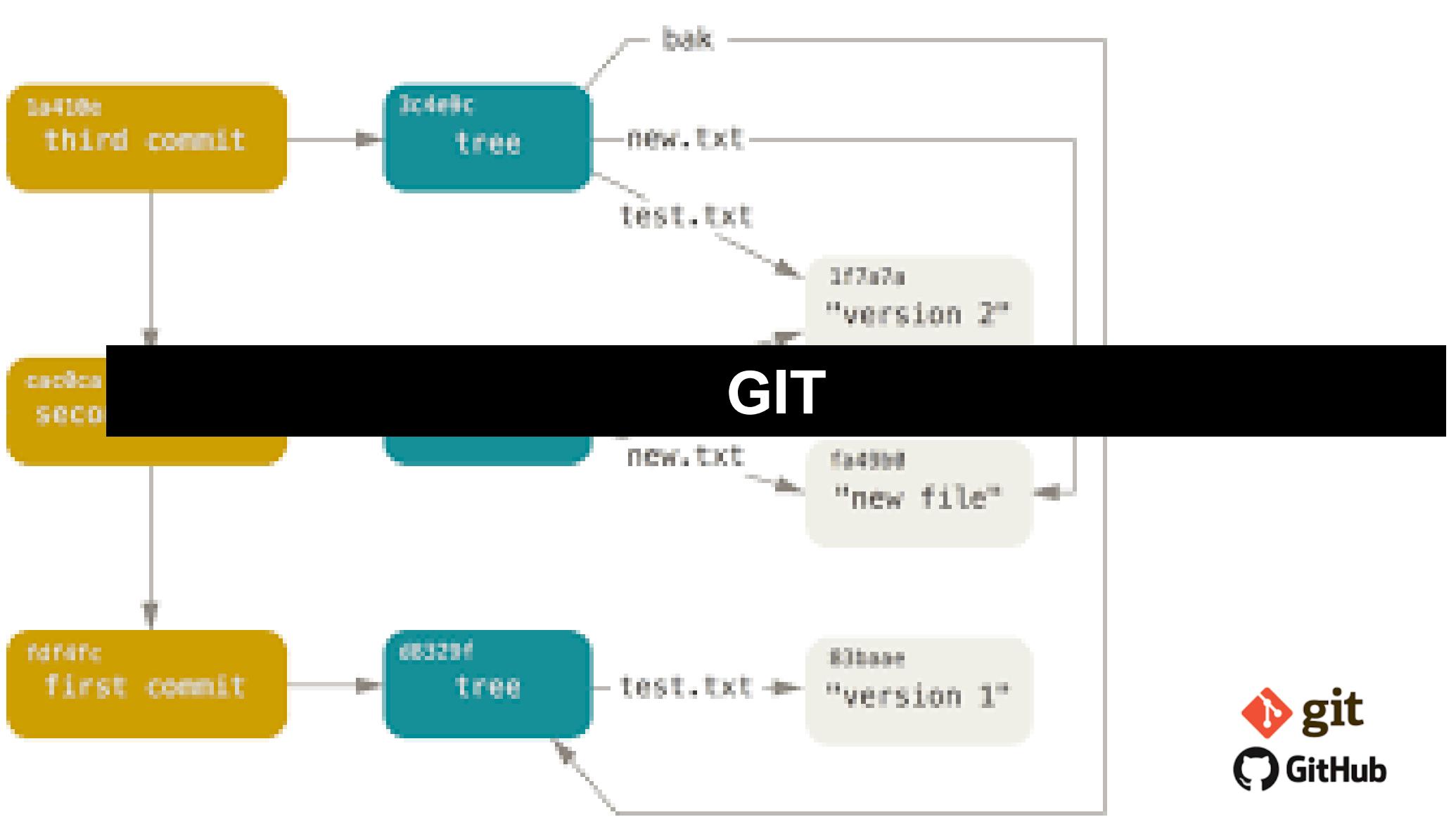
## .Tipos de controle de versionamento

- Centralizado:

- Versões no servidor;
- Padrão dos sistemas Subversion, Perforce e Team Foundation Server.

- Distribuído:

- Cada computador possui todo o histórico de versões;
- Ambiente de replicação;
- Mais segurança;
- Exemplos: Git, Mercurial.



## .Git

- Criado em 2005 por Linus Torvalds;
- Vantagens:
  - Gratuito e open source
  - Distribuído
  - Alta performance
  - Extensível
  - Suporte a diferentes tipos de workflow

## .Git

- O Git por ser distribuído tem operações locais de:
  - Navegação pelo histórico de mudanças;
  - Criação de ramificações (branches)
- Integridade
  - Quando um arquivo é adicionado, todo o seu histórico é mantido.
- Somente adição
  - Arquivos removidos não são excluídos, ou seja, são marcados como excluídos.
- Colaboração e autonomia
  - Usuários podem realizar alterações sem depender de arquivos do servidor.

## .Git

- Muitas Ferramentas e serviços foram criadas baseados no Git;
- Um exemplo disso é o GitHub que é um site que funciona como serviço de hospedagem de código fonte de projetos que utilizam Git como controle de versão.
- Sendo a ferramenta que mais disseminou o uso do Git ao redor do mundo.
- E sim: **GitHub e Git são coisa diferentes!**

## .Git - Instalação

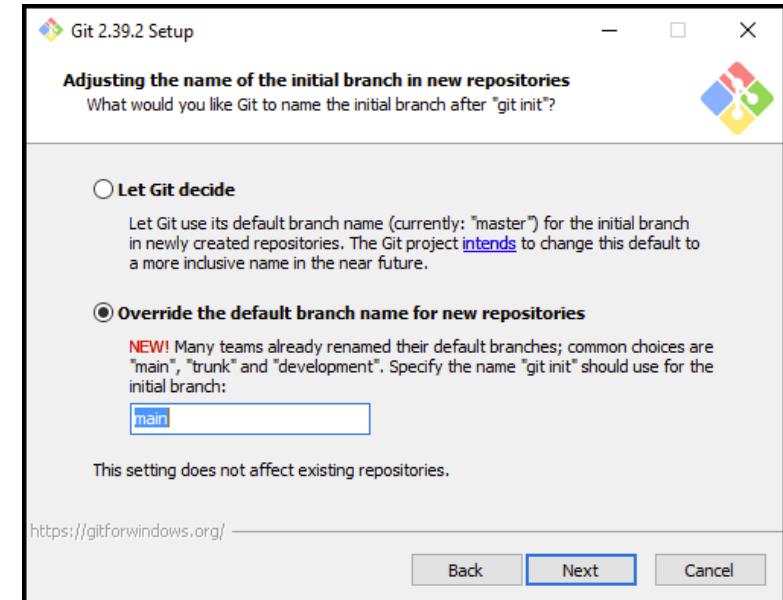
- Faça o download da versão de acordo com o seu sistema operacional: <https://git-scm.com>



.Para instalar: next, next, next, ... Finish!!

## .Git - Instalação

- A partir da versão 2.29.0, o Git começa a sugerir a mudança de nome do branch **master** para **main**.
- Alterar na instalação
- Senão:
  - **git branch -m master  
main**

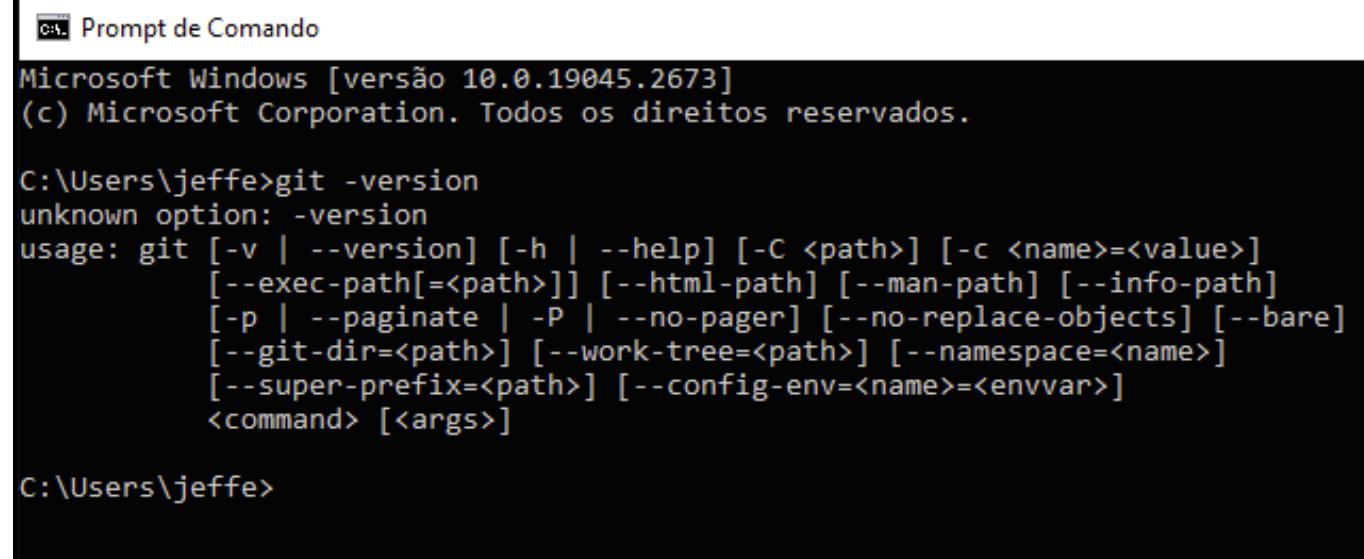


## .Git

- Existem duas principais maneiras de se utilizar o Git: via linha de comando e via interface gráfica.
- As IDEs de desenvolvimento geralmente implementam ferramentas para trabalhar com o Git.
- Mas ao longo do curso vamos trabalhar com o Git via linha de comando, utilizando o terminal, para não ficar dependente de uma ferramenta ou plugin em específico.

## .Git

- Depois de instalado, abra o prompt de comando (CMD) ou PowerShell e digite o seguinte comando
- **git --version**



```
C:\ Prompt de Comando
Microsoft Windows [versão 10.0.19045.2673]
(c) Microsoft Corporation. Todos os direitos reservados.

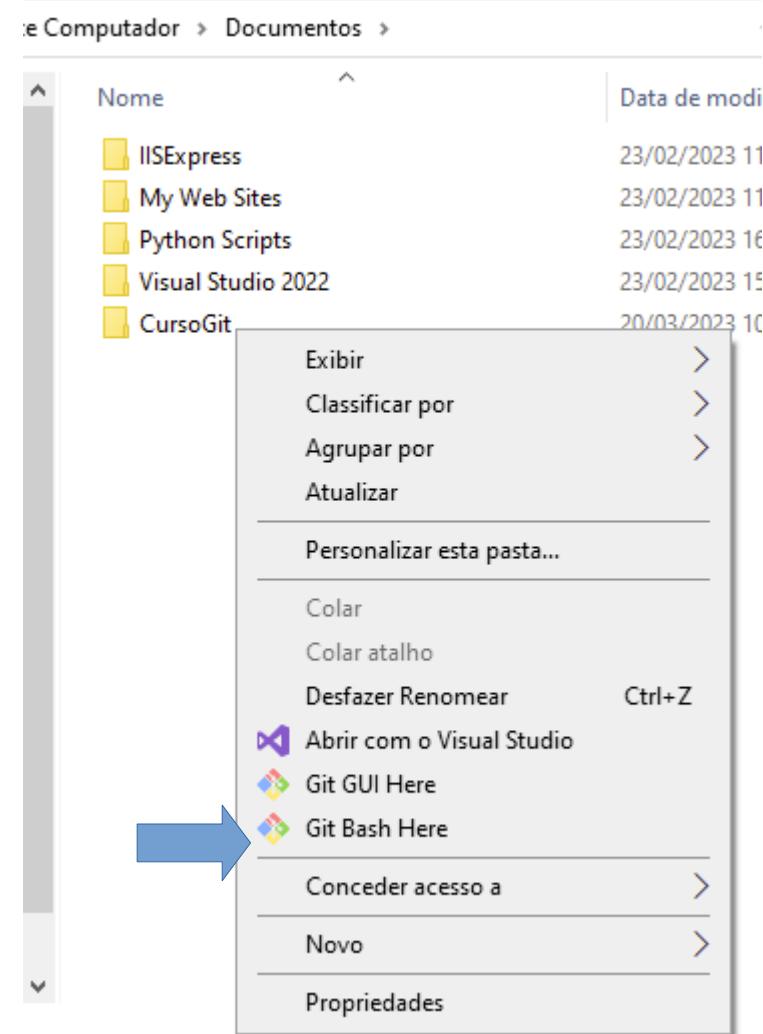
C:\Users\jeffe>git -version
unknown option: -version
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

C:\Users\jeffe>
```

.Crie uma pasta para nossos testes – CursoGit (nos documentos);

.Com o Windows Explorer clique com o botão direito na área de trabalho desta pasta e escolha a opção Git Bash Here.

.O Bash é uma interface de linha instalada com o git que permite a utilização dos comandos git de maneira intuitiva



## .Git – Configurações Globais

- As duas principais configurações a serem realizadas no git é o nome e o e-mail do usuário;
- Isso serve para identificar quem está fazendo as alterações no projeto;
- Digite os comandos:
  - **git config --global user.name “Nome do Usuário”**
  - **git config --global user.email “email@email.com”**
- Depois verifique:
  - **git config --list**

## .Git – Configurações Globais

- O parâmetro --global indica ao Git que esse mesmo nome e e-mail deve ser utilizado em todos os novos repositórios Git que forem criados ou baixados a partir desse computador.
- Para que assim você não precise ficar repetindo o comando anterior a cada novo repositório que for trabalhar.

## .Git – Configurações Globais

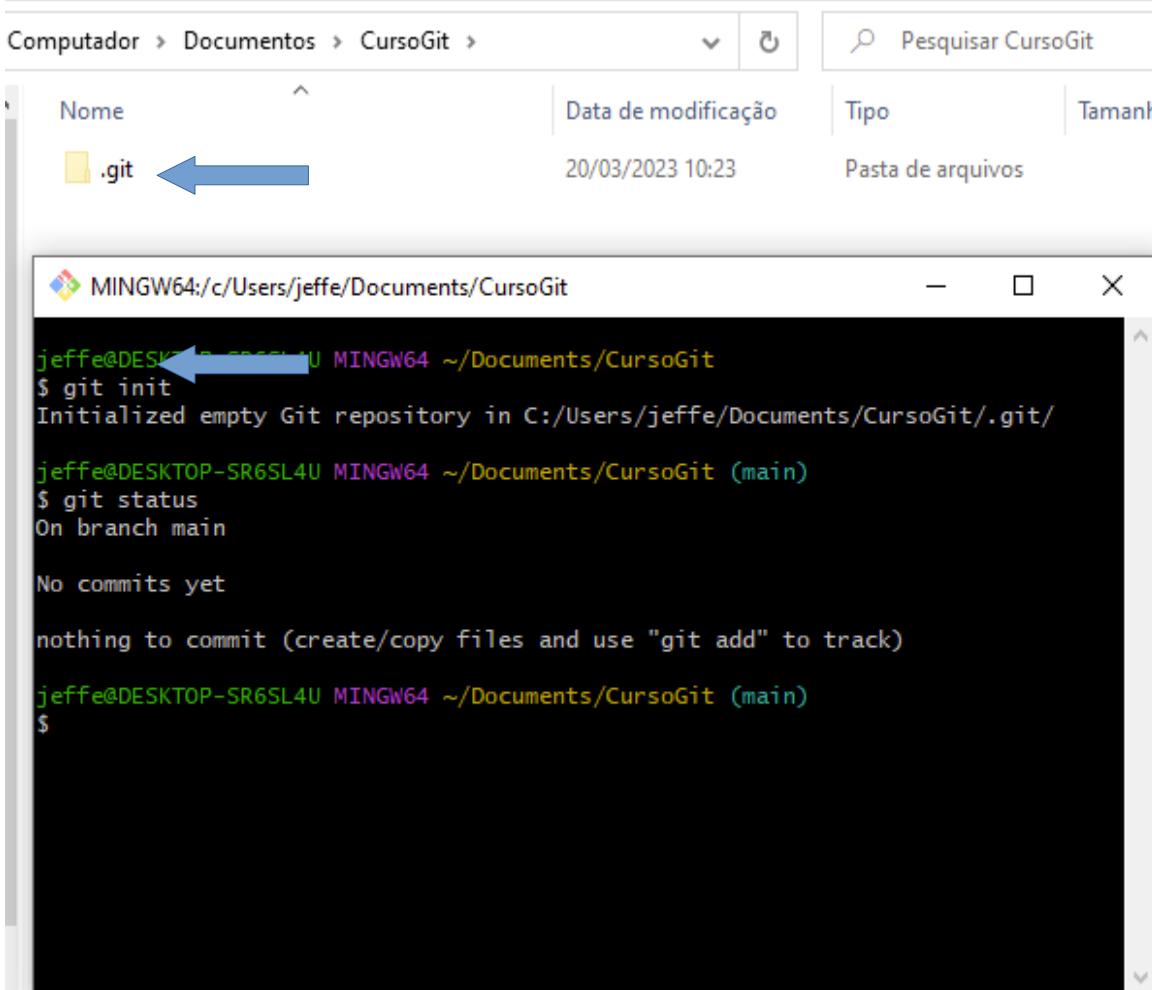
- Para desconfigurar as configurações globais de user, temos:
  - **git config --global --unset-all user.name**
  - **git config --global --unset-all user.email**
- Depois verifique:
  - **git config --list**

## .Git – Repositório

- No git existe um termo importante chamado repositório, que nada mais é do que o diretório raiz da aplicação, no qual o time de desenvolvimento vai criar o código fonte dele.
- Cada aplicação deve ter seu próprio repositório separado.

# .Git – Repositório

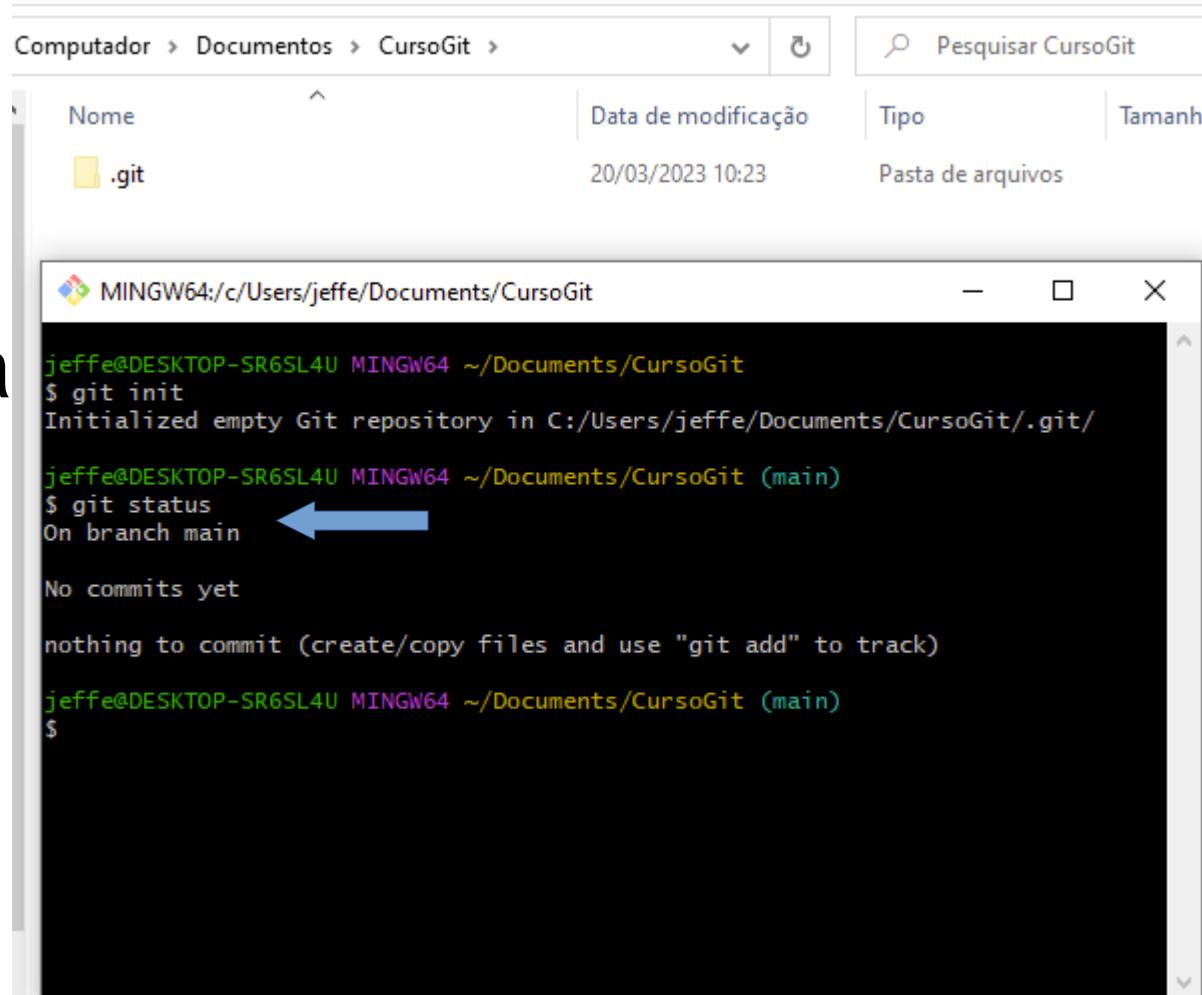
- Para criar o repositório basta rodar: **git init**;
- O git então criará um diretório oculto chamado **.git**;
- Esse diretório será utilizado para armazenar o histórico de alterações dos arquivos dentre outras informações.



The image shows a screenshot of a Windows operating system. At the top, there is a taskbar with several pinned icons. Below the taskbar, the Start menu is open, showing search results for 'git'. The main window is a File Explorer showing the file structure: 'Computador > Documentos > CursoGit'. Inside 'CursoGit', there is a folder named '.git'. A blue arrow points from the text 'basta rodar: git init;' in the slide to this '.git' folder. In the bottom right corner of the slide, there is a small terminal window icon. This icon is also highlighted with a blue arrow. When clicked, it opens a terminal window titled 'MINGW64:/c/Users/jeffe/Documents/CursoGit'. The terminal shows the following session:  
jeffe@DESKTOP-SR6SL4U MINGW64 ~ /Documents/CursoGit  
\$ git init  
Initialized empty Git repository in C:/Users/jeffe/Documents/CursoGit/.git/  
  
jeffe@DESKTOP-SR6SL4U MINGW64 ~ /Documents/CursoGit (main)  
\$ git status  
On branch main  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)  
  
jeffe@DESKTOP-SR6SL4U MINGW64 ~ /Documents/CursoGit (main)  
\$

# .Git – Repositório

- O comando **git status** permite verificar o status da branch;
- No exemplo foi criada como main conforme a recomendação do git.



Computador > Documentos > CursoGit >

Nome	Data de modificação	Tipo
.git	20/03/2023 10:23	Pasta de arquivos

MINGW64:/c/Users/jeffe/Documents/CursoGit

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit
$ git init
Initialized empty Git repository in C:/Users/jeffe/Documents/CursoGit/.git/

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main ←
No commits yet
nothing to commit (create/copy files and use "git add" to track)

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$
```

## .Git – Repositório

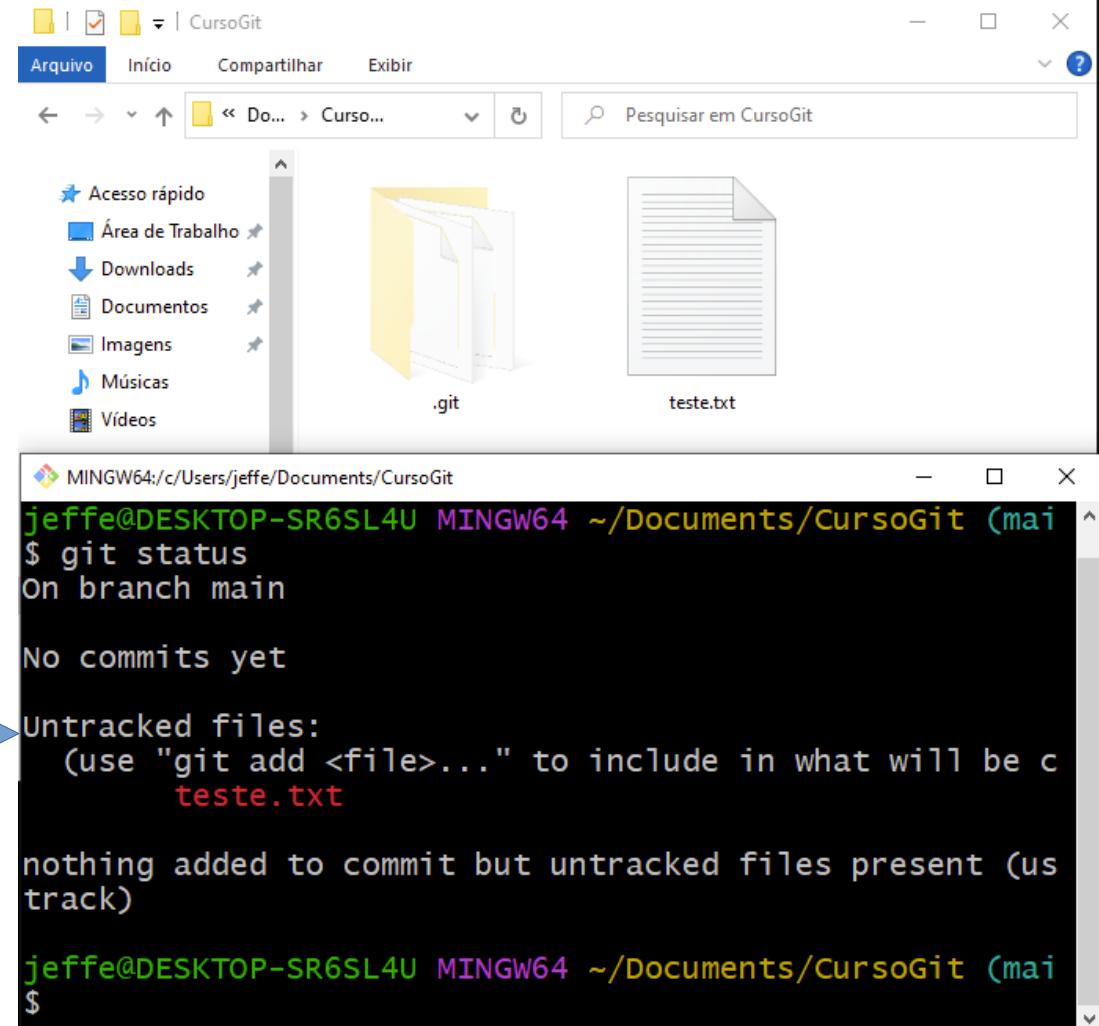
- Caso o seu **branch default** esteja configurado como **master** e;
- Se no processo de criação do repositório ele foi criado como **master** você tem a opção de alterá-lo para **main**, com o comando:
  - **git branch -m master main**

## .Git – Exibindo arquivos criados pelo git

- Para exibir os arquivos criados pelo git, você pode alterar as configurações do Windows permitindo que pastas e arquivos ocultos sejam exibidos;
- Pode também **usar comandos do Bash** para listar os arquivos ocultos, veja:
  - **dir -a** → mostra a pasta oculta
  - **dir .git** → mostra o conteúdo da pasta
  - **cd .git** → entra na pasta (sai do branch main)
  - **find .** → Mostrar todas as subpastas

## . Adicionando arquivos ao diretório

- Dentro da pasta CursoGit crie um arquivo chamado teste.txt usando o Windows para isso.
- Depois digite o comando
  - **git status**
- Então será exibida uma mensagem que há um arquivo não rastreado (**untracked**), ou seja, o git entende que esse arquivo não pertence ao repositório.



```
MINGW64:/c/Users/jeffe/Documents/CursoGit
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (mai
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be c
    teste.txt

nothing added to commit but untracked files present (us
track)

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (mai
$
```

.Adicionando arquivos ao diretório

.Assim, é necessário adicioná-lo ao repositório:

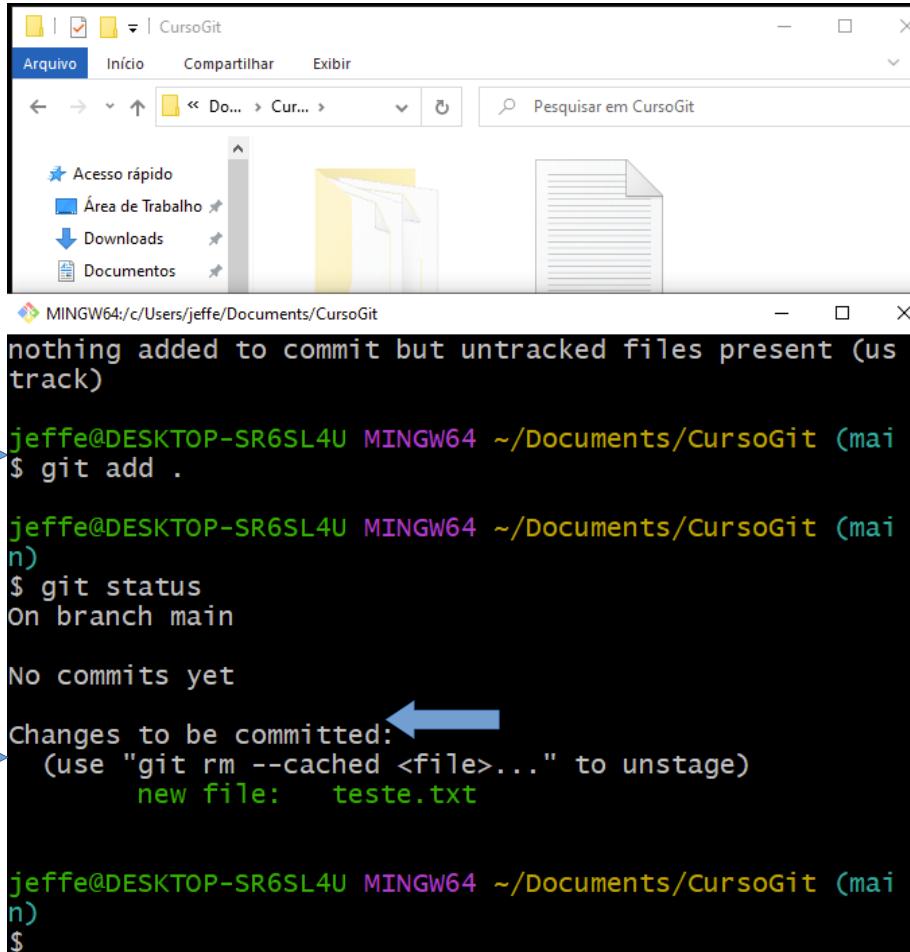
- **git add <nome do arquivo>**

.Ou todos os arquivos:

- **git add -all** (ou)
- **git add .**

.Agora verifique o status do arquivo novamente com o **git status**.

.O arquivo está rastreado (**tracked**).



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

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git add .

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main

No commits yet

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

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$
```

. Adicionando arquivos ao diretório

. Como visto podemos adicionar apenas alguns arquivos indicando o nome do arquivo específico, ou ainda indicar todos só arquivos através do **git add .;**

. Temos que enfatizar:

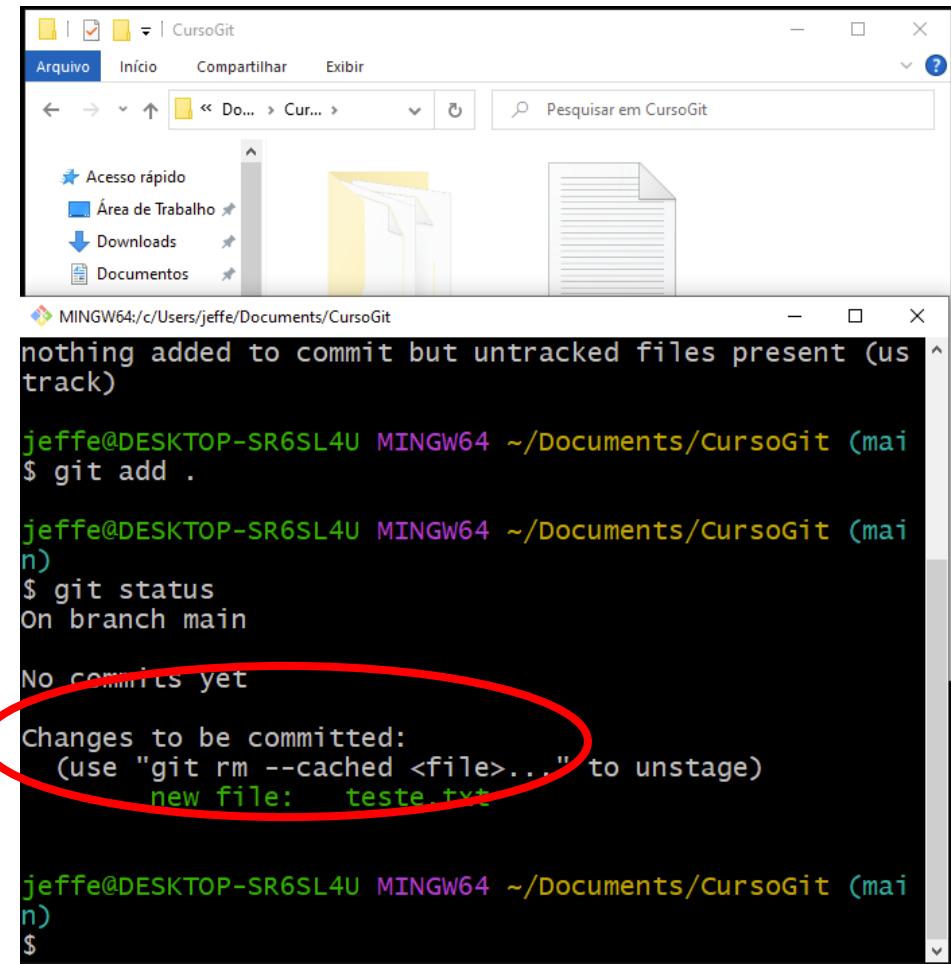
- O ponto “.” utilizando indica ao git que queremos adicionar quaisquer alterações que tiverem sido realizadas no repositório.
- Inclusive em arquivos que estejam em subdiretórios, ou seja, o comando é recursivo.

- .Salvando as alterações (commit)
- .Outro termo importante e recorrente no git é o **commit**, que nada mais é do que a representação de um registro de alteração efetuada por alguém em um determinado repositório.
- .Sempre que a pessoa do time de desenvolvimento desejar registrar no histórico do repositório alguma alteração, ela precisará fazer um commit.
- .Assim o git deixará registrado quem o efetuou, quanto foi feito e, principalmente, quais arquivos foram modificados.

.Salvando as alterações  
(commit)

.Em nosso exemplo o arquivo teste.txt está com o status de tracked, e ele está pronto para ter suas alterações salvas e registradas no histórico.

.Atenção: quando um arquivo é alterado, essas alterações ainda não foram efetivas e oficializadas.



```
MINGW64:/c/Users/jeffe/Documents/CursoGit
nothing added to commit but untracked files present (use "git add" to track)

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git add .

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main

No commits yet

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

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$
```

.Salvando as alterações (commit)

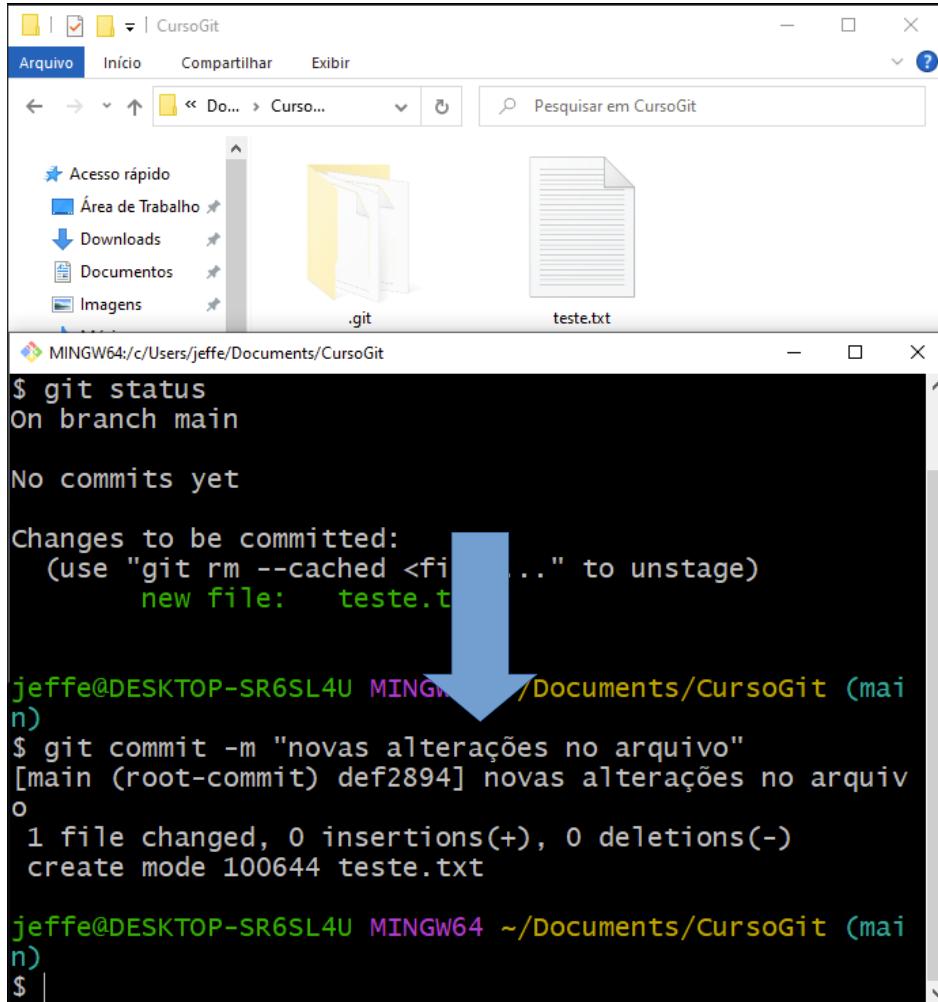
.Para efetuar um commit:

- **git commit -m <mensagem do commit>**

.O parâmetro -m é utilizado para escrever uma mensagem que explique o que aquele commit representa para a aplicação. Então:

- **git commit -m “novas alterações no arquivo”**

.Lógico que essa não é uma boa mensagem.



```
$ git status
On branch main
No commits yet
Changes to be committed:
  (use "git rm --cached <file>" to unstage)
    new file:   teste.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git commit -m "novas alterações no arquivo"
[main (root-commit) def2894] novas alterações no arquivo
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 teste.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ |
```

.Salvando as alterações (commit)

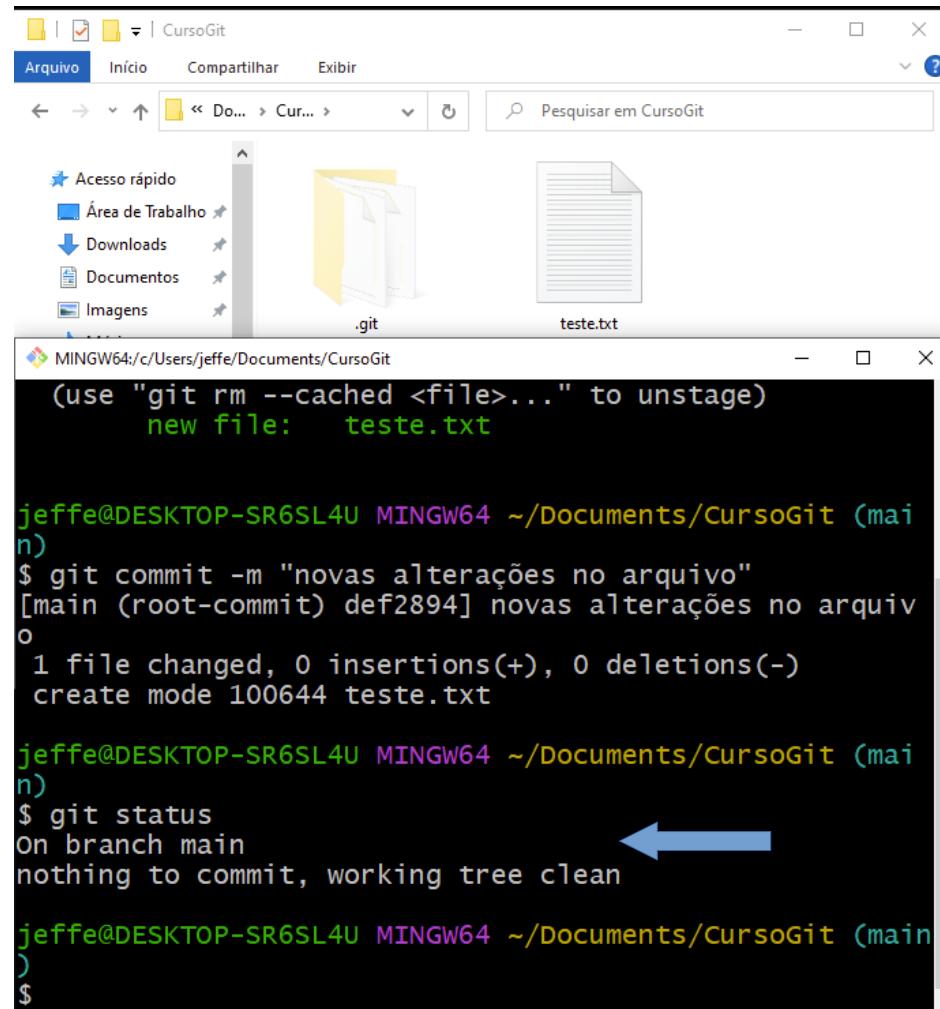
.Verifique o commit com o comando:

- **git status**

.Agora nossa branch apresenta como status:

- Nothing to commit
- Working tree clean.

.Nossas alterações foram “comitadas”.



```
MINGW64:/c/Users/jeffe/Documents/CursoGit
(use "git rm --cached <file>..." to unstage)
new file:   teste.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git commit -m "novas alterações no arquivo"
[main (root-commit) def2894] novas alterações no arquivo
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 teste.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
nothing to commit, working tree clean

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$
```

A blue arrow points to the last line of the terminal output: "nothing to commit, working tree clean".

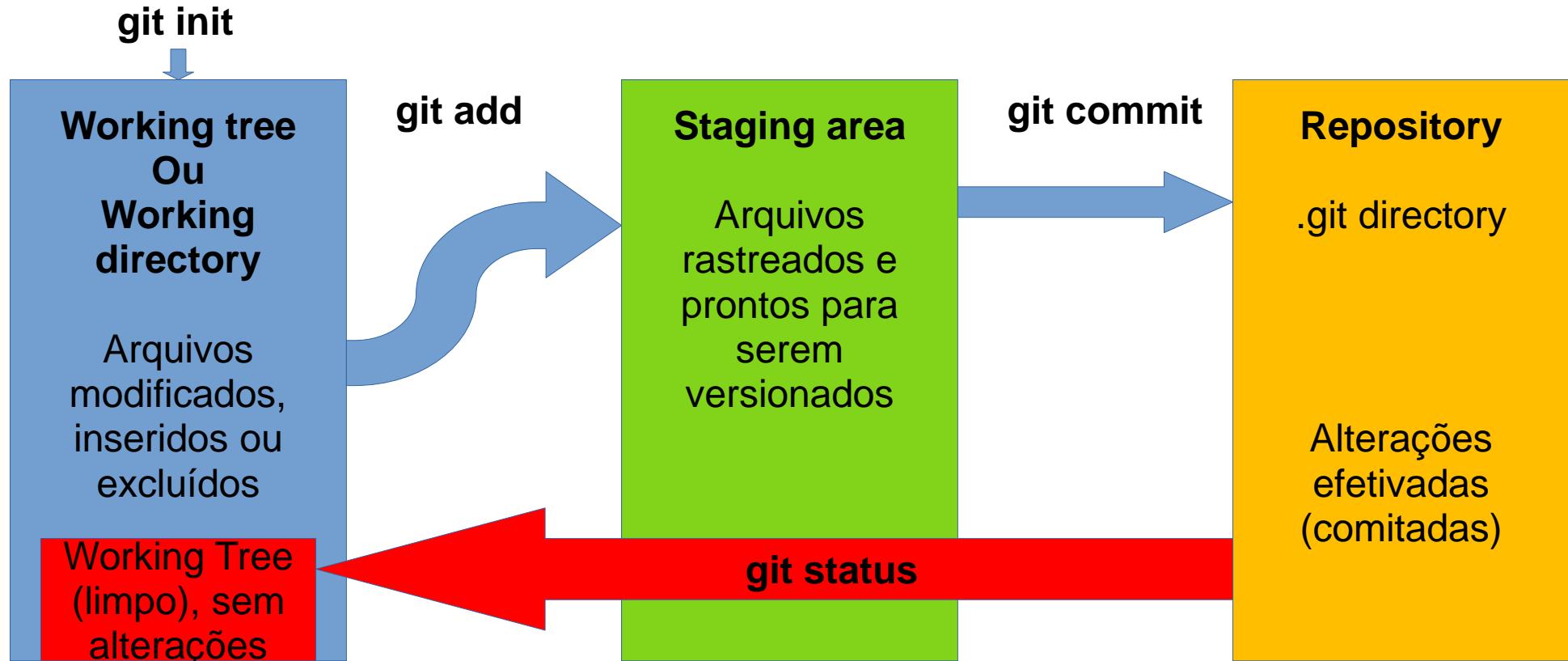
.Salvando as alterações (commit)

.Alguns exemplos e mensagens:

- Correção do bug no botão de logout
- Funcionalidade de login
- Ajuste no tamanho das fontes
- Alteração da cor de fundo da tabela de novos clientes
- Correção ortográfica na mensagem de erro
- Adicionado campo de telefone no cadastro de cliente
- Adicionado novo item na barra de menu superior
- Simplificando código que calcula o valor do saldo devedor.

.Perceba que as mensagens costumam ser curtas e objetivas em relação ao que aquele commit representa, facilitando assim quando necessário analisar os commits do repositório.

## .Compreendendo os estado do git.



Dentro do diretório de trabalho os arquivos podem assumir os estados:  
untracked(não rastreado), unmodified (não modificado) e modified(modificado).

## • Praticando

- Abra o arquivo teste.txt e adicione uma linha ao arquivo.
- Verifique agora o status do arquivo com o **git status** – as alterações estão “not staged”.
- Torne as alterações rastreadas com o **git add** e verifique com o **git status** – agora as alterações estão prontas para serem comitadas.
- Realize o commit e verifique o status novamente: **git commit -m “nova linha inserida no arquivo”**. Alterações comitadas e working tree limpa novamente.

```
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:  teste.txt

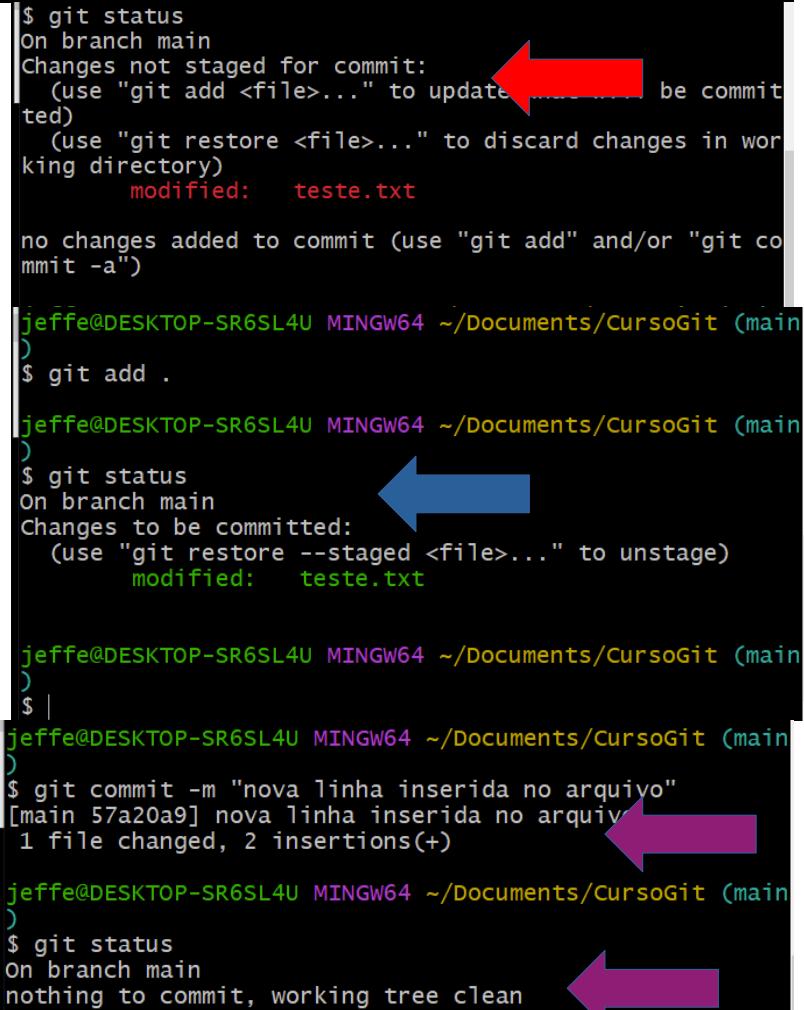
no changes added to commit (use "git add" and/or "git commit -a")

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git add .

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:  teste.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ |
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git commit -m "nova linha inserida no arquivo"
[main 57a20a9] nova linha inserida no arquivo
 1 file changed, 2 insertions(+)

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
nothing to commit, working tree clean
```



## .Praticando

- Agora crie o arquivo **trabalho.txt**.
- Verifique agora o status do arquivo com o **git status** – as alterações estão “**not staged**”.
- Torne as alterações rastreadas com o **git add** e verifique com o **git status** – agora as alterações estão prontas para serem comitadas.
- Realize o commit e verifique o status novamente: **git commit -m “arquivo trabalho.txt criado”**. Alterações comitadas e working tree limpa novamente.

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    trabalho.txt ← Red arrow pointing to the file listed

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

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   trabalho.txt ← Blue arrow pointing to the file listed

$ git commit -m "arquivo trabalho.txt criado"
[main 00a2877] arquivo trabalho.txt criado ← Purple arrow pointing to the commit message
  1 file changed, 1 insertion(+)
  create mode 100644 trabalho.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
nothing to commit, working tree clean ← Purple arrow pointing to the final status message
```

## • Listando os commits do repositório

- Outro comando importante do git é o **git log**, utilizado para listar todos os commits registrados no repositório. Veja como exemplo nossos commits realizados até o momento:

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git log
commit 00a2877c3be4164ed2077419fd29f73e27d6f5ef (HEAD ->
  main)
Author: jeffersonarpasserini <jefferson.passerini@gmail.
com>
Date:   Mon Mar 20 17:04:23 2023 -0300

  arquivo trabalho.txt criado

commit 57a20a94233a4205686292e0f628b4e2b7bf3067
Author: jeffersonarpasserini <jefferson.passerini@gmail.
com>
Date:   Mon Mar 20 16:57:29 2023 -0300

  nova linha inserida no arquivo

commit def28945404855bd9b133147d77900e80643a9fa
Author: jeffersonarpasserini <jefferson.passerini@gmail.
com>
Date:   Mon Mar 20 16:20:19 2023 -0300

  novas alterações no arquivo
```

## • Listando os commits do repositório

- Repare que para cada commit, o git exibe o autor do commit, a data e a mensagem.
- Além disso repare que cada commit possui um **identificador único**.
- Esse identificador, também é **chamado de id**, é gerado automaticamente pelo git e serve para diferenciar um commit do outro.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git log
commit 00a2877c3be4164ed2077419fd29f73e27d6f5ef (HEAD ->
  main)
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 17:04:23 2023 -0300

    arquivo trabalho.txt criado

commit 57a20a94233a4205686292e0f628b4e2b7bf3067
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 16:57:29 2023 -0300

    nova linha inserida no arquivo

commit def28945404855bd9b133147d77900e80643a9fa
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 16:20:19 2023 -0300

    novas alterações no arquivo
```

.É possível um arquivo estar em dois estados, ou seja no working directory e no staging area?

- Sim!!

.Abra o arquivo trabalho.txt e adicione uma nova linha.

.Agora adicione o arquivo com o comando **git add --all**; a partir de agora ele está no **staging área**.

.Agora adicione outra modificação (uma nova linha por exemplo), e verifique o status do arquivo com o **git status**.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>" to unstage)
    modified:   trabalho.txt ← green arrow

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:   trabalho.txt ← red arrow
```

.Problema – o que fazer?

.Se nesse momento usarmos o **commit**, apenas a alteração que está no **staging** será **efetivada**.

.Se usarmos novamente o **git add**, a nova alteração será colocada no **staging area** e será fundida com a que já está lá. Assim apenas uma alteração será registrada? O que fazer ?

.Vamos adicionar e ver o que acontece. Use o comando **git add --all** e depois o **git status**.

.Apenas uma alteração estará no staging.

.Agora faça o commit:

– **git commit -m “alteração no trabalho.txt”**

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   trabalho.txt ← green arrow

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:   trabalho.txt ← red arrow
```

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git add .
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   trabalho.txt
```

## .Problema – o que fazer?

.Se nesse momento usarmos o **commit**, apenas a alteração que está no staging será efetivada.

.Se usarmos novamente o **git add**, a nova alteração será colocada no staging area e será fundida com a que já está lá. Assim apenas uma alteração será registrada? O que fazer ?

.Vamos adicionar e ver o que acontece. Use o comando **git add --all** e depois o **git status**.

.Apenas uma alteração estará no staging.

.Agora faça o commit:

- **git commit -m “alteração no trabalho.txt”**
  
- Como ficou o arquivo?
- Ambas as alterações foram gravadas

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   trabalho.txt ← green arrow

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:   trabalho.txt ← red arrow
```

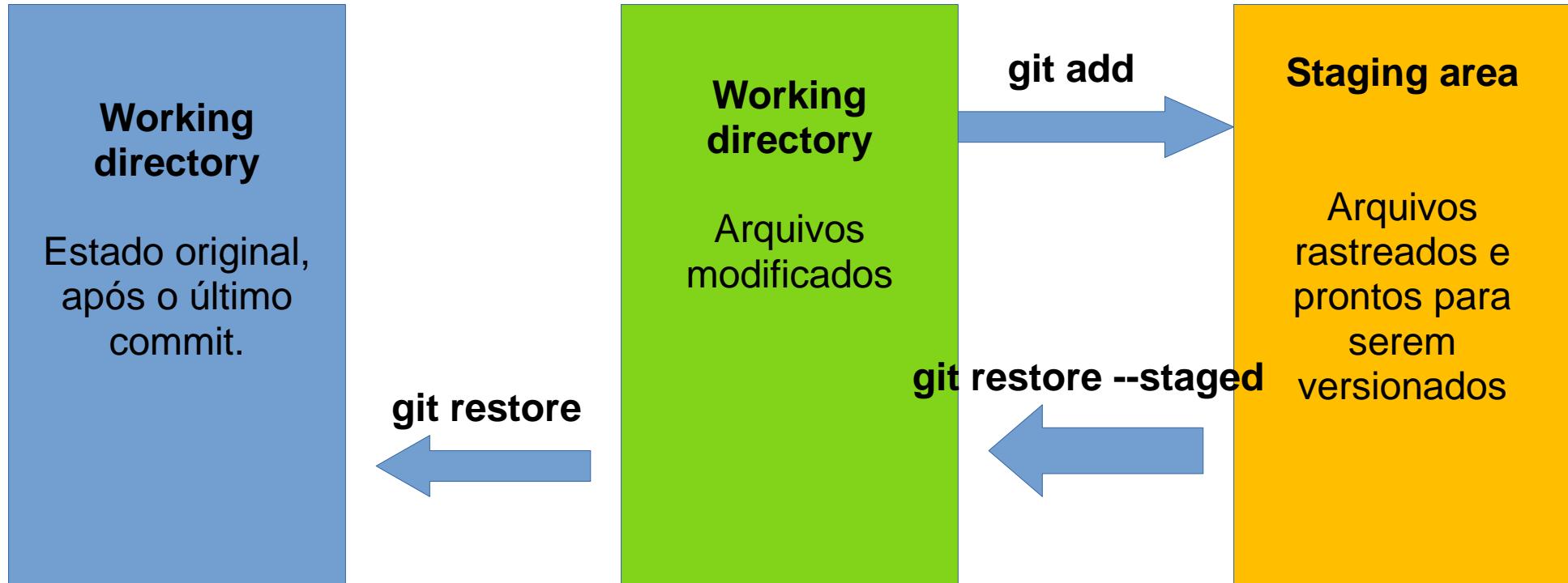
```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git add .

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   trabalho.txt
```

## .Restaurando alterações locais antes do commit.

- Quando um arquivo está na staging area, é possível fazer com que ele retorne ao working directory para fazer novas alterações ou descartar as alterações realizadas:
  - **git restore --staged <arquivo>** - um arquivo específico
  - **git restore --staged .** - todos os arquivos.
- Depois que ele estiver de volta ao working directory, caso desejar voltar ao status original do arquivo, ou seja, antes de qualquer modificação, use o comando:
  - **git restore <arquivo>** - um arquivo específico
  - **git restore .** - todos os arquivos.

## .Compreendendo os estado do git.



Dentro do diretório de trabalho os arquivos podem assumir os estados:  
untracked(não rastreado), unmodified (não modificado) e modified(modificado).

## .Desfazendo alterações - antes do commit.

- Para desfazer todas as alterações, ou seja, voltar ao estado inicial do branch, independentemente se os arquivos estão no working directory ou no staging area, use o comando:
  - **git reset --hard**
- Eses comandos servem para arquivos rastreados. Para excluir arquivos não rastreados, use o comando:
  - **git clean -f**

## .Verificando Alterações

- Vamos lá:
  - Agora exclua o arquivo “teste.txt”
  - Abra o arquivo “trabalho.txt” e apague uma linha existente
  - Ainda no arquivo “trabalho.txt”, atualize/altere uma linha existente adicionando um novo conteúdo a ela.
  - Crie um novo arquivo “configuracao.txt” e adicione uma linha qualquer a ele.
  - Agora execute o comando **git status** para verificar as alterações.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
on branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      deleted:  teste.txt
      modified: trabalho.txt

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

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

## .Verificando Alterações

- Existem modificações que não estão no staging area e o arquivo “configuracao.txt” não está rastreado.
- Entretanto, **antes de usar** o comando **git add**, vamos **verificar as diferenças** entre o último **commit** e essas **modificações pendentes**.

.Verificando Alterações (diferenças)

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents
$ git diff
diff --git a/teste.txt b/teste.txt
deleted file mode 100644
index 3f697d5..0000000
--- a/teste.txt
+++ /dev/null
@@ -1,2 +0,0 @@
-nova linha
-nova nova linha
\ No newline at end of file
diff --git a/trabalho.txt b/trabalho.txt
index a184397..c5b4d29 100644
--- a/trabalho.txt
+++ b/trabalho.txt
@@ -1,3 +1,2 @@
    trabalho01
-linha 02
-linha 03
\ No newline at end of file
+linha 03 alterada para teste
\ No newline at end of file
```

-Execute: **git diff**

-Observa-se:

- As modificações no arquivo trabalho.txt aparecem coloridas:
  - O sinal (-) e em cor vermelha indica exclusão de conteúdo;
  - O sinal (+) em cor verde indica a adição de conteúdo;
- Os arquivos excluídos aparecem especificados, porém, os novos arquivos não. Isso porque não há uma versão anterior do “configuracao.txt”, logo não há o que mostrar.
- Atenção ! Use a **tecla q** para sair do **git diff** caso ele exiba mais de uma página.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git diff
```

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$
```

.Após isso, adicione todas as alterações (**git add –all**) e faça o **git diff** novamente;

.Observe que agora o comando **git diff** não nos demonstra nenhuma diferença, isso porque esse comando tem a função de determinar a diferença entre o último **commit** e o diretório de trabalho.

.A partir do momento que o comando **git add –all** foi utilizado as modificações estão na **staging area**;

.Para visualizar a **diferença** entre o **último commit** e o que está na **staging area**, é necessário adicionar o **parâmetro cached** no comando **git diff**:

- **git diff --cached** ou **git diff --staged** (execute os comandos)

.Para efetivar as alterações:

- **git commit -m** “arquivo teste.txt excluído, trabalho.txt alterado e configuracao.txt adicionado”

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git commit -m "arquivo teste.txt excluido, trabalho.txt alterado e configuração.txt adicionado"
[main a02ae20] arquivo teste.txt excluido, trabalho.txt alterado e configuração.txt adicionado
 3 files changed, 2 insertions(+), 4 deletions(-)
 create mode 100644 configuração.txt
 delete mode 100644 teste.txt
```

## .Voltando ao **git log**:

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git log
commit a02ae20d93f71a359d7022d3526f1a5dc4e50596 (HEAD -> main)
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 27 07:47:06 2023 -0300

    arquivo teste.txt excluido, trabalho.txt alterado e config
uracao.txt adicionado

commit d5d9083baa72f615ccebc1b4b3af3885b289e2de
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 17:28:20 2023 -0300

    alteracao no trabalho.txt

commit 00a2877c3be4164ed2077419fd29f73e27d6f5ef
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 17:04:23 2023 -0300

    arquivo trabalho.txt criado

commit 57a20a94233a4205686292e0f628b4e2b7bf3067
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 16:57:29 2023 -0300

    nova linha inserida no arquivo

commit def28945404855bd9b133147d77900e80643a9fa
Author: jeffersonarpasserini <jefferson.passerini@gmail.com>
Date:   Mon Mar 20 16:20:19 2023 -0300
```

- Execute o comando git log para visualizar o histórico de alterações;
- Exibe o histórico das alterações ( da mais recente para a mais antiga), mostrando cada um dos commits e seus respectivos identificadores;
- Os identificadores são determinados por um hash usando o algoritmo SHA-1 (Secure Hash Algorithm);
- O HEAD é o ponteiro que determina a última modificação em um branch.
- Para cada commit, além do hash temos:
  - O autor (nome, e-mail), data e hora e a mensagem do commit.
- Se na exibição do log houver mais de uma página utilize a tecla q para sair.

. Voltando ao **git log** - outros parâmetros úteis:

- **git log -n [n]** – substitui **[n]** por um número inteiro para limitar o número de commits a ser exibido.
- **git log -p** – exibe as alterações de cada commit, como se tivéssemos data um diff em cada exibição de commit.

- **git log --oneline** – mostra os commits de maneira resumida, em apenas uma linha;
- **git log --graph** – mostra o log junto com um conjunto de caracteres que evidenciam os branchs e merges, se houver.
- **git log --stat** – mostra apenas a quantidade de modificações nos arquivos;

Para mais opções, consulte a documentação oficial: <https://git-scm.com/docs/git-log>

## .Navegando por *commits*

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/cursoGi
t (main)
$ git log --oneline
a02ae20 (HEAD -> main) arquivo teste.txt excluido
, trabalho.txt alterado e configuracao.txt adicio
nado
d5d9083 alteracao no trabalho.txt
00a2877 arquivo trabalho.txt criado
57a20a9 nova linha inserida no arquivo
def2894 novas alterações no arquivo
```

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/cursoGi
t (main)
$ |
```

- Sim, é possível navegar pelas alterações, para isso basta usar o comando checkout da seguinte maneira:
- Primeiro use o comando **git log --oneline** para mostrar os *commits*;
- Observe que os respectivos *hashes* aparecem resumidos.

## .Navegando por *commits*

- Escolha um dos hashes que apareceram no comando anterior e digite:
- Use o comando:
  - **git checkout [hash]**
  - onde vamos substituir **[hash]** por uma sequência hash real de um commit;

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git log --oneline
a02ae20 (HEAD -> main) arquivo teste.txt excluido, t
rabalho.txt alterado e configuracao.txt adicionado
d5d9083 alteracao no trabalho.txt
00a2877 arquivo trabalho.txt criado
57a20a9 nova linha inserida no arquivo
def2894 novas alterações no arquivo

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git checkout def2894
```



```
$ git checkout def2894  
Note: switching to 'def2894'.
```

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>
```

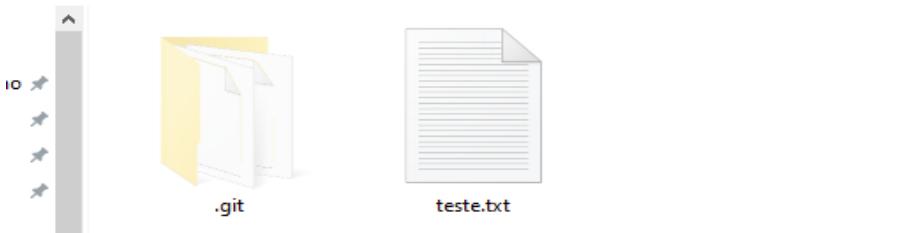
or undo this operation with:

```
git switch -
```

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

HEAD is now at def2894 novas alterações no arquivo

> Este Computador > Documentos > CursoGit



## .Navegando por **commits**

- A partir desse momento estamos no modo “**detached HEAD**”;
- Ou seja, não estamos mais na última alteração.
- Navegue na pasta do repositório e verifique que somente os arquivos corretos daquele commit estão disponíveis.
- Para voltar ao main (**HEAD**):
  - **git checkout main**

## .Desfazendo **commits**

- Para desfazer o último commit, é necessário usar o comando reset;
- Basicamente podemos usar o reset “soft”, ou seja, apenas retornar o HEAD ao status staged, após o ultimo commit anterior.
  - **git reset HEAD~1**
- Nesse caso, podemos fazer novas alterações e commitar novamente, ou simplesmente desfazer as alterações usando o comando:
  - **git reset --hard**

## .Desfazendo **commits**

- Para desfazer de verdade o último commit, ou seja, excluir as alterações, use:
  - **git reset --hard HEAD~1**
- Note que o número 1 após o **HEAD** é o número de commits que serão desfeitos.
- Se colocar 3, por exemplo, os últimos 3 commits serão desfeitos.
- O parâmetro **--hard** indica que tudo será desfeito, ou seja, o commit voltará ao estado inicial antes do último commit.
- Se for usado o parâmetro **--soft**, o commit será desfeito, porém, todas as modificações serão colocadas no **working directory**, ou seja, antes de serem adicionados a **staging area**.

## .Desfazendo **commits**

- Para desfazer um commit que não é o último, usamos o comando revert. Para isso é necessário identificar o hash resumido do commit que desejamos desfazer e executar o comando:
  - **git revert [hash]**
- **Atenção!** O revert não exclui um commit, ele cria um “commit reverso”, ou seja, se o commit desfeito adicionava 2 linhas de código em um arquivo, o revert irá adicionar um commit que exclui essas duas linhas de código.
- A vantagem disso é a preservação do histórico de alterações:
  - Note que o comando revert normalmente abre um editor de texto para que você edite e salve a mensagem do commit reverso. Se abrir no próprio bash, use:
    - Tecla **insert** para modificar a mensagem;
    - **Alt+Q** para sair do modo edição
    - Digite **:wq** para confirmar a mensagem e sair do editor usando (w = write, q = quit)

## .Desfazendo **commits**

- Também é possível desfazer vários commits usando o comando:
  - **git revert HEAD~3**
- No exemplo acima, é criado um commit que desfaz os últimos 4 commits, contando a partir do zero.

## .Desfazendo **commits**

- Outras duas formas de usar o revert são:
  - **git revert -n master~5..master~2**
- No exemplo acima, serão revertidos os commits a partir do 5º até o segundo. Note que a sintaxe do comando deve iniciar do mais recente para o mais antigo.
- **git revert -n f33dba..f3378a**
- No exemplo acima, passou como intervalo os hashes dos commits, do mais recente para o mais antigo.

## .Ignorando arquivos: `.gitignore`

- Existem determinados arquivos que não precisam ser versionados em um projeto, ou seja, eles devem ser ignorados pelo git.
- Há diversos cenários em que isso é usado:
  - Arquivos de senhas;
  - Arquivos compilados;
  - Pastas *vendor* de *frameworks*;
  - Arquivos locais do projeto, etc.
- O objetivo é manter o repositório mais enxuto, apenas com os arquivos essenciais do projeto.
- Para isso, o git usa um arquivo denominado com a extensão `.gitignore` em que são listados os arquivos e pastas que devem ser ignorados.

## .Ignorando arquivos: **.gitignore** -

### Testando

- Crie um arquivo na pasta do curso denominado: “***ignore.me***”
- Cria uma pasta com o nome ***figuras*** e coloque um arquivo qualquer dentro desta pasta.
- Agora crie um arquivo chamado ***.gitignore*** usando o comando **touch .gitignore**.
- Execute um git status para perceber que o git identificou que há arquivos não rastreados;

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ ls
configuracao.txt trabalho.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ touch .gitignore

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    figuras/
    ignore.me

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

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ |
```

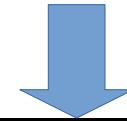
## .Ignorando arquivos: .gitignore - Testando

- Agora abra o arquivo **.gitignore** usando o bloco de notas, e adicione as linhas:
  - ignore.me
  - figuras/\*.\*
- Execute o **git status** novamente e veja o que mudou.
- Os arquivos listados no **.gitignore** não aparecem
- Obviamente o arquivo **.gitignore** deve ser versionado.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    figuras/
    ignore.me

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



```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

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

## .Atividade:

- Siga as instruções (passo-a-passo) e vá documentando um arquivo do word com os comandos executados e quando houver saída de tela (git status, log, etc) printar a tela para documentar.
- Entregar o arquivo word documentação com a execução da atividade no teams.

## .Parte 1 – Criando repositório local

- 1 - Usando o explorer, crie uma pasta chamada atv-git
- 2 - Dentro da pasta, clique com o botão direito do mouse e selecione “Git Bash Here...”
- 3 - Inicialize o repositório com o comando: git init
- 4 - Configure o e-mail e o usuário usando os comandos:
  - a. git config --local user.name “seu nome”
  - b. git config --local user.email “seu email”
- 5 - Adicione 2 arquivos nesta pasta: index.htm e config.ini
- 6 - Adicione algum código ou conteúdo aleatório nos arquivos criados
- 7 - Use o comando git status para perceber que há arquivos não rastreados que precisam ser adicionados

## .Parte 1 – Continuação

- 8 - Para adicionar esses arquivos ao repositório, use o comando `git add .` (ou `git add -all`)
- 9 - Use o comando `git status` novamente para verificar que há mudanças a serem comitadas – o status atual do repositório é “Staged” ou “Staging”.
- 10 - Agora vamos fazer o primeiro commit. Use o comando `git commit -m "First Commit!"` – lembre-se que o parâmetro `-m` indica a mensagem do commit.
- 11 - Agora que o primeiro commit foi realizado, use o comando `git status` para ver que o status atual do diretório (working directory) está “clean”, ou seja, o estado atual é igual ao último commit;

## .Parte 2 – Realizando mais um commit

- 12 - Vamos fazer mais um commit! Abra o arquivo index.html e adicione linhas a ele.
- 13 - Use o comando git status para verificar que há alterações;
- 14 - Use o comando git diff para verificar as diferenças entre o que você está fazendo e o último commit.
- 15 - Adicione essas alterações ao staging, preparando para o commit. Para isso, use o comando git add . (ou git add --all)
- 16 - Use o comando git status para verificar que os arquivos estão prontos para serem “comitados”
- 17 - Faça o segundo commit. Use o comando git commit -m “Second Commit” – se desejar mudar a mensagem do commit, fique à vontade.
- 18 - Verifique os commits realizados usando o comando git log. Se desejar ver os commits de maneira resumida, use o parâmetro --oneline, assim: git log --oneline

## .Parte 3 – Desfazendo coisas (antes do commit)

- 19 - Agora é importante que você abra o código fonte em um editor, para verificar as alterações;
- 20 - Faça uma alteração em um dos arquivos, depois use o git status para verificar que há alterações a serem realizadas.
- 21 - Agora vamos imaginar que você quer desfazer essas alterações, ou seja, quer voltar ao estado original após o último commit. Como as alterações ainda não foram adicionadas ao staging, podemos usar o comando git restore . para desfazer todas as alterações. Faça isso e veja no editor que o arquivo voltou ao estado original.
- 22 - Agora faça mais algumas alterações em um ou mais arquivos. Depois use o comando git status para verificar que eles foram modificados.
- 23 - Agora adicione esses arquivos ao staging, ou seja, prepare-os para o commit. Use o comando git add .

## .Parte 3 – Desfazendo coisas (antes do commit)

- 24 - Agora, vamos imaginar que você quer desfazer as alterações. Nesse caso o comando git restore . Não resolve, pois os arquivos estão no staging. Faça o teste, use git restore . e veja que as alterações continuam lá!
- 25 - Para desfazer as alterações, considerando que há arquivos no staging, é possível usar o comando git restore --staged. Isso faz com que as modificações voltem ao working directory, mas ainda não foram desfeitas. Para desfazê-las, use git restore . novamente.
- 26 - Note que, para desfazer as alterações (antes do commit), os arquivos estando no staging ou no working directory, podemos usar o comando git reset --hard .

## .Parte 4 – Desfazendo commit's

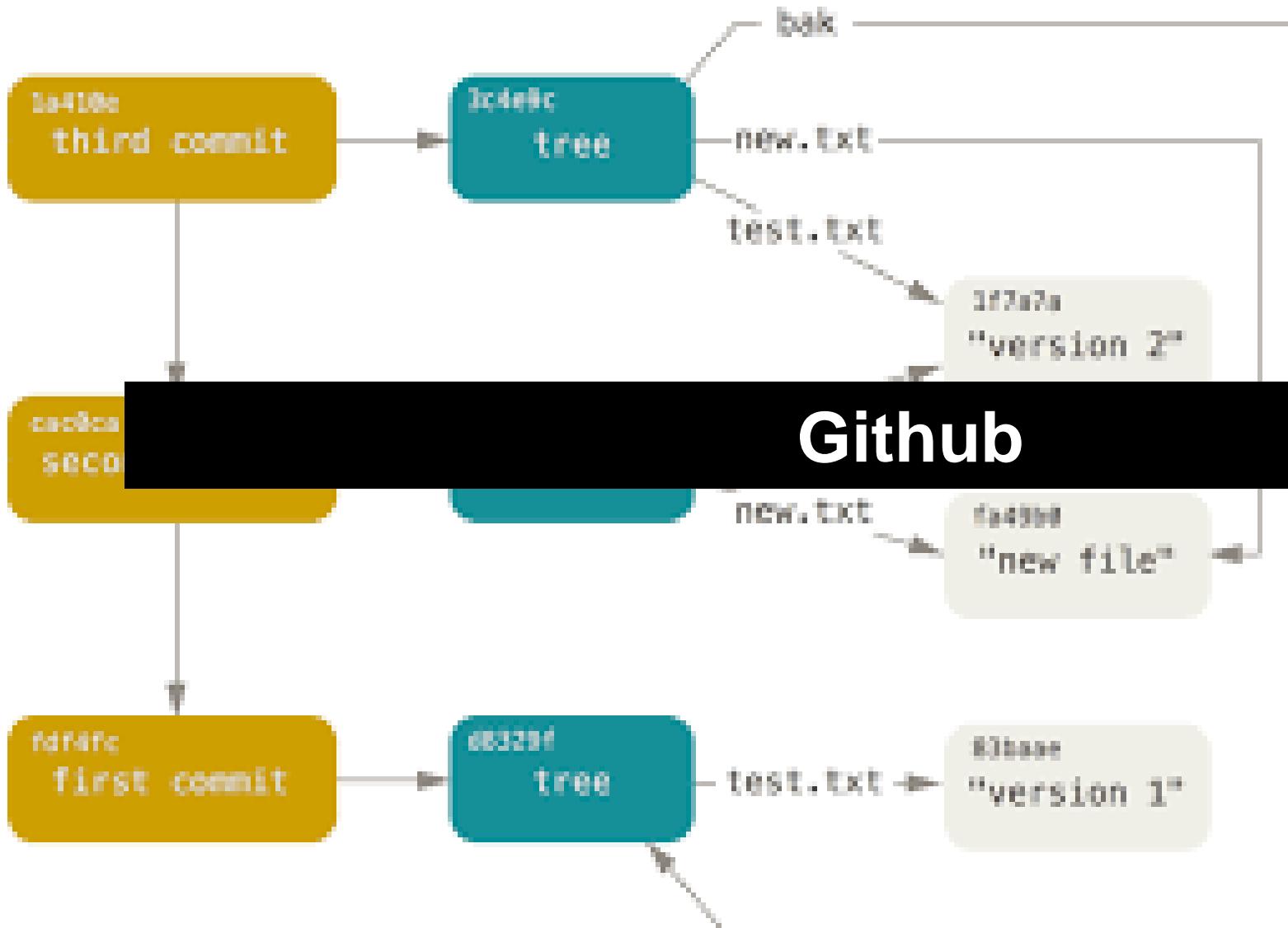
- Há dois modos de desfazer um commit: o modo soft e o modo hard. no modo soft, o registro do commit é desfeito, porém, suas alterações são preservadas. Isso significa que o repositório volta ao status modified. Assim, podemos fazer novas alterações ou simplesmente adicionar as alterações ao staging e comitar novamente.
- 27 - Para defazer o último comit no modo soft, use o comando git reset HEAD~1 – note que ~1 é o número de commits que serão desfeitos. É recomendável desfazer de um em um.
- 28 - Depois, use o git status para verificar que o repositório se encontra em estado modificado. Use também o git log --oneline para verificar que o registro do último commit foi excluído.

## .Parte 4 – Desfazendo commit's

- 29 - Agora, faça algumas alterações adicionais, use o git add . e faça mais um commit.
- 30 - Use o comando git log --oneline para visualizar os commits
- 31 - Agora, vamos desfazer esse último commit no modo hard. Use o comando git reset --hard HEAD~1.
- 32 - Use o commando git log --oneline e depois git status para verificar que não há mais commit nem alterações.
- 33 - Agora vamos o usar o revert, que consiste em criar um commit desfazendo um anterior. Isso é muito útil para preservar o histórico de commits.

## .Parte 4 – Desfazendo commit's

- 34 - Faça algumas alterações no arquivo, adicione ao staging e faça mais um commit.
- 35 - Agora, vamos reverter esse último commit.
  - Use o comando git log –oneline e verifique o hash do commit mais recente. Agora use o comando git revert [hash].
  - Isso irá fazer com que abra um editor (VIM editor) onde poderemos editar a mensagem de commit.
  - Para isso basta usar as setas do teclado e a tecla Insert para alterar o modo de edição da mensagem – Insert ou Replace.
  - Após alterar a mensagem, pressione as teclas **ALT + SHIFT + :** - note que no canto esquerdo da tela irá mostrar os dois pontos (".").
  - Agora basta digitar as telas **wq** (write + quit) e pressionar Enter. Pronto, foi criado um commit que desfaz as alterações do último.
- 36 - Se preferir não mexer na mensagem, quando o editor for aberto, basta teclar **SHIFT + q** (quit) e pressionar Enter.

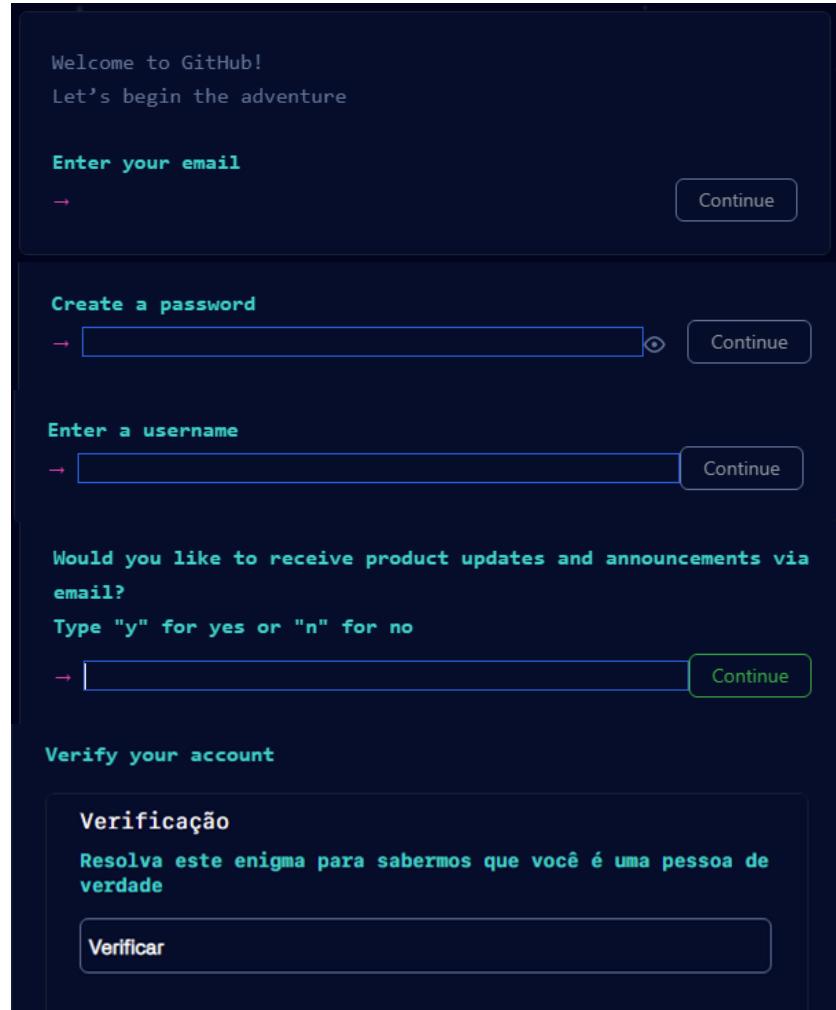


## .O que é GitHub?

- É um repositório online de código-fonte que usa Git como sistema de controle de versão;
- Mais que um repositório, o GitHub pode ser considerado uma rede social de desenvolvedores, permitindo o compartilhamento de projetos e o desenvolvimento colaborativo;
- Foi criado em 2007 em Ruby, por quatro programadores: Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon;
- Iniciou as atividades em 2008, sendo comprada pela Microsoft em 2018 por US\$7,5 bilhões.

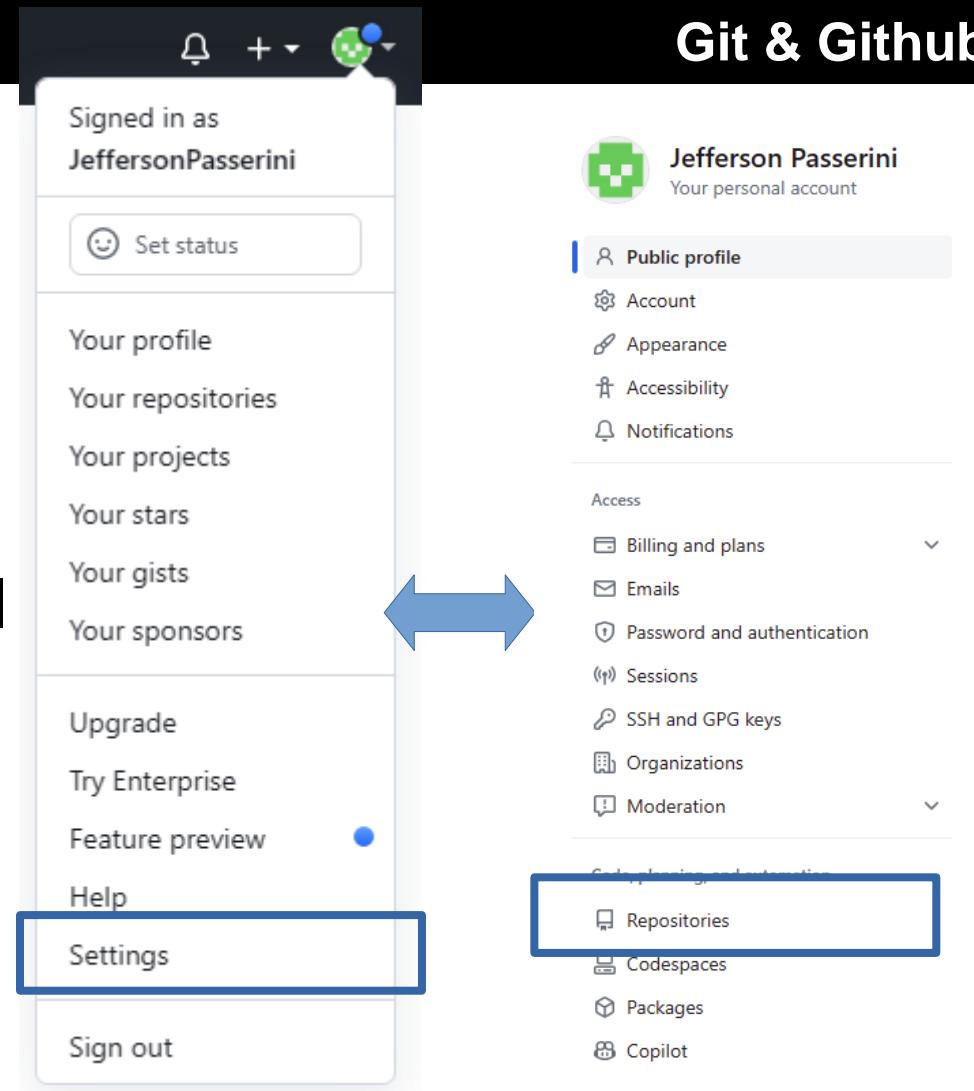
## .Criando uma conta

- Acesse o site: <https://github.com/>
- Clique no link Sign up no canto superior direito
- Siga as instruções
- Confirme a conta
- E faça login.



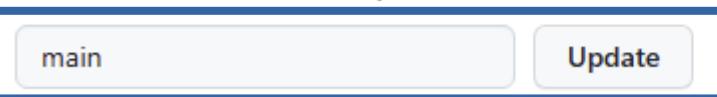
## .Verifique o repositório padrão

- No menu no canto superior direito de sua tela, escolha a opção: ***settings***;
- A seguir na tela de configurações no menu lateral a direita escolha a opção: ***Repositories***;



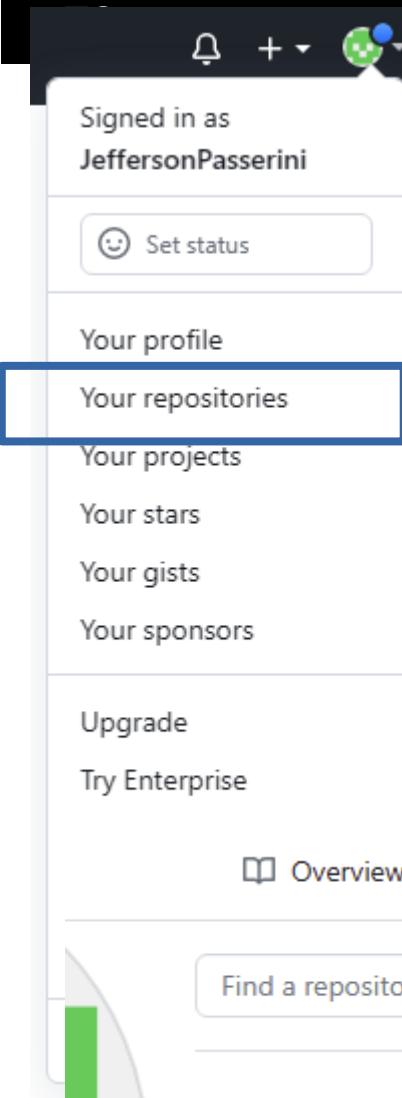
## Repository default branch

Choose the default branch for your new personal repositories. You might want to change the default name due to different workflows, or because your integrations still require "master" as the default branch name. You can always change the default branch name on individual repositories. [Learn more about default branches.](#)



### .Verifique o repositório padrão

- Verifique se o seu repositório padrão é o ***main***;
- Lembre-se que para trocar de master para main no repositório local utilizamos o comando:  
***. git branch -m master main***



## .Criando seu primeiro repositório no GitHub

- Acesse a opção “**Your repositories**”;
- No botão “New” crie seu novo repositório;

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner \* Repository name \*

 JeffersonPasserini /

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

Description (optional)

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

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

Initialize this repository with:

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

Add a README file  
This is where you can write a long description for your project. [Learn more](#).

Add .gitignore  
Choose which files not to track from a list of templates. [Learn more](#).

.gitignore template:

Choose a license  
A license tells others what they can and can't do with your code. [Learn more](#).

 You are creating a public repository in your personal account.

# Criando repositório no GitHub

- Informe o nome de seu repositório;
- Uma descrição e deixei público para todos vejam seus trabalhos;
- O resto dos parâmetros podemos deixar que depois podem ser configurados;

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

 Set up in Desktop

or

HTTPS

SSH

git@github.com:JeffersonPasserini/Curso-git.git



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

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

```
echo "# Curso-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:JeffersonPasserini/Curso-git.git
git push -u origin main
```

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

```
git remote add origin git@github.com:JeffersonPasserini/Curso-git.git
git branch -M main
git push -u origin main
```

...or import code from another repository

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

 Import code

## ***.Criando Repositório no GitHub***

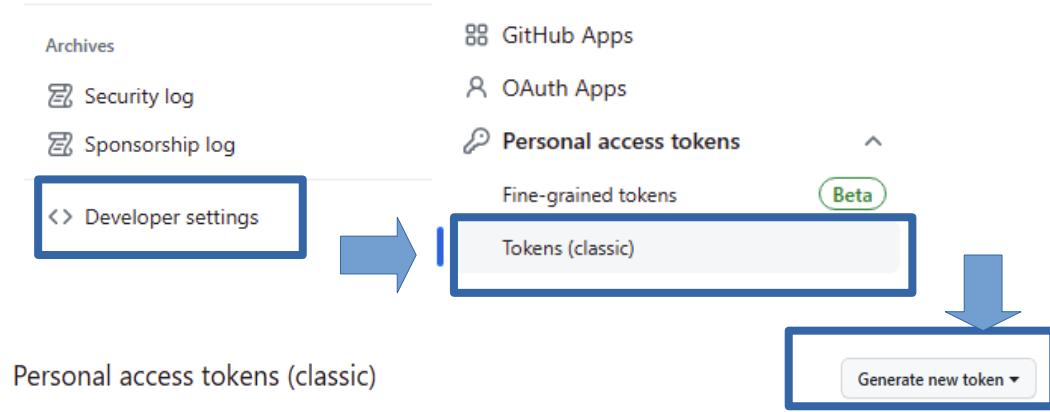
- Após a criação do repositório será exibida essa página com instruções de utilização.

## .Vinculando o Git local com sua conta no GitHub

- Normalmente, por meio das configurações globais, é possível definir nome de usuário e e-mail;
  - git config --global user.name “Nome do Usuário”
  - git config --global user.email “email@email.com”
- Se desejar realizar apenas configurações locais, ou seja, específica para cada repositório, troque o parametro:
  - **--global por –local;**
- Até 31/08/2021 o GitHub suportava acesso via HTTS através de usuário e senha, mas por questões de segurança desativou essa opção;
- Agora é necessário gerar um token via *Personal Access Token* (PAT);
- Documentação: <https://docs.github.com/pt/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>.

## Personal Access Tokens

- Os tokens de acesso pessoal são uma alternativa para o uso de senhas quando vamos fazer autenticação via API do GitHub ou pela linha de comando;
- É importante gerar um Token, caso o Git peça um quando formos atualizar o repositório remoto.
- Para gerar um novo Token:
  - Clique no menu com seu avatar e depois em Settings;
  - Selecione a opção Developer Settings
  - Depois clique em Personal access tokens
  - Agora clique em Generate new token
  - Preencha o campo Note e marque os escopos dele. Qdo é pessoal, marque todos os scopes.
  - Clique em Generate Token
  - Depois de gerado, Copie-o e salve em algum arquivo.



### Personal access tokens (classic)

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

## Personal Access Tokens

- Os tokens de acesso pessoal são uma alternativa para o uso de senhas quando vamos fazer autenticação via API do GitHub ou pela linha de comando;
- É importante gerar um Token, caso o Git peça um quando formos atualizar o repositório remoto.
- Para gerar um novo Token:
  - . Clique no menu com seu avatar e depois em Settings;
  - . Selecione a opção Developer Settings
  - . Depois clique em Personal access tokens
  - . Agora clique em Generate new token
  - . Preencha o campo Note e marque os escopos dele. Qdo é pessoal, marque todos os scopes.
  - . Clique em Generate Token
  - . Depois de gerado, Copie-o e salve em algum arquivo.

### New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

What's this token for?

Expiration \*

30 days The token will expire on Tue, May 2 2023

Select scopes

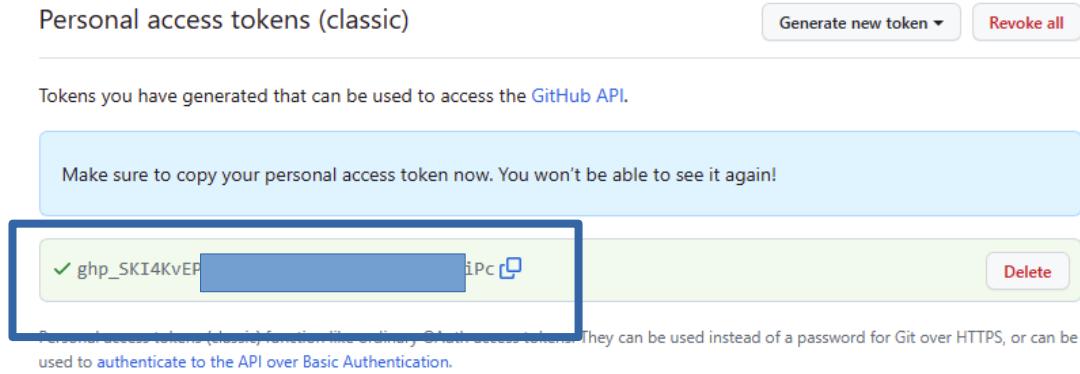
Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

**Generate token** Cancel

## Personal Access Tokens

- Os tokens de acesso pessoal são uma alternativa para o uso de senhas quando vamos fazer autenticação via API do GitHub ou pela linha de comando;
- É importante gerar um Token, caso o Git peça um quando formos atualizar o repositório remoto.
- Para gerar um novo Token:
  - . Clique no menu com seu avatar e depois em
  - . Settings;
  - . Selecione a opção Developer Settings
  - . Depois clique em Personal access tokens
  - . Agora clique em Generate new token
  - . Preencha o campo Note e marque os escopos
  - . dele. Qdo é pessoal, marque todos os scopes.
  - . Clique em Generate Token
  - . Depois de gerado, Copie-o e salve em algum
  - . arquivo.



## *.Resetar as configurações “remote/origin” do seu repositório local*

- Para garantir que esse tutorial funcione, vamos zerar as configurações de “remote/origin” do seu repositório local, isto é, caso você já esteja utilizando um. Para isso, vamos fazer o seguinte:
  - Abra a pasta .git que fica oculta em um de seus repositórios locais;
  - A seguir, abra o arquivo config e exclua
  - as configurações [remote “origin”] – vide imagem a seguir.

Este Computador > Documentos > CursoGit > .git

config - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

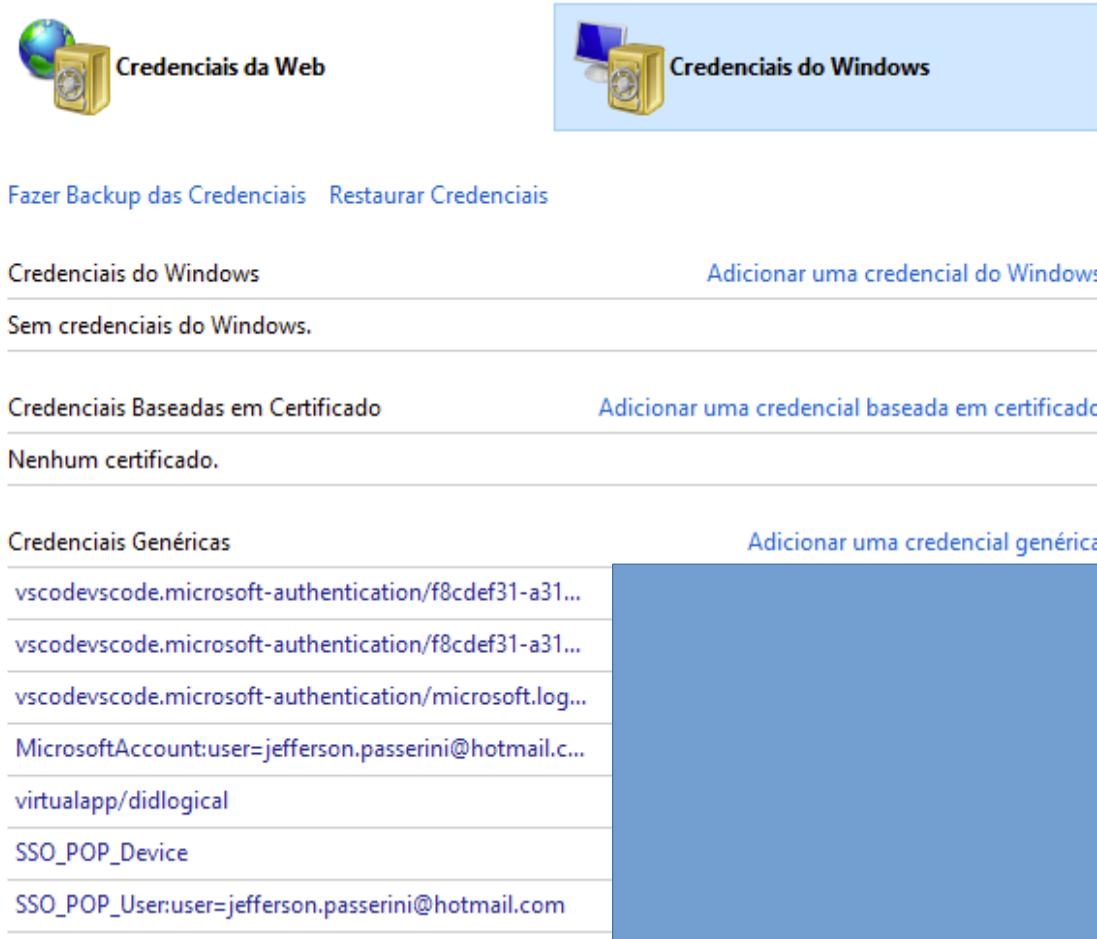
```
[core]
repositoryformatversion = 0
filemode = false
bare = false
logallrefupdates = true
symlinks = false
ignorecase = true

[gui]
wmstate = normal
geometry = 893x435+78+78 175 196
```

1 [core]
2 repositoryformatversion = 0
3 filemode = false
4 bare = false
5 logallrefupdates = true
6 symlinks = false
7 ignorecase = true
8 [branch "main"]
9 remote = origin
10 merge = refs/heads/main
11
12 [remote "origin"]
13 url =
14 fetch = +

## *. Excluir as credenciais do GitHub localmente*

- Para garantir as configurações estejam de fato zeradas, abra o Gerenciador de Credenciais do Windows e exclua todas as credenciais relacionadas Git que estejam cadastradas. Veja a imagem a seguir.



## *.Configurar seu repositório local*

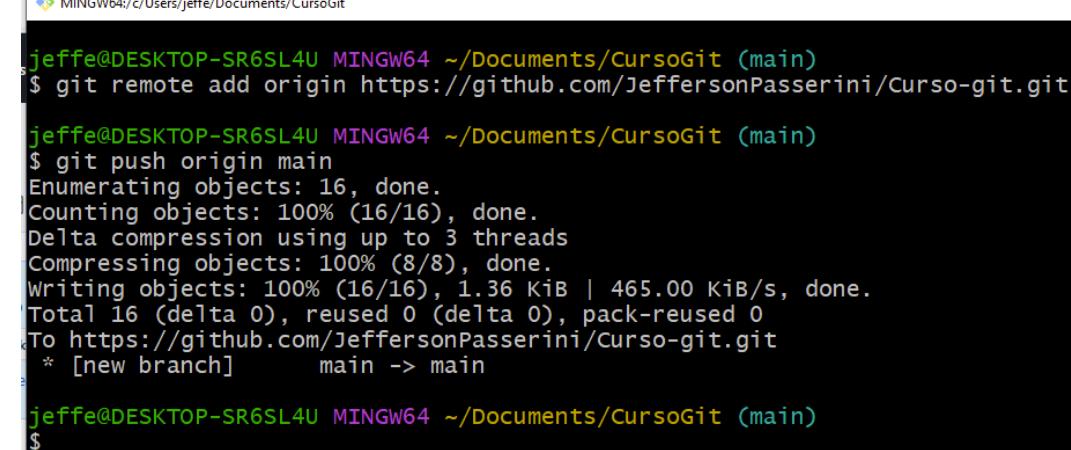
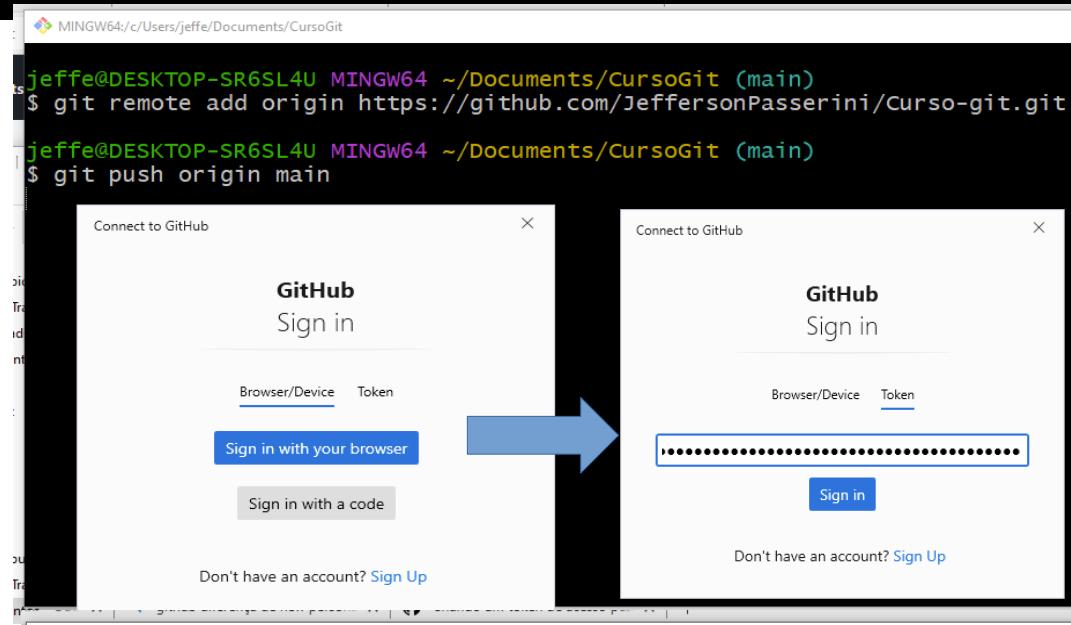
- Agora precisamos setar as configurações **remote/origin** de um repositório local já existente. Para isso, precisamos **confirmar a URL HTTPS** do seu repositório, acessar a pasta do repositório via **Git Bash** (ou outro terminal) e digitar o seguinte código:
  - **git remote add origin [URL-HTTPS-DO-REPOSITÓRIO]**
- Caso você tenha excluído o repositório local, será necessário realizar o git clone para baixá-lo novamente:
  - **git clone [URL-HTTPS-DO-REPOSITÓRIO]**
- **Observação:** Para pegar o [URL-HTTPS-DO-REPOSITÓRIO] vá ao repositório e copie o link HTTPS ali fornecido.

## .Configurar seu repositório local

- Agora para testar vamos executar nosso primeiro push.

- git push origin main

- Será solicitado seu acesso ao GitHub e escolha a opção token e forneça o PAT que foi gerado anteriormente.



## *.E se não funcionar?*

- Caso o comando git push ou git pull retorne uma mensagem de erro, informando que não foi possível autenticar com usuário, senha ou token, confirme os passos 1 e 2, e tente novamente.
- Se o erro persistir é possível informar o PAT diretamente nas configurações do repositório. Para isso, faça o seguinte:
  - Abra a pasta oculta .git na pasta do seu repositório;
  - Na configuração [remote “origin”], edite a URL do repositório da seguinte maneira:
    - Url = `https://$(git_token)@github.com/user_name/repo_name.git`

## *.E se não funcionar?*

- Alternativamente, você pode executar o push/pull informando diretamente a URL como parâmetro, veja:
  - `git push https://$(git_token)@github.com/user_name/repo_name.git`
- Obviamente em ambos os casos, você deve substituir a porção `git_token` pelo seu PAT.
- Agora tente fazer o push/pull, deve funcionar!

## .Autenticação com chave SSH

- Acesse o Git Bash e digite o seguinte comando:
  - ssh-keygen -t ed25519 -C **seu-email@exemplo.com**
- Após a execução desse comando, será pedido um nome para o arquivo em que a chave será salva. Digite o nome e pressione <Enter>;

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CURSOGIT2 (main)
$ ssh-keygen -t ed25519 -c "jefferson.passerini@hotmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/jeffe/.ssh/id_ed25519): curso-ssh-ke
y
```

- Depois será pedido uma “passphrase” (senha). Se preferir pode deixar em branco. Pressione <Enter>.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CURSOGIT2 (main)
$ ssh-keygen -t ed25519 -c "jefferson.passerini@hotmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/jeffe/.ssh/id_ed25519): curso-ssh-ke
y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

## *.Autenticação com chave SSH*

- Após confirmar será exibida a seguinte mensagem:

```
Your public key has been saved in curso-ssh-key.pub
The key fingerprint is:
SHA256:jhv+LPHx+pdMZS9L96NiqljHwMZgJMd+v7CYQbwU3fA jefferson.passerini@hotmail.com
The key's randomart image is:
---[ED25519 256]---
| ..o o.o
| +. + ...
| .oo . E
| .o+o o
| . .=. S o .
| ...o+o . o o
| + ==o+ o o +
| o o+=o .o+ ...
| .o+=++.....
+---[SHA256]---
```

- Note que foi criado um arquivo (curso-ssh-key) e um **.pub** na pasta escolhida. Ao abrir o arquivo **.pub** é possível ver a chave ssh.

## **.Adicionar a chave ao ssh-agent**

- Para iniciar o ssh-agent em background digite:
  - eval "\$(ssh-agent -s)"
- Para adicionar sua chave ao ssh-agent, digite:
  - ssh-add [my-ssh-key]
  - Note que [my-ssh-key] é o nome do arquivo gerado.
- A chave foi adicionada localmente. Agora é necessário adicioná-la no GitHub.

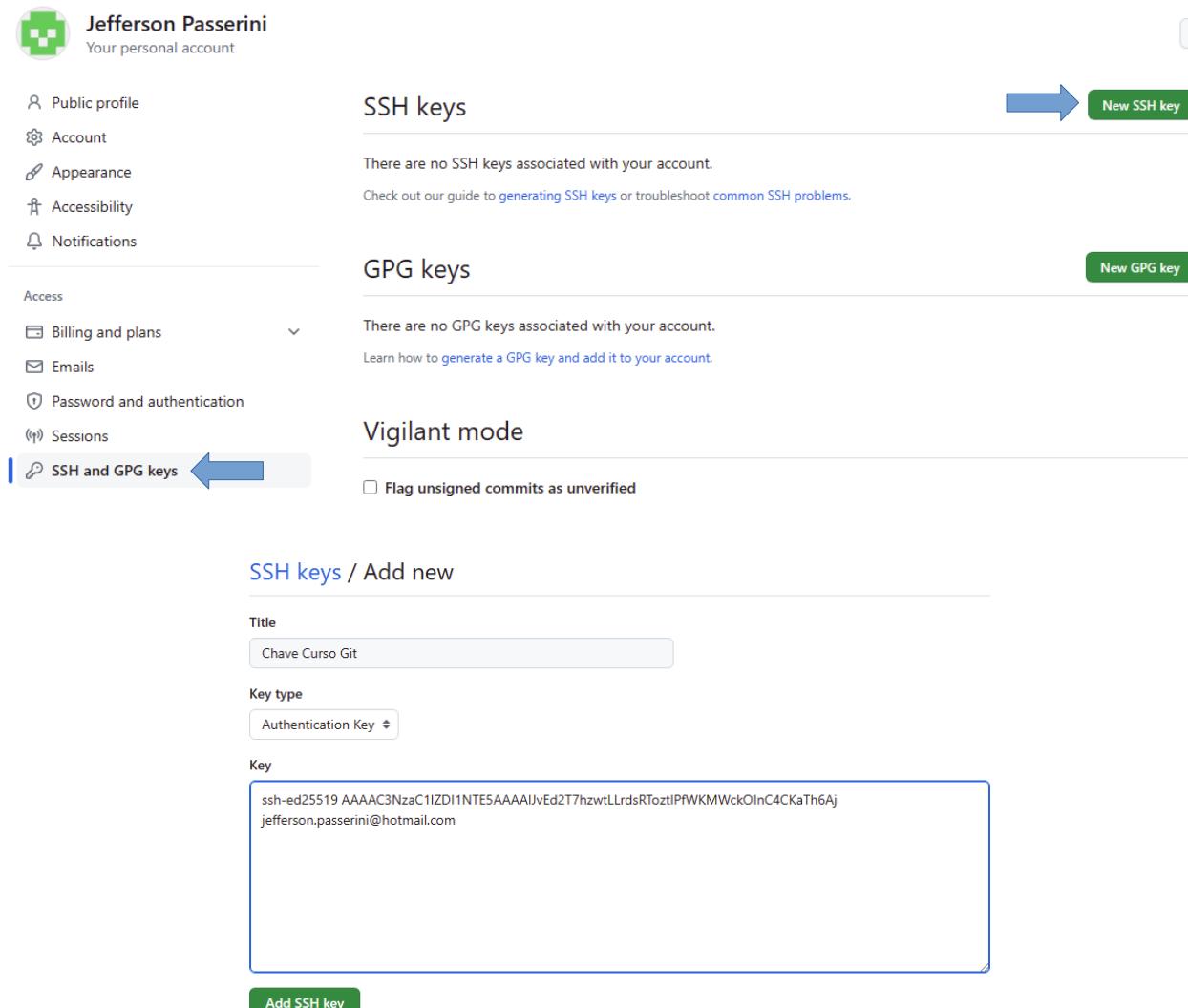
```
r
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents
$ eval "$(ssh-agent -s)"
Agent pid 1088

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents
$ ssh-add curso-ssh-key
Enter passphrase for curso-ssh-key:
Identity added: curso-ssh-key (jefferson.passerini@hotmail.com)

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents
$ |
```

## **.Adicionar chave SSH ao GitHub**

- Abra o arquivo de chave (extensão .pub) criado com chave e copie-a (CTRL+C);
- Abra o GitHub no navegador;
- Vá até configurações (settings);
- Clique no menu esquerdo SSH and GPG Keys;
- Em SSH keys, clique em New SSH Key;
- Dê um título a sua chave e cole o conteúdo copiado no item 1;



The screenshot shows the GitHub settings interface for the user 'Jefferson Passerini'. The left sidebar includes links for Public profile, Account, Appearance, Accessibility, Notifications, Access (Billing and plans, Emails, Password and authentication, Sessions), and SSH and GPG keys (which is highlighted with a blue arrow). The main content area is titled 'SSH keys' and displays a message stating there are no SSH keys associated with the account. It includes a link to a guide on generating SSH keys and troubleshooting common SSH problems. A green button labeled 'New SSH key' is visible. Below the SSH keys section is a 'GPG keys' section with a similar message and a 'New GPG key' button. Further down is a 'Vigilant mode' section with a checkbox for 'Flag unsigned commits as unverified'. At the bottom is a 'SSH keys / Add new' form with fields for 'Title' (set to 'Chave Curso Git'), 'Key type' (set to 'Authentication Key'), and a large 'Key' text area containing a public SSH key and the user's email ('jefferson.passerini@hotmail.com'). A blue box highlights the 'Add SSH key' button at the bottom of the form.

## **.Adicionar chave SSH ao GitHub**

- Pronto! Agora o seu Git Local está devidamente conectado ao GitHub. Assim é possível gerenciar repositórios remotos sem informar login e senha, visto que a conexão é feita via SSH.
- Para saber mais, consulte a documentação:
  - . *<https://docs.github.com/pt/enterprise-server@3.1/github/authenticating-to-github/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>*

## .O que fazer com os arquivos de chaves?

- Nesse exemplo, geramos os arquivos que contém as chaves na própria pasta do repositório;
- Logicamente, vamos mover esses arquivos de lugar;
- Crie um pasta em um local seguro do seu computador e mova os dois arquivos gerados;
- Caso *inicie outro repositório*, essa chave poderá ser reaproveitada. Para isso basta executar os comandos de SSH a partir do:
  - eval "\$(ssh-agent -s)"
  - Veja os slides anteriores.
- Como a chave já está adicionada ao GitHub, não é necessário adicionar novamente.

## **.Entendendo os comandos**

- ***git remote add origin [link-http-do-repositório-github]***
  - *Vincula o repositório local ao remoto.*
- ***git push origin main***
  - *Envia o conteúdo do repositório local para o servidor (repositório no GitHub)*
  - *Este comando se for a primeira vez no repositório deve solicitar sua autenticação (PAT).*
- *Assim temos nosso repositório local vinculado ao remoto, e nossas informações salvas.*

# .Navegando pelo repositório remoto

Clique na aba **code** para verificar os arquivos do projeto.



JeffersonPasserini / Curso-git (Public)

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

jeffersonarpasserini arquivo teste.txt excluido, trabalho.txt alterado e configuracao... a02ae20 last week 5 commits

configuracao.txt arquivo teste.txt excluido, trabalho.txt alterado e configuracao.txt ... last week

trabalho.txt arquivo teste.txt excluido, trabalho.txt alterado e configuracao.txt ... last week

Add a README

Clique no menu **commits** para verificar o histórico de alterações;



Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main

Commits on Mar 27, 2023

arquivo teste.txt excluido, trabalho.txt alterado e configuracao.txt ... a02ae20 jeffersonarpasserini committed last week

Commits on Mar 20, 2023

alteracao no trabalho.txt d5d9083 jeffersonarpasserini committed 2 weeks ago

arquivo trabalho.txt criado 00a2877 jeffersonarpasserini committed 2 weeks ago

nova linha inserida no arquivo 57a20a9 jeffersonarpasserini committed 2 weeks ago

novas alteracoes no arquivo def2894 jeffersonarpasserini committed 2 weeks ago

Newer Older

# .Navegando pelo repositório remoto

Clique em um dos **commits** para verificar os detalhes das alterações

The screenshot shows a GitHub commit history for a repository. The commit details are as follows:

**arquivo teste.txt excluido, trabalho.txt alterado e configuracao.txt ...**  
... adicionado  
main  
jeffersonarpasserini committed last week  
1 parent d5d9083 commit a02ae20

Showing 3 changed files with 2 additions and 4 deletions.

**Filter changed files**

**configuracao.txt**

- 1 @@ -0,0 +1 @@  
1 + teste de configuracao

**teste.txt**

**trabalho.txt**

Load diff  
This file was deleted.

0 comments on commit a02ae20

Lock conversation

## .Fazendo alterações online usando a interface GitHub

Vamos adicionar um arquivo README em nosso repositório.

Para isso clique no botão “add a README”

The screenshot shows a GitHub repository page for 'JeffersonPasserini / Curso-git'. The repository is public and has 1 branch and 0 tags. Recent commits by 'jeffersonarpasserini' show changes to 'configuracao.txt' and 'trabalho.txt'. At the bottom, there is a message encouraging the user to add a README, with a green 'Add a README' button highlighted by a blue border.

Help people interested in this repository understand your project by adding a README.

Add a README

## .Fazendo alterações online usando a interface GitHub

O arquivo README utiliza a linguagem de marcação chamada Markdown, faça a alteração conforme abaixo.

The screenshot shows the GitHub interface for the repository 'JeffersonPasserini / Curso-git'. The 'Code' tab is selected. In the main area, the 'README.md' file is open, showing the following content:

```
1 # Curso-git
2
3 ## Projeto desenvolvido em sala de aula
4
5 Curso de Git e Github
6
```

A blue rectangular box highlights the entire code block. At the top right of the editor, there are buttons for 'Cancel changes', 'Spaces', '2', and 'No wrap'. Above the editor, there are buttons for 'Pin', 'Unwatch 1', 'Fork 0', and 'Star 0'.

## .Fazendo alterações online usando a interface GitHub

Na mesma tela faça o commit das alterações efetuadas.

O GitHub já define uma mensagem padrão, mas você pode alterá-la

The screenshot shows a GitHub repository page for 'JeffersonPasserini / Curso-git'. The 'Code' tab is selected. In the code editor, a new file 'README.md' is being created with the following content:

```
1 # Curso-git
2
3 ## Projeto desenvolvido em sala de aula
4
5 Curso de Git e Github
6
```

A modal window titled 'Commit new file' is open, prompting the user to 'Create README.md'. A blue arrow points to this input field. Below it is a text area for an optional extended description. At the bottom of the modal, there are two radio button options: one selected for 'Commit directly to the main branch.' and another for 'Create a new branch for this commit and start a pull request.' The 'Commit new file' button is highlighted with a green background and white text.

Clique em  
Commit new file

## ***.Fazendo alterações online usando a interface GitHub***

O arquivo README.md foi criado em seu repositório no Github

 JeffersonPasserini	Create README.md	c02f344 now	6 commits
 README.md	Create README.md	now	
 configuracao.txt	arquivo teste.txt excluido, trabalho.txt alterado e configuracao.txt ...	last week	
 trabalho.txt	arquivo teste.txt excluido, trabalho.txt alterado e configuracao.txt ...	last week	

## ***E o repositório local? Verifique.***

## **.Baixando as atualizações para o repositório local**

.Se você verificou, pode observar que o arquivo README.md não existe no repositório local.

.Execute o comando git status, e verifique que não há nada para “*commitar*”.

.Para baixar as atualizações do repositório do GitHub, execute o comando

**git pull origin main**

.Então: **Push** envia para o servidor e **Pull** baixa do servidor.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git status
On branch main
nothing to commit, working tree clean

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git pull origin main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 796 bytes | 132.00 KiB/s, done.
From https://github.com/JeffersonPasserini/CursoGit2
 * branch            main      -> FETCH_HEAD
   45b79d5..8977ac8  main      -> origin/main
Updating 45b79d5..8977ac8
Fast-forward
 README.md | 5 +////
 1 file changed, 5 insertions(+)
 create mode 100644 README.md

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ |
```

## *.Agora vamos estudar o git clone.*

**.Para testarmos o git clone, apague a pasta de seu repositório local.**

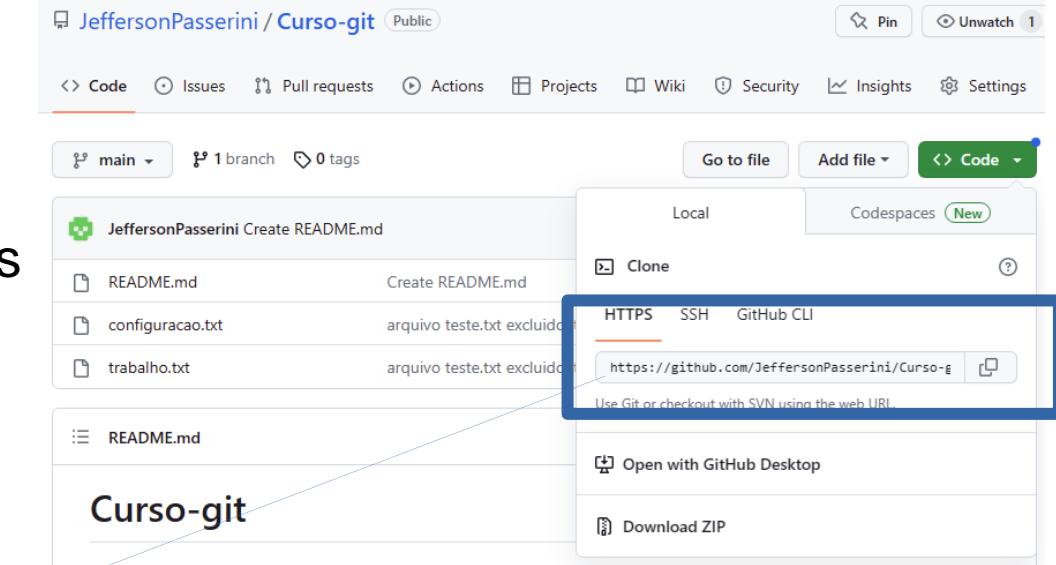
**.Fique tranquilo seus arquivos estão salvos no repositório do GitHub.**

**.Agora abra o git bash no local onde você deseja baixar o seu projeto novamente.**

**.E digite o comando:**

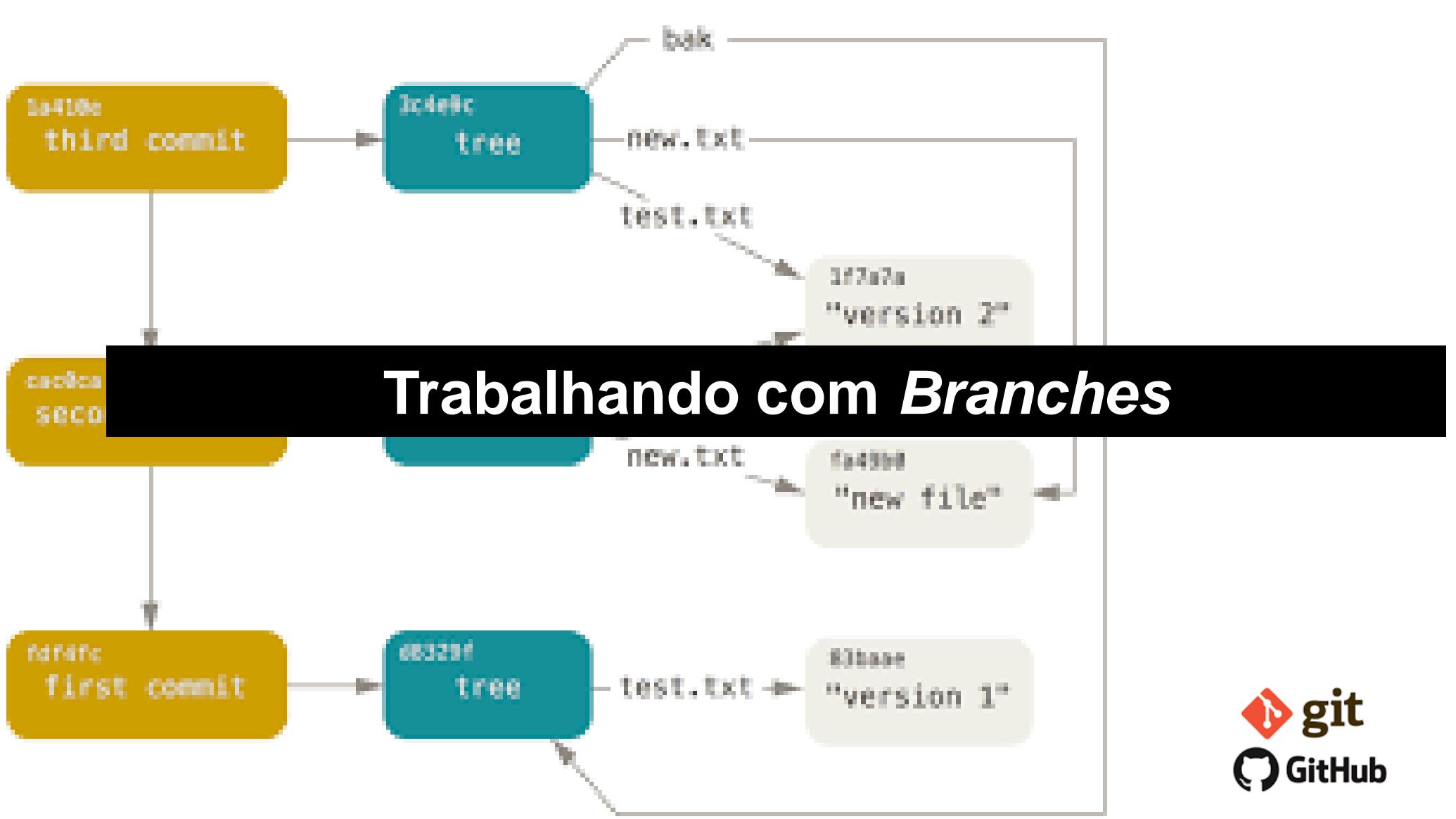
***.git clone [link-repositorio-github]***

***.Agora você tem novamente seu projeto em sua máquina no repositório local.***



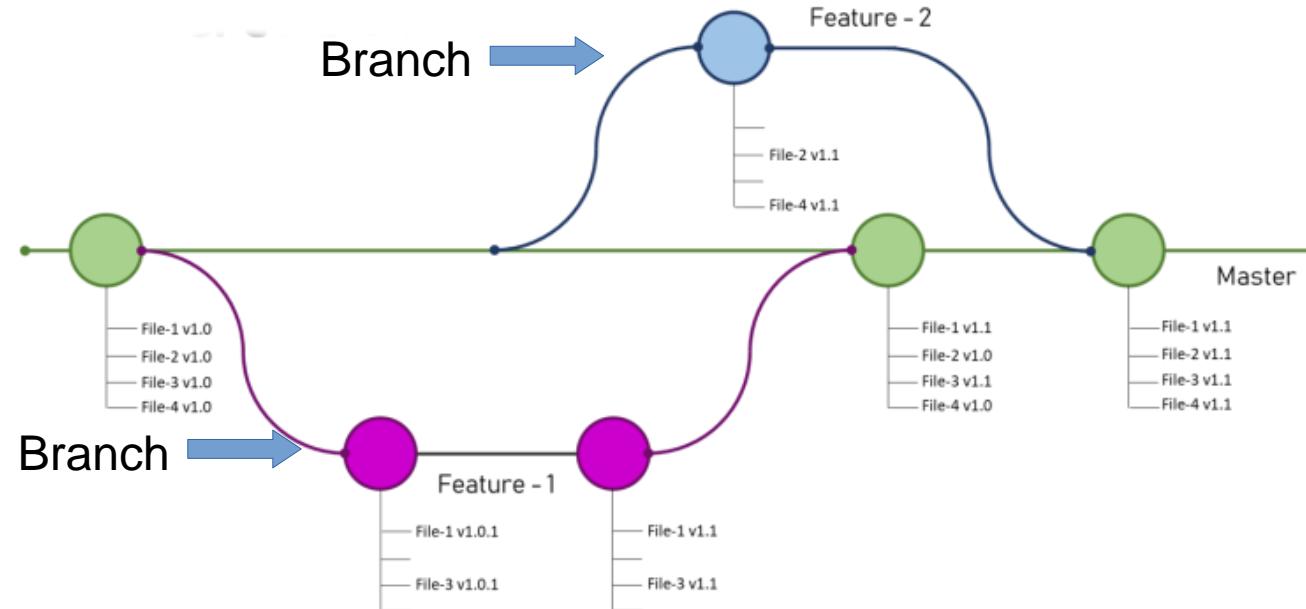
```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents
$ git clone https://github.com/JeffersonPasserini/Curso-git.git
Cloning into 'Curso-git'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 19 (delta 0), reused 16 (delta 0), pack-reused 0
Receiving objects: 100% (19/19), done.

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents
$ |
```



## .Trabalhando com branches

- Uma **branch** é uma ramificação em um projeto;
- É usada para desenvolver novas funcionalidades em um projeto sem impactar versões estáveis;
- Uma vez que a **branch** é desenvolvida, testada e aprovada, é possível fazer o **merge**, ou seja, mesclar as alterações.



## **.Listar e criar branches**

- Para listar as branches do repositório atual, use o comando:
  - . **git branch**
- Para criar uma nova branch, use o comando **git branch [nome\_branch]**, exemplo:
  - . **git branch alterando\_projeto**
- Use o comando **git branch** para listar as branches. Observe que há um asterisco na branch ativa, nesse caso **main**.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git branch
  alterando_projeto
* main
```

## .Listar e criar branches

- Para mudar para a branch criada, use o comando:
  - **git checkout nome\_branch** ou
  - **git switch nome\_branch**
- Ao listar as branches novamente, o asterisco que indica a branch atual está na branch selecionada. A linha do prompt também muda, indicando a branch ativa.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git checkout alterando_projeto
Switched to branch 'alterando_projeto'

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (alterando_pro
jeto)
$ git branch
* alterando_projeto
  main
```

## **.Enviar branches para o repositório remoto**

- Uma vez na branch, você pode fazer todas as modificações necessárias;
- Para executar o push (enviar para o servidor) utilize o comando:
  - **git push origin alterando\_projeto**
- Como no servidor do Github não existe a branch “alterando\_projeto” é necessário criá-la, neste modo o próprio git dá o comando.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/cursoGit2 (alterando_projeto)
$ git push origin alterando_projeto
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 3 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 337 bytes | 337.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'alterando_projeto' on GitHub by visiting:
remote:     https://github.com/JeffersonPasserini/cursoGit2/pull/new/alterando_projeto
remote:
To https://github.com/JeffersonPasserini/cursoGit2.git
 * [new branch]      alterando_projeto -> alterando_projeto
```

## • Visualizando a branch no GitHub

- Após o push, o repositório recebeu a atualização da branch, assim, é possível visualizá-la. Para isso clique no link “branches”

The screenshot shows a GitHub repository interface. At the top, there are navigation buttons: 'main' (with a dropdown arrow), '2 branches' (with a dropdown arrow), '0 tags', 'Go to file', 'Add file', and a green 'Code' button with a dropdown arrow. A modal window titled 'Switch branches/tags' is open, showing a search bar with 'ted' typed in, a status message 'deletion, or require status checks before merging. [Learn more](#)', and a 'Protect this branch' button. Below the search bar, there are tabs for 'Branches' (selected) and 'Tags'. Under 'Branches', there is a list of branches: 'main' (marked with a checkmark and labeled 'default'), 'alterando\_projeto', and a 'View all branches' link. To the right of the branches, commit history is shown for each. For 'main', the commits are: 'commit inicial do repo' (2 weeks ago), 'Create README.md' (2 weeks ago), 'configuracao.txt' (2 weeks ago), and 'trabalho.txt' (2 weeks ago). Below the commit history, there is a 'README.md' file editor. The content of the file is:

```
CursoGit2
Projeto desenvolvido em sala de aula
Curso de Git e Github
```

# • Visualizando a branch no GitHub

## – Comparando as branches

The screenshot shows a comparison between two GitHub repositories. On the left, the repository 'main' is shown, which has 2 branches and 0 tags. It contains several files: '.gitignore', 'README.md', 'configuracao.txt', and 'trabalho.txt', all of which are 'commit inicial do repo'. A file named 'README.md' is expanded, showing the content 'CursoGit2'.

On the right, the repository 'alterando\_proj...' is shown, also with 2 branches and 0 tags. It contains the same files as the main repository, plus a new file 'Branch\_alterando\_novo\_arquivo.txt'. This new file is highlighted with a blue border. The commit history for this branch shows:

- '.gitignore' - commit inicial do repo (2 weeks ago)
- 'Branch\_alterando\_novo\_arquivo.txt' - novo arquivo (9 minutes ago) [highlighted]
- 'README.md' - Create README.md (2 weeks ago)
- 'configuracao.txt' - commit inicial do repo (2 weeks ago)
- 'trabalho.txt' - commit inicial do repo (2 weeks ago)

The commit for 'Branch\_alterando\_novo\_arquivo.txt' is timestamped '9 minutes ago' and has a SHA of '2ba6eef'. The 'Code' button is visible at the top right of the right-hand repository view.

**Left Repository (main):**

- .gitignore: commit inicial do repo
- README.md: Create README.md
- configuracao.txt: commit inicial do repo
- trabalho.txt: commit inicial do repo

**Right Repository (alterando\_proj...):**

- .gitignore: commit inicial do repo
- Branch\_alterando\_novo\_arquivo.txt: novo arquivo
- README.md: Create README.md
- configuracao.txt: commit inicial do repo
- trabalho.txt: commit inicial do repo

**Content of README.md (Left):**

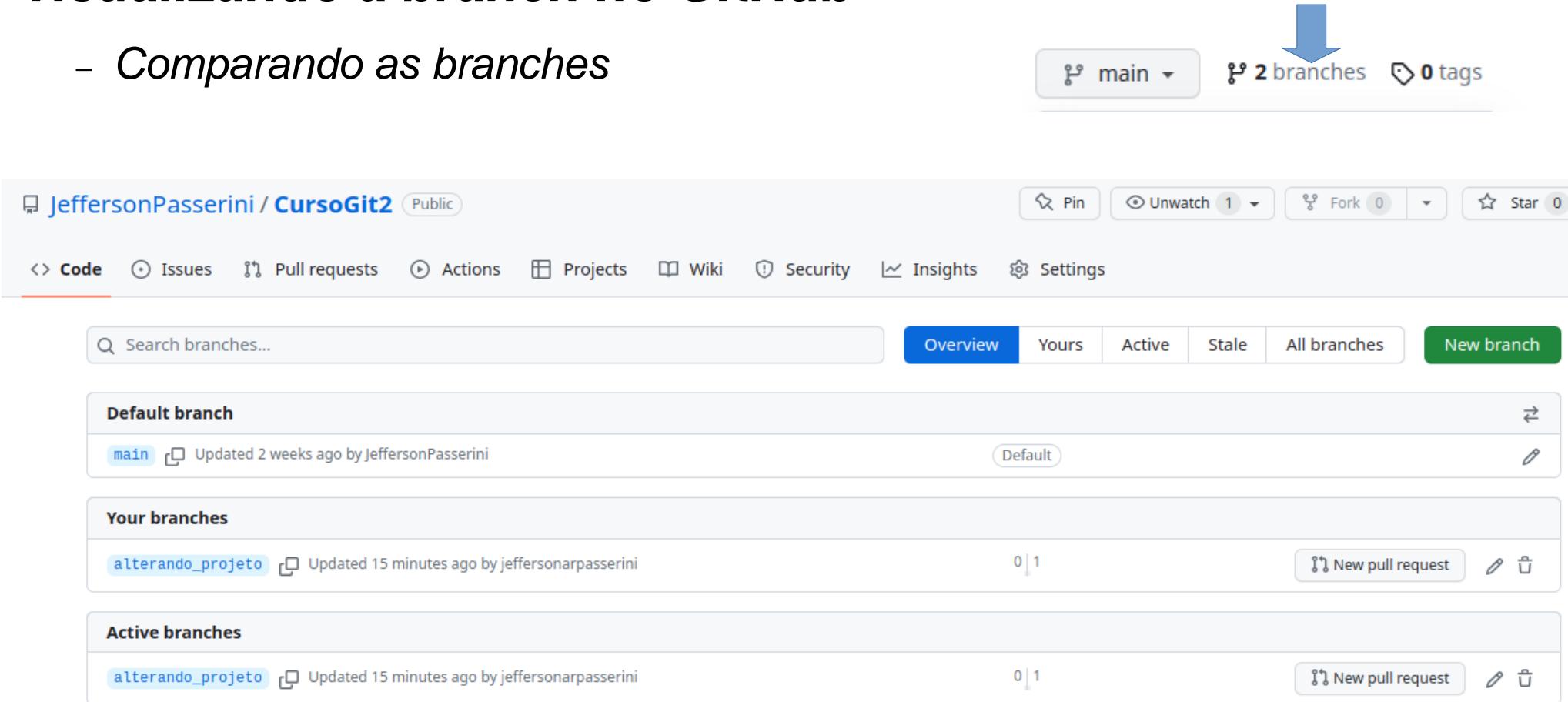
CursoGit2

**Content of README.md (Right):**

CursoGit2

## • Visualizando a branch no GitHub

- Comparando as branches



The screenshot shows a GitHub repository page for 'JeffersonPasserini / CursoGit2'. The top navigation bar includes links for 'main', '2 branches', and '0 tags'. Below the navigation bar, there are buttons for 'Pin', 'Unwatch', 'Fork', and 'Star'. The main content area features a search bar for branches, followed by tabs for 'Overview', 'Yours', 'Active', 'Stale', and 'All branches', with 'Overview' being the active tab. The 'Default branch' section shows the 'main' branch, which was updated 2 weeks ago by JeffersonPasserini. The 'Your branches' section lists the 'alterando\_projeto' branch, which was updated 15 minutes ago by jeffersonarpasserini. The 'Active branches' section also lists the 'alterando\_projeto' branch with the same update information. There are buttons for creating new pull requests and managing branches.

JeffersonPasserini / CursoGit2 Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Search branches...

Overview Yours Active Stale All branches New branch

**Default branch**

main Updated 2 weeks ago by JeffersonPasserini Default

**Your branches**

alterando\_projeto Updated 15 minutes ago by jeffersonarpasserini 0 | 1 New pull request

**Active branches**

alterando\_projeto Updated 15 minutes ago by jeffersonarpasserini 0 | 1 New pull request

## .Remover branches no repositório local

- Para excluir uma branch:
  - Primeiro saia da branch, ou seja, não é possível excluir uma branch ativa;
  - Use o comando *switch* para mudar para outra branch, por exemplo:
    - *git switch main*
  - Agora para excluir efetivamente uma branch local, execute o comando:
    - *git branch -d [nome\_branch]*
  - Ou para remoção forçada, use:
    - *git branch -D [nome\_branch]*

## **.Remover branches no repositório remoto**

- Ao remover uma branch localmente, ela permanece no servidor;
- Se quiser recuperá-la de lá, basta usar o comando:
  - *git switch [nome\_branch]*
- Para remover do repositório remoto, execute:
  - *git push --delete origin [nome\_branch]*
- Quando se exclui do repositório remoto, não é possível recuperar!
- Cuidado! Mantenha isso muito bem alinhado com sua equipe.

## .Mesclar alterações - merge

- O merge é uma operação simples em essência, porém perigosa, pois irá mesclar dois branches;
- É recomendável fazer com bastante cautela, e só atualizar o repositório remoto quando tiver a plena certeza de que tudo ocorreu como o esperado;
- Para fazer um merge em um branch, segue-se a lógica:
  - Branch ativo ← outro branch (outro branch fará parte do branch ativo)
  - Torne ativa a branch main → **git switch main**;
  - Execute o comando: **git merge [branch que será usada]**;
- Atenção! Caso não haja conflitos, ou seja, linha de código e arquivos que se sobreponem, o commit será feito automaticamente. Caso contrário, será necessário resolver os conflitos antes de efetuar o commit.
- Use o comando **git push** para atualizar o repositório remoto.

## .Mesclar alterações - merge

- Executando os comandos

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (alterando_projeto)
$ git switch main
Switched to branch 'main'

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git merge alterando_projeto
Updating 8977ac8..2ba6eef
Fast-forward
  Branch.Alterando_novo_arquivo.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 Branch.Alterando_novo_arquivo.txt

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git status
On branch main
nothing to commit, working tree clean

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/JeffersonPasserini/CursoGit2.git
  8977ac8..2ba6eef  main -> main
```

## .Mesclar alterações - merge

- Você pode verificar que as alterações foram mescladas pois agora o arquivo criado está na branch main.

JeffersonPasserini / CursoGit2 Public

Code Issues Pull requests Actions Projects Wiki Security

main ▾ 2 branches 0 tags

jeffersonarpasserini novo arquivo

.gitignore commit initial do repo

Branch\_alterando\_novo\_arquivo.txt novo arquivo

README.md Create README.md

configuracao.txt commit initial do repo

trabalho.txt commit initial do repo

A branch “alterando\_projeto” continua existindo no repositório local e no remoto, temos que apagá-la.

main ▾ 2 branches 0 tags

Switch branches/tags

Find or create a branch...

Branches Tags

✓ main default

alterando\_projeto

[View all branches](#)

## .Mesclar alterações - merge

- Para apagar siga os comandos do slides 114 e 115
- No repositório local:

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git branch
  alterando_projeto
* main

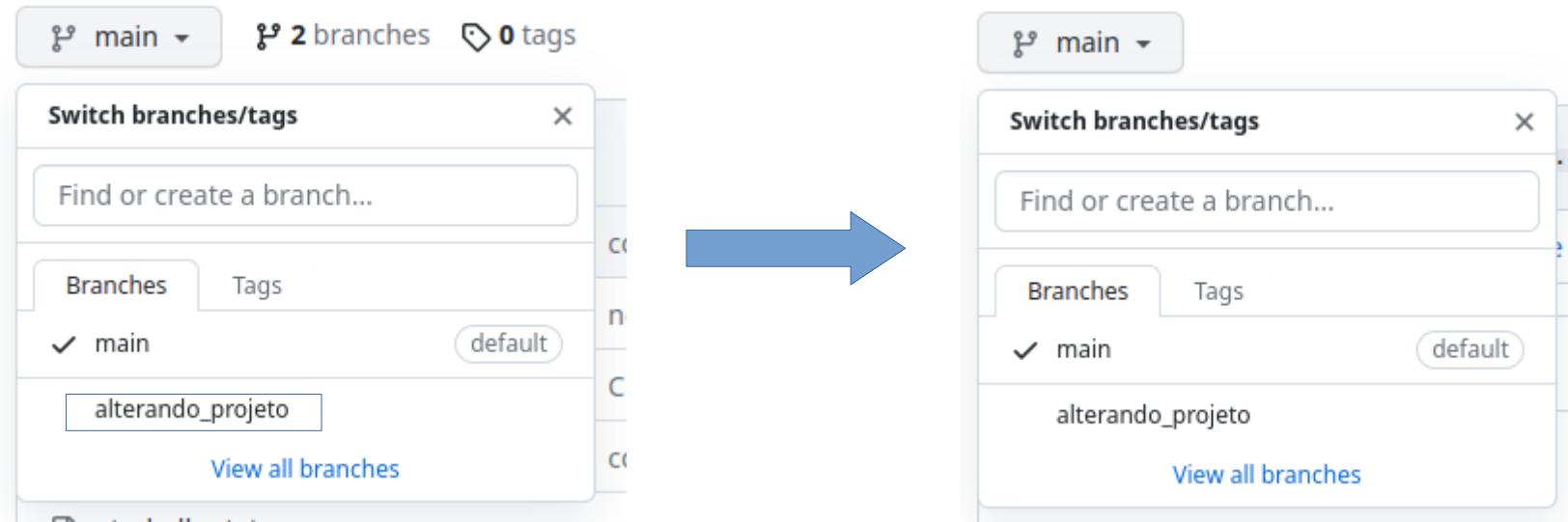
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git branch -d alterando_projeto
Deleted branch alterando_projeto (was 2ba6eef).

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git branch
* main
```

## .Mesclar alterações - merge

- Para apagar siga os comandos do slides 114 e 115
- No repositório remoto:

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git push --delete origin alterando_projeto
To https://github.com/JeffersonPasserini/CursoGit2.git
 - [deleted]          alterando_projeto
```



## **.Mesclar alterações com conflitos**

- *Vamos criar um cenário para execução de nossos testes.*
- *Certifique-se que o repositório local está atualizado com o remoto – faça os commits e pushes necessários.*
- *Usando a interface do GitHub abra o repositório remoto e edite um arquivo em uma ou mais linhas de código possíveis. Salve e dê o commit usando os recursos do GitHub na web;*
- *Volte ao repositório local, faça alterações nas mesmas linhas de código;*
- *Execute o git add, o git commit e o git push.*

## .Mesclar alterações com conflitos

- Temos nosso problema!

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2
(main)
$ git push origin main
To https://github.com/JeffersonPasserini/CursoGit2.git
  ! [rejected]      main -> main (fetch first)
error: failed to push some refs to 'https://github.com/JeffersonPasserini/CursoGit2.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by an
other 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.
```

## .Mesclar alterações com conflitos

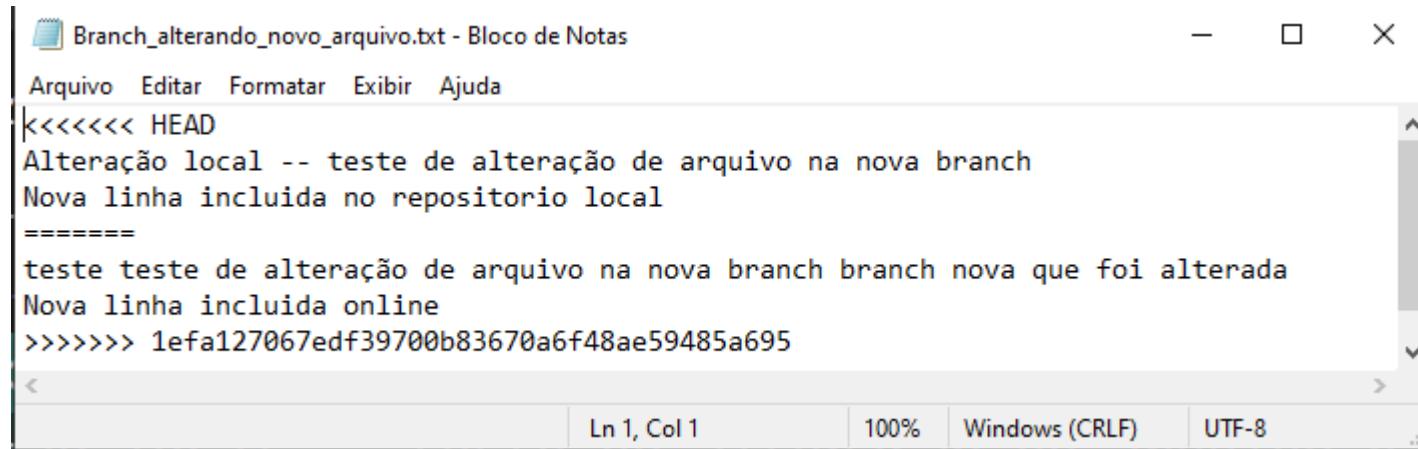
- A alteração é rejeitada por que o **git** detectou que o repositório remoto está diferente do local.
- Ele avisa que há trabalhos no repositório remoto que não há no local, logo temos que fazer um **git pull** para atualizar o repositório.
- Sim, faça o **git pull**.
- Até ai tudo certo, o problema é que pode haver conflitos, ou seja, as mesmas linhas de código podem estar diferentes local e remotamente.
- Nesse caso, o branch *main* entra num estado chamado **MERGING** (veja a figura), ou seja, ele identifica que há conflitos a serem resolvidos.

```
jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main)
$ git pull origin main
From https://github.com/JeffersonPasserini/CursoGit2
 * branch           main      -> FETCH_HEAD
Auto-merging Branch_alterando_novo_arquivo.txt
CONFLICT (content): Merge conflict in Branch_alterando_novo_arquivo.txt
Automatic merge failed; fix conflicts and then commit the result.

jeffe@DESKTOP-SR6SL4U MINGW64 ~/Documents/CursoGit2 (main|MERGING)
$ |
```

## .Mesclar alterações com conflitos

- O que podemos fazer? Você tem que dizer ao **git** qual é a modificação correta, logo, **será necessário editar manualmente o arquivo.**
- Ao abrir o arquivo em um editor, veja que ele mostra o local do conflito;
- Dependendo do editor, se há integração com o GitHub, será mais fácil acertar as alterações, pois ele mostra um pequeno menu que auxilia nas alterações.



A screenshot of a Windows Notepad window titled "Branch\_alterando\_novo\_arquivo.txt - Bloco de Notas". The window shows the following text content:

```
<<<<< HEAD
Alteração local -- teste de alteração de arquivo na nova branch
Nova linha incluida no repositorio local
=====
teste teste de alteração de arquivo na nova branch branch nova que foi alterada
Nova linha incluida online
>>>>> 1efa127067edf39700b83670a6f48ae59485a695
```

The text is divided into three sections by conflict markers: the first section is under the HEAD, the second is a local change, and the third is an online change. The bottom status bar shows "Ln 1, Col 1", "100%", "Windows (CRLF)", and "UTF-8".

Editor sem  
integração

## .Mesclar alterações com conflitos

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Branch\_alterando\_novo\_arquivo.txt - CursoGit2 - Visual Studio Code
- Sidebar (EXPLORER):** CURSOGIT2 folder containing files: figuras, .gitignore, Branch\_alterando... ! (highlighted), configuracao.txt, ignore.me, README.md, trabalho.txt.
- Central Area:** Editor tab for Branch\_alterando\_novo\_arquivo.txt. The content of the file is:

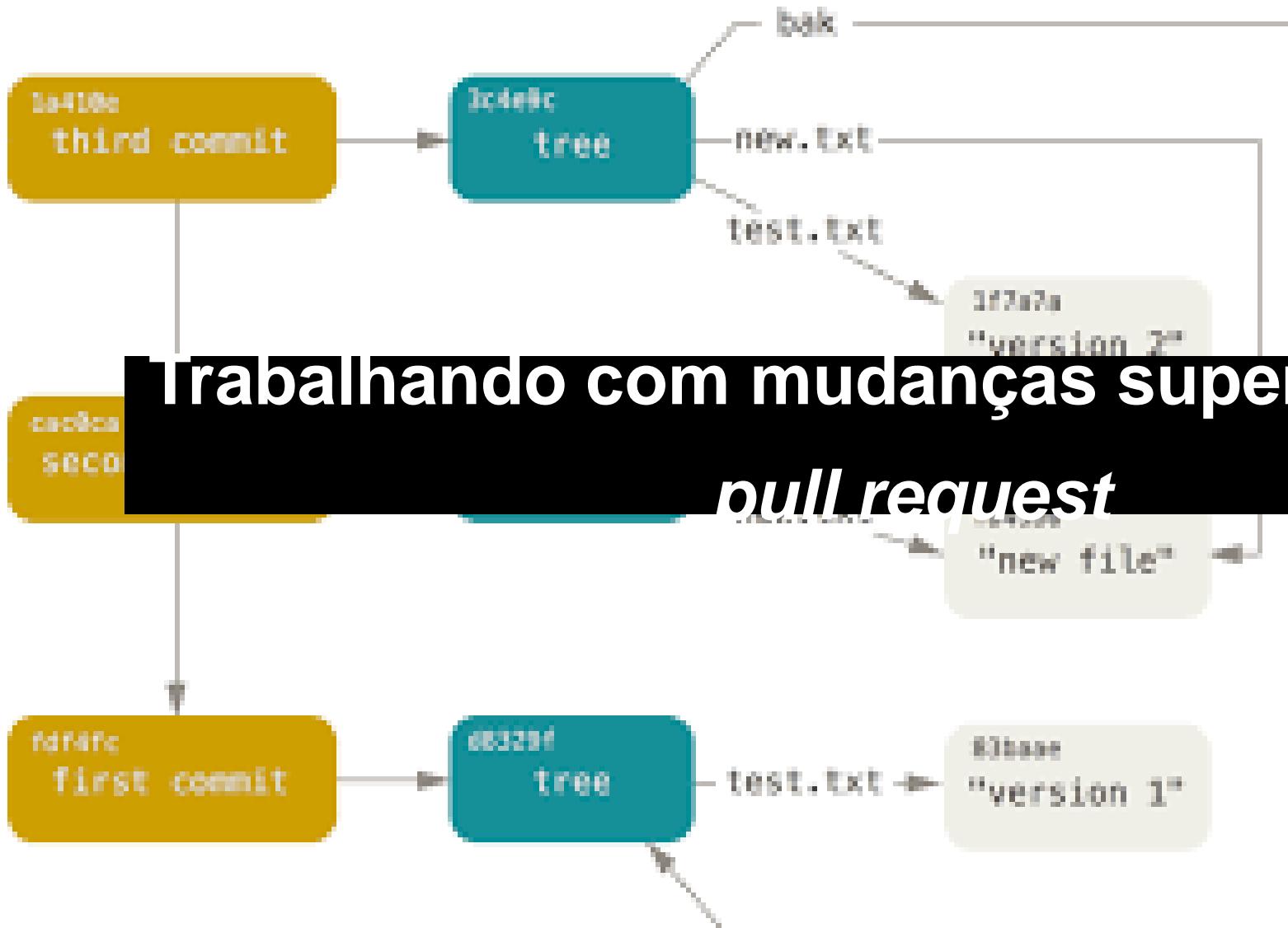
```
1 <<<< HEAD (Alteração Atual)
2 Alteração local -- teste de alteração de arquivo na nova branch
3 Nova linha incluida no repositorio local
4 =====
5 teste teste de alteração de arquivo na nova branch branch nova que foi
6 Nova linha incluida online
7 >>>> 1efa127067edf39700b83670a6f48ae59485a695 (Alteração da Entrada)
```
- Bottom Right:** A blue button labeled "Resolve in Merge Editor".
- Bottom Navigation:** Includes icons for main!, up, down, Git Graph, and status indicators (Ln 1, Col 1, Spaces: 4, CRLF, Plain Text).

## .Mesclar alterações com conflitos

- **Aceitar Alteração Atual (Accept Current Change)**: aceita as modificações locais;
- **Aceitar Alteração de Entrada (Accept Incoming Change)**: aceita as modificações remotas – vindas com o git pull;
- **Aceitar Ambas as Alterações (Accept Both Change)**: necessário edição manual para consolidar as mudanças;
- **Comparar Alterações (Compare Change)**: abre uma janela de comparação entre as duas modificações;
- **Depois de resolver o conflito, faça o git add, git commit e depois o git push para manter o repositório atualizado.**

## .Rebase

- Existe uma alternativa ao merge que é o **rebase**;
- Na prática o rebase é capaz de criar um histórico de branches mais limpo e linear, mas também é extremamente perigoso;
- Esse artigo do blog TreinaWeb explica as diferenças entre merge e rebase, recomendo a leitura.
- <https://www.treinaweb.com.br/blog/git-merge-e-git-rebase-quando-usa-los/>

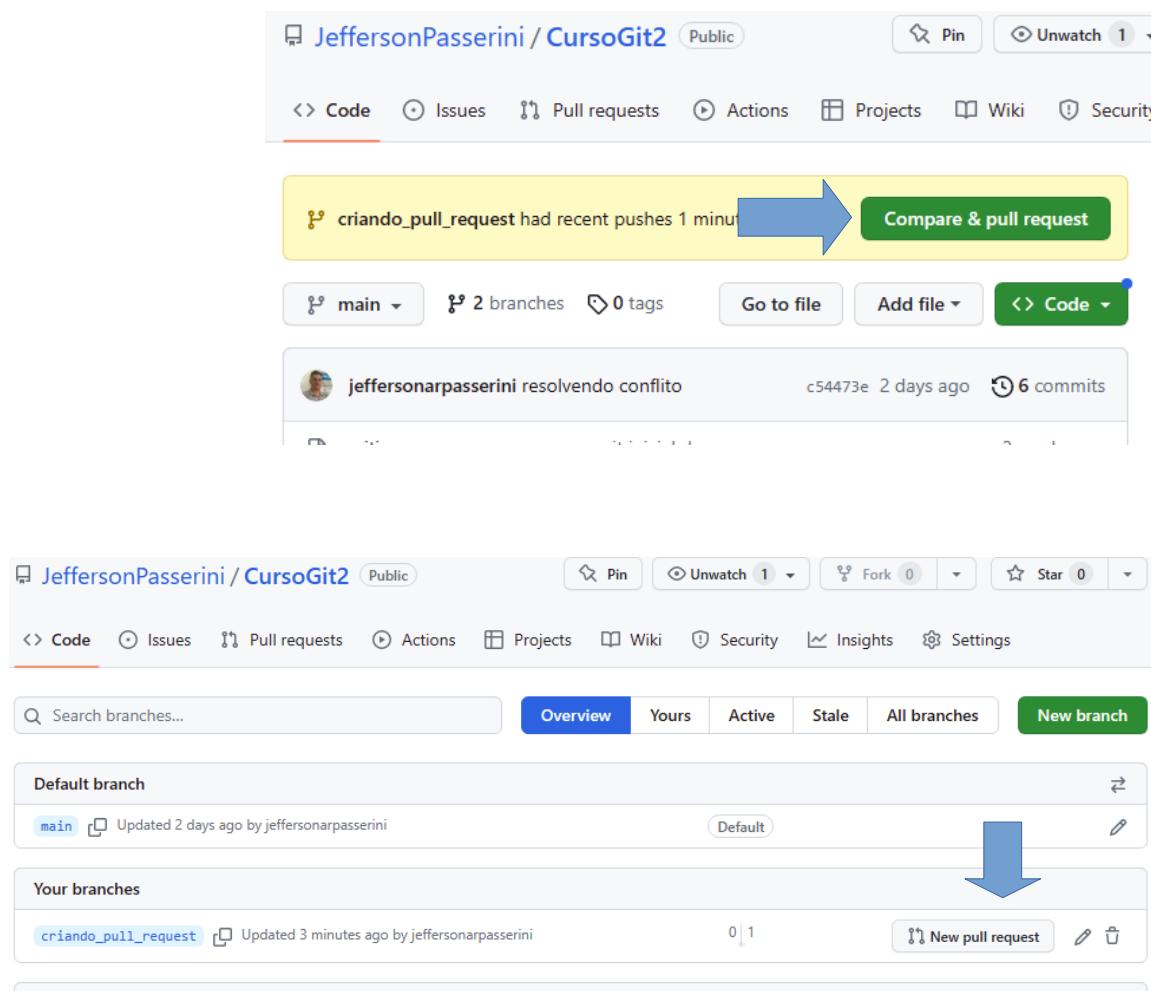


## .Pull Request

- Quando há projetos envolvendo equipes e, principalmente, modificações importantes, é recomendável que todas as alterações de *merging* sejam supervisionadas;
- Normalmente essa supervisão é realizada por outra pessoa, que faz parte da equipe de desenvolvimento e tem capacidade de analisar os impactos das alterações;
- O GitHub oferece o recurso de *pull request* – um meio para que as alterações seja revisadas e validades antes do merge.

## .Criando um Pull Request

- Crie uma nova branch no repositório local;
- Promova algumas modificações e atualize no repositório remoto;
- Abra o ambiente web do GitHub e clique no botão New Pull Request – veja a figura.



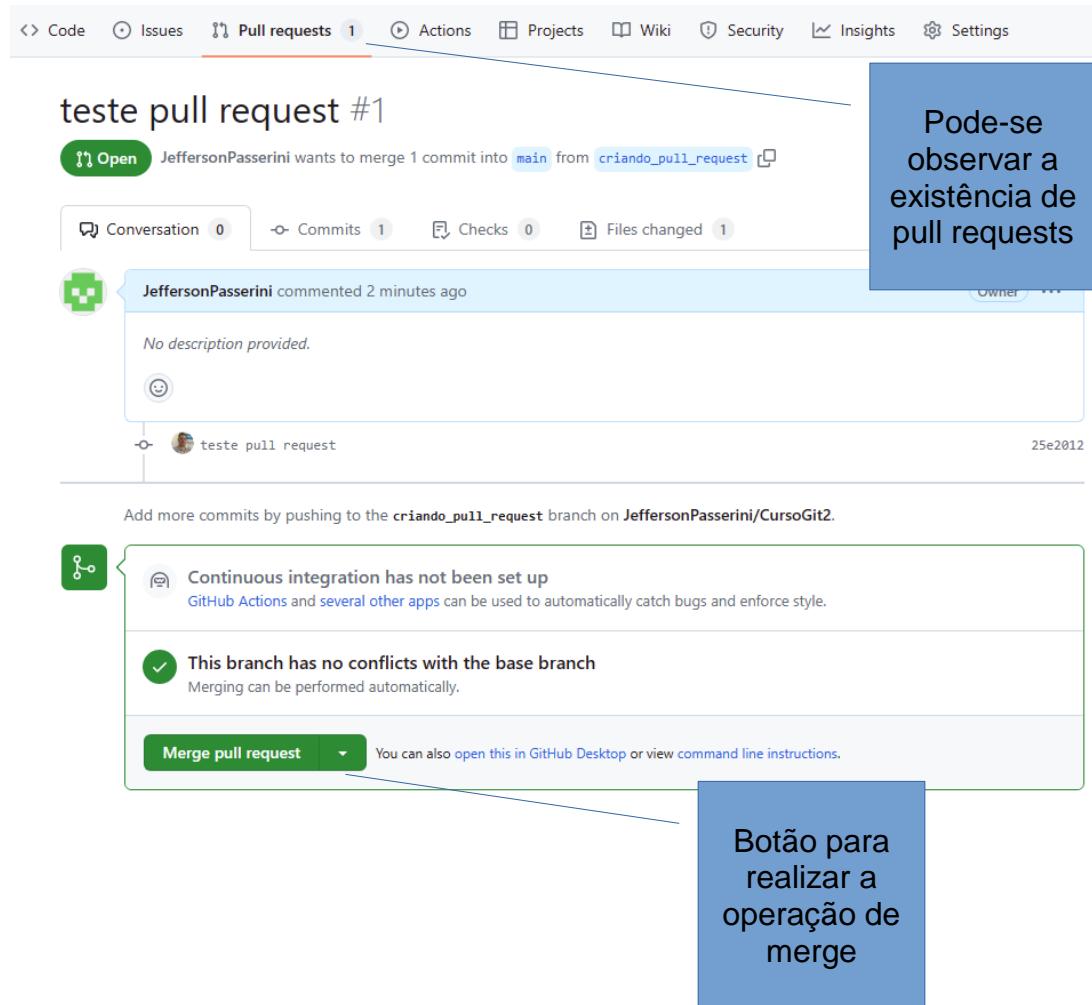
## .Criando um Pull Request

- Na tela de criação da branch podemos visualizar algumas informações importantes (vide a figura);
- Você também pode observar rolando a tela o detalhamento das alterações contidas na branch.
- Após verificar do detalhes do pull request clique em “Create pull request”



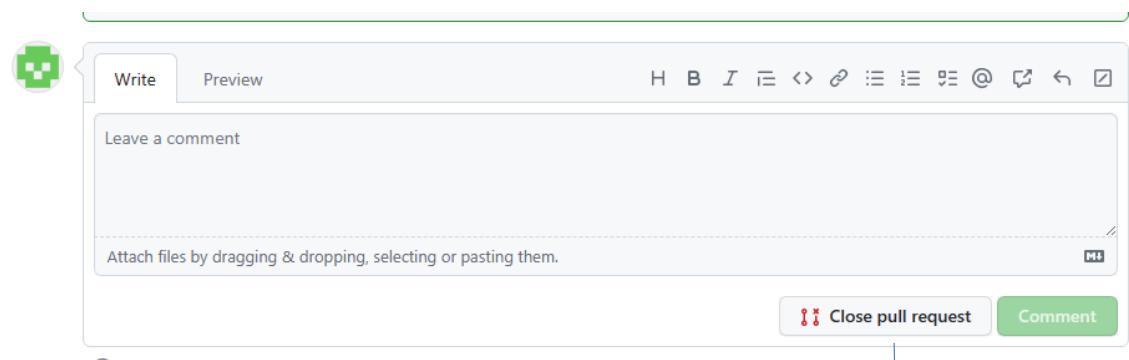
## .Criando um Pull Request

- Após a criação do “pull request” podemos fazer o merge;
- Após o merge ser realizado no ambiente web do GitHub não se esqueça de executar um “git pull” no bash de seu repositório local para atualizar os commits existem com o repositório remoto;
- Também não deve-se esquecer de remover a branch do repositório local e do remoto.



## .Criando um Pull Request

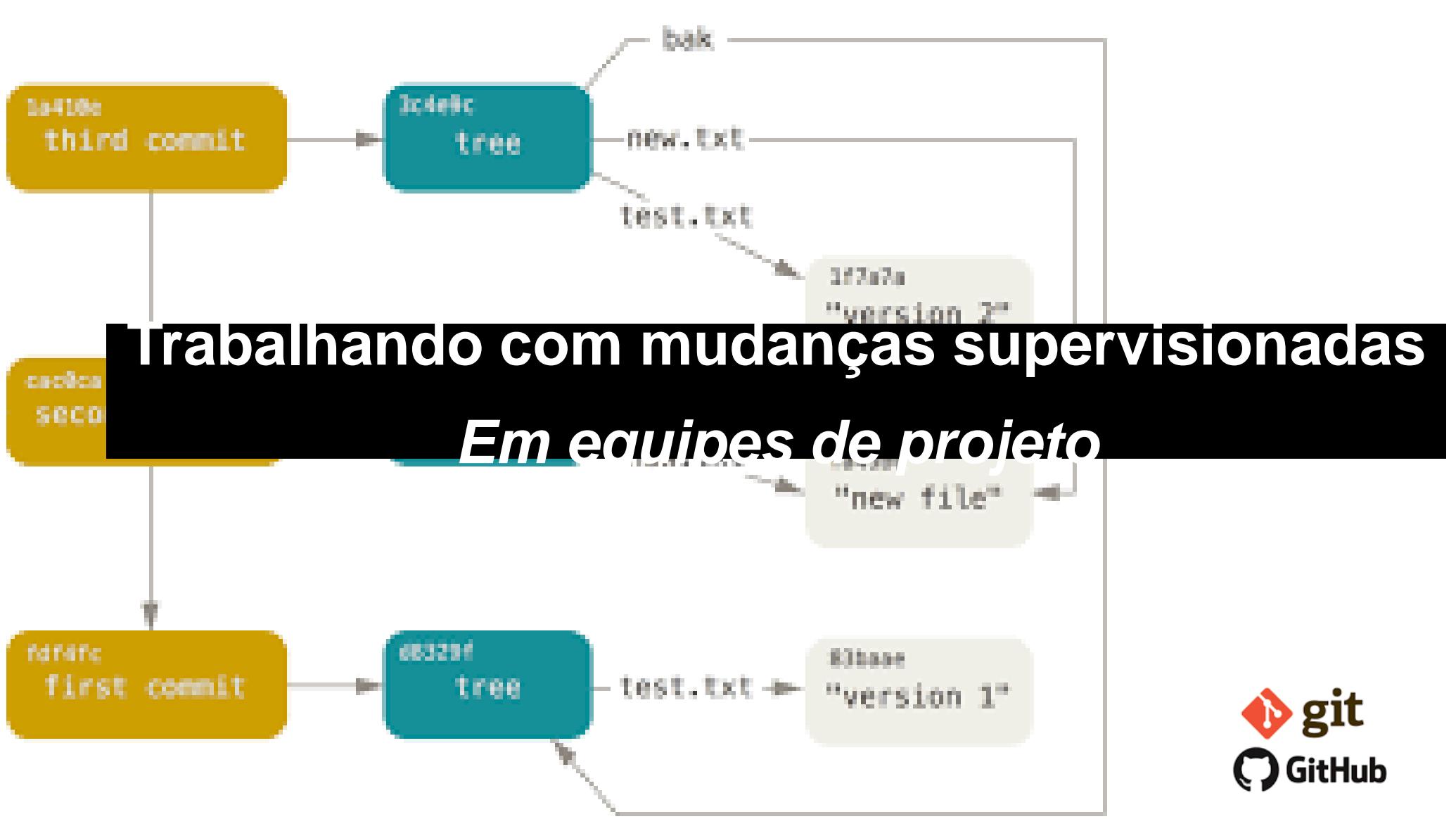
- Além de realizar o merge eventualmente quando estiver trabalhando em equipe você pode querer comentar e encerrar o pull request sem o merge.
- Ocorre quando a branch não atende aos requisitos necessários ou precisa de correções.



Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

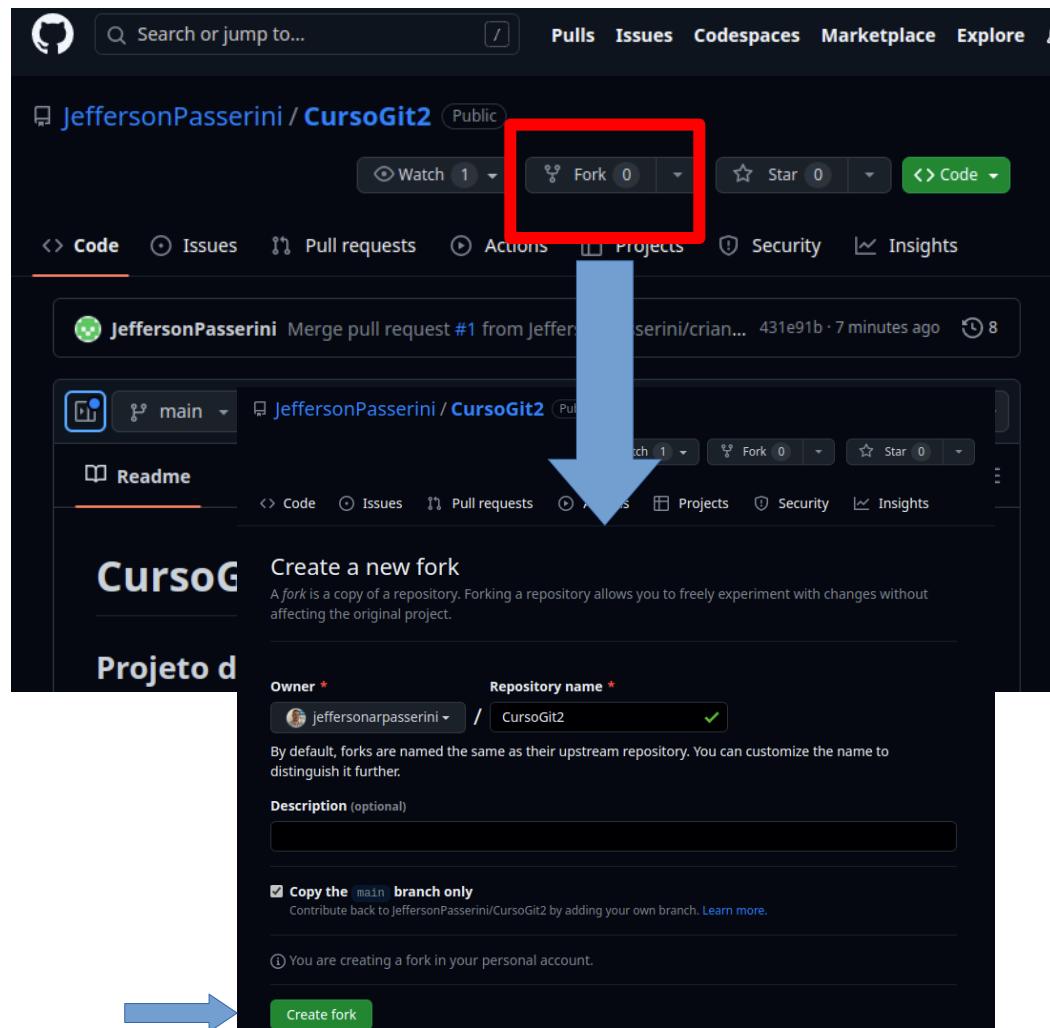
ProTip! Add `.patch` or `.diff` to the end of URLs for Git's plaintext views.

Botão para realizar a operação de merge



## *.Pull Request em equipe de projeto*

- Vamos preparar nosso laboratório de teste, para isso junte-se em grupos de 3 alunos;
- Um elemento do grupo irá criar um novo repositório remoto no GitHub;
- E irá fazer um clone deste repositório para o repositório local.
- Os outros dois componentes do grupo irão realizar um fork do projeto que o primeiro aluno criou:
  - Para isso vá a página do GitHub do aluno que criou o projeto e localize o repositório no GitHub deste aluno;
  - Clique no botão “Fork” para criar um fork do projeto para o seu GitHub;
  - O Fork é uma cópia do projeto.



## *.Pull Request em equipe de projeto*

- *Uma vez criando o Fork em seu perfil, pode-se observar (quadro vermelho) de qual projeto esse repositório foi forkado;*
- *Pode-se também resincronizar o fork em sua conta com o repositório original (quadro verde);*
- *A partir de agora podemos contribuir com o projeto.*



## .Pull Request em equipe de projeto

- Os alunos que fizeram o fork do projeto podem agora clonar o projeto para o seu repositório local;
- Após isso verifiquem se estão na branch main e criem uma nova branch para realizar alterações (git branch).
- Crie um novo arquivo e faça os comandos git add e git commit.

```
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch
* main
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch alteracao_jefferson
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch
    alteracao_jefferson
* main
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git switch alteracao_jefferson
Switched to branch 'alteracao_jefferson'
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch
* alteracao_jefferson
  main
(base) jeffersonpasserini@pop-os:~/CursoGit2$ 

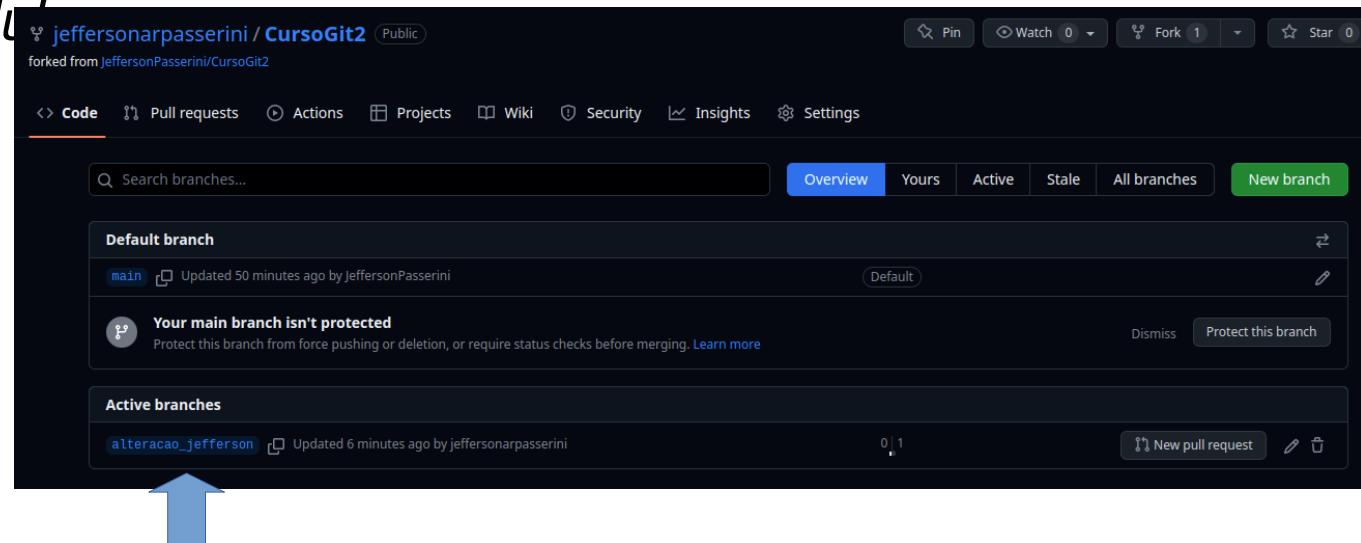
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git status
No ramo alteracao_jefferson
Arquivos não monitorados:
  (utilize "git add <arquivo>..." para incluir o que será submetido)
    Jefferson_pull_request.txt ←

nada adicionado ao envio mas arquivos não registrados estão presentes (use "git ad
d" to registrar)
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git add .
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git commit -m "teste pull request em
equipe"
[alteracao_jefferson f529798] teste pull request em equipe
  1 file changed, 2 insertions(+)
   create mode 100644 Jefferson_pull_request.txt
(base) jeffersonpasserini@pop-os:~/CursoGit2$ █
```

# .Pull Request em equipe de projeto

- Agora os alunos que criaram uma nova branch em seus forks devem realizar o push;
- Verifique no seu GitHub se a nova branch foi criada no repositório remoto.

```
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git push origin alteracao_jefferson
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), 338 bytes | 338.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'alteracao_jefferson' on GitHub by visiting:
remote:     https://github.com/jeffersonarpasserini/CursoGit2/pull/new/alteracao_
jefferson
remote:
To github.com:jeffersonarpasserini/CursoGit2.git
 * [new branch]      alteracao_jefferson -> alteracao_jefferson
(base) jeffersonpasserini@pop-os:~/CursoGit2$
```



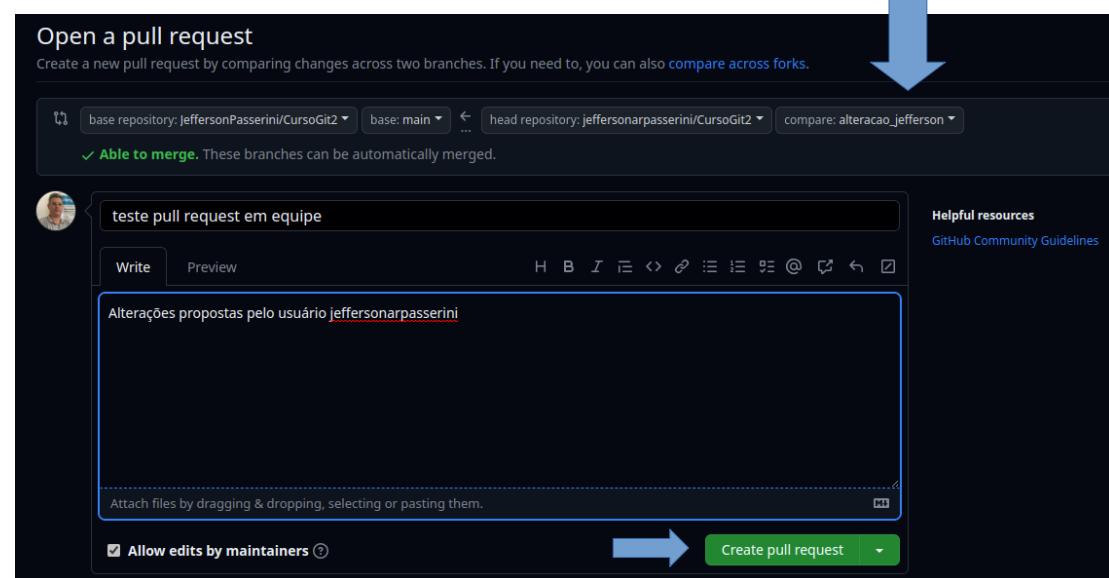
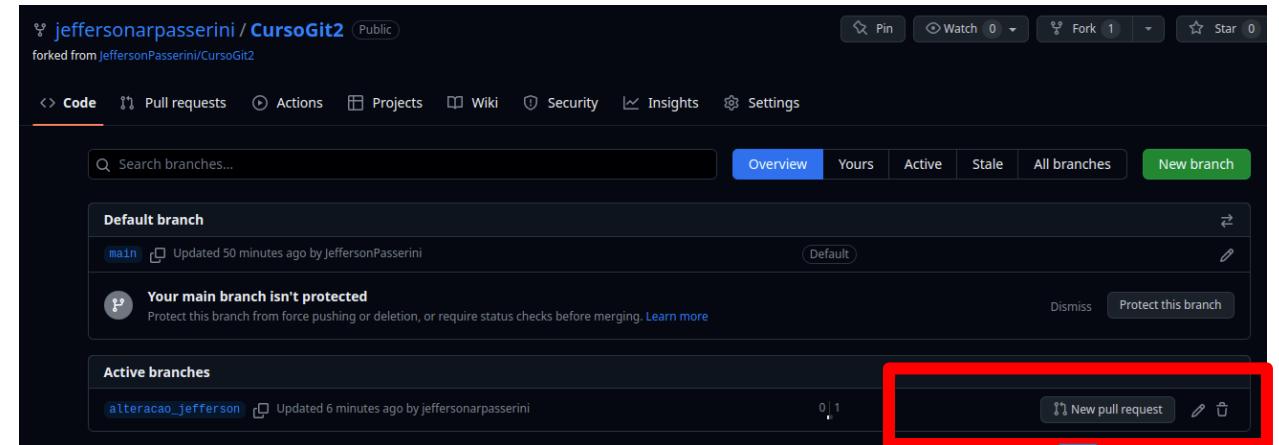
A screenshot of a GitHub repository page for 'JeffersonPasserini / CursoGit2'. The repository is public. The main navigation bar includes 'Code', 'Issues', 'Pull requests', 'Actions', and 'Projects'. Below the navigation is a search bar for branches. A large blue callout box highlights the text 'Repositório original com apenas a branch main'. The repository details show it was updated 53 minutes ago by JeffersonPasserini. A green 'New branch' button is visible.

A screenshot of a GitHub repository page for 'jeffersonarpasserini / CursoGit2'. It is a fork of 'JeffersonPasserini / CursoGit2'. The main navigation bar includes 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. Below the navigation is a search bar for branches. A red callout box highlights the text 'Repositório forkado na segunda conta github, com duas branches'. The repository details show it was updated 50 minutes ago by JeffersonPasserini. A message 'Your main branch isn't protected' is displayed, along with a 'Protect this branch' button. A green 'New branch' button is also present.

A screenshot of a GitHub repository page for 'jeffersonarpasserini / CursoGit2'. It is a fork of 'JeffersonPasserini / CursoGit2'. The main navigation bar includes 'Code', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. Below the navigation is a search bar for branches. A red callout box highlights the text 'Repositório forkado na segunda conta github, com duas branches'. The repository details show it was updated 6 minutes ago by jeffersonarpasserini. A message 'Your main branch isn't protected' is displayed, along with a 'Protect this branch' button. A green 'New branch' button is also present.

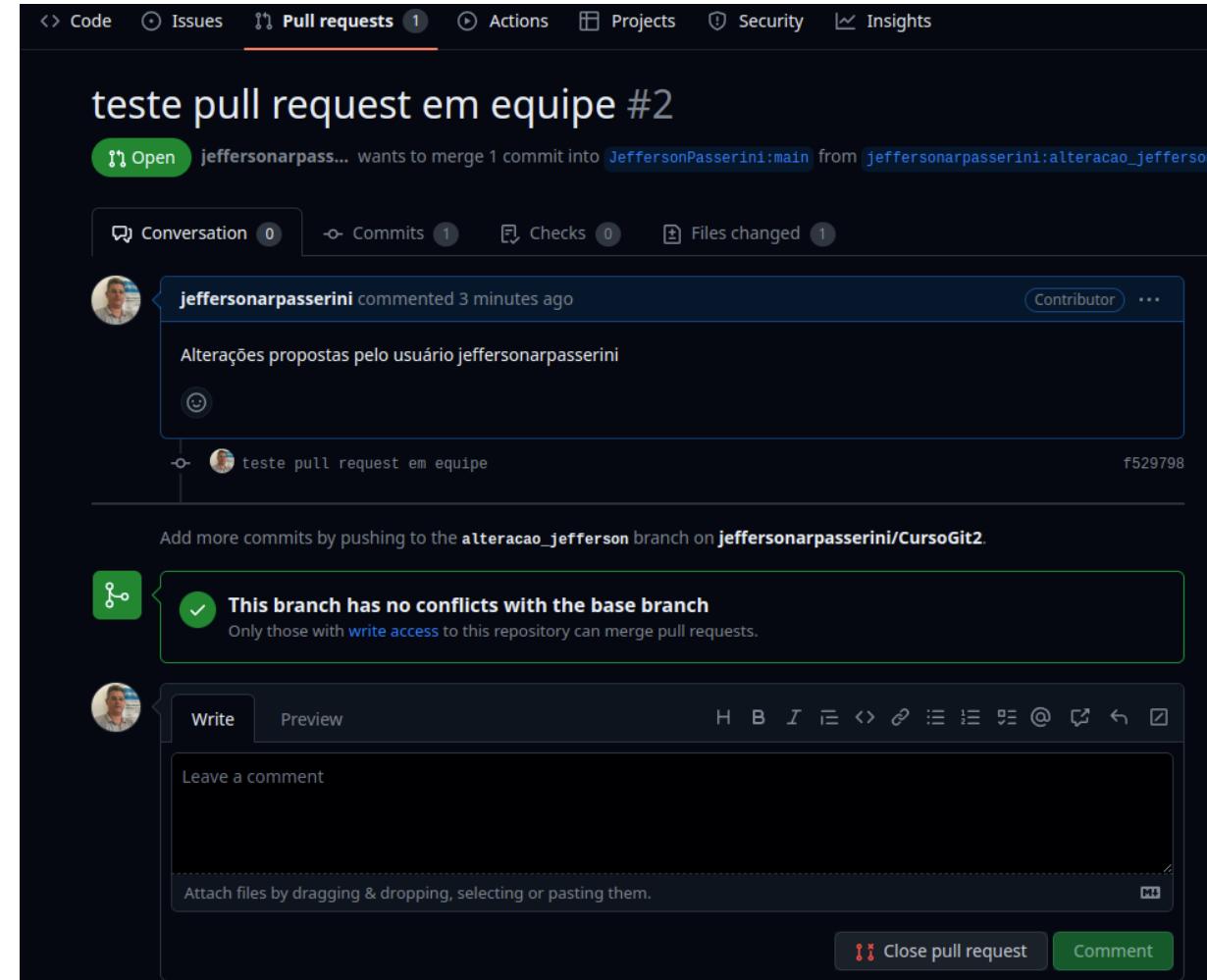
# .Pull Request em equipe de projeto

- Da branch criada com as alterações pretendidas os alunos que possuem os repositórios forkados devem abrir um pull request.



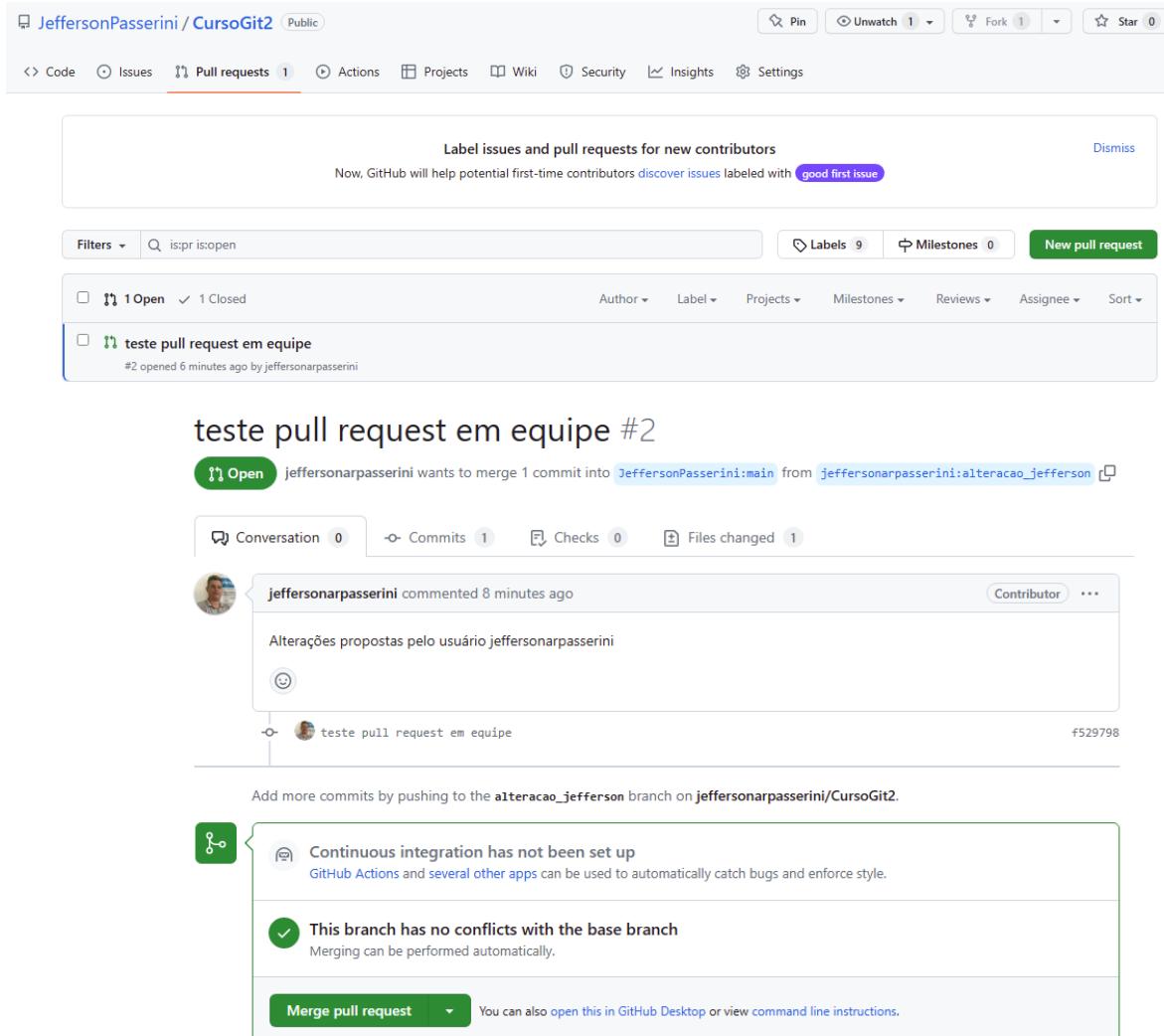
## .Pull Request em equipe de projeto

- Com o pull request gerado você perceberá que não conseguirá dar o merge nas branchs;
- Apenas fechar a solicitação de pull request cancelando ou ainda comentar na pull request;
- Isso ocorre porque o merge deve ser dados pelo proprietário do repositório.



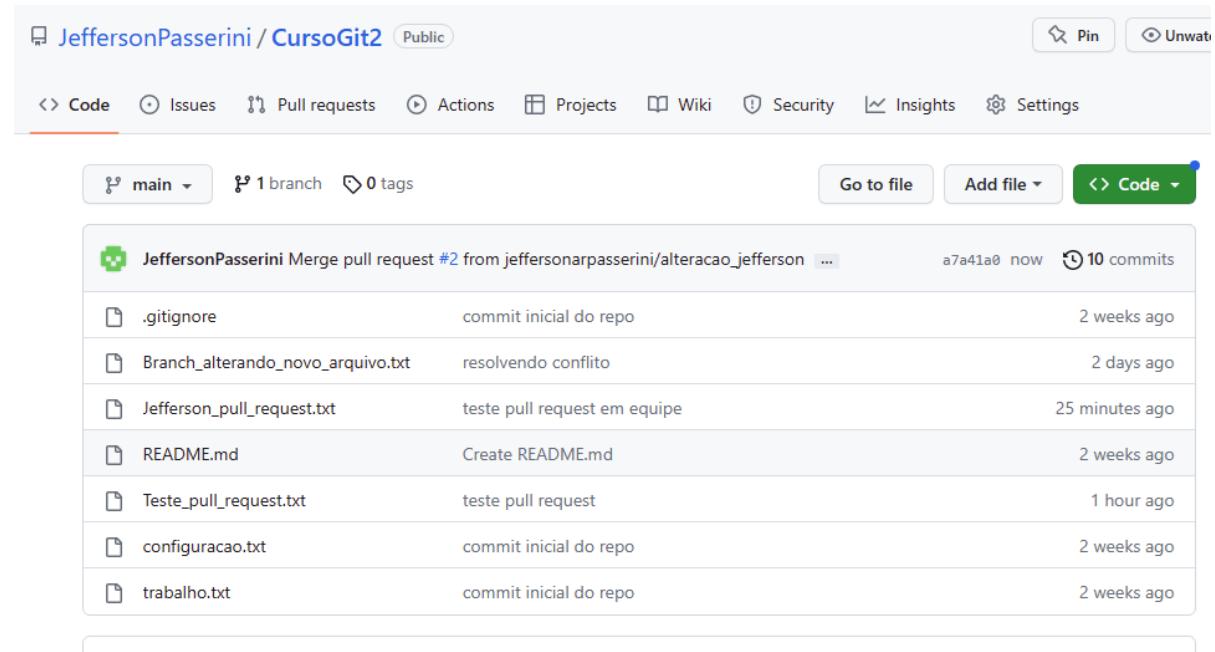
# .Pull Request em equipe de projeto

- Na conta do usuário proprietário do repositório agora tem uma requisição de pull request pendente de análise;
- Abra a requisição faça suas considerações, solicite alterações ou ainda faça o merge com a branch main em seu projeto;
- Clique no botão “Merge pull request”.



## *.Pull Request em equipe de projeto*

- Agora o arquivo que foi incluído na branch que foi feito o merge consta da branch main do repositório original;
- O proprietário do repositório de um git pull no repositório local para sincronizar as alterações.



The screenshot shows a GitHub repository page for 'JeffersonPasserini / CursoGit2' (Public). The 'Code' tab is selected. At the top, it shows 'main' branch, '1 branch', and '0 tags'. Below the header, there's a list of commits from a pull request merge:

Commit	Message	Date	
a7a41a0	JeffersonPasserini Merge pull request #2 from jeffersonarpasserini/alteracao_jefferson ...	now	
	commit inicial do repo	2 weeks ago	
	Branch_alterando_novo_arquivo.txt	resolvendo conflito	2 days ago
	Jefferson_pull_request.txt	teste pull request em equipe	25 minutes ago
	README.md	Create README.md	2 weeks ago
	Teste_pull_request.txt	teste pull request	1 hour ago
	configuracao.txt	commit inicial do repo	2 weeks ago
	trabalho.txt	commit inicial do repo	2 weeks ago

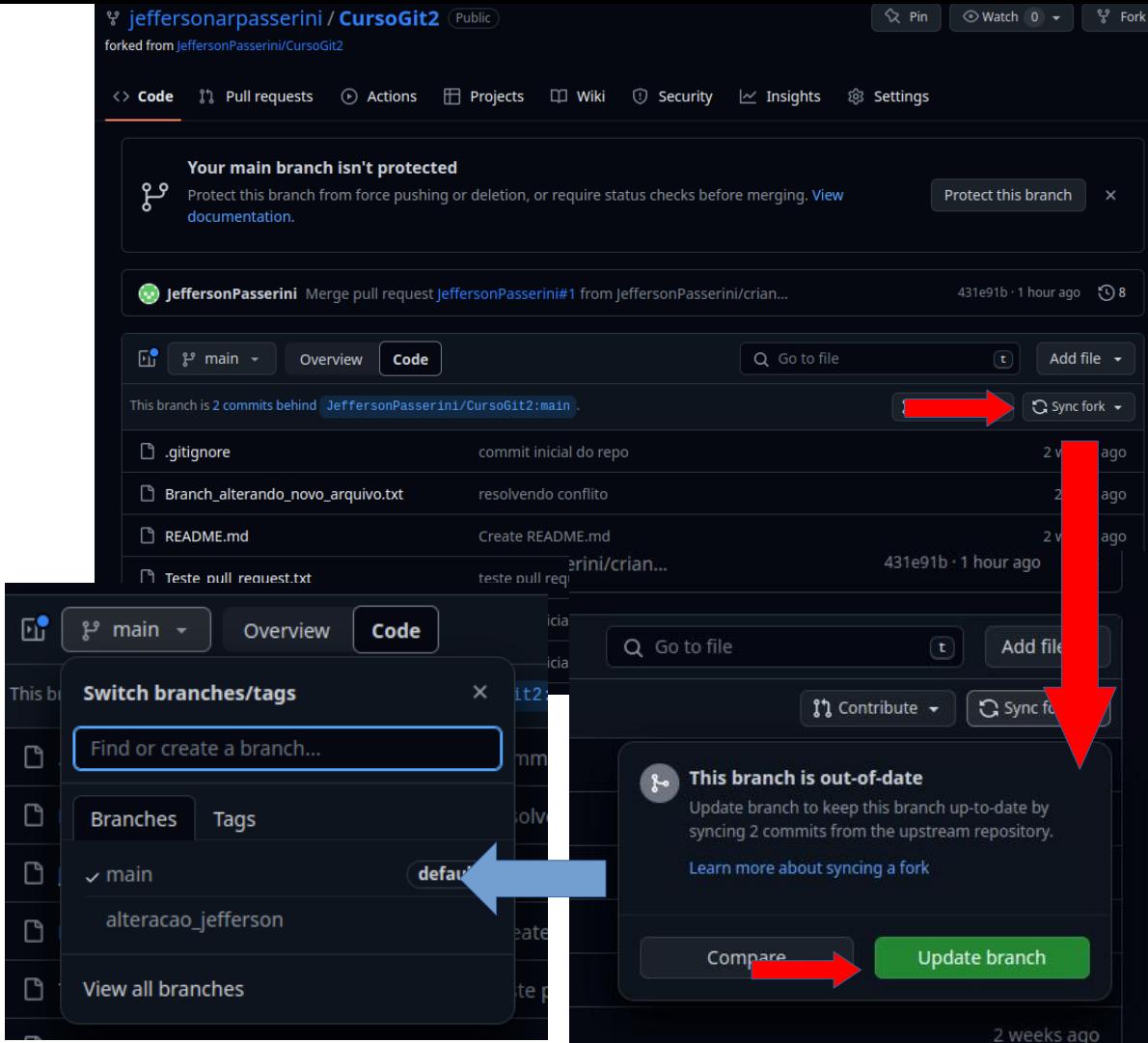
# *.Pull Request em equipe de projeto*

- Para o usuário forkado que requisitou o pull request aparecerá a atualização do status para “Merged”;*



# *.Pull Request em equipe de projeto*

- Para atualizar o repositório forkado deverá clicar no seu repositório no botão “sync fork”;
- Atualizando o repositório remoto do usuário que realizou o fork do projeto;
- Agora temos a branch main atualizada no fork e ainda consta a branch “alteracao\_jefferson”.



## .Pull Request em equipe de projeto

- Agora volte para o repositório local (usuário do fork) e garanta que esteja na branch main;
- Faça um git pull para atualizar o repositório local;

```
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch
* alteracao_jefferson
  main
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch
  alteracao_jefferson
* main
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git pull origin main
From github.com:jeffersonarpasserini/CursoGit2
 * branch            main      -> FETCH_HEAD
Already up to date.
(base) jeffersonpasserini@pop-os:~/CursoGit2$
```

```
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git pull origin main
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 638 bytes | 638.00 KiB/s, done.
From github.com:jeffersonarpasserini/CursoGit2
 * branch            main      -> FETCH_HEAD
   431e91b..a7a41a0  main      -> origin/main
Updating 431e91b..a7a41a0
Fast-forward
  Jefferson_pull_request.txt | 2 ++
    1 file changed, 2 insertions(+)
    create mode 100644 Jefferson_pull_request.txt
```

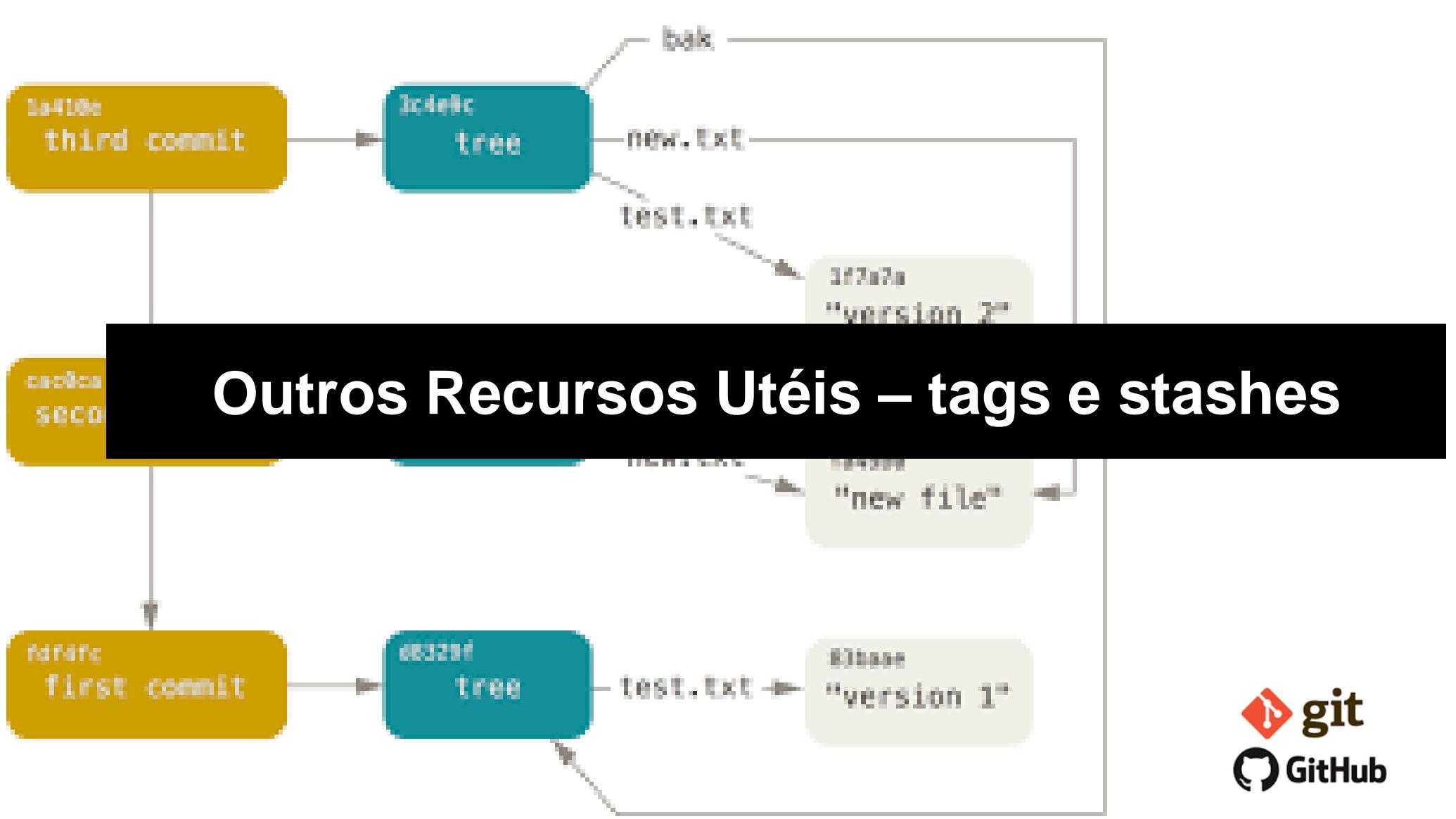
## .Pull Request em equipe de projeto

- Remova a branch local e depois remova a branch remota.

```
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch
  alteracao_jefferson
* main
(base) jeffersonpasserini@pop-os:~/CursoGit2$ git branch -d alteracao_jefferson
Deleted branch alteracao_jefferson (was f529798).

(base) jeffersonpasserini@pop-os:~/CursoGit2$ git push --delete origin alteracao_jefferson
To github.com:jeffersonarpasserini/CursoGit2.git
 - [deleted]          alteracao_jefferson
(base) jeffersonpasserini@pop-os:~/CursoGit2$
```

- Todos os usuários forkados devem fazer o “sync fork” regularmente no github;
- E todos os usuários é recomendável sempre fazer o git pull para atualizar o repositório local antes de abrir uma nova branch.



## Outros Recursos Utéis – tags e stashes