**Curso de Git e Github**

**Prof. Anderson Coimbra Choren**

Vídeos: https://www.youtube.com/watch?v=beMnH51P-T4&list=PLmMxPWmzYRGcTabffOwHBORBjtKa2wCXS

GitHub: https://github.com/professorchoren/curso\_de\_git\_e\_github

Resumo feito por Roberto Pinheiro (https://github.com/betopinheiro1005)

# Aula 01 - Controle de versão

## SCV, o que são e qual sua importância

### Modelo tradicional de versionamento

Gravando os arquivos de cada versão do projeto em um arquivo compactado ZIP diferente.

Desvantagens:

* Não sabemos o que mudou de uma versão para outra;
* Não sabemos quem e nem quando essas modificações foram feitas;
* Grande chance de haver divergências de código caso estejamos trabalhando em equipe;
* Gasto excessivo de espaço em disco;

A solução é usar um sistema de controle de versão (SCV).

### Vantagens de se usar um SCV

* Controle de histórico;
* Trabalho em equipe;
* Marcação e resgate de versões estáveis;
* Ramificação do projeto;
* Segurança;
* Rastreabilidade;
* Organização;
* Confiança;

## Tipos de controle de versão

Os SCV se dividem em dois grupos:

1. Centralizados
2. Distribuídos

### Sistema de controle centralizado

- O servidor possui todo o controle de versão do código-fonte.

- Basicamente, os clientes sempre trabalham em rede com os arquivos no servidor.

Exemplos:

* CVS
* Microsoft Visual SourceSafe 6.0
* Apache Subversion ou SVN

### Sistema de controle distribuído

- Não há a necessidade de um servidor mas pode haver;

- Os clientes possuem toda a cópia do código-fonte localmente;

- Ou seja: o cliente tem a cópia local (em seu computador) e a cópia remota (no servidor);

Exemplos:

* GIT (o mais conhecido)
* Mercurial

# Aula 02 - Introdução ao GIT e sua importância para nós desenvolvedores

- O Git trabalha com o conceito de repositórios.

- Um repositório é uma pasta a qual o Git deverá monitorar. Ou seja, é a pasta onde o Git atuará gerenciando as diversas versões dos arquivos ali presentes.

## Os três estados do Git

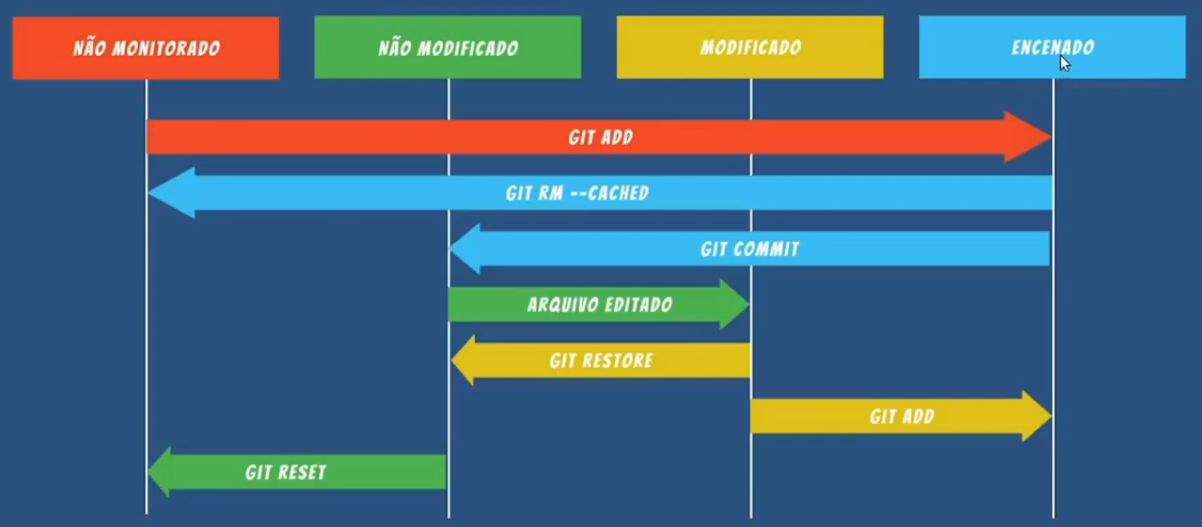
1. Diretório de trabalho;
2. Área de preparo;
3. Repositório local



## Os quatro estados dos arquivos

Quando trabalhamos com Git, nossos arquivos podem ser classificados em quatro estados:

1. Untracked (não monitorado)
2. Unmodified (não modificado)
3. Modified (modificado)
4. Staged (Encenado)

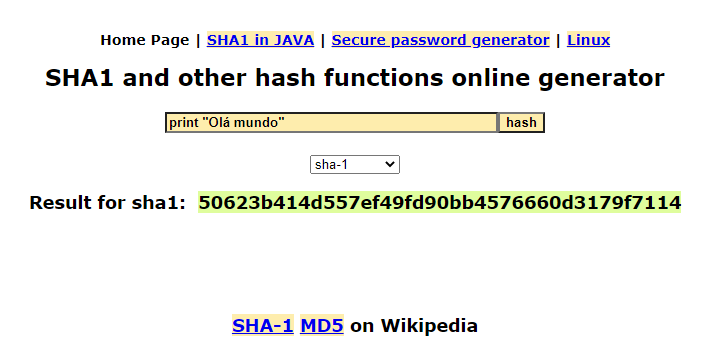


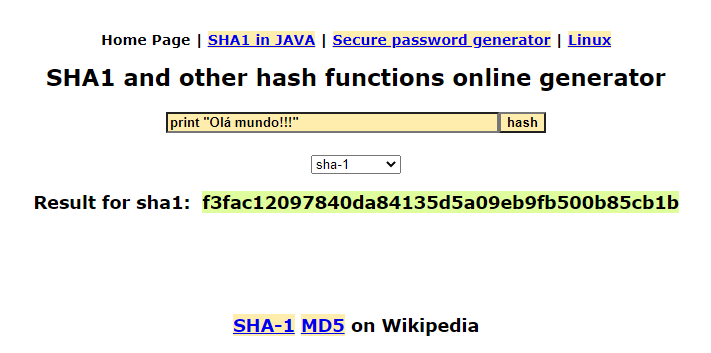
## O Hash SHA1

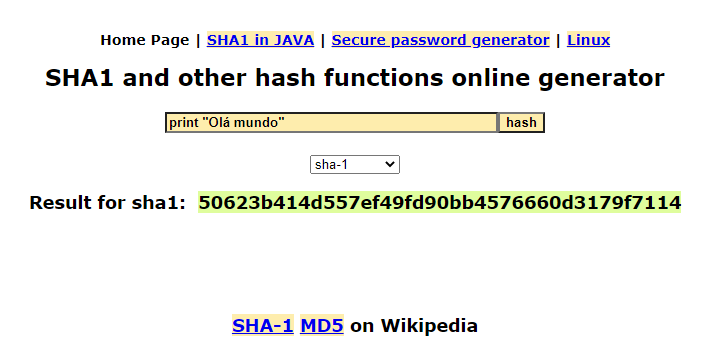
O Git utiliza o algoritmo de hash sha1 para definir quando um arquivo foi modificado.

Dessa forma o Git é capaz de identificar quais linhas do arquivo foram modificadas, mantendo assim um histórico de versões para esse arquivo.

http://www.sha1-online.com/







Versão 1: 50623b414d557ef49fd90bb4576660d3179f7114

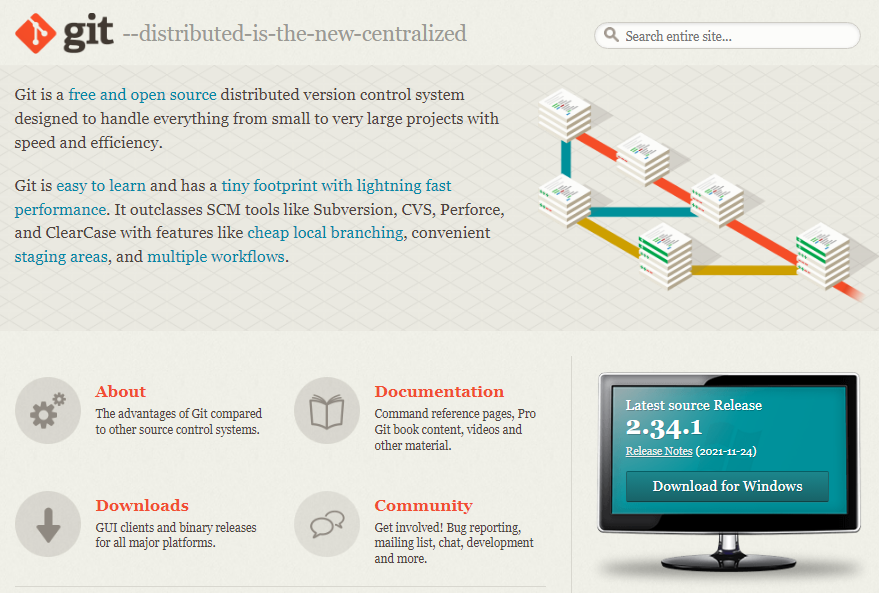
Versão 2: f3fac12097840da84135d5a09eb9fb500b85cb1b

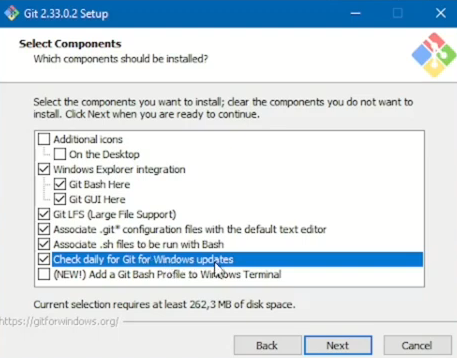
Versão 3: 50623b414d557ef49fd90bb4576660d3179f7114

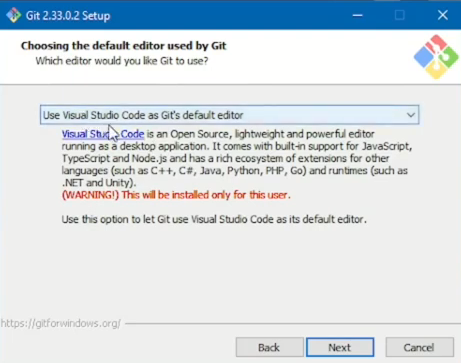
# Aula 03 - Instalando o GIT no Windows

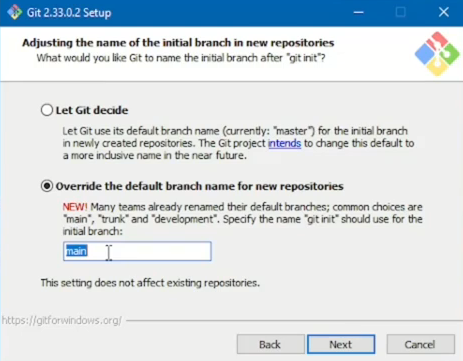
- Acesse o site e faça o download.

https://git-scm.com/









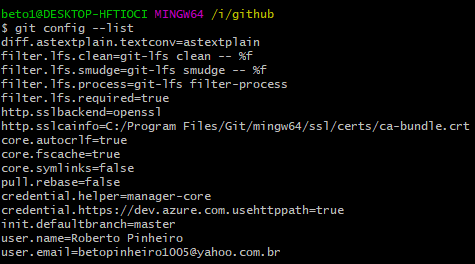
## Configurando as variáveis de ambiente

git config --global user.name "Roberto Pinheiro"

git config --global email.name "betopinheiro1005@yahoo.com.br"

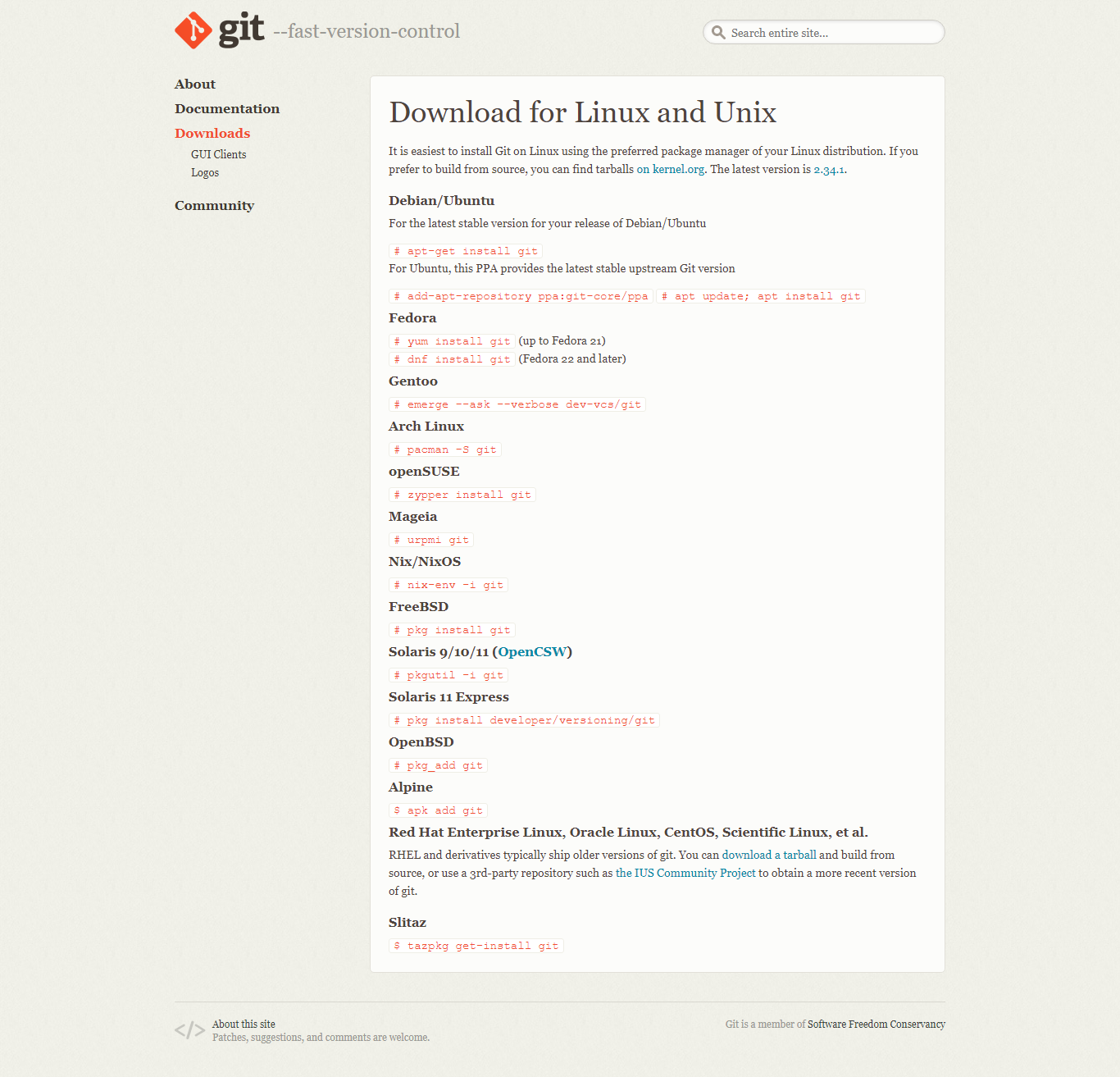
### Checando se está configurado corretamente

git config --list



# Aula 04 - Instalando o GIT no Linux

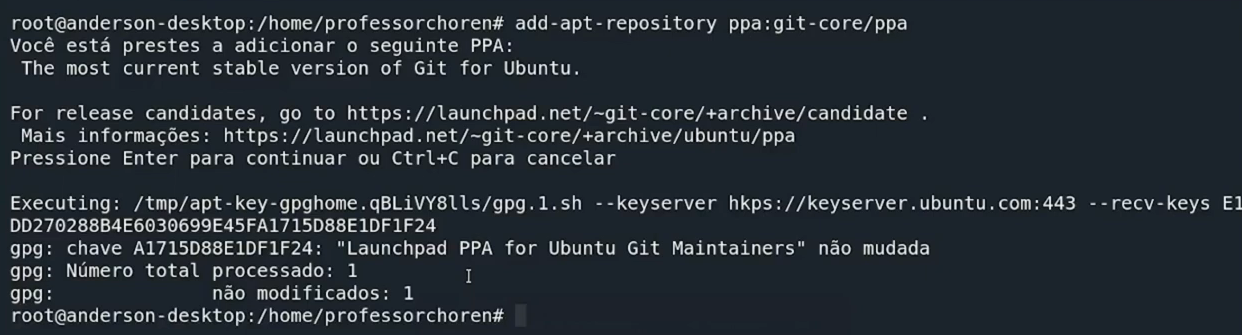
https://git-scm.com/download/linux



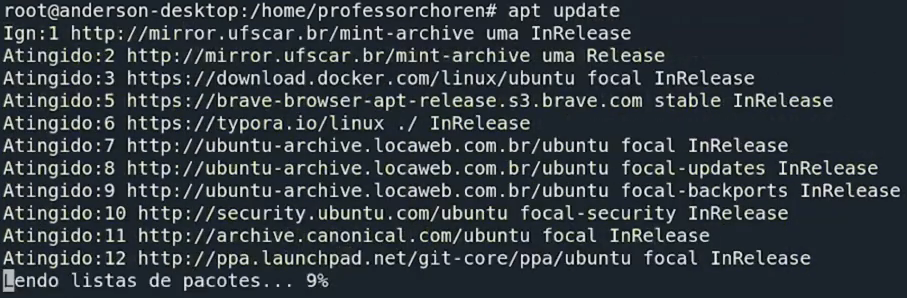
### Debian/Ubuntu

For Ubuntu, this PPA provides the latest stable upstream Git version

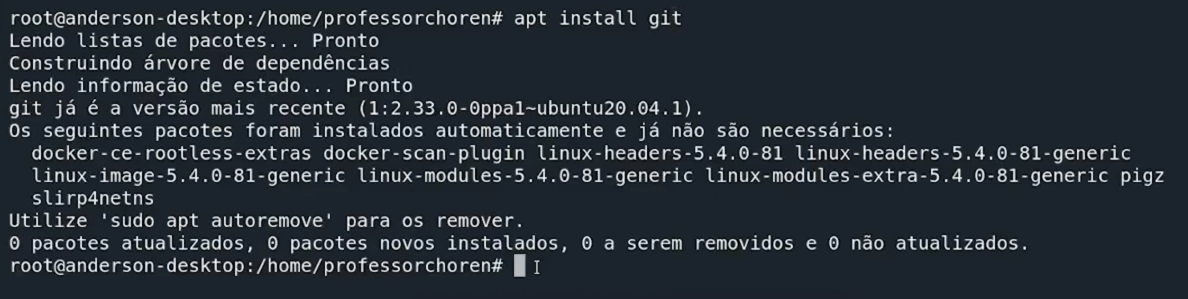
add-apt-repository ppa:git-core/ppa



apt update

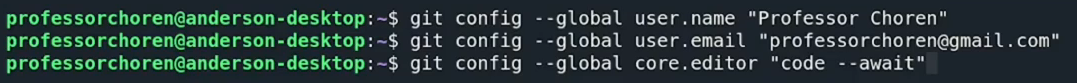


apt install git





## Configurando as variáveis de ambiente



# Aula 05 - Comandos básicos do GIT

- Crie a pasta git\_github\_choren

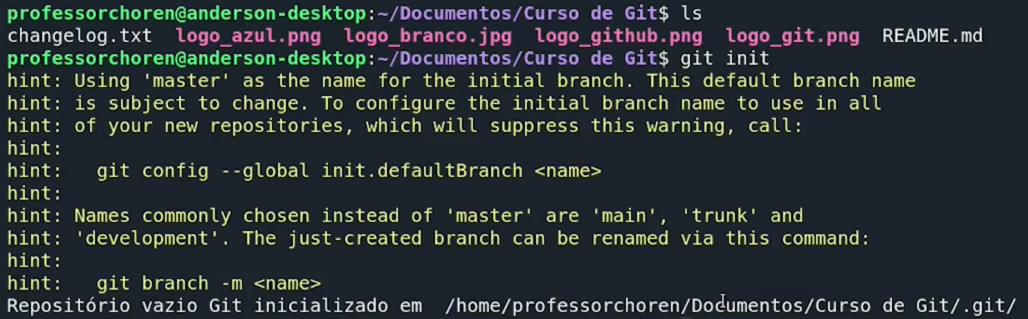
## Listando os arquivos na pasta

ls



## Inicializando um repositório git

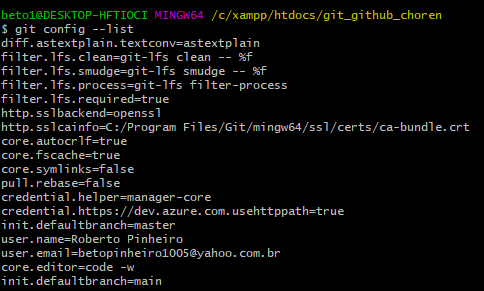
git init



git config --global init.defaultBranch main



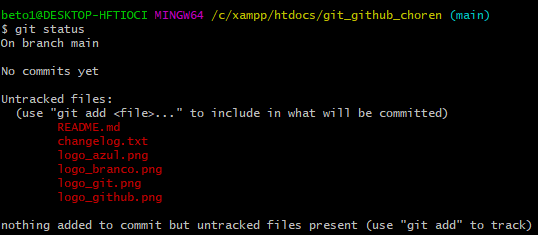
git config --list





É criada uma pasta oculta chamada .git

git status



## Removendo o repositório git

rm -rf .git



git status

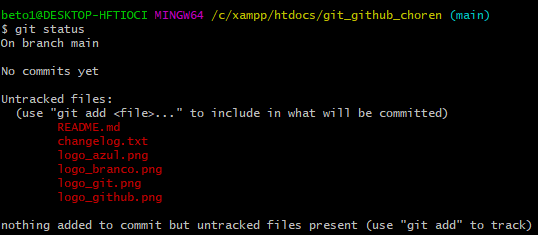


## Inicializando novamente o repositório git

git init



git status



- Os arquivos não estão sendo monitorados pelo Git.

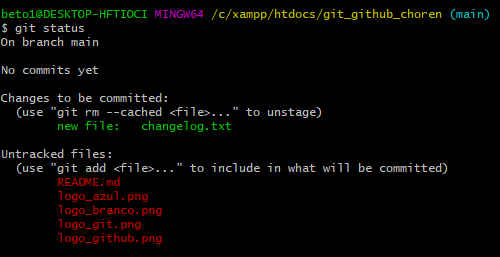
## Adicionando arquivos ao controle de versão

### Adicionando um arquivo específico pelo nome

git add changelog.txt



git status

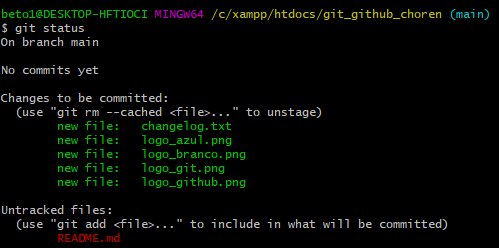


### Adicionando arquivos com uma determinada extensão

git add \*.png



git status



### Adicionando todos os arquivos

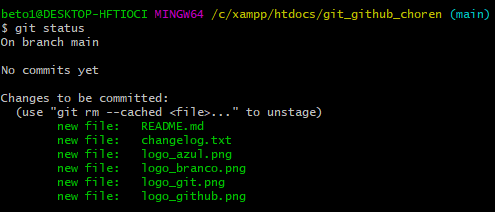
git add --all

ou

git add .



git status



# Aula 06 - Comando git commit

## Configurando o editor padrão

- O editor padrão é o vi.

- Para configurar o Visual Studio Code como o editor padrão:

git config --global core.editor "code -w"

-w para aguardar o editor fechar.



## Convenção para commits

A estrutura dos commits é descrita por uma convenção

https://www.conventionalcommits.org/pt-br/v1.0.0/

A especificação do Conventional Commits é uma convenção simples para utilizar nas mensagens de commit. Ela define um conjunto de regras para criar um histórico de commit explícito, o que facilita a criação de ferramentas automatizadas baseadas na especificação. Esta convenção se encaixa com o SemVer, descrevendo os recursos, correções e modificações que quebram a compatibilidade nas mensagens de commit.

A mensagem do commit deve ser estruturada da seguinte forma:

<tipo>[escopo opcional]: <descrição>

[corpo opcional]

[rodapé(s) opcional(is)]

O commit contém os seguintes elementos estruturais, para comunicar a intenção ao utilizador da sua biblioteca:

fix: um commit do tipo fix soluciona um problema na sua base de código (isso se correlaciona com PATCH do versionamento semântico).

feat: um commit do tipo feat inclui um novo recurso na sua base de código (isso se correlaciona com MINOR do versionamento semântico).

BREAKING CHANGE: um commit que contém no rodapé opcional o texto BREAKING CHANGE:, ou contém o símbolo ! depois do tipo/escopo, introduz uma modificação que quebra a compatibilidade da API (isso se correlaciona com MAJOR do versionamento semântico). Uma BREAKING CHANGE pode fazer parte de commits de qualquer tipo.

Outros tipos adicionais são permitidos além de fix: e feat:, por exemplo @commitlint/config-conventional (baseado na Convenção do Angular) recomenda-se build:, chore:, ci:, docs:, style:, refactor:, perf:, test:, entre outros.

Outros rodapés diferentes de BREAKING CHANGE: <descrição> podem ser providos e seguem a convenção simular a git trailer format.

Observe que esses tipos adicionais não são exigidos pela especificação do Conventional Commits e não têm efeito implícito no versionamento semântico (a menos que incluam uma BREAKING CHANGE). Um escopo pode ser fornecido ao tipo do commit, para fornecer informações contextuais adicionais e está contido entre parênteses, por exemplo feat(parser): adiciona capacidade de interpretar arrays.

## Versionamento semântico

https://semver.org/lang/pt-BR/

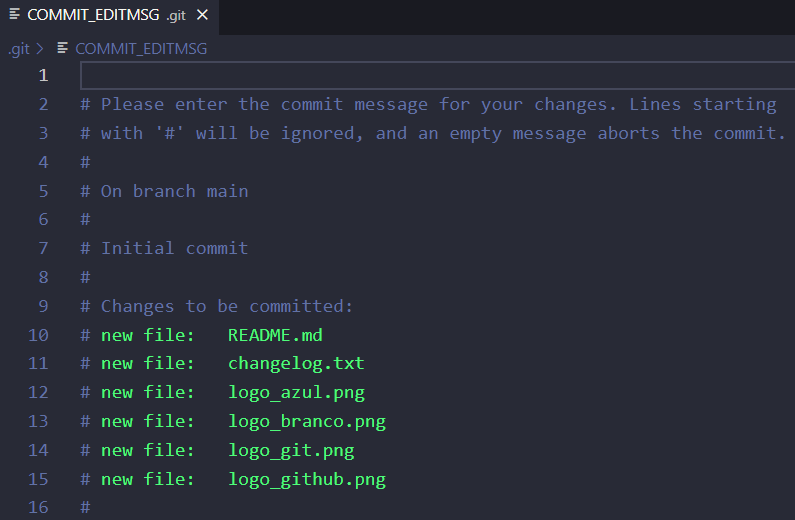
Versionamento Semântico 2.0.0

Dado um número de versão MAJOR.MINOR.PATCH, incremente a:

* versão Maior(MAJOR): quando fizer mudanças incompatíveis na API,
* versão Menor(MINOR): quando adicionar funcionalidades mantendo compatibilidade, e
* versão de Correção(PATCH): quando corrigir falhas mantendo compatibilidade.
* Rótulos adicionais para pré-lançamento(pre-release) e metadados de construção(build) estão disponíveis como extensão ao formato MAJOR.MINOR.PATCH.

## Realizando o primeiro commit

git commit





**.git\COMMIT\_EDITMSG**

feat: adicionar os recursos base do projeto

adição dos arquivos:

README.md

changelog.txt

logo\_azul.png

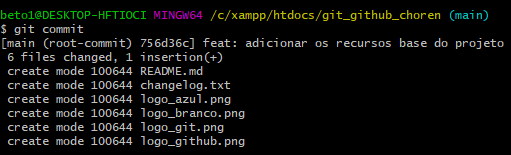
logo\_branco.png

logo\_git.png

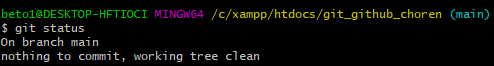
logo\_github.png

- Nunca se deve utilizar . (ponto final) no commit

- Salve o arquivo e feche o editor.



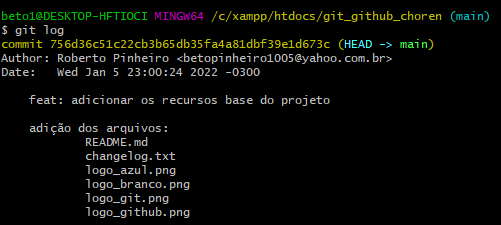
git status



# Aula 07 - Comando git log

- Para saber o que já foi commitado:

git log



HEAD é a cabeça (ou ponteiro) do repositório, ou seja: para onde o Git está apontando naquele momento.

O HEAD é o apontamento do Git para um commit, ou seja: que commit atualmente eu estou trabalhando.

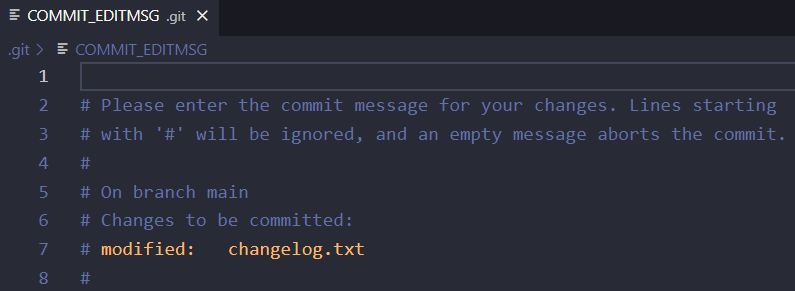
- Altere o conteúdo do arquivo changelog.txt

**changelog.txt**

05/01/2022 - Aprendendo a usar o comando git log.

### Adicionando a modificação e realizando o commit

git commit -a



**.git\COMMIT\_EDITMSG**

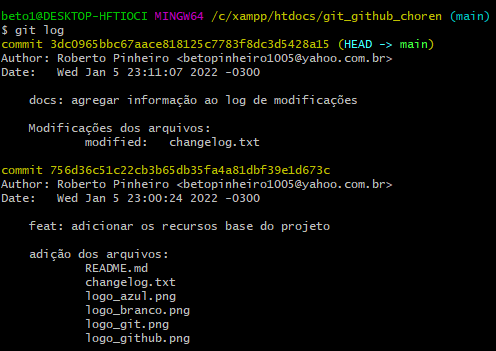
docs: agregar informação ao log de modificações

Modificações dos arquivos:

modified: changelog.txt



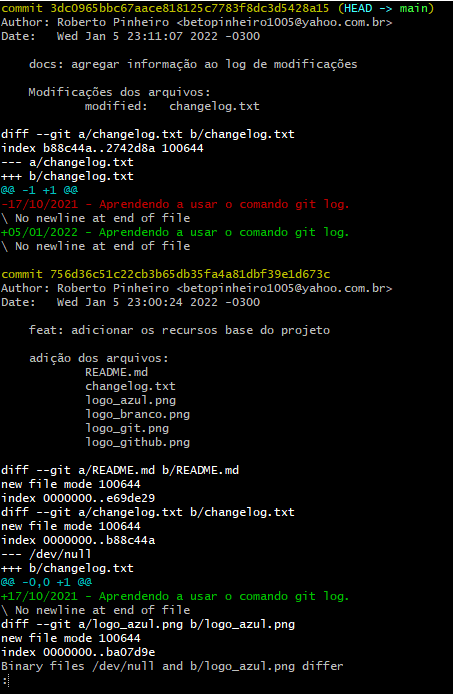
git log



- Observe que a cabeça do repositório (HEAD) foi movida para o novo commit.

## Exibição de commits mais detalhada

git log -p

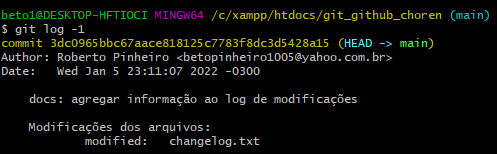


- Pressione a tecla <q> para sair.

## Filtrando logs

- Para apresentar apenas o último log:

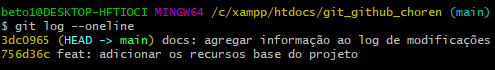
git log -1



- Conta de trás para frente.

## Log com informação resumida

git log --oneline



- Apresenta os sete primeiros caracteres do hash.

## Filtro resumido do último commit

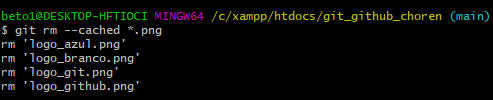
git log --oneline -1



# Aula 08 - Comando git rm

O comando git rm basicamente é uma forma de remover um arquivo do controle de versão.

git rm --cached \*.png



ls



- Perceba que os arquivos não foram removidos do diretório, apenas do controle de versão.

git status



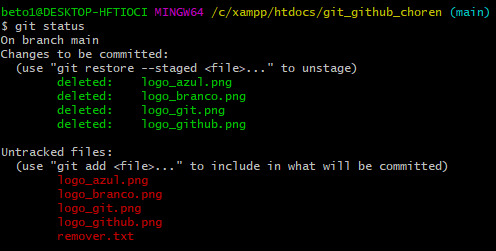
### Criando um novo arquivo de texto

echo > remover.txt

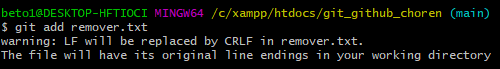
ls



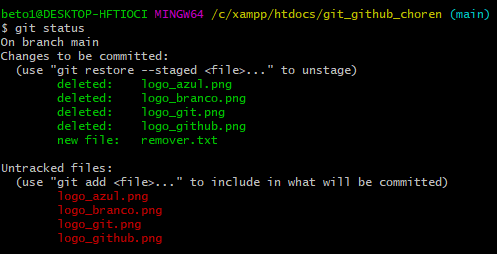
git status



git add remover.txt



git status



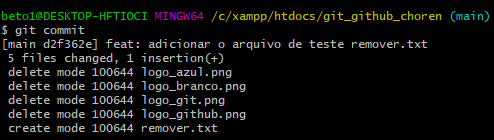
git commit

**.git\COMMIT\_EDITMSG**

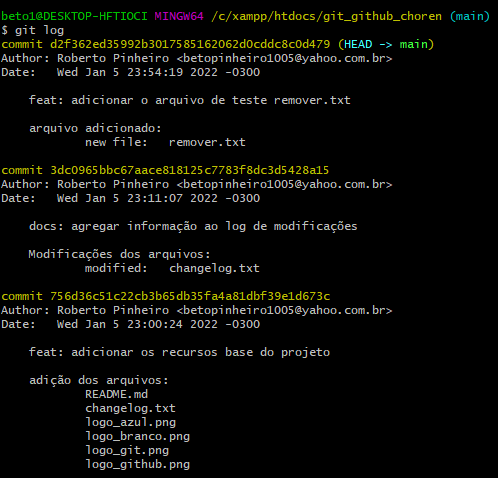
feat: adicionar o arquivo de teste remover.txt

arquivo adicionado:

new file: remover.txt



git log



ls



git rm remover.txt

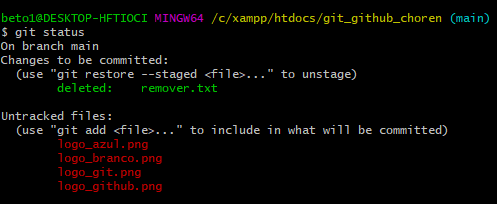


Sem o --cached na linha de comandos acima, o arquivo é removido do controle de versão e da pasta de arquivos.

ls



git status



# Aula 09 - Comando git diff

**README.txt**

# Curso de Git e GitHub

1. Controle de versão

2. Conhecendo o Git

3. Instalando o Git CLI no Windows

4. Instalando o Git CLI no Linux

5. Comandos básicos do Git

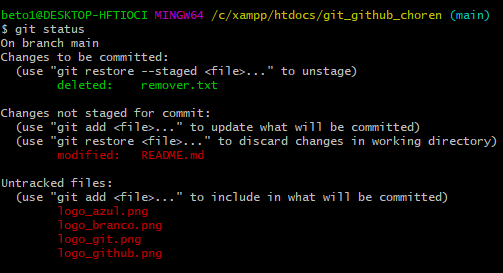
6. Comando git commit

7. Comando git log

8. Comando git rm

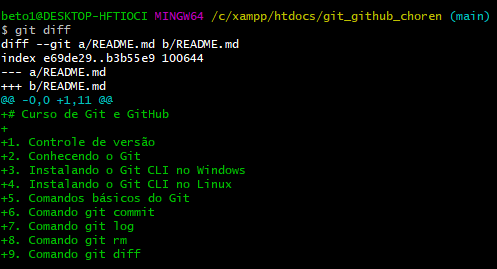
9. Comando git diff

git status



## Mudanças em relação ao último commit

git diff

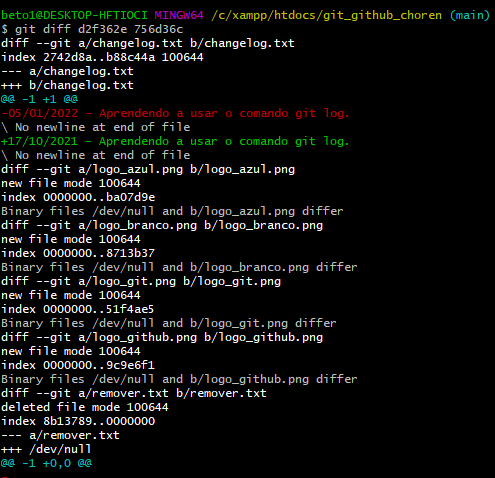


## Comparação entre commits

git log --oneline

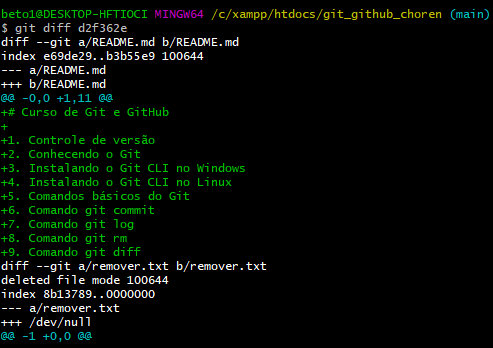


git diff d2f362e 756d36c



## Diferença da versão atual para o último commit

git diff d2f362e



# Aula 10 - O arquivo .gitignore

.gitignore é um arquivo onde é declarado quais são os arquivos ou pastas que devem ser ignorados no monitoramento do versionamento do código.

## Criando o arquivo .gitignore

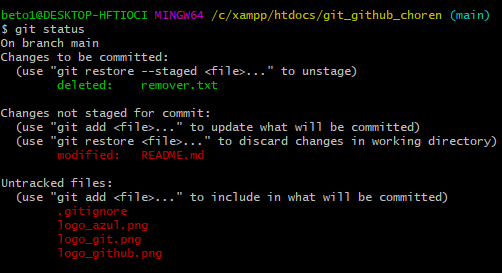
touch .gitignore

**.gitignore**

# Ignorando um único arquivo

logo\_branco.png

git status



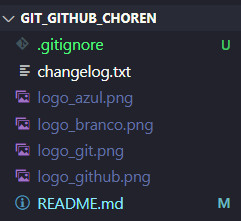
**.gitignore**

# Ignorando um único arquivo

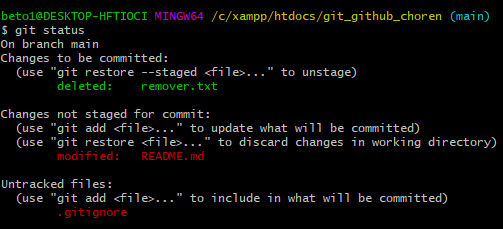
logo\_branco.png

# Ignorando arquivos de uma extensão pré-definida

\*.png

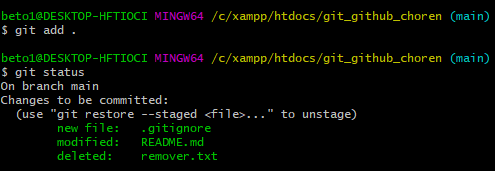


git status



git add .

git status



git commit

**.git\COMMIT\_EDITMSG**

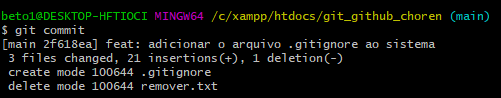
feat: adicionar o arquivo .gitignore ao sistema

arquivos modificados:

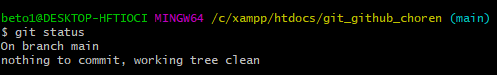
adicionado: .gitignore

modificado: README.md

removido: remover.txt



git status



# Aula 11 - Comandos git restore e git reset

Ambos comandos são utilizados para desfazer uma alteração feita no repositório.

- Crie um arquivo vazio chamado: logs\_06\_01\_2022.txt:

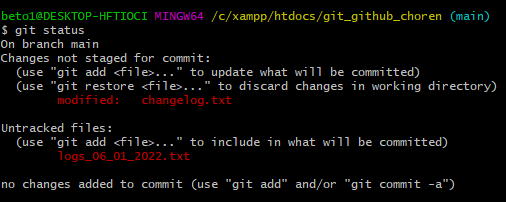
touch logs\_06\_01\_2022.txt

- Altere o conteúdo do arquivo changelog.txt

**changelog.txt**

06/01/2022 - Aprendendo a usar os comandos git restore e git reset.

git status

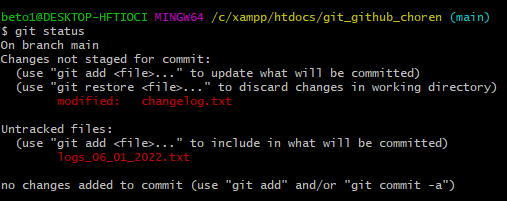


**changelog.txt**

05/01/2022 - Aprendendo a usar o comando git log.

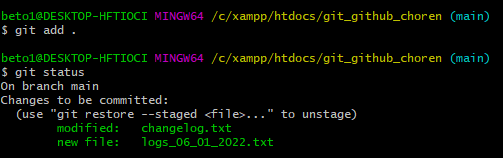
06/01/2022 - Aprendendo a usar os comandos git restore e git reset.

git status



git add .

git status



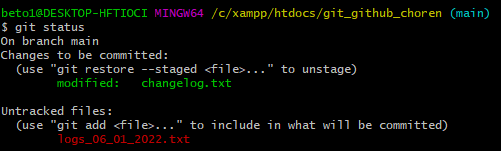
## Desfazendo alterações ainda não commitadas

- Não era para ter adicionado o arquivo logs\_06\_01\_2022.txt.

- Para desfazer as alterações que estão no staging area preparadas para serem commitadas, entre com:

git restore --staged logs\_06\_01\_2022.txt

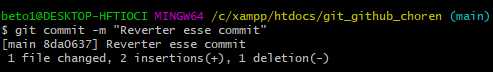
git status



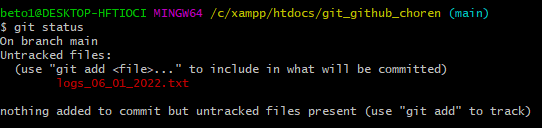
## Desfazendo alterações que já foram commitadas

### Realizando o commit

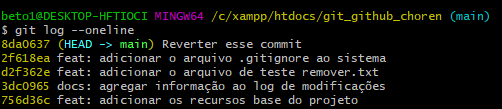
git commit -m "Reverter esse commit"



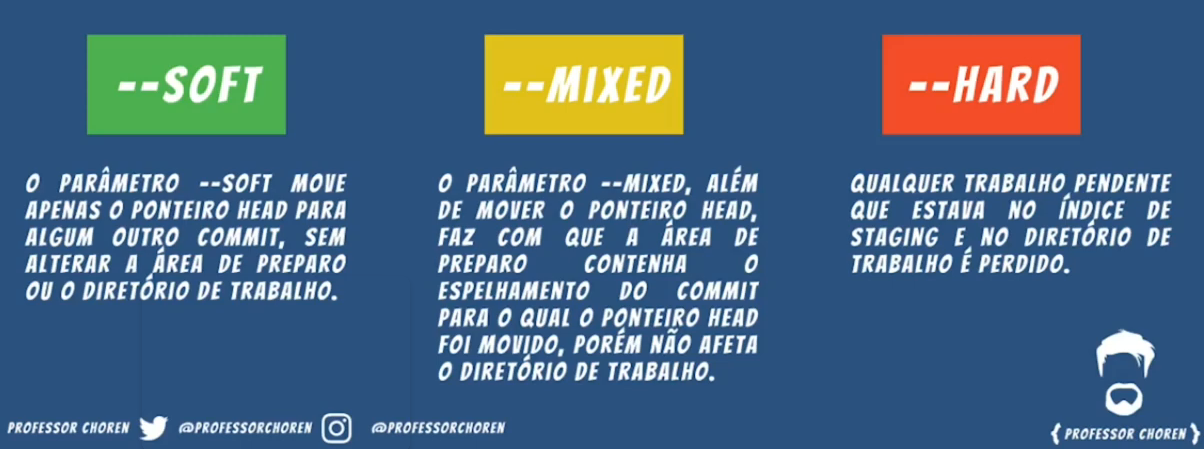
git status



git log --oneline



## Modos de reversão



### --soft

Desfaz o commit mas não descarta nada da área de preparo ou do diretório de trabalho.

### --mixed

É o valor padrão caso não se coloque nenhum desses três parâmetros.

Vai desfazer o commit e manter os arquivos no staging area (não desfaz o git add, desfaz apenas o git commit). Este comando não afeta o diretório de trabalho.

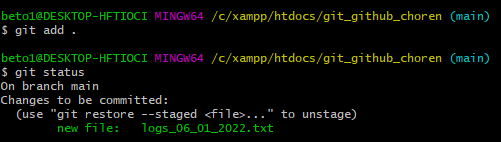
### --hard

Desfaz tudo. Qualquer trabalho que não tenha sido commitado é desfeito. Volta para como era no commit anterior ao qual eu estou desfazendo.

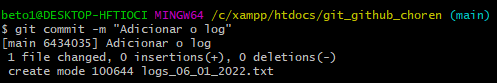
Exemplos:

git add .

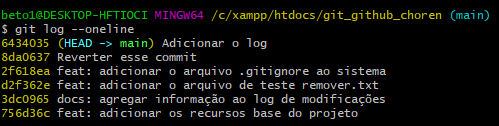
git status



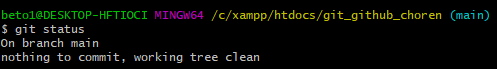
git commit -m "Adicionar o log"



git log --oneline



git status

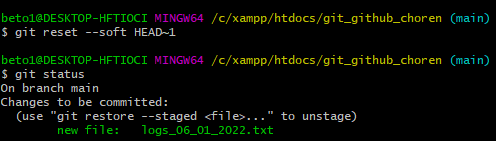


### Usando --soft

A partir do ponteiro voltar 1 commit:

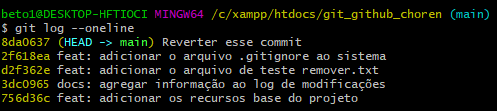
git reset --soft HEAD~1

git status



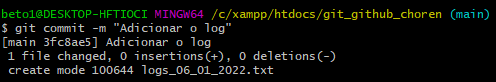
- Commit desfeito, porém o arquivo está monitorado (na staging area), pronto para ser commitado.

git log --oneline

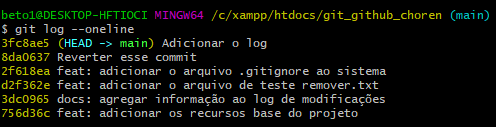


Refazendo o commit:

git commit -m "Adicionar o log"



git log --oneline

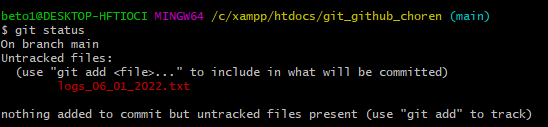


### Usando --mixed

git reset --mixed HEAD~1

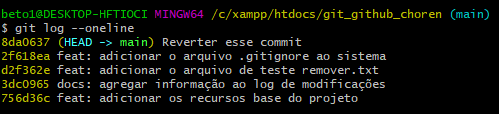


git status



- Agora o arquivo está não monitorado (não se encontra mais na staging area).

git log --oneline

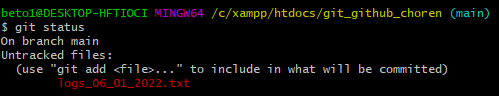


### Usando --hard

git reset --hard HEAD~1



git status

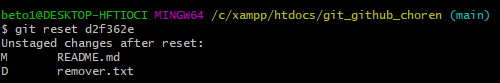


git log --oneline

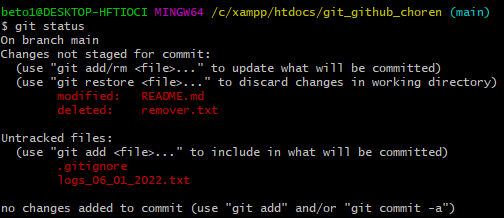


## Revertendo o commit a partir de um ponto específico

git reset d2f362e

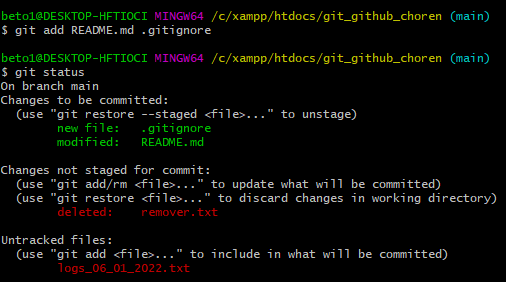


git status



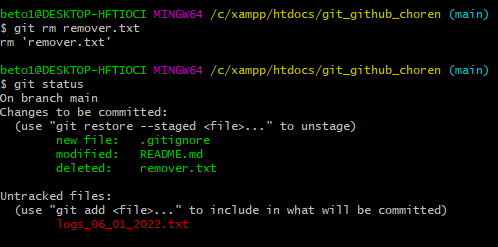
git add README.md .gitignore

git status

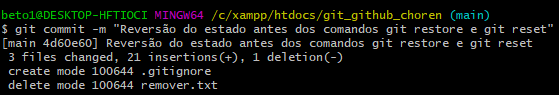


git rm remover.txt

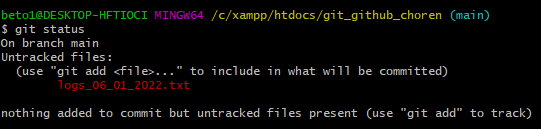
git status



git commit -m "Reversão do estado antes dos comandos git restore e git reset"



git status



# Aula 12 - Comando git checkout

O comando git checkout permite alternar entre as branches.

Uma branch é um ramo dentro do repositório.



- O que acontece nos ramos secundários não afeta o ramo principal.

git branch



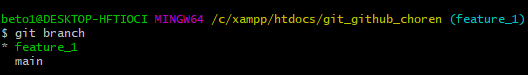
## Criando uma nova branch

git checkout -b feature\_1

-b → cria a branch, caso ela não exista.



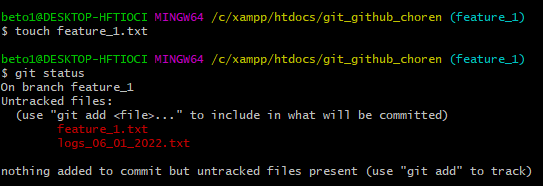
git branch



- A branch foi criada e HEAD aponta para ela.

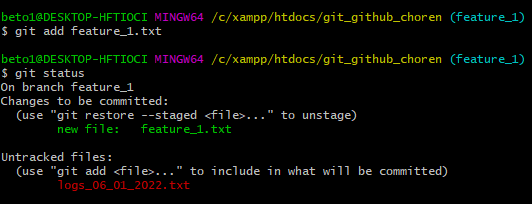
touch feature\_1.txt

git status



git add feature\_1.txt

git status



git stash save "Trabalhando na feature 1"



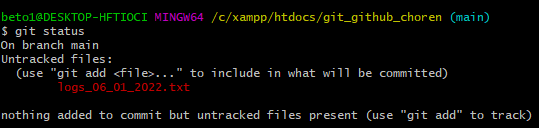
- Dessa forma, é salvo o estado com um nome.

### Voltando para a branch main

git checkout main



git status



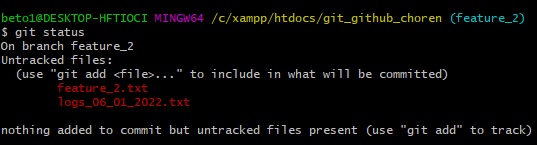
- Perceba que aquela pendência não veio junto.

## Criando uma nova branch

git checkout -b feature\_2

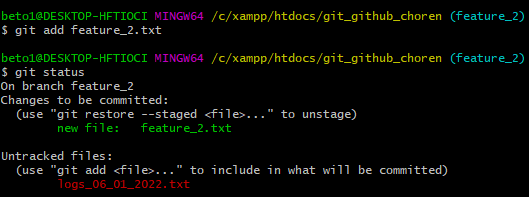


git status



git add feature\_2.txt

git status



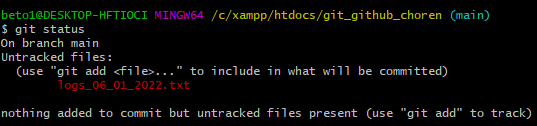
git stash save "Trabalhando na feature 2"



### Voltando para a branch main

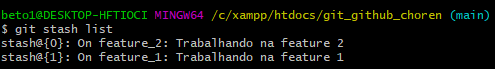
git checkout main

git status



### Exibindo quais são as pendências

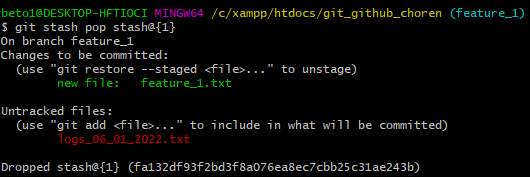
git stash list



## Desempilhando a feature\_1

git checkout feature\_1

git stash pop stash@{1}



git commit

**.git\COMMIT\_EDITMSG**

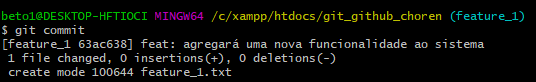
feat: agregará uma nova funcionalidade ao sistema

No ramo feature\_1

Mudanças submetidas:

feature\_1.txt

- Salve e feche o arquivo.



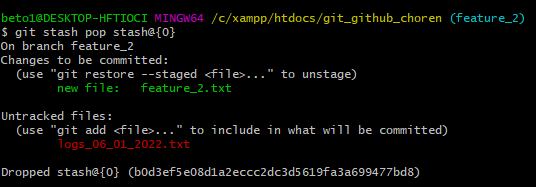
git status



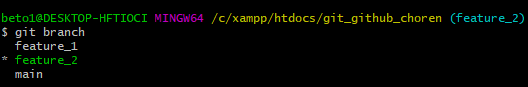
### Alternando para a branch feature\_2

git checkout feature\_2

git stash pop stash@{0}



git branch



git commit

**.git\COMMIT\_EDITMSG**

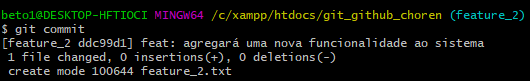
feat: agregará uma nova funcionalidade ao sistema

No ramo feature\_2

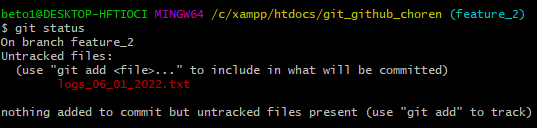
Mudanças submetidas:

feature\_2.txt

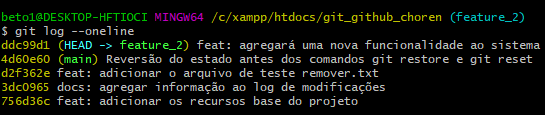
- Salve e feche o arquivo.



git status

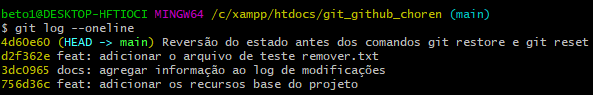


git log --oneline



git checkout main

git log --oneline



# Aula 13 - Comando git merge

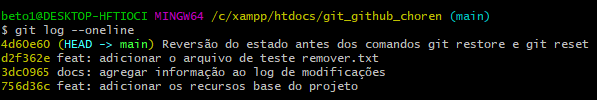
git merge serve para unir o conteúdo de uma branch com alguma outra branch.

## Unindo as branches feature\_1 e feature\_2 ao ramo principal (main)

git branch



git log --oneline



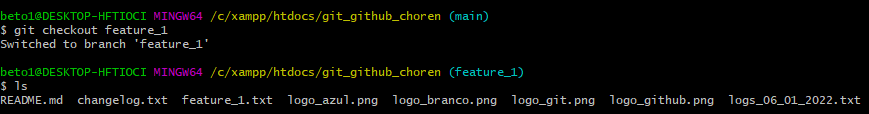
ls



- Os arquivos criados nas branches feature\_1 e feature\_2 não existem na branch main.

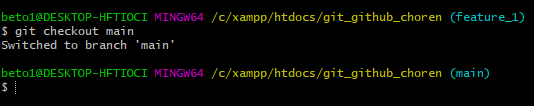
git checkout feature\_1

ls



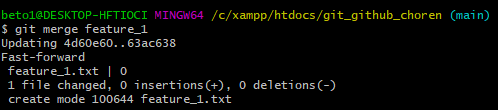
### Voltando para a branch main

git checkout main



### Unindo a branch feature\_1 ao ramo principal (branch main)

git merge feature\_1

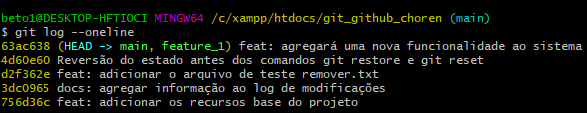


ls



Agora temos o arquivo feature\_1.txt na branch main.

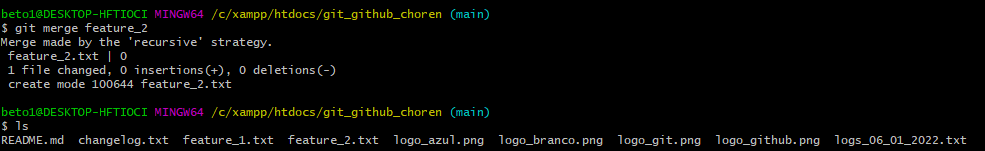
git log --oneline



### Unindo a branch feature\_2 ao ramo principal (branch main)

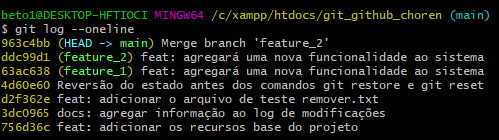
git merge feature\_2

ls

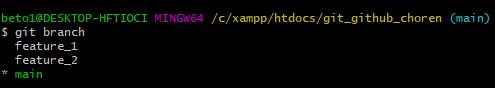


Agora temos o arquivo feature\_2.txt na branch main.

git log --oneline

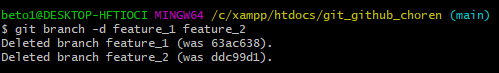


git branch



## Deletando as branches feature\_1 e feature\_2

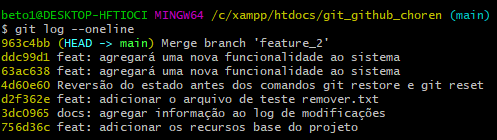
git branch -d feature\_1 feature\_2



git branch



git log --oneline



- Importante: Para evitar problemas, recomenda-se que após fazer o merge do branch main com o branch criado, o branch criado seja sempre deletado.

# Aula 14 - Comando git tag

Tags são forma de definir marcadores no nosso repositório onde a versão do projeto pode ser utilizável.

## Criando uma tag

git tag -a v1.0.0-beta -m "Versão 1.0.0 beta 1"

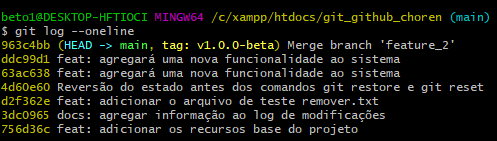
## Exibindo as tags criadas

git tag



Importante: a tag é criada a partir do commit que está na cabeça (HEAD) do repositório.

git log --oneline



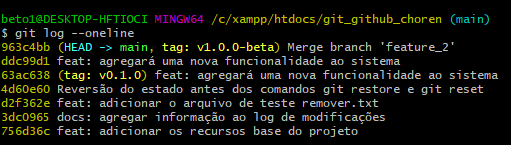
## Adicionando uma tag para uma determinada funcionalidade

Funcionalidade 1 → hash resumido 63ac638

Funcionalidade 2 → hash resumido ddc99d1

git tag -a v0.1.0 -m "Funcionalidade 1" 63ac638

git log --oneline



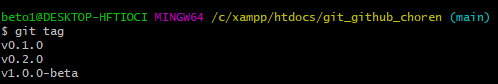
git tag -a v0.2.0 -m "Funcionalidade 2" ddc99d1

git log --oneline



### Exibindo as tags criadas

git tag



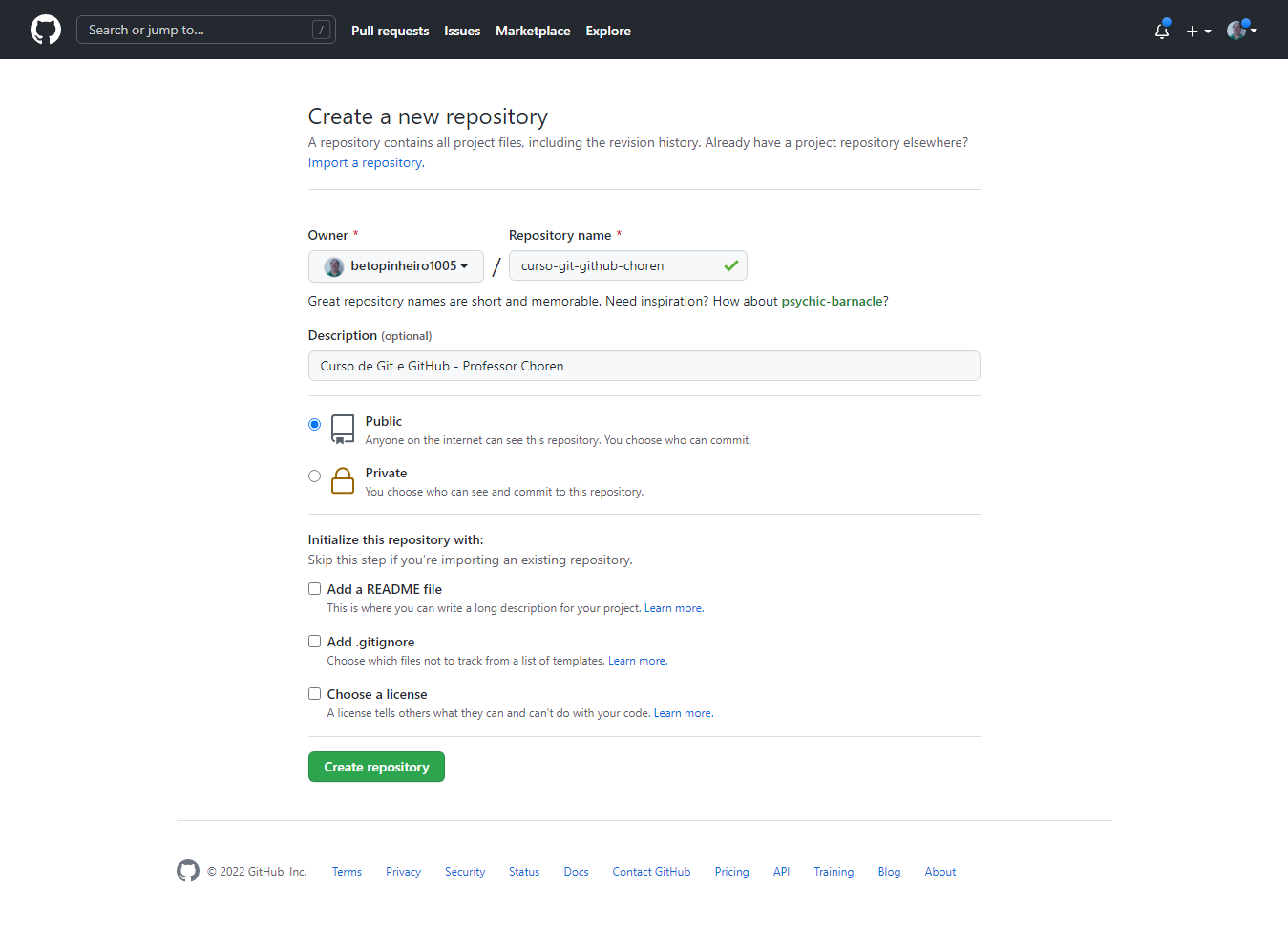
# Aula 15 - Conhecendo o GitHub

## Ferramentas de versionamento de código

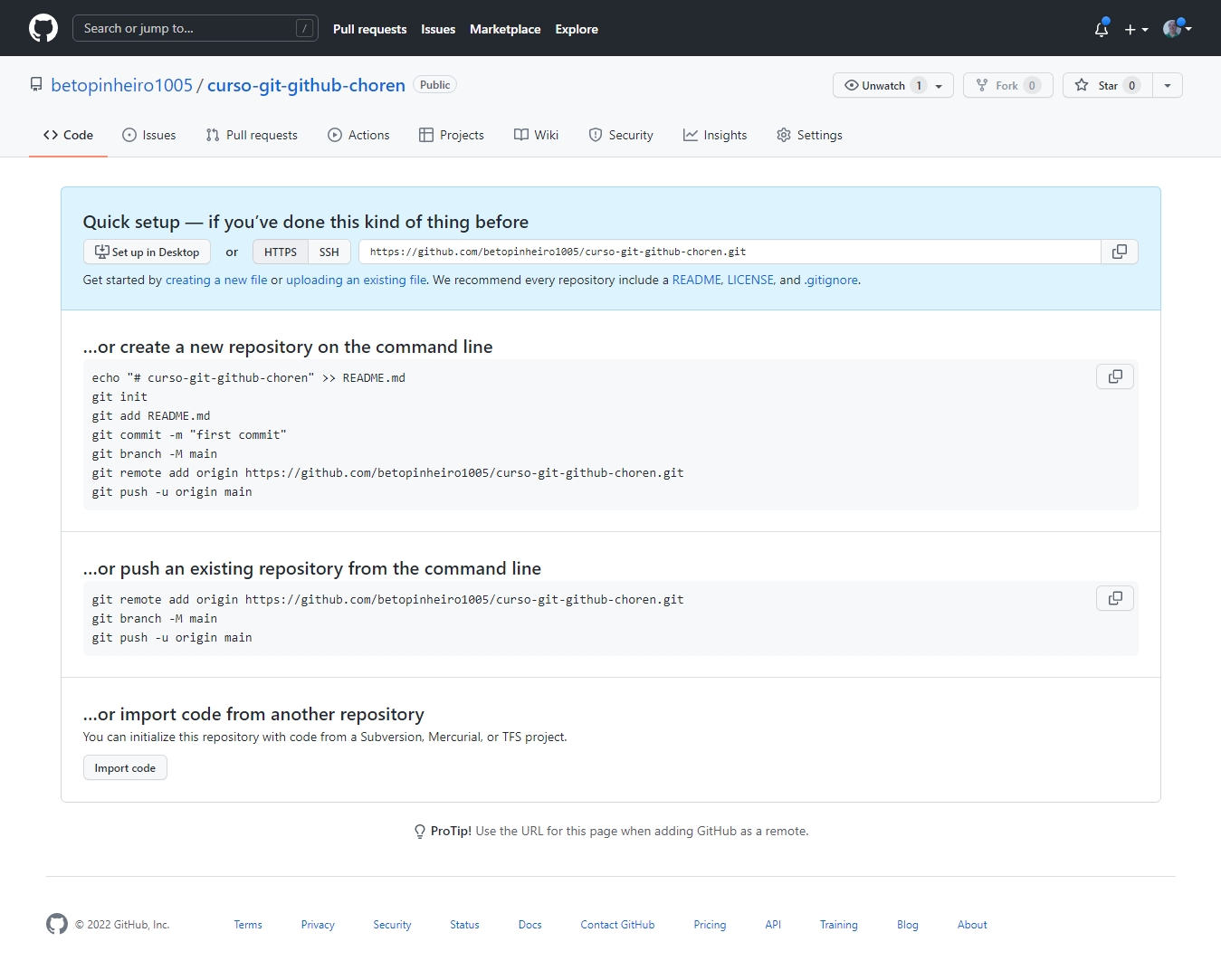
1. GitHub
2. GitLab
3. Bitbucket

- Crie uma conta no GitHub (Sign up) e logue nela (Sign in).

## Criando um novo repositório



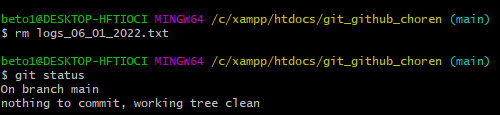
- Clique no botão "Create repository"



- Inicialmente, remova o arquivo logs\_06\_01\_2022.txt:

rm logs\_06\_01\_2022.txt

git status



- Para enviar para o GitHub não pode haver nenhuma pendência, portanto antes sempre confira com o comando git status

## Adicionando o endereço remoto

git remote add origin https://github.com/betopinheiro1005/curso-git-github-choren.git

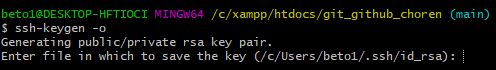


git remote

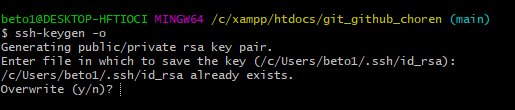


## Gerando uma chave SSH

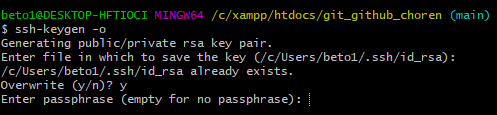
ssh-keygen -o



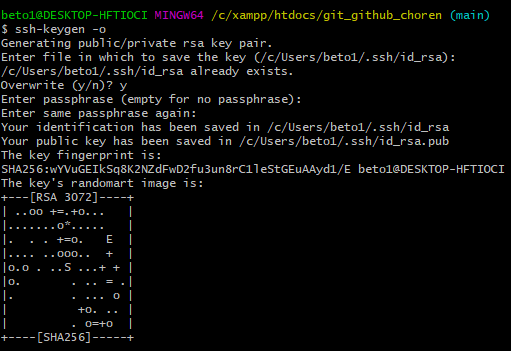
- Pressione a tecla <Enter>



- Digite y e pressione a tecla <Enter>



Essa é a senha que será usada para enviar os arquivos.

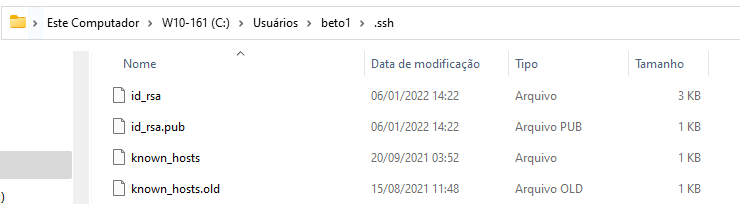


Your identification has been saved in /c/Users/beto1/.ssh/id\_rsa

Your public key has been saved in /c/Users/beto1/.ssh/id\_rsa.pub

A chave pública e privada foram criadas na pasta "/c/Users/beto1/.ssh/"

Vá até essa pasta e abra o arquivo: id\_rsa.pub

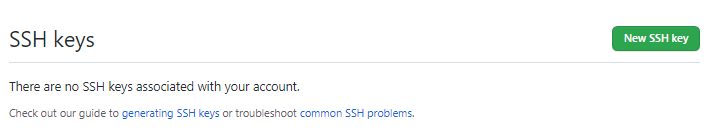


- Copie todo o conteúdo dessa chave.

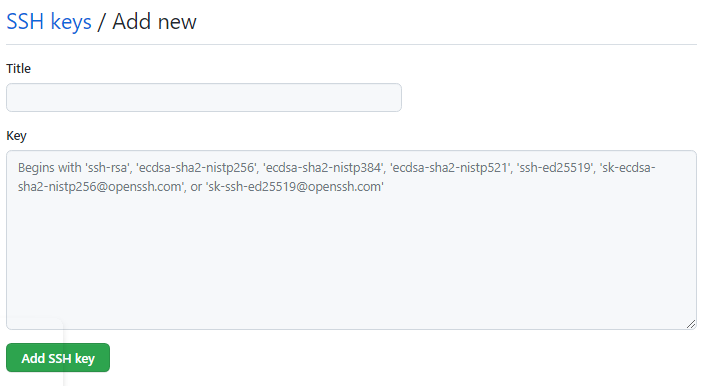
- No GitHub, clique na sua conta em "Settings"

- Clique na opção "SSH and GPC keys".

- Delete a chave existente.

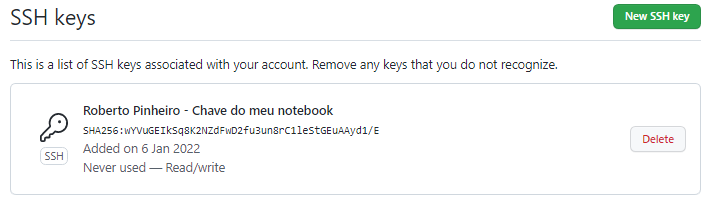


- Clique no botão "New SSH Key"



- Cole a chave copiada anteriormente e dê um título para essa chave.

- Em seguida clique no botão Add SSH key



git branch



Atenção: O ramo principal deve se chamar main, não pode se chamar master.

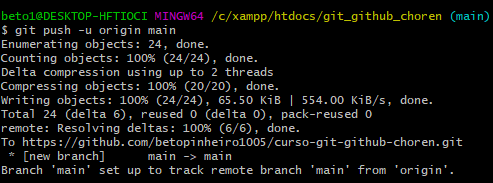
- Caso na sua máquina apareça master, entre com o seguinte comando:

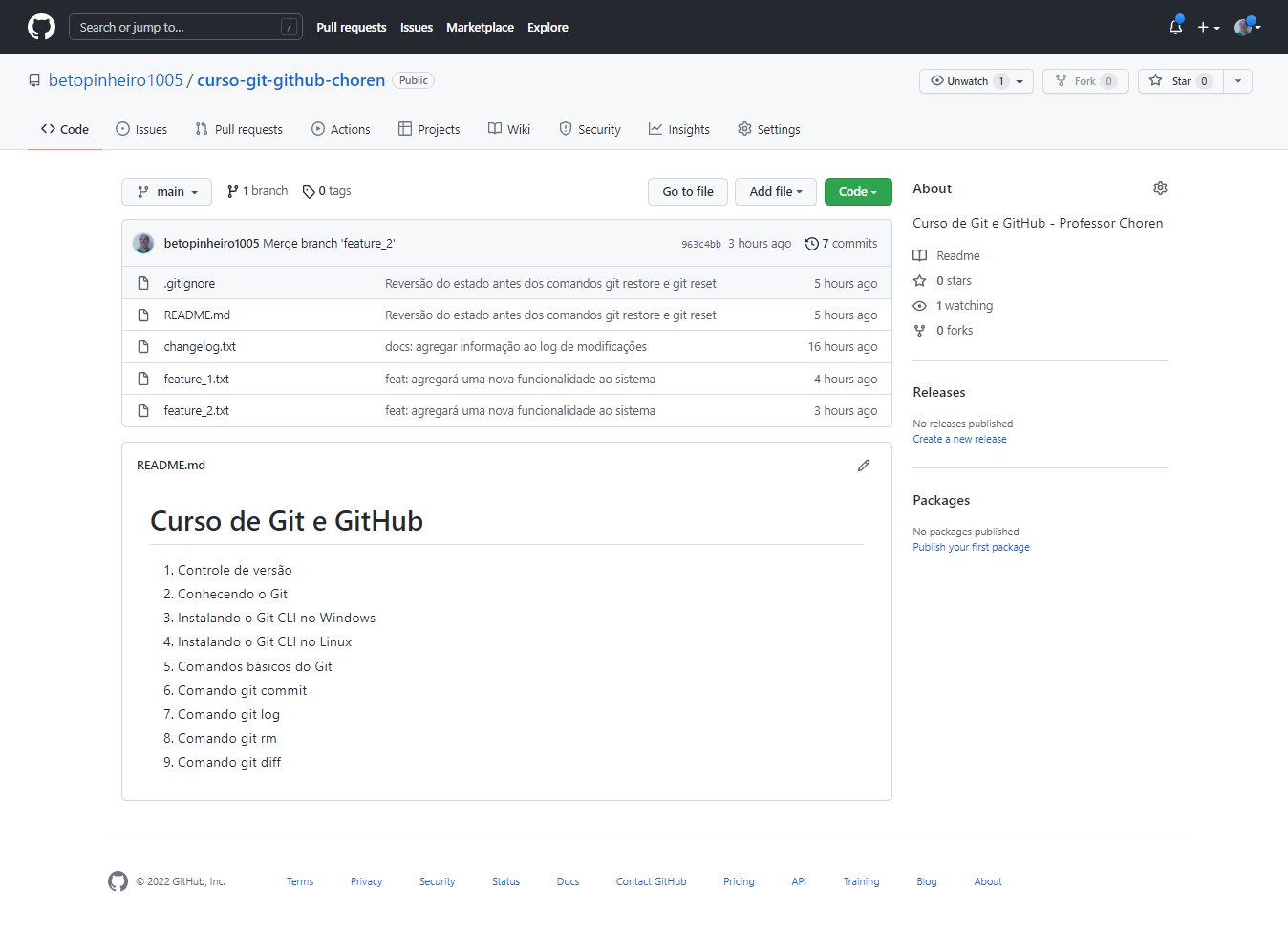
git branch -M main

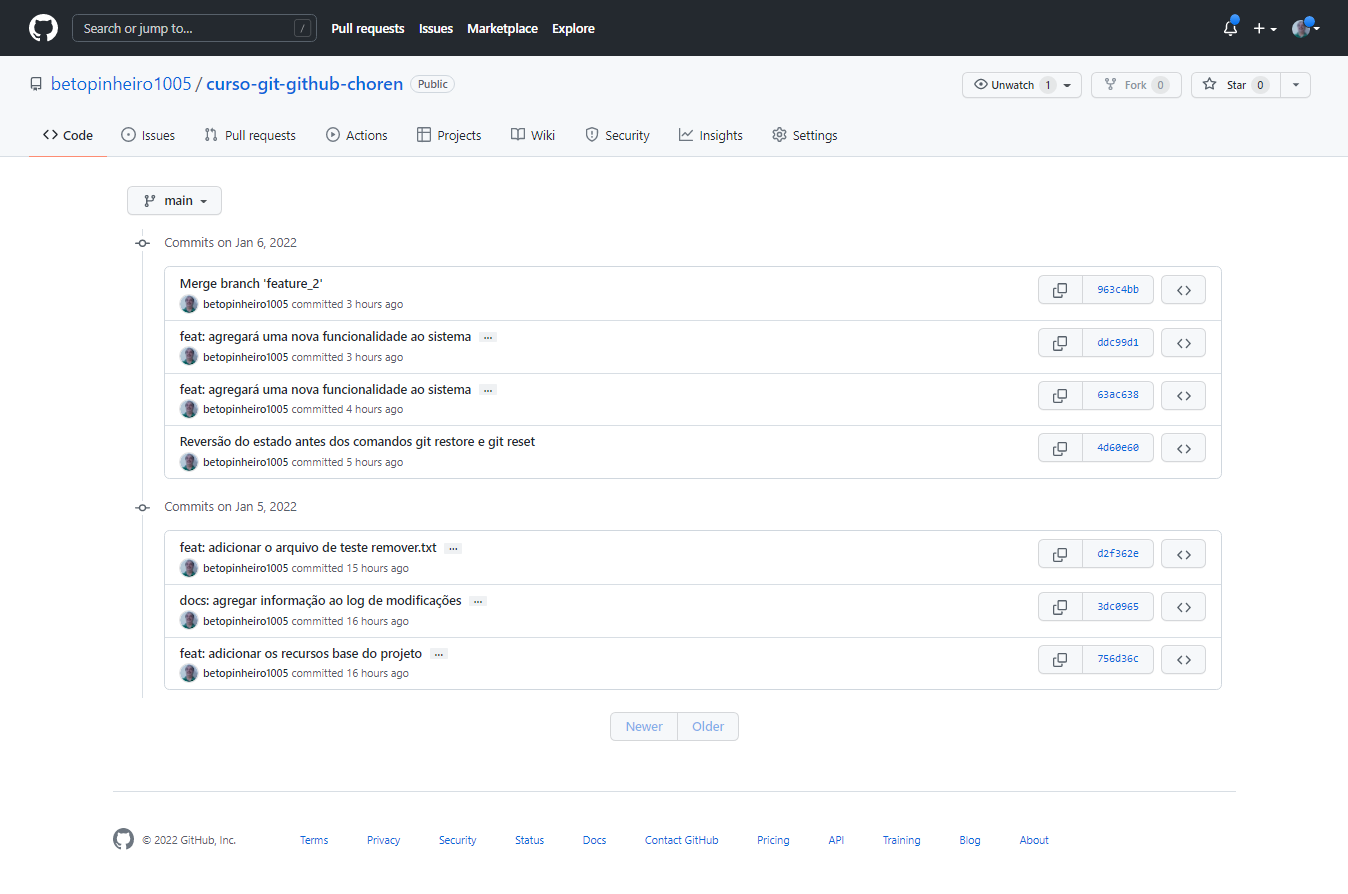
## Enviando os arquivos para o GitHub

- Na primeira vez que enviar arquivos, entre com o comando:

git push -u origin main







- Na próxima aula iremos atualizar o arquivo README.md

# Aula 16 - Comandos git clone e git push

- Apague a pasta que contém o repositório local (pasta C:\xampp\htdocs\git\_github\_choren).

## Clonando o repositório remoto

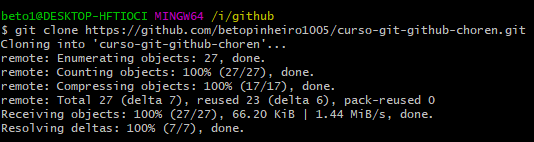
- No GitHub, acesse o repositório remoto.

- Clique no botão "Code" e copie o endereço SSH:

https://github.com/betopinheiro1005/curso-git-github-choren.git

- Na pasta que ficará o repositório local (I:\github), entre com o comando:

git clone https://github.com/betopinheiro1005/curso-git-github-choren.git



- Será criada a pasta curso-git-github-choren.

cd curso-git-github-choren

ls



## Alterando o arquivo README.md

**README.md**

# Curso de Git e GitHub

## Professor Choren

### Lista de aulas - [Vídeos do curso](https://www.youtube.com/watch?v=beMnH51P-T4&list=PLmMxPWmzYRGcTabffOwHBORBjtKa2wCXS)

Aula 01 - Controle de versão

Aula 02 - Conhecendo o Git

Aula 03 - Instalando o Git CLI no Windows

Aula 04 - Instalando o Git CLI no Linux

Aula 05 - Comandos básicos do Git

Aula 06 - Comando git commit

Aula 07 - Comando git log

Aula 08 - Comando git rm

Aula 09 - Comando git diff

Aula 10 - Comando git restore e git reset

Aula 11 - Arquivo .gitignore

Aula 12 - Comando git checkout

Aula 13 - Comando git merge

Aula 14 - Comando git tag

Aula 15 - Conhecendo o GitHub e utilizando o comando git push

Aula 16 - Comando git clone e git push

Aula 17 - Comando git pull

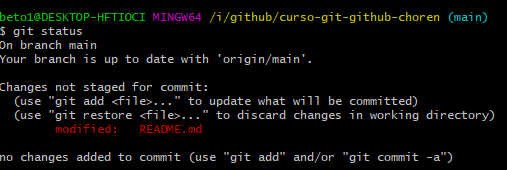
Aula 18 - Publicando tags no GitHub

Aula 19 - Gestão de conflitos no merge

Aula 20 - Contribuição com repositório no GitHub

## Enviando as alterações para o GitHub

git status



git commit -a

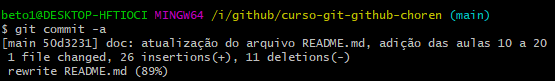
**.git\COMMIT\_EDITMSG**

doc: atualização do arquivo README.md, adição das aulas 10 a 20

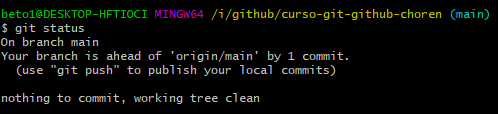
Mudanças submetidas:

README.md

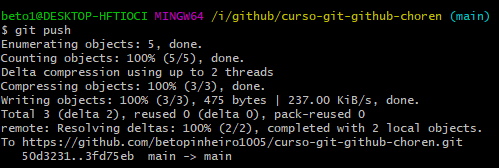
- Salve e feche o arquivo



git status

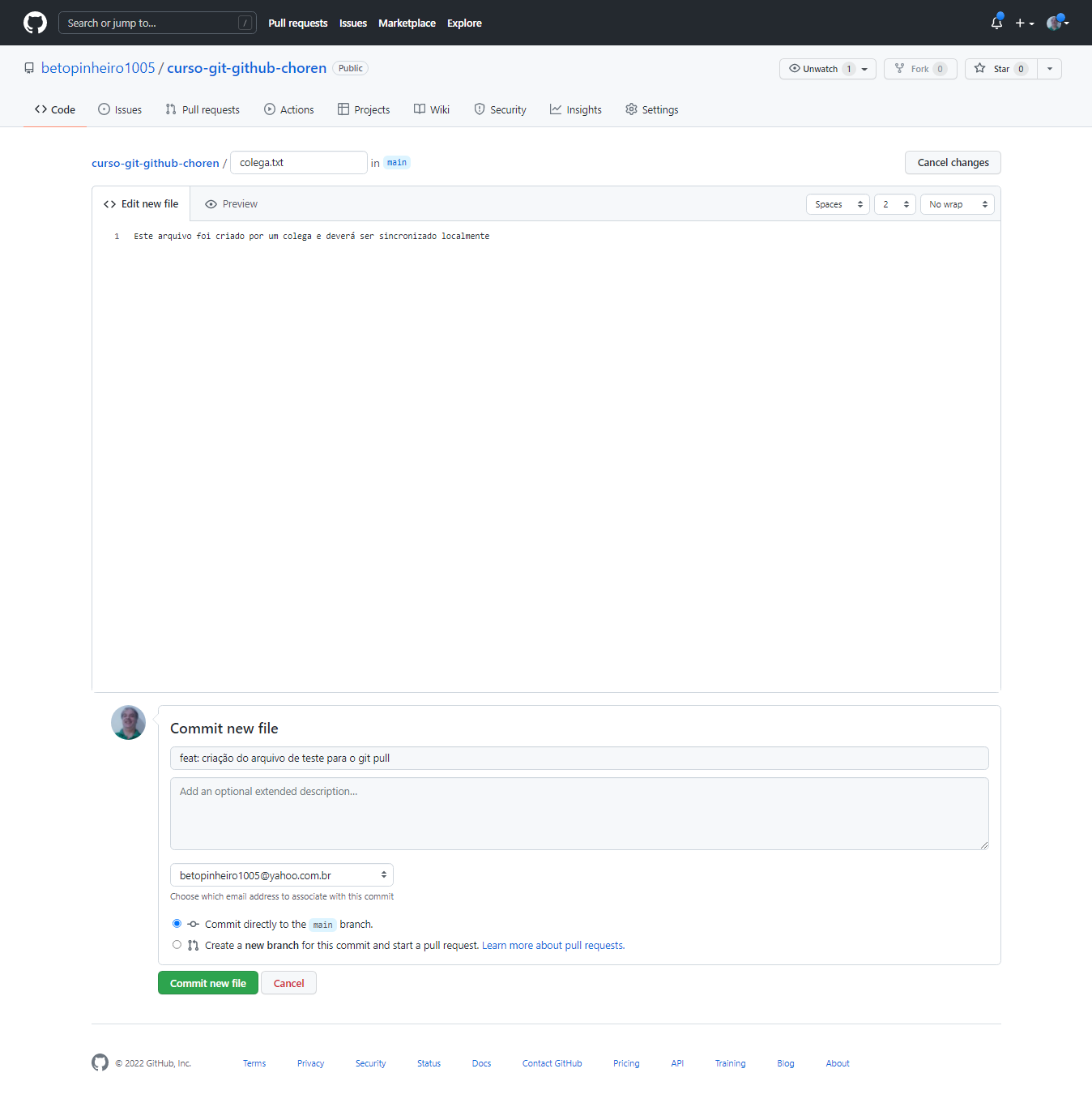


git push

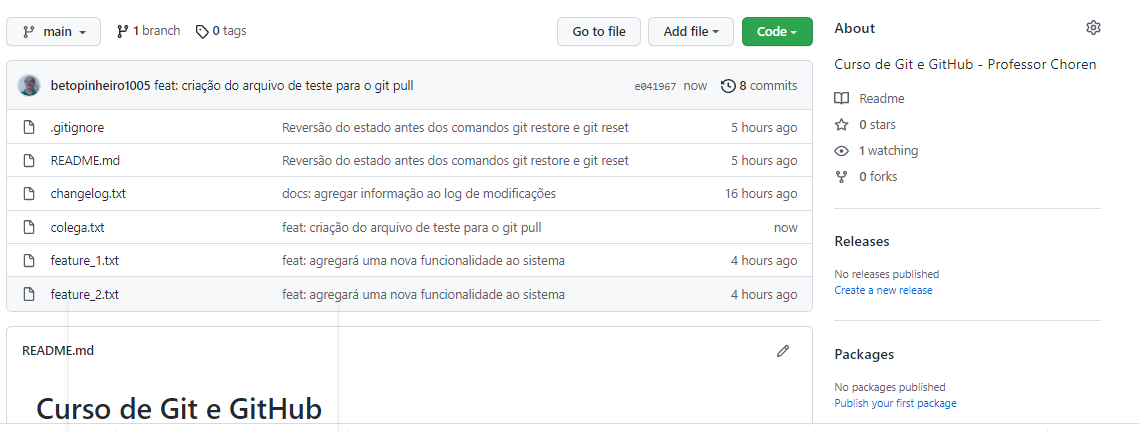


# Aula 17 - Comando git pull

- No GitHub clique no botão "Add File → Create new file"



- Faça as anotações e em seguida clique no botão "Commit new file"

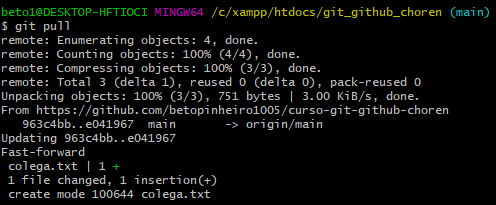


## Baixando a alteração para o repositório local

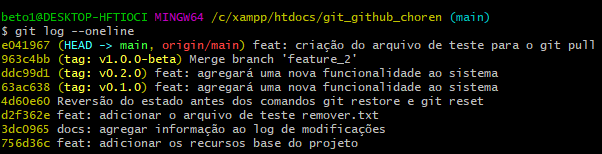
git pull

ou:

git pull origin main



git log --oneline



cat colega.txt

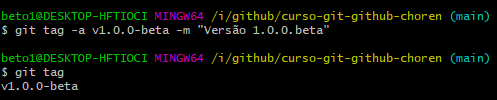


# Aula 18 - Publicando tags no GitHub

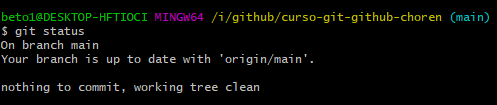
## Criando uma tag no repositório local

git tag -a v1.0.0-beta -m "Versão 1.0.0.beta"

git tag

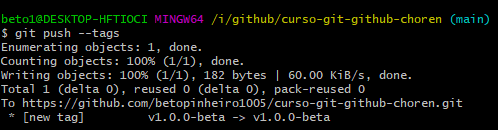


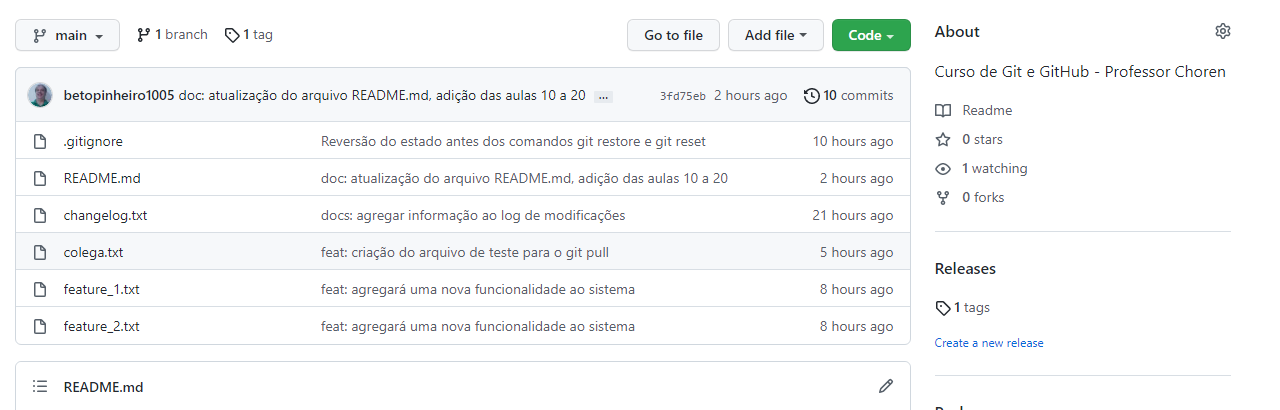
git status

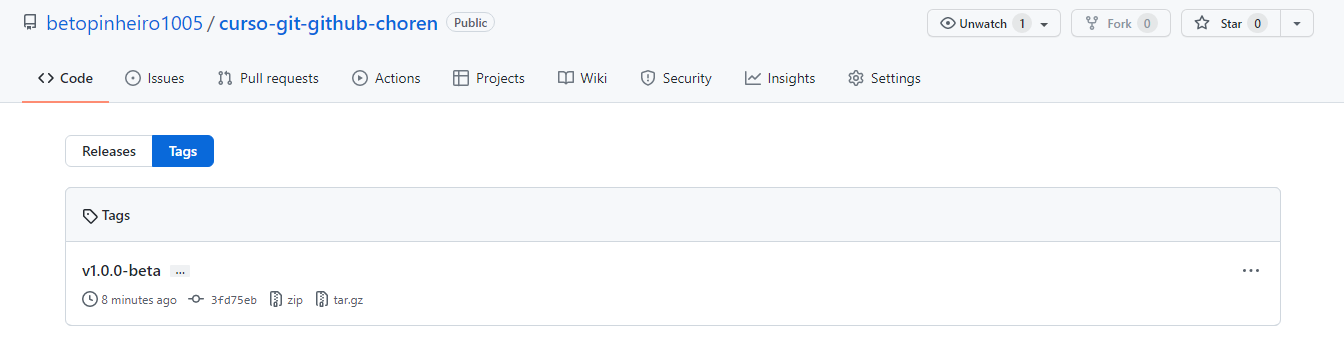


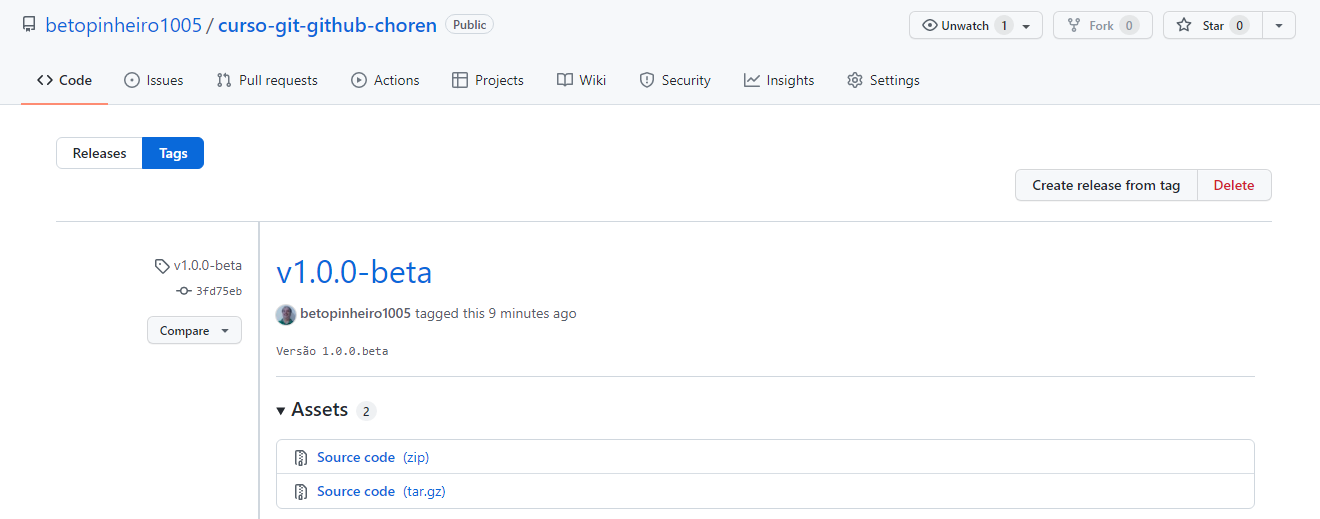
## Enviando a tag para o repositório remoto

git push --tags









# Aula 19 - Gestão de conflitos no merge

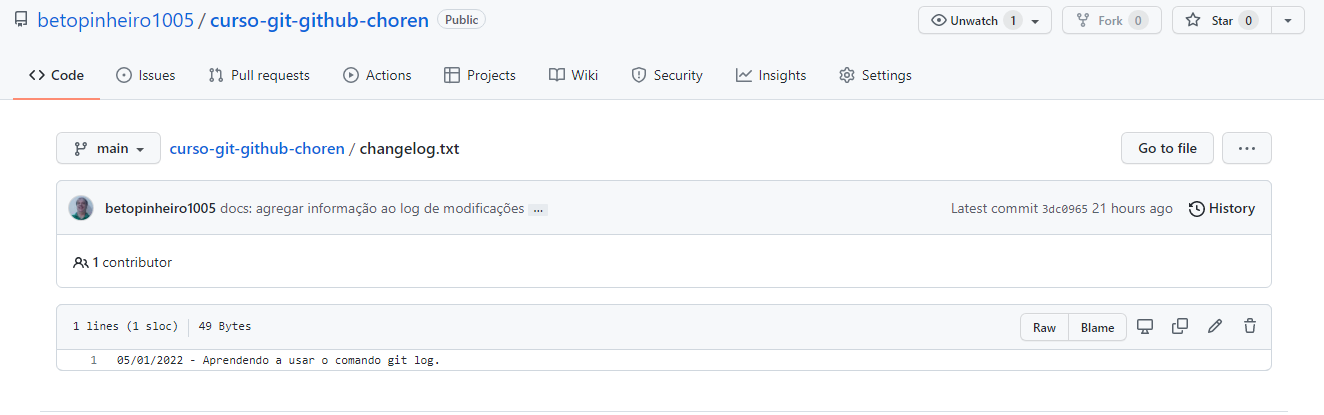
É muito comum, que em projetos grandes ou até mesmo pequenos, haja conflitos entre versões de códigos, principalmente quando trabalhamos em equipes.

Um conflito ocorre quando dois programadores estão trabalhando encima do mesmo arquivo e eles modificam as mesmas linhas, cada um em seu ambiente de trabalho. Será necessário informar qual é a versão que se quer manter.

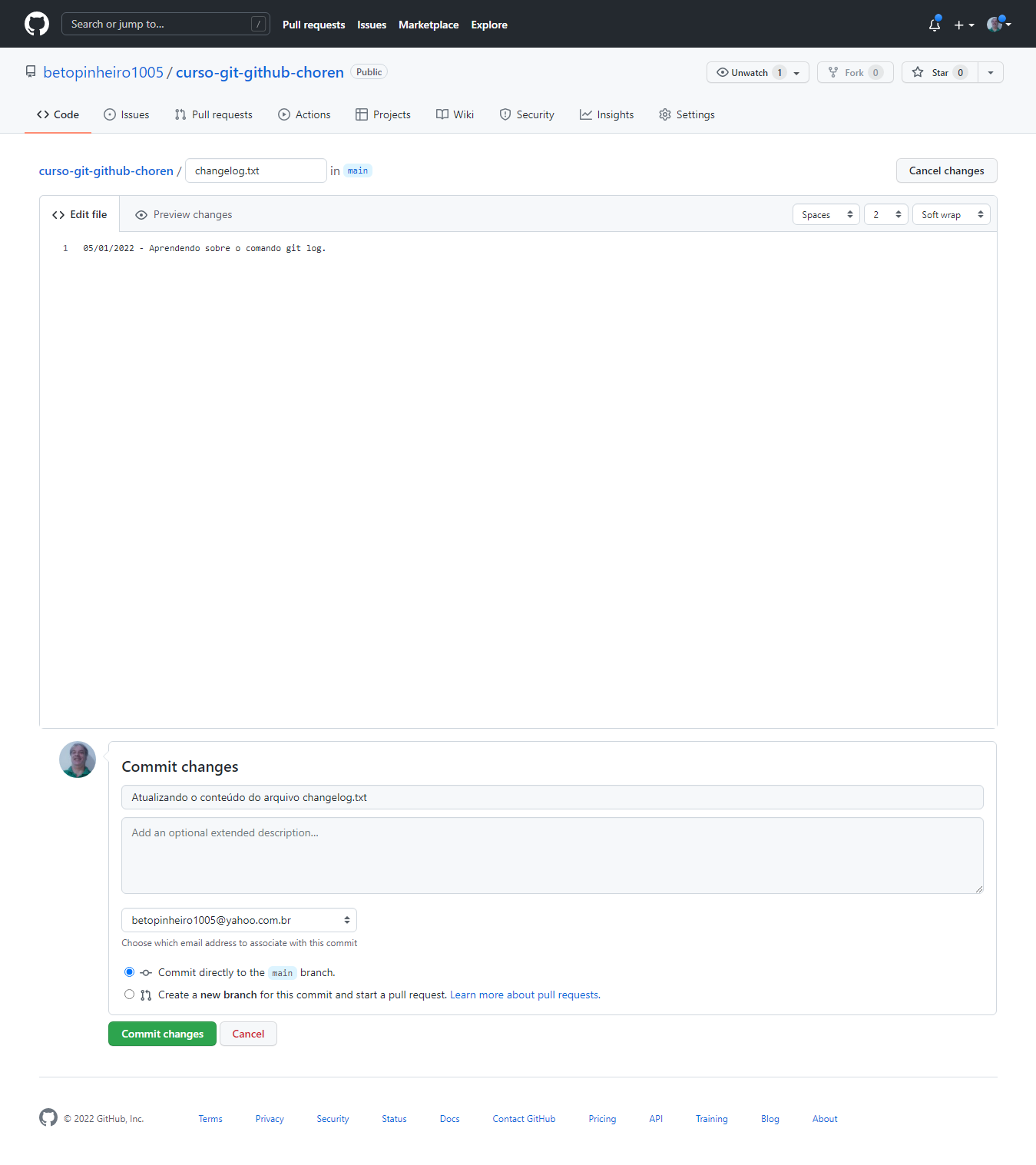
Não pode haver duas versões do mesmo código. Só podemos ter uma única fonte de dados, uma única fonte de código.

## Exemplo de conflito

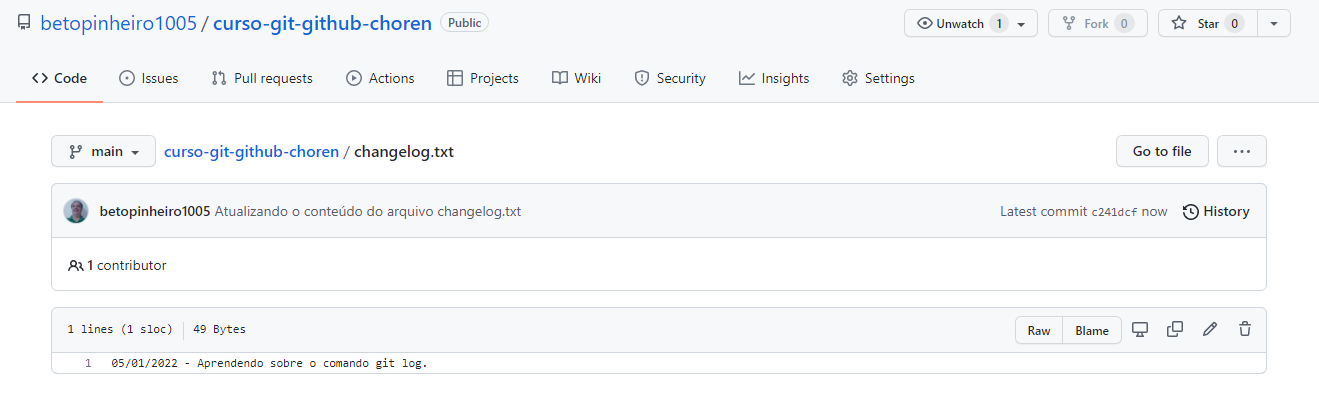
- No repositório remoto (servidor):



### Fazendo uma edição do arquivo changelog.txt diretamente pelo GitHub



- Clique no botão "Commit changes"



- No repositório local:

cat changelog.txt



- Edite o arquivo changelog.txt:

**changelog.txt**

05/01/2022 - A

06/01/2022 - Aprendendo sobre a gestão de conflitos

git commit -a

**.git\COMMIT\_EDITMSG**

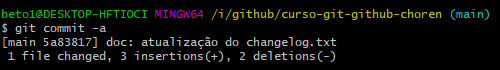
doc: atualização do changelog.txt

Adição da linha de log sobre gestão de conflitos

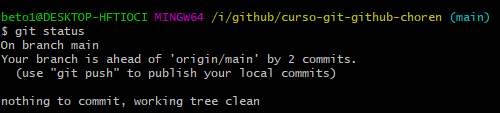
Mudança submetida:

changelog.txt

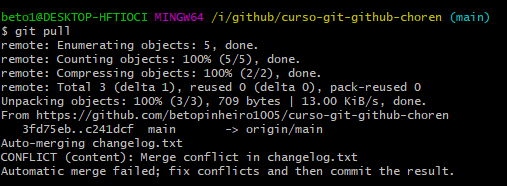
- Salve e feche o arquivo



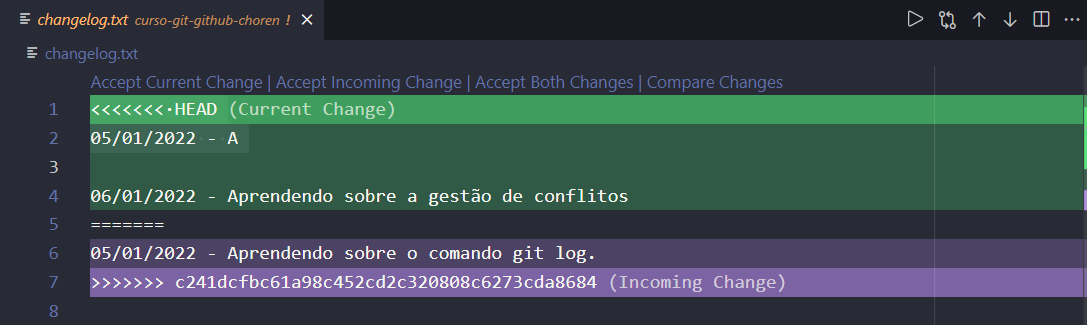
git status



git pull



No arquivo changelog.txt:



O conflito é exibido no arquivo.

São exibidas as seguintes opções:

* Accept Current Change (Aceite a mudança atual)
* Accept Incoming Change (Aceite a mudança recebida)
* Accept Both Changes (Aceite ambas as mudanças)
* Compare Changes (Compare mudanças)

- Se for clicada na opção Accept Current Change, teremos como resultado:

05/01/2022 - A

06/01/2022 - Aprendendo sobre a gestão de conflitos

- Se for clicada na opção Accept Incoming Change, teremos como resultado:

05/01/2022 - Aprendendo sobre o comando git log.

- Se for clicada na opção Accept Both Changes, teremos como resultado:

05/01/2022 - A

06/01/2022 - Aprendendo sobre a gestão de conflitos

05/01/2022 - Aprendendo sobre o comando git log.

Clique na última opção:

**changelog.txt**

05/01/2022 - A

06/01/2022 - Aprendendo sobre a gestão de conflitos

05/01/2022 - Aprendendo sobre o comando git log.

- Apague as duas primeiras linhas:

**changelog.txt**

06/01/2022 - Aprendendo sobre a gestão de conflitos

05/01/2022 - Aprendendo sobre o comando git log.

- Salve o arquivo

git commit -a

**.git\COMMIT\_EDITMSG**

Merge branch 'main' of https://github.com/betopinheiro1005/curso-git-github-choren

# Conflicts:

# changelog.txt

#

# It looks like you may be committing a merge.

# If this is not correct, please run

# git update-ref -d MERGE\_HEAD

# and try again.

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

#

# On branch main

# Your branch and 'origin/main' have diverged,

# and have 2 and 1 different commits each, respectively.

# (use "git pull" to merge the remote branch into yours)

#

# All conflicts fixed but you are still merging.

#

# Changes to be committed:

# modified: changelog.txt

#

- Altere para:

**.git\COMMIT\_EDITMSG**

fix: Merge branch 'main' of https://github.com/betopinheiro1005/curso-git-github-choren

Conflitos:

changelog.txt

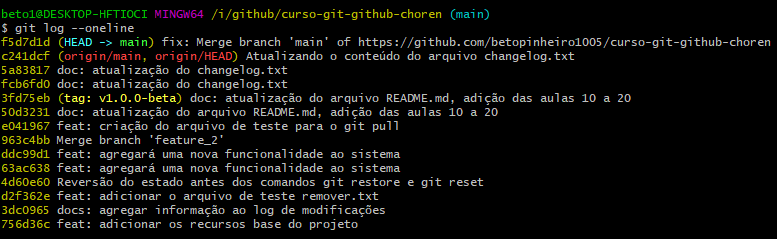
Mudança submetida:

changelog.txt

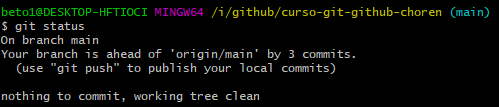
- Salve e feche o arquivo



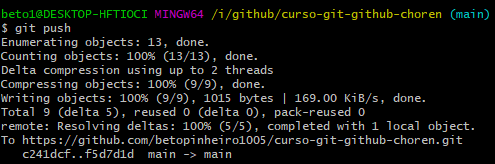
git log --oneline



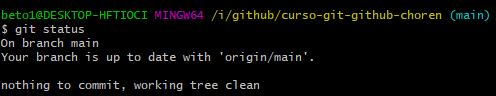
git status



git push

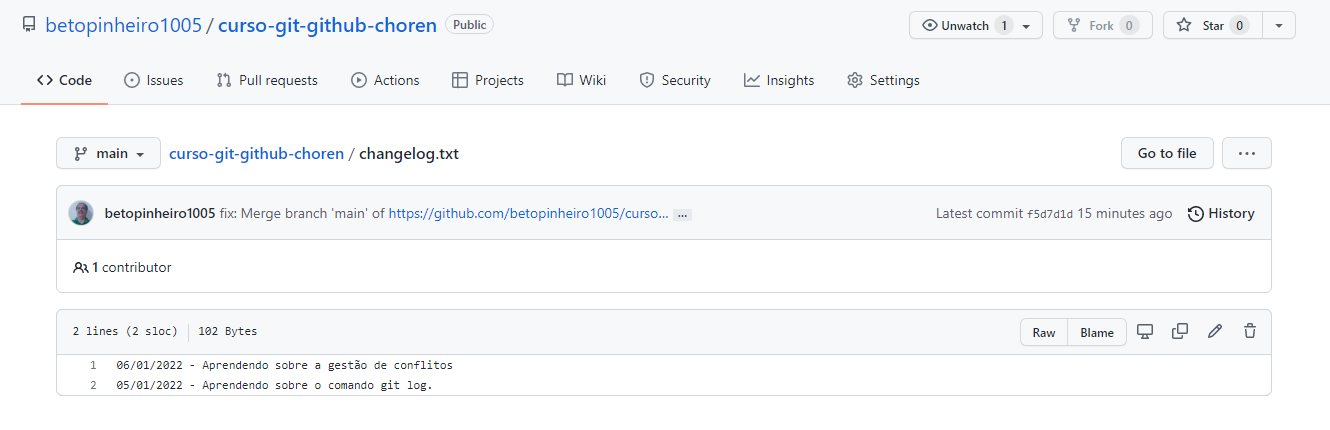


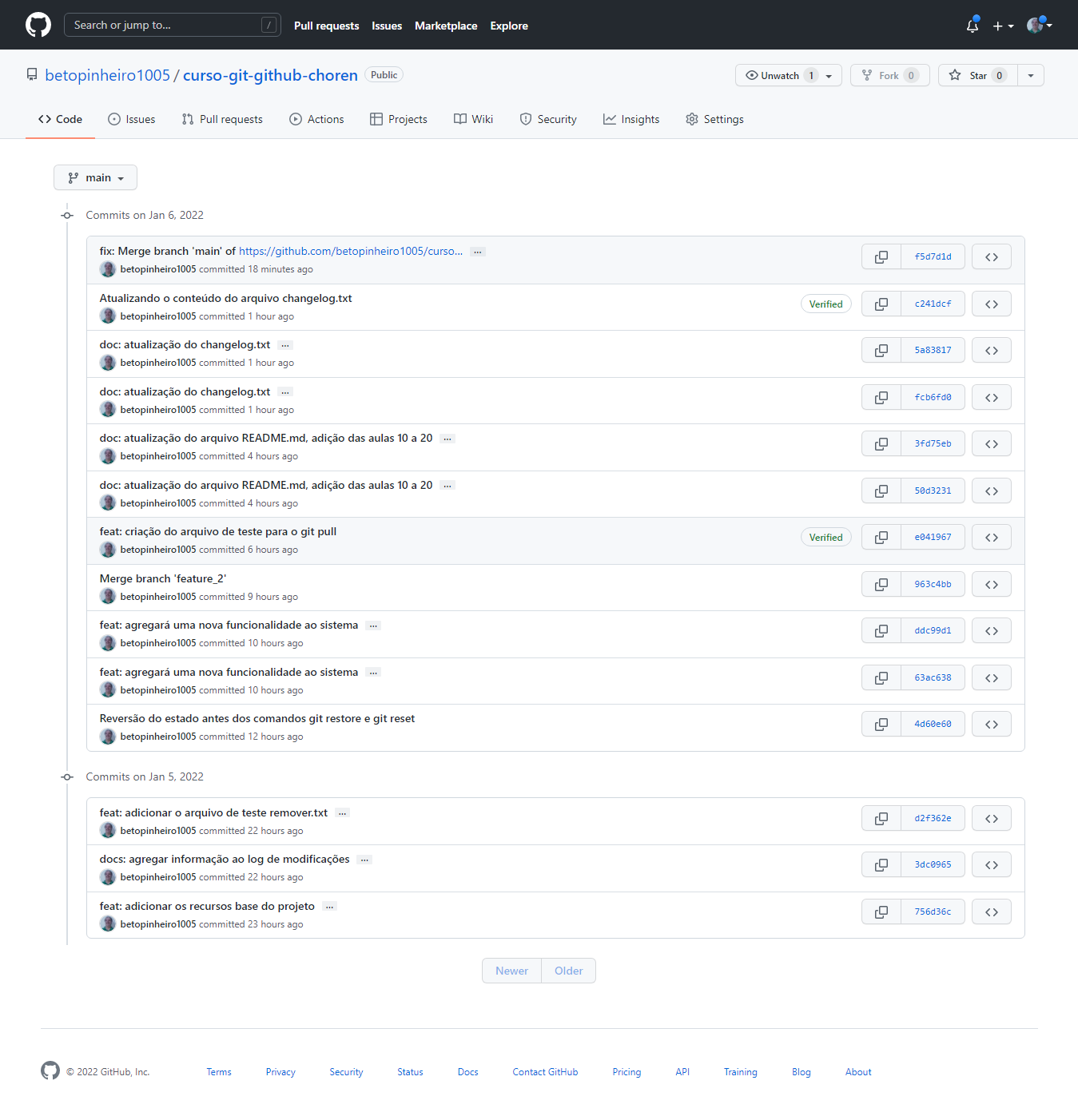
git status



- No GitHub:







# Aula 20 - Contribuição com repositório no GitHub

## Comando pull request

pull request é uma forma de enviar solicitações de alterações para o repositório cujo dono não é você, ou que você não faz parte da equipe de desenvolvimento.

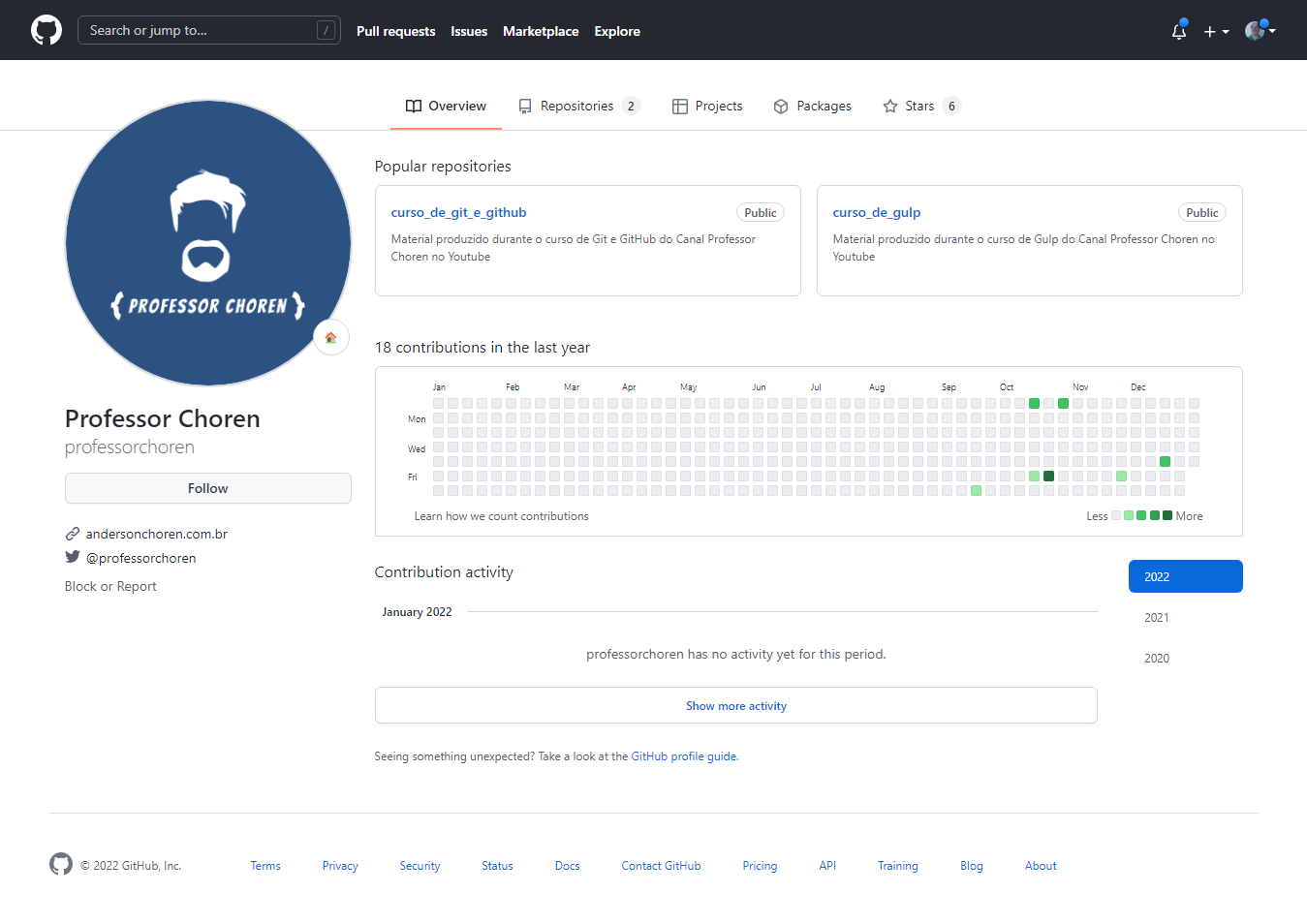
Na realidade significa solicitar um request a partir de um fork que você faz.

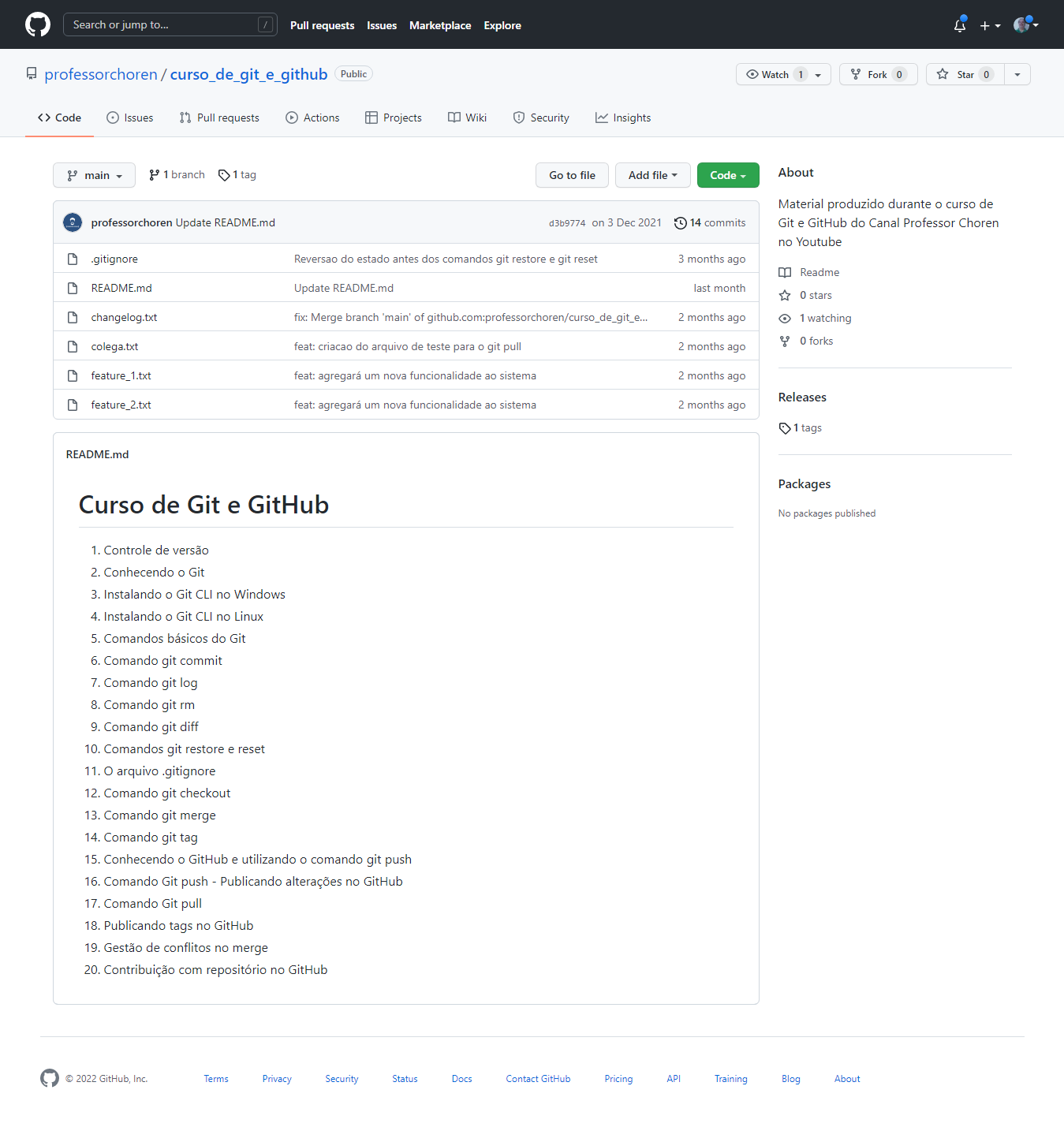
fork é uma cópia exata do projeto para sua conta no GitHub que possibilita que você faça melhorias.

Após isso, é solicita-se ao dono desse repositório que puxe essas informações do seu repositório para o repositório dele, aceitando suas modificações e fazendo esse merging dentro do projeto.

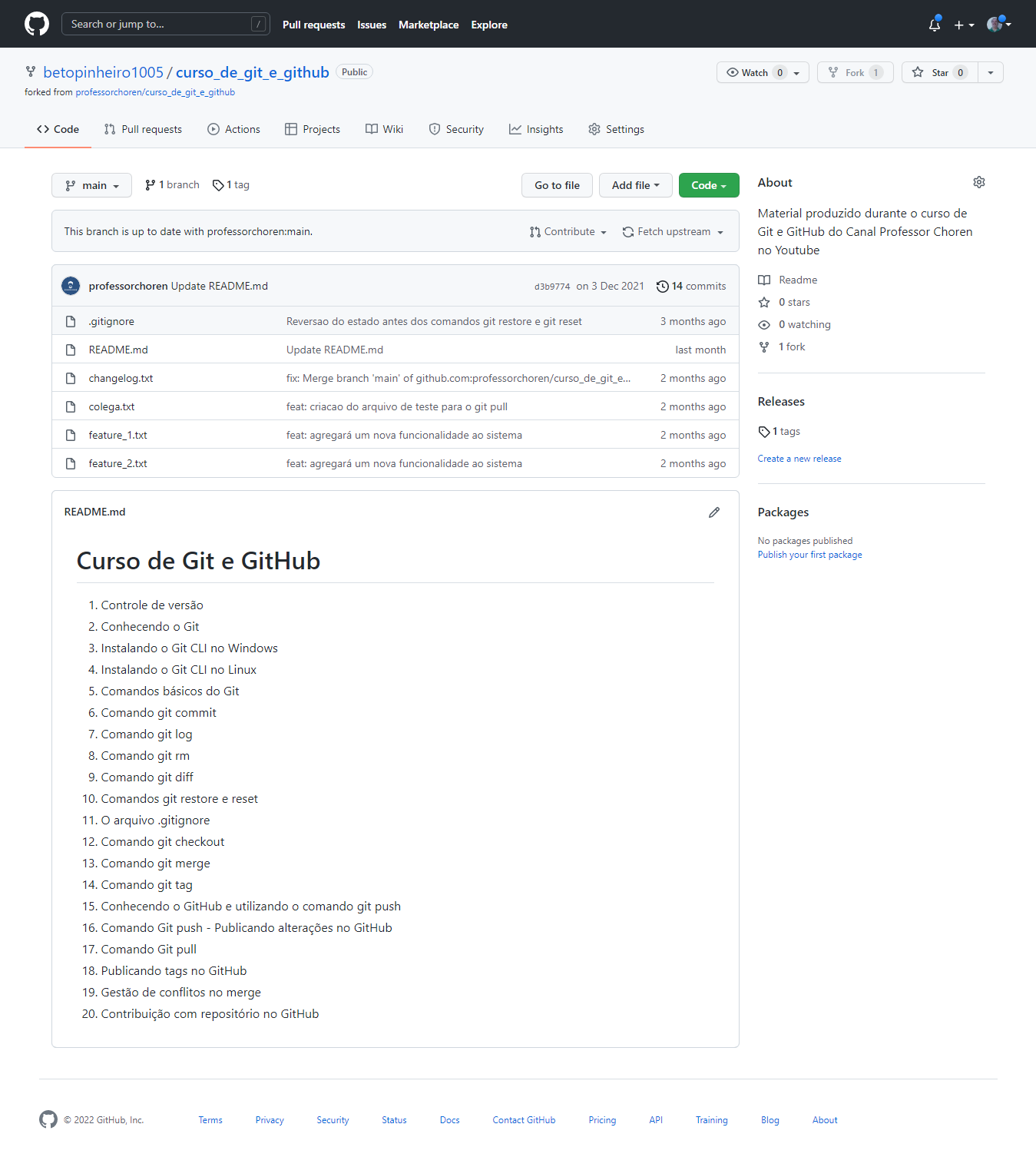
## Fazendo o fork do repositório

https://github.com/professorchoren

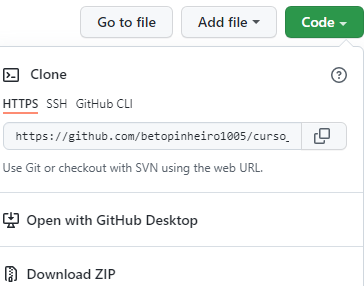




- Clique no botão "Fork"



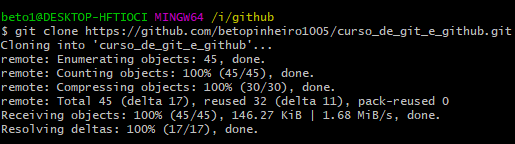
- Clique no botão "Code" e copie o endereço SSH:



https://github.com/betopinheiro1005/curso\_de\_git\_e\_github.git

## Clonando o repositório remoto para sua máquina

git clone https://github.com/betopinheiro1005/curso\_de\_git\_e\_github.git



cd curso\_de\_git\_e\_github



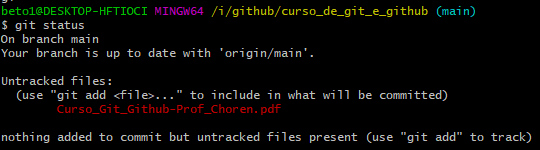
ls



- Faça alterações no projeto original (curso\_de\_git\_e\_github), que está em sua máquina, para acrescentar alguma melhoria, recurso ou funcionalidade que seja útil ao projeto.

OBS.: Como contribuição adicionei esta apostila (em formato PDF) chamada "Curso\_Git\_Github-Prof\_Choren.pdf" que elaborei e que contém o resumo do curso, com o código parcial de cada uma das aulas, além de anotações importantes visando facilitar os estudos deste curso.

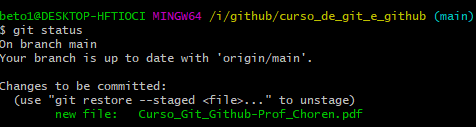
git status



git add .



git status



git commit

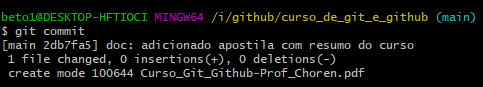
**.git\COMMIT\_EDITMSG**

doc: adicionado apostila com resumo do curso

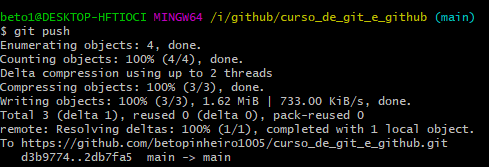
Mudança submetida:

Curso\_Git\_Github-Prof\_Choren.pdf

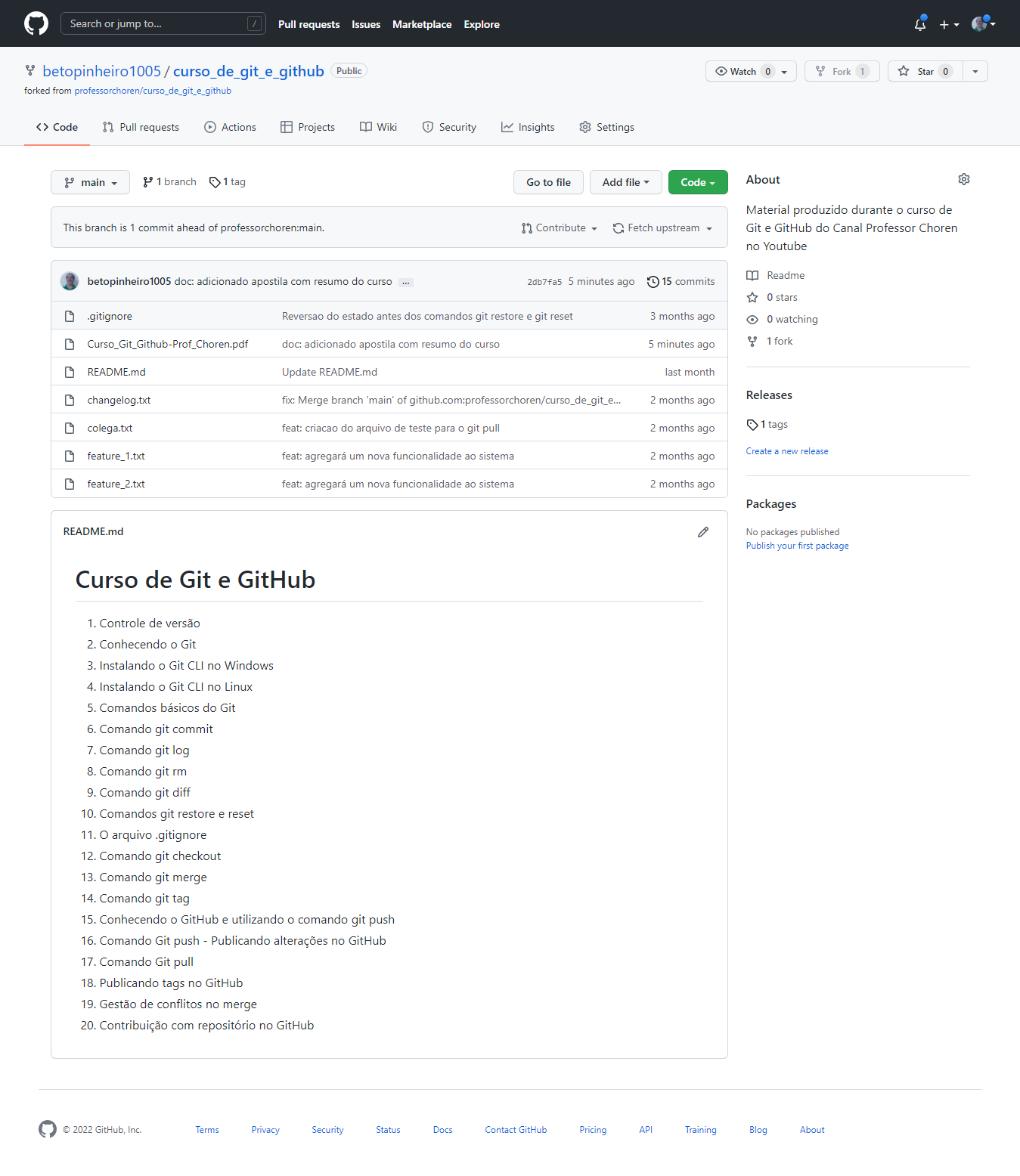
- Salve e feche o arquivo



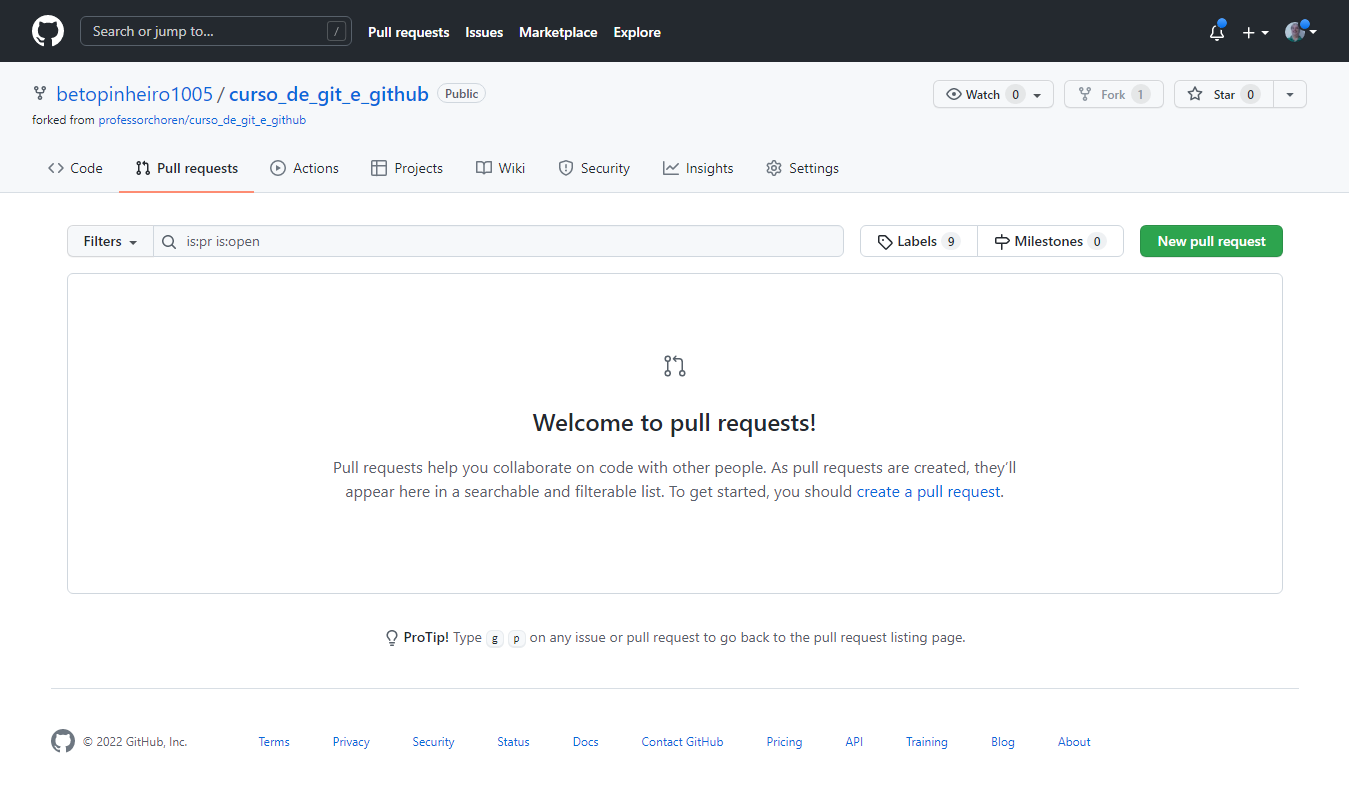
git push



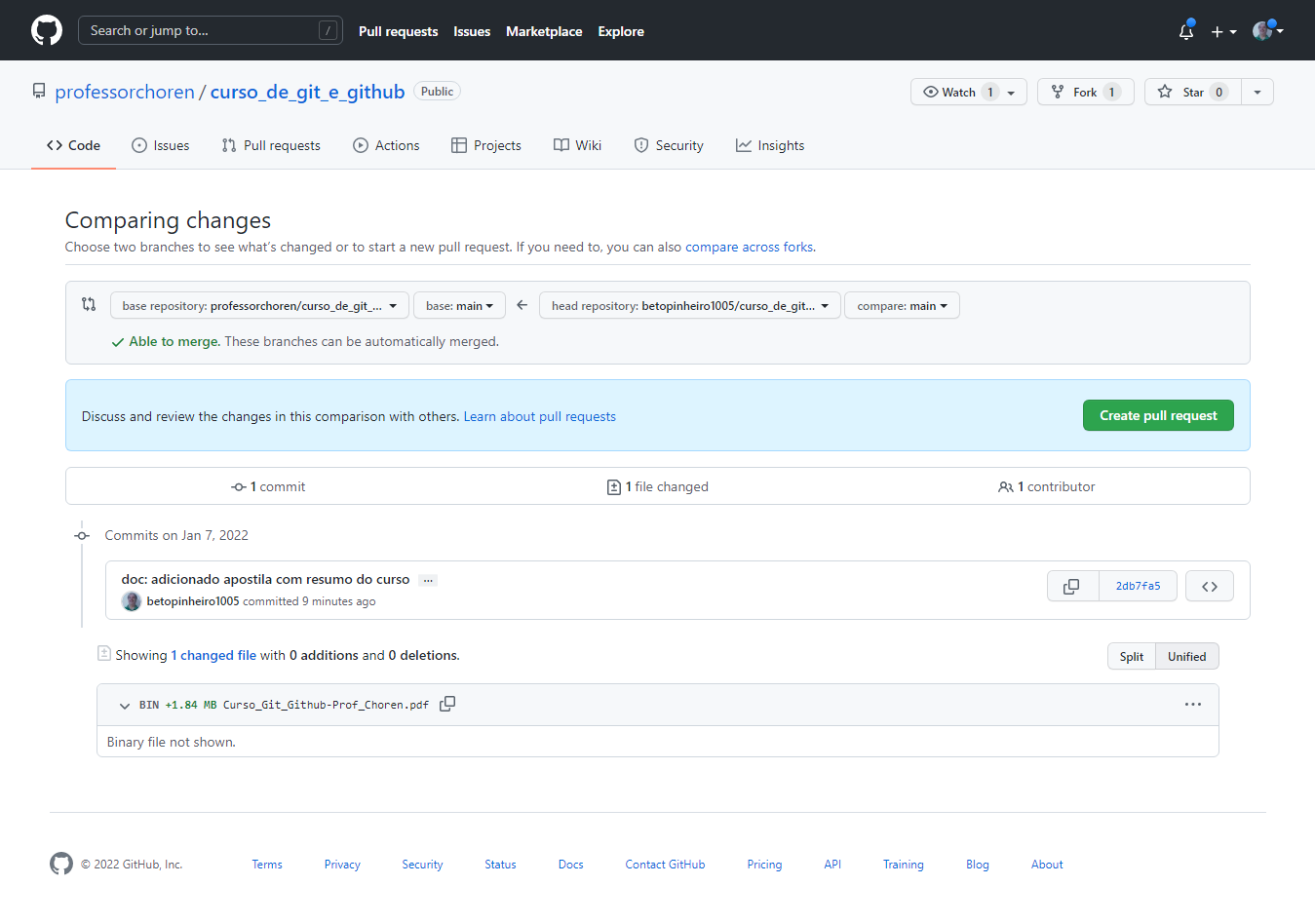
- No GitHub:



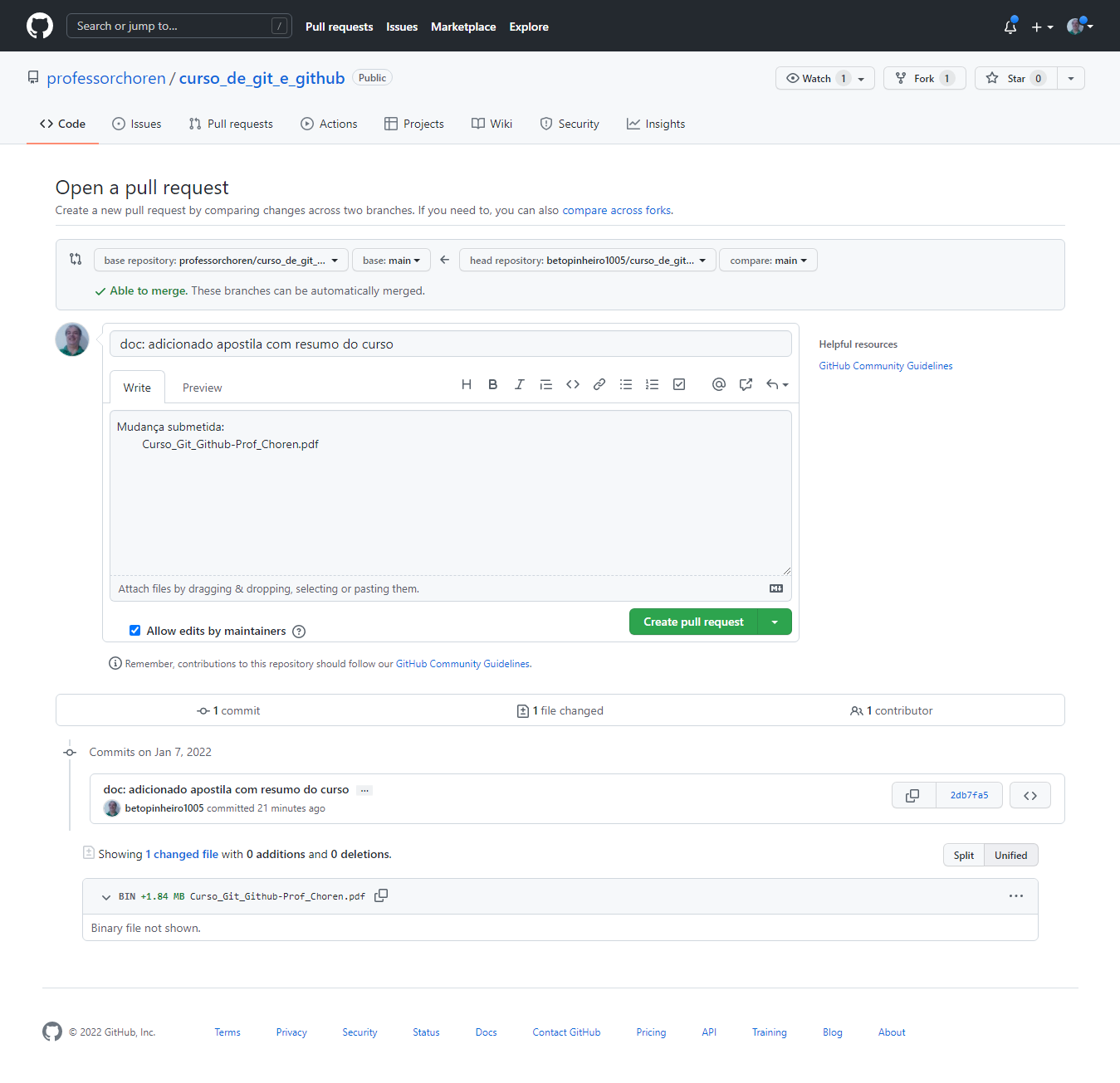
- clique no botão "Pull requests"



- Clique no botão "New pull request"



- Clique no botão "Create pull request"



- Clique no botão "Create pull request"

- Na tela que se abrir, há um campo (Leave a comment) que permite que você deixe um comentário e/ou crie uma mensagem para o dono do canal.

- Neste campo, digite uma solicitação ao autor do projeto para que ele analise sua contribuição e caso esteja de acordo aprove-a fazendo a aceitação para incorporá-la ao repositório dele.

- Em seguida clique no botão "Comment". Faça quantos comentários desejar.

- Aguarde a resposta.