

CURSO COMPLETO DE GIT (2 HORAS E 30 MINUTOS)

https://youtu.be/OuOb1_qADBQ

O versionamento é de extrema importância para o programador conseguir alterar o código sem grandes problemas, é uma ferramenta essencial. Caso a alteração realizada tenha causado algum tipo de problema, ou até mesmo que surja a necessidade de desfazer ou até refazer as alterações feitas, o versionamento facilitará isso para você, pois ele deixa tudo registrado.

Vamos supor que você e um colega estejam trabalhando em um projeto. Você alterou a linha de código número 15, seu colega alterou 15 também. Quando vocês forem “juntar todo o código novamente” (realizar o merge), o versionamento **git** identificará um conflito de código.

Versionamento é ter um controle de versões de controle

Pense na seguinte situação: você escreveu um programa de até cinco mil linhas de código, mas ocorreu um bug causado pelo código, então surge agora a necessidade de descobrir em qual parte do código ocorreu esse problema, perceba que analisar cinco mil linhas novamente demora bastante e não tem necessidade disso se o código estiver versionado pelo **git**, pois ele grava tudo o que é alterado no código.

Com o **git** podemos comparar com as versões anteriores, ele irá nos mostrar exatamente o que foi alterado entre as versões.

Sem o git, o trabalho em equipe seria ainda mais difícil porque as alterações feitas por alguém poderia sobrescrever as alterações feitas por outra pessoa. O controle de versões também não existiria.

Com o git, tudo o que for alterado por determinada pessoa, ficará gravado o nome do usuário. O git recebe tudo isso, de cada pessoa, junta tudo em um arquivo só.

Quando falarmos “iniciar um repositório” significa configurar o git no projeto. Fazer uso do **git** no projeto.

Para saber se o **git** está instalado digite: **git --version**

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~ (main)
$ git --version
git version 2.32.0.windows.1
```

```
C:\Windows\system32>git --version
git version 2.32.0.windows.1
```

Depois de instalar, antes de começarmos a usar o **git**, precisamos configurar com nossas informações. Isso é necessário porque precisamos identificar os usuários, por exemplo, se uma alteração for feita em um projeto, ficará registrado que foi ela que fez a alteração, então isso estará associado ao usuário que fez a alteração.

O comando abaixo realiza essa configuração que falamos agora. O resultado dessa operação bem sucedida não informa nada para nós, mas não se preocupe, tudo deu certo.

Definimos o nome do usuário

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~ (main)
$ git config --global user.name "Sckoofer"
```

Definimos o email do usuário

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~ (main)
$ git config --global user.email "sckoofer@gmail.com"
```

Qual editores estamos usando

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~ (main)
$ git config --global core.editor vscode
```

Saber o nome que foi cadastrado na máquina

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~ (main)
$ git config user.name
Sckoofer
```

Com o **git config --list** podemos visualizar todas as configurações gravadas. A imagem abaixo só está mostrando algumas informações salvas.

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~ (main)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
```

27:15 Criando repositório (vamos usar no windows)

Para navegar entre arquivos, usamos o comando **cd**

Exemplo usando o comando: **cd /nome_da_pasta/nome_da_pasta**

Quando acessarmos uma pasta, tudo o que fizemos, acontecerá nela.

Descobrir o que tem na pasta, usamos o comando **dir** (se for linux, usamos o **tree /f**)

Para voltar uma pasta anterior a atual onde você está: **cd ..**

33:02 Muito bem, antes de começar, vamos acessar a pasta onde está nosso projeto. Vamos fazer isso, a partir de agora, usando o prompt de comando

Vamos agora iniciar o repositório usando o comando: **git init**

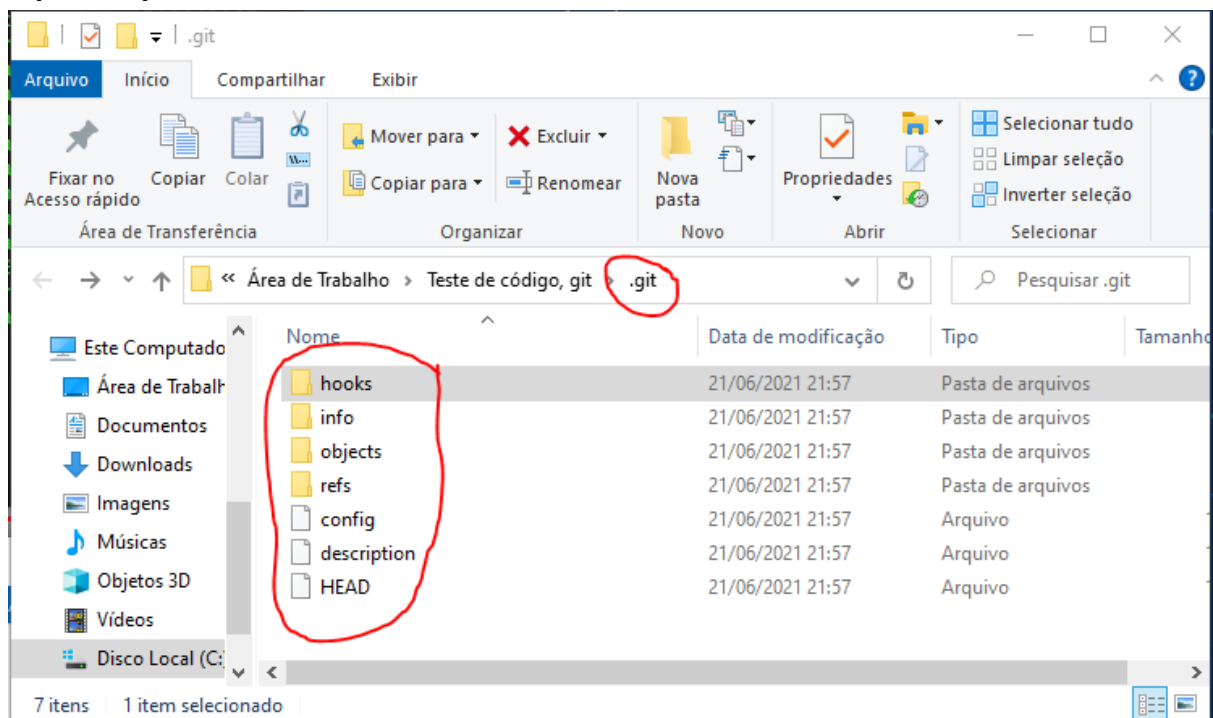
```
C:\Users\SckooferWin10\Desktop\Teste de código, git>git init
Initialized empty Git repository in C:/Users/SckooferWin10/Desktop/Teste de código, git/.git/
```

Fizemos isso dentro do arquivo **Teste de código, git** então o **git** está levando em consideração tudo o que está dentro do arquivo **Teste de código, git**.

Note que, uma pasta chamada **.git** será criada dentro do repositório atual, ela estará invisível oculta. Para visualizá-la via linha de comando, através do **powershell**, use o comando:

Get-ChildItem . -Force

Aqui o arquivo abaixo foi criado.



Só para deixar claro, o **git** não é **github**. Mas o **git** permite sincronizar com o **github**.
O git, na verdade, funciona localmente, para dá pra fazer ele funcionar remotamente.

36:30

O que é um BRANCH?

São ramificações do projeto. Vamos dizer assim, o projeto foi copiado, você tem essa cópia e pode modificar ela o quanto quiser. Não importa o que você faça com essa cópia, ela não é o projeto original. Isto é, são versões diferentes. Se você atualiza a cópia, ou o original, isso não vai afetar os outros. Tudo o que você faz alí, fica alí (se você quiser, é claro)

A **branch master** é onde fica a versão principal, não é uma cópia, é o original

Vamos explicar em outro exemplo:

Tem o código do projeto de um site, o principal, que é a versão 1.0

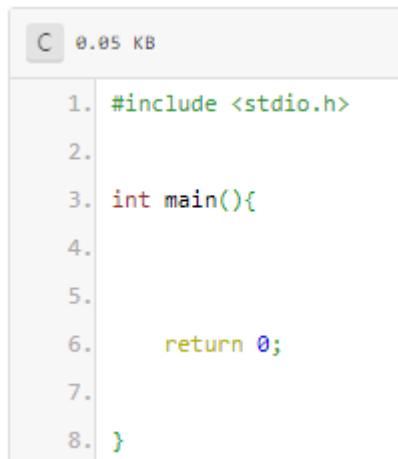
Mas agora vão criar a versão 2.0 do mesmo projeto.

Para fazer isso criamos uma **branch master** para a equipe continuar trabalhando no projeto 1.0 e a outra equipe ficará trabalhando na versão 2.0 em **outra branch** (essa branch pode ter qualquer nome que você quiser).

“**Dar o commit**” ou “**comitar**” é fazer alterações, não apenas no código, e sim pastas e modificando e criando arquivos e documentos etc. E o **git** vai gravar somente o que foi alterado.

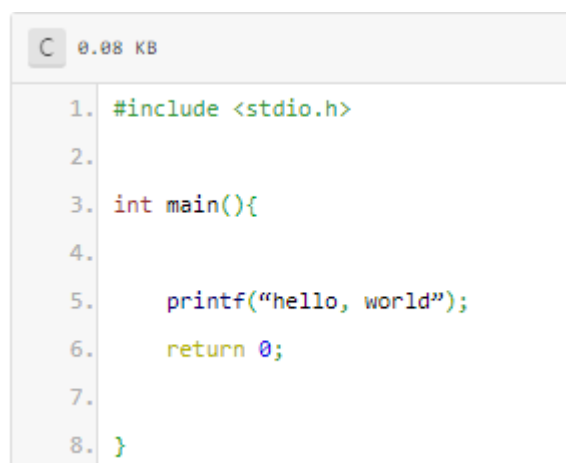
Realizar o commit é mandar para o git a alteração

Exemplo de um commit feito por código:



```
C 0.05 KB
1. #include <stdio.h>
2.
3. int main(){
4.
5.
6.     return 0;
7.
8. }
```

commit: linha 5 - adicionar código linha printf(“hello, world”);



```
C 0.08 KB
1. #include <stdio.h>
2.
3. int main(){
4.
5.     printf(“hello, world”);
6.     return 0;
7.
8. }
```

Você pode mandar comentários com cada commit, se você quiser, para explicar alguma coisa. É uma boa prática comentar as modificações que você está fazendo com cada commit, assim ajuda todo mundo a entender o que você está tentando fazer ou fazendo.

Explique as alterações que você fez.

O arquivo principal do **git** é o arquivo **readme.md**

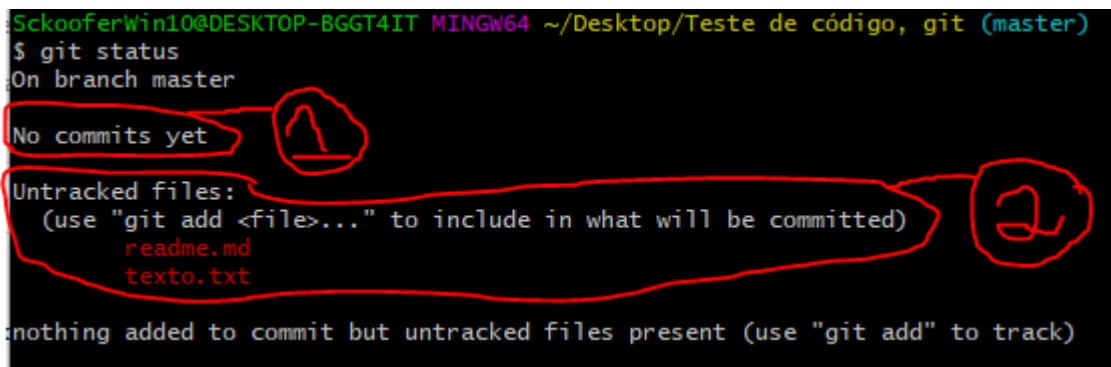
O arquivo **readme.md** é o arquivo que todo programador, quando entra em um projeto, vai ler.

O arquivo **readme.md** contém instruções, comentários etc. enfim, tudo o que for descritivo.

O arquivo pode se chamar **readme.txt** ou **readme.md**

O comando **git status** vai fazer uma varredura na pasta onde está aberta no terminal.

Nessa imagem acima podemos notar algumas coisas que estão circuladas em vermelho:

A terminal window showing the output of the 'git status' command. The output is as follows: '\$ git status', 'On branch master', 'No commits yet', 'Untracked files:', '(use "git add <file>..." to include in what will be committed)', 'readme.md', 'texto.txt', and 'nothing added to commit but untracked files present (use "git add" to track)'. There are two red annotations: a circle with the number '1' around 'No commits yet', and a circle with the number '2' around the 'Untracked files:' section and its contents. A red line also underlines the text '(use "git add <file>..." to include in what will be committed)'.

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
readme.md
texto.txt
nothing added to commit but untracked files present (use "git add" to track)
```

1 - Não ocorreu nenhum commit

2 - Untracked files são arquivos que não foram registrados dentro do git, então o git não está monitorando alterações feitas neles.

Para incluir apenas um arquivo no git, usamos o comando:

git add **Nome_e_extensão_do_arquivo**

Para incluir todos os arquivos (da pasta) no git, usamos o comando:

git add -A

Na imagem abaixo nós aplicamos os seguintes passos:

```
MINGW64:/c/Users/SckooferWin10/Desktop/Teste de código, git

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master

No commits yet

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

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

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git add -A

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master

No commits yet

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

```
MINGW64:/c/Users/SckooferWin10/Desktop/Teste de código, git

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master

No commits yet

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

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

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git add -A

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master

No commits yet

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

3 - Perceba que todos os arquivos da pasta foram adicionados no git

O comando: **git status** revela todas as alterações. Até os que não foram commitadas
Depois de incluir todas as pastas no git, faremos agora nosso primeiro commit. Você se lembra que cada commit aceita comentários que o programador pode explicar o que foi alterado, o motivo etc.? Esse comentário é inserido entre aspas.

Exemplo: **git commit -m "Aqui_voce_coloca_o_comentário"**

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git commit -m "Primeiro Commit"
[master (root-commit) cfa4a22] Primeiro Commit
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 readme.md
create mode 100644 texto.txt
```

A mensagem que apareceu informa as características do commit (os dois arquivos inseridos)

46:34 Como visualizar todos os commits realizados?

Com o comando **git log**, todos os commits realizados em um branch específico serão demonstrados

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git log
commit cfa4a22af199e0f39ee3f83ca8cd67ed19f98f9f (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 14:08:41 2021 -0300

    Primeiro Commit
```

Handwritten red annotations:
- "BRANCH" with an arrow pointing to "(HEAD -> master)"
- "COMENTÁRIO" with an arrow pointing to the commit message "Primeiro Commit"

Na terceira linha: **commit cfa4a22af199e0f39ee3f83ca8cd67ed19f98f9f (HEAD -> master)**
A palavra **HEAD ->** significa qual **branch** está sendo analisado

O comando: **git status** revela todas as alterações!

O comando: git commit -am "comentário" já salva a alteração em um commit

É mais rápido. Você altera lá o arquivo que quer alterar, dá um **git commit -am** e tá pronto, o commit foi realizado com sucesso, parabéns!

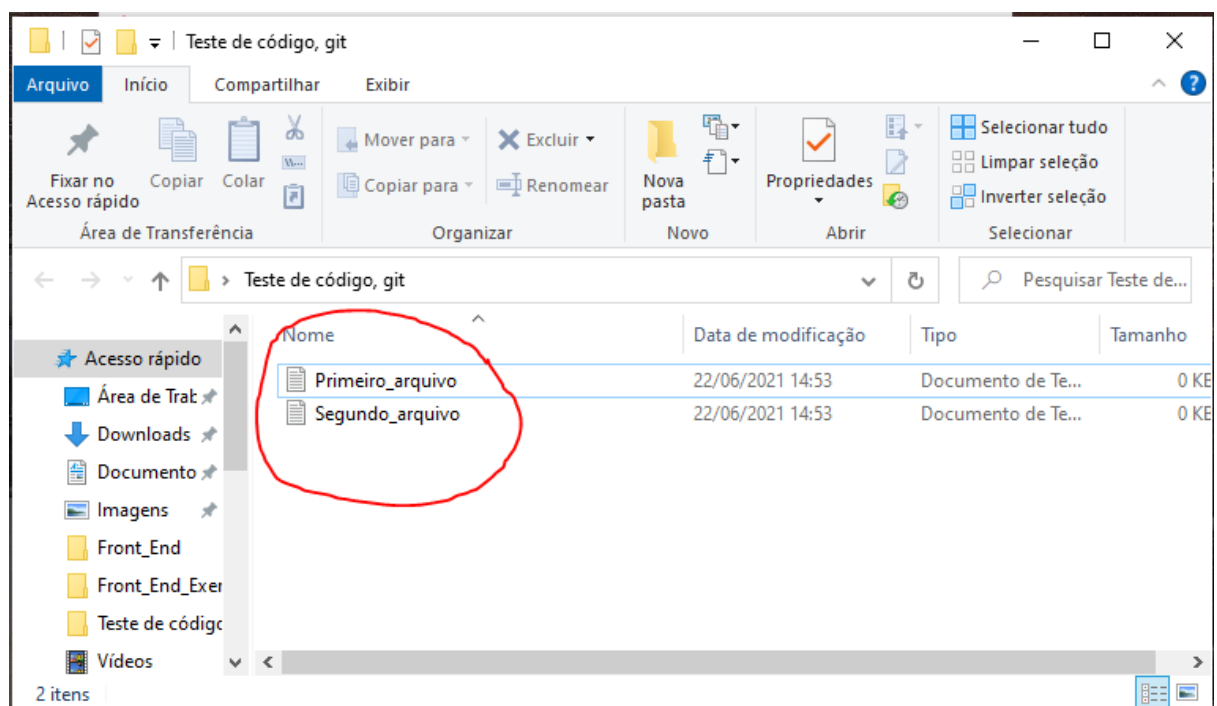
Vamos para um exemplo prático, todos os arquivos que você ver abaixo já estão sendo monitorados pelo **git**, para isso usamos o comando **git add -A**

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git add -A

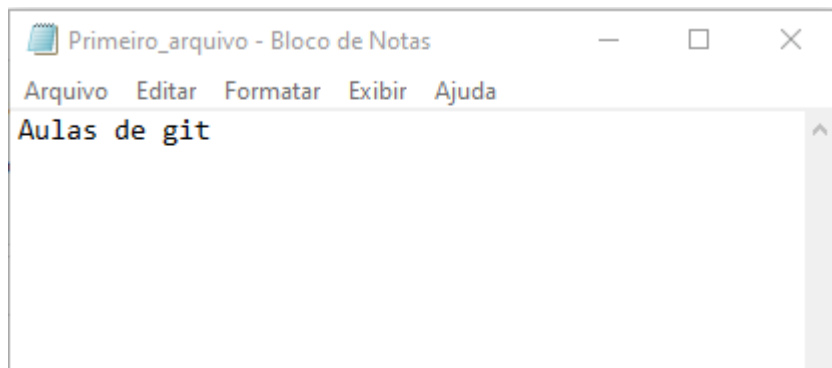
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    readme.md -> Primeiro_arquivo.txt
        renamed:    texto.txt -> Segundo_arquivo.txt

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    readme.md -> Primeiro_arquivo.txt
        renamed:    texto.txt -> Segundo_arquivo.txt

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ |
```



Vamos alterar o primeiro arquivo colocando o seguinte texto:
Aulas de git



Vamos salvar no bloco de notas.

Agora e digitar o comando: **git status**

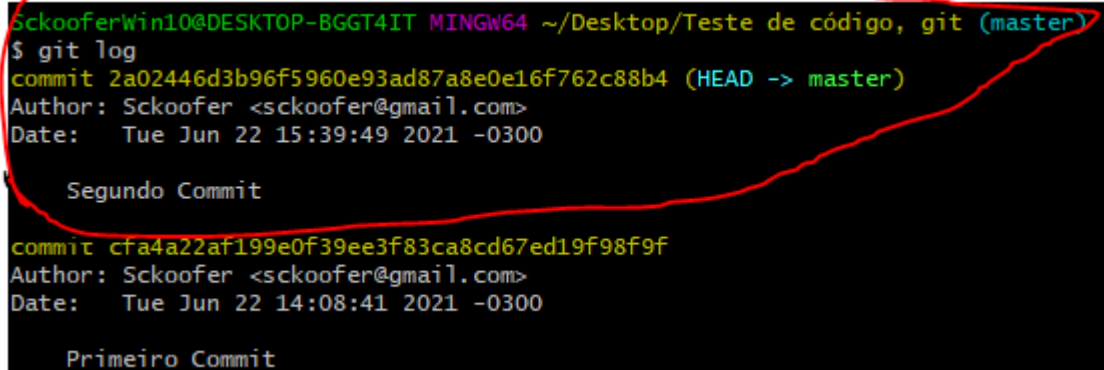
```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        README.md              | 1 +
        Primeiro_arquivo.txt    | 1 +
        Segundo_arquivo.txt     | 1 +
  3 files changed, 3 insertions(+)
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:   Primeiro_arquivo.txt
```

Está circulado em vermelho um recado do git, informando que o arquivo **primeiro_arquivo.txt** foi alterado e o não foi **commitado**

Vamos commitar com o comando: **git commit -m** e escrever na área de comentários a frase “Segundo Commit”

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git commit -m "Segundo Commit"
[master 2a02446] Segundo Commit
3 files changed, 1 insertion(+)
create mode 100644 Primeiro_arquivo.txt
rename readme.md => Segundo_arquivo.txt (100%)
delete mode 100644 texto.txt
```

Para verificar o commit realizado usaremos o comando: **git log**



```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git log
commit 2a02446d3b96f5960e93ad87a8e0e16f762c88b4 (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 15:39:49 2021 -0300

    Segundo Commit

commit cfa4a22af199e0f39ee3f83ca8cd67ed19f98f9f
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 14:08:41 2021 -0300

    Primeiro Commit
```

Pronto! Lembrando que tudo isso ocorreu no branch master.

Se você não der o commit, todas as alterações que você fizer não serão gravadas no git, portanto, só ficarão gravadas no seu computador.

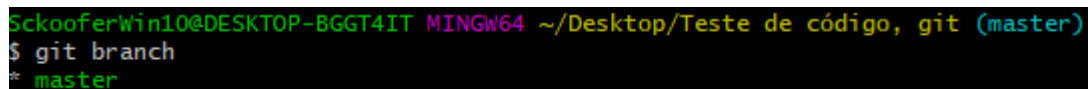
---> 50:39 - Aprendemos agora os principais comandos que utilizaremos no git

- 1) comando: git status.....serve para verificar
- 2) comando: git add -A.....para trackear (monitorar) arquivos
- 3) comando: git commit -m "aqui você escreve comentários sobre o commit"

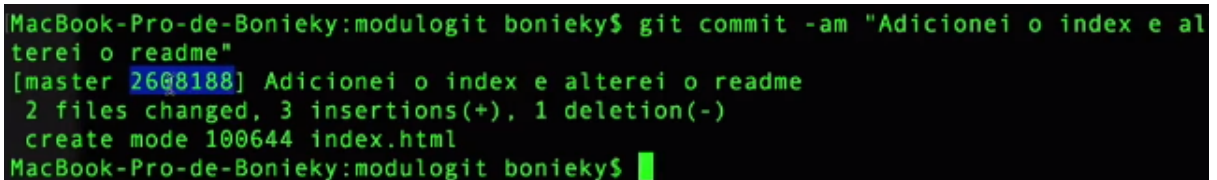
Vamos agora remover as modificações que criamos? 52:00

Consultamos em qual branch estamos: **git branch**

Na imagem abaixo, indica que estamos no branch **master**



```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Teste de código, git (master)
$ git branch
* master
```



```
MacBook-Pro-de-Boniekky:modulogit boniekky$ git commit -am "Adicionei o index e alterei o readme"
[master 26@8188] Adicionei o index e alterei o readme
2 files changed, 3 insertions(+), 1 deletion(-)
create mode 100644 index.html
MacBook-Pro-de-Boniekky:modulogit boniekky$
```

Em azul são só os primeiros **dígitos** do hash de commit

Uma das grandes vantagens de usar o **git** é a possibilidade de deixar o código como estava antes. Para isso acontecer, antes você precisaria ter commitado a versão que gostaria de poder voltar, aquele hash com vários números e letras, aquilo é a identificação do commit.

Para voltar para uma versão anterior do código, usamos o comando:

git reset --o_tipo_do_reset

No o_tipo_do_reset pode você escolher por:

1) soft

Retorna à uma versão, mas mantém as alterações realizadas posteriormente, porém, sem estar registrado no commit.

2) mixed

3) hard

Desfaz tudo o que foi feito posteriormente. Essa opção não é recomendada para ser utilizada em equipe.

Vamos fazer um teste usando a opção hard

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git log
commit cb5f95d136ad8552f69176232fc1da1635078631 (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:36:28 2021 -0300

    Quinta e última alteração

commit fbed516952f55d5d898ac2547e2ed18f5e306589
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:35:44 2021 -0300

    Quarta Alteração

commit 2e182ab75a532f6c23a6385a74cffdb28dbfe08f
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:34:21 2021 -0300

    Novo Documento de Texto.txt

commit 16013d87c8ffb5fbdeee22eb17db07709af3cf78
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:58:25 2021 -0300

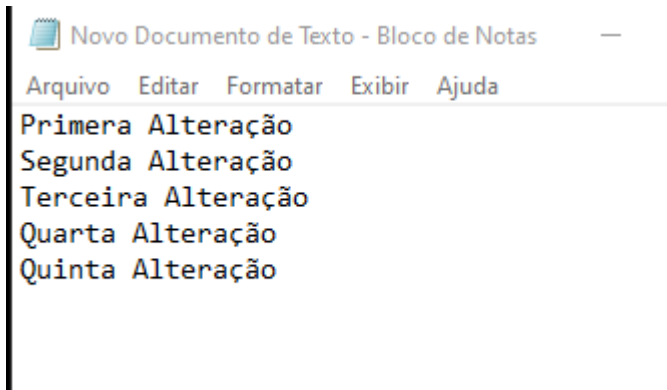
    Segunda Alteração

commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:53:33 2021 -0300

    Primeiro Arquivo

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$
```

Situação atual do arquivo



```
commit cb5f95d136ad8552f69176232fc1da1635078631 (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:36:28 2021 -0300

    Quinta e última alteração

commit fbed516952f55d5d898ac2547e2ed18f5e306589
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:35:44 2021 -0300

    Quarta Alteração

commit 2e182ab75a532f6c23a6385a74cfffdb28dbfe08f
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:34:21 2021 -0300

    Novo Documento de Texto.txt

commit 16013d87c8ffb5fbdeee22eb17db07709af3cf78
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:58:25 2021 -0300

    Segunda Alteração

commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:53:33 2021 -0300

    Primeiro Arquivo

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git reset --hard commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867
```

Executamos o comando

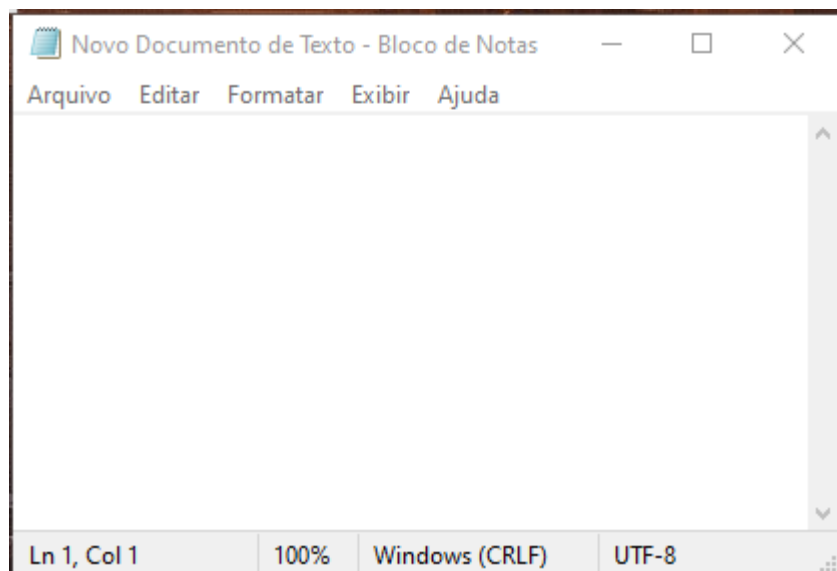
```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git reset --hard 2e3480d46d557960c7b2bfc87dd3c609c8ca0867
HEAD is now at 2e3480d Primeiro Arquivo
```

Esse é o resultado do log após a execução da tarefa. Perceba que todos os **commits** que vieram depois daquele que você escolheu, sumiram.

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git log
commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867 (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:53:33 2021 -0300

    Primeiro Arquivo
```

E abaixo temos a imagem de como ficou o arquivo após executar o comando.



Para não ter que exemplificar novamente, vamos explicar mais uma vez. A opção **soft** apaga os commits posteriores da aquela que você escolheu, **MAS NÃO DESFAZ ALTERAÇÕES NOS ARQUIVOS, ELA SÓ ACABA CERTOS COMMITS.**

```

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git log
commit 8421bd803042144dda86f023cd3e36a19d9fecfc (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 21:01:41 2021 -0300

    Terceiro Impacto

commit 690a5cfffcc535959cc3abffc93d28c6d504676e1
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 20:59:00 2021 -0300

    Pós alteração

commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:53:33 2021 -0300

    Primeiro Arquivo

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git reset --soft commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867
fatal: ambiguous argument 'commit': unknown revision or path not in the working
tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

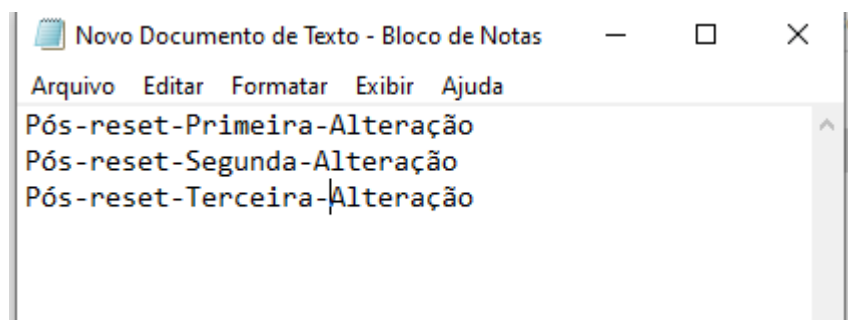
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git reset --soft 2e3480d46d557960c7b2bfc87dd3c609c8ca0867

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git log
commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867 (HEAD -> master)
Author: Sckoofer <sckoofer@gmail.com>
Date: Tue Jun 22 19:53:33 2021 -0300

    Primeiro Arquivo

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ |

```



Perceba na cima acima, nada mudou no arquivo, diferente dos commits que sumiram.

1:06 Vamos criar um branch.

Lembrando que, quando criamos um novo branch, é como se a gente tivesse copiado todo o conteúdo do projeto, e colocado em outro lugar.

Para criar um novo branch usamos o comando: **git branch nome_do_branch**

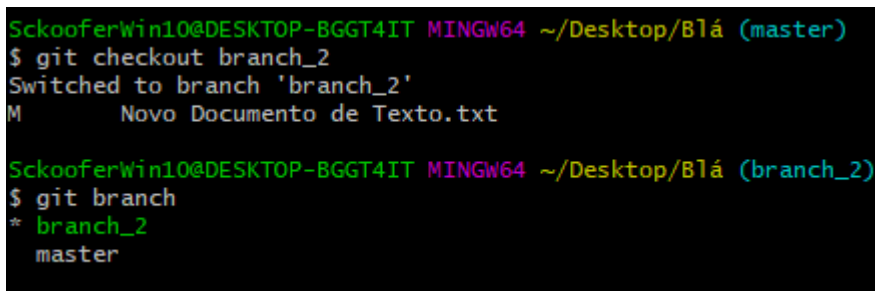


A terminal window showing the command `git branch branch_2` being executed. The prompt is `SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)`. The command is circled in red, and a red circle with the number 1 is next to it. The output shows the command being repeated, and then `* master` is displayed, with a red circle and the number 2 next to it.

O novo branch foi criado, colocamos o nome de **branch_2**. Para visualizar todos os branches do projeto usamos o comando: **git branch_2**.

Perceba que, na imagem acima, temos o **branch master** colorido em verde com asterisco na frente. Isso significa que, estamos visualizando apenas o **branch master**.

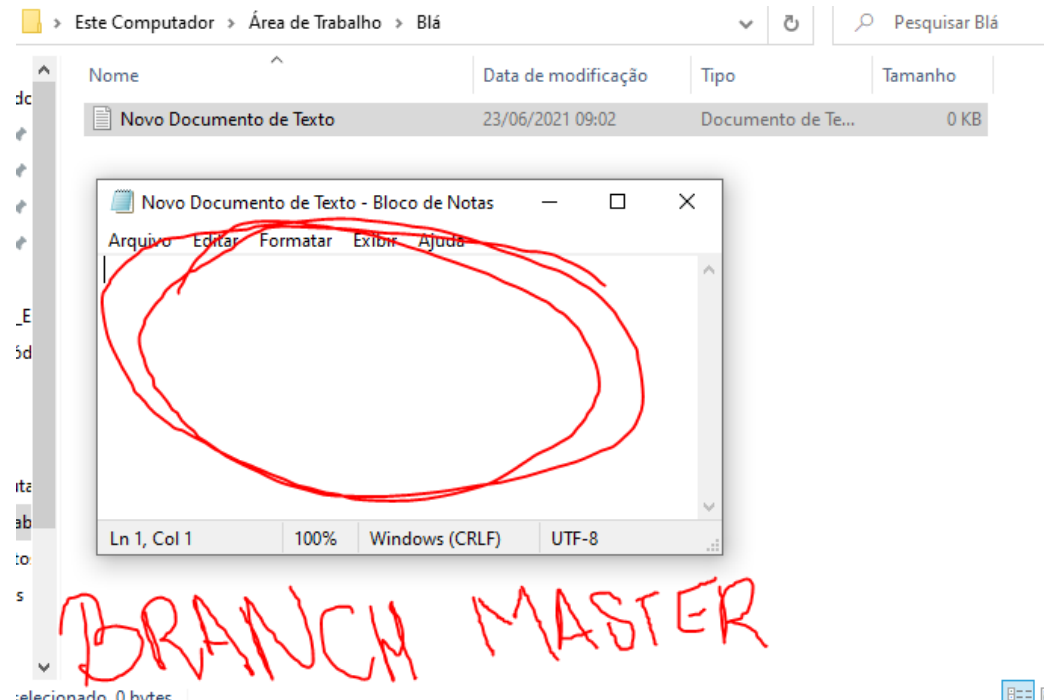
Para visualizar outros branches, precisamos antes sair do atual e entrar no qual queremos, para isso usamos o comando: **git checkout nome_do_branch**



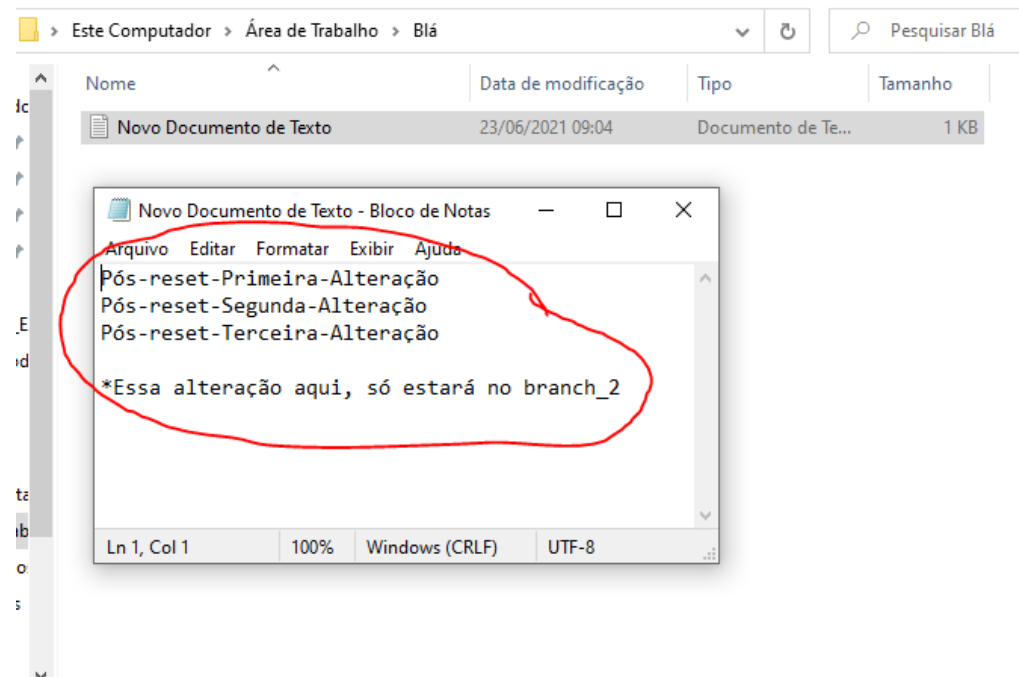
A terminal window showing the command `git checkout branch_2` being executed. The prompt is `SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)`. The command is executed, and the output shows `Switched to branch 'branch_2'` and `M Novo Documento de Texto.txt`. The prompt then changes to `SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (branch_2)`. The command `git branch` is executed, and the output shows `* branch_2` and `master`.

Lembrando, novamente, quando criamos um novo branch, é como se copiássemos o que tem dentro de um branch para outro.

Olha que loucura, no bom sentido, é claro,
Trocamos para o branch master e abrimos o arquivo usando bloco de notas. Não tem nada ali, está vazio!



Vamos trocar para o branch_2. Veja que o arquivo contém coisas que só aparecem se estivermos no branch_2 e que tenham sido commitadas.

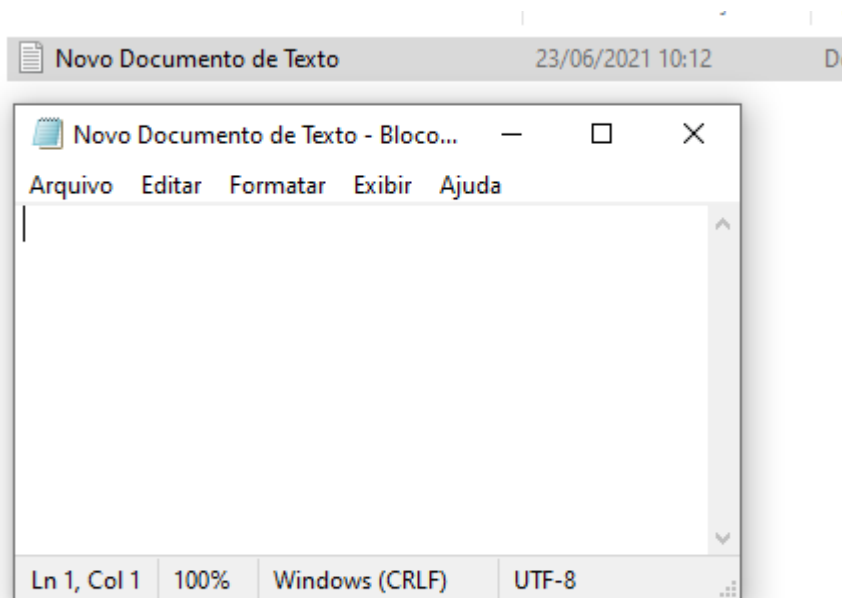


Podemos criar novos branches, lembrando que são cópias do estado que você estava no momento antes de fazer o branch. Você pode alterar depois os outros, não afeta os demais. É só uma cópia, você modifica essa cópia o quanto quiser, não afeta os outros branches.

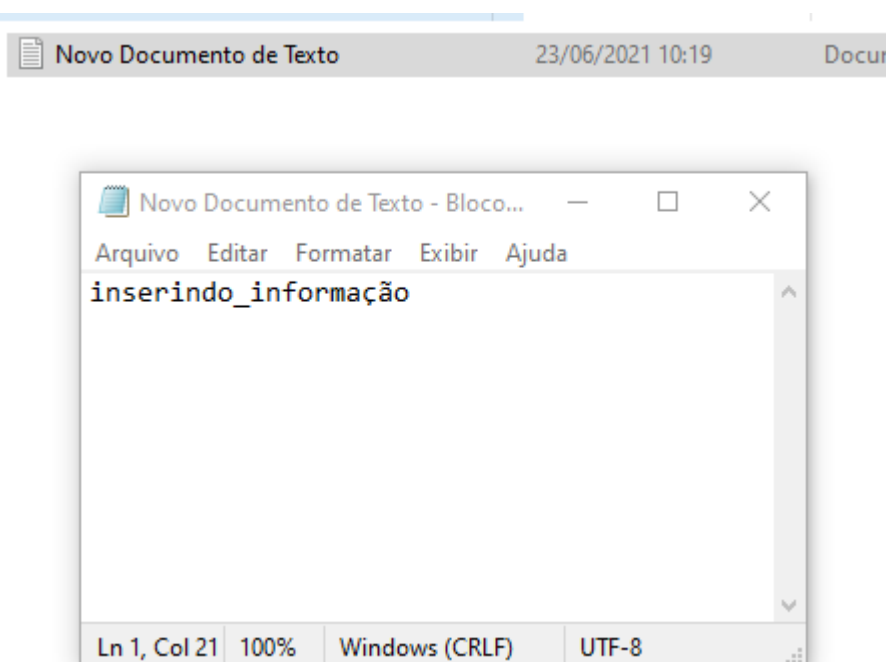
Cada equipe pode trabalhar em branches diferentes, é como se fossem projetos a parte.

1:17

Quando fizemos a alteração, mas não fizemos o commit, o git nos informará que ocorreu uma alteração nos arquivos que ele estava monitorando. Com o comando: **git diff** conseguimos saber exatamente quais linhas de código foram alteradas



Nada aqui em cima, vamos alterar, salvar e usar o comando **git diff**



```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git diff
diff --git a/Novo Documento de Texto.txt b/Novo Documento de Texto.txt
index e69de29..64fe8fd 100644
--- a/Novo Documento de Texto.txt
+++ b/Novo Documento de Texto.txt
@@ -0,0 +1 @@
+inserindo_informação
\ No newline at end of file
```

Como pode ver, o **git** já identificou a alteração no arquivo que ainda não foi commitado. Nesse caso, a frase “inserindo_informação” está sendo colocada dentro do arquivo, por isso a frase está colorida em verde.

Vamos commitar a alteração, abrir novamente o arquivo, apagar a frase “inserindo_informação” para visualizar novamente o monitoramento feito pelo **git**

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git diff
diff --git a/Novo Documento de Texto.txt b/Novo Documento de Texto.txt
index 64fe8fd..e69de29 100644
--- a/Novo Documento de Texto.txt
+++ b/Novo Documento de Texto.txt
@@ -1,0 +0,0 @@
-inserindo_informação
\ No newline at end of file
```

Agora a frase está em vermelho, quando commitar essa alteração que fizemos, o **git** não encontrará alterações, pois já foi commitado.

Arquivo Editar Formatar Exibir Ajuda

Propriedades

Abrir

Limpar seleção

Inverter seleção

Selecionar

Pesquisar Blá

Tipo

Tamanho

Documento de Te...

0 KB

Ln 1, Col 1 100% Windows (CRLF) UTF-8

MINGW64:/c/Users/SckooferWin10/Desktop/Blá

Date: Wed Jun 23 10:28:24 2021 -0300

Inserindo Frase

commit 3e3480d46d557960c7b2bfc87dd3c609c8ca0867

Author: Sckoofer <sckoofer@gmail.com>

Date: Tue Jun 22 19:53:33 2021 -0300

Primeiro Arquivo

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)

\$ git diff

diff --git a/Novo Documento de Texto.txt b/Novo Documento de Texto.txt

index 64f8fd1..e69de29 100644

--- a/Novo Documento de Texto.txt

+++ b/Novo Documento de Texto.txt

@@ 1,1 @@

insere aqui o conteúdo

\ No newline at end of file

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)

\$ git commit -am "Apagando nova frase"

[master 3ebd689] Apagando nova frase

1 file changed, 1 deletion(-)

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)

\$ git log

commit 3ebd6891d848e1b5707c3213f7238dad124b51e1 (HEAD -> master)

Author: Sckoofer <sckoofer@gmail.com>

Date: Wed Jun 23 10:30:00 2021 -0300

Apagando nova frase

commit d2044db39a40088ed83412632497998c942b8819

Author: Sckoofer <sckoofer@gmail.com>

Date: Wed Jun 23 10:28:24 2021 -0300

Inserindo Frase

commit 2e3480d46d557960c7b2bfc87dd3c609c8ca0867

Author: Sckoofer <sckoofer@gmail.com>

Date: Tue Jun 22 19:53:33 2021 -0300

Primeiro Arquivo

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)

\$ git diff

Mas e se tivéssemos alterado vários arquivos e só quiséssemos saber apenas o nome dos arquivos que foram alterados, mas ainda não foram commitados?

Usamos o comando: **git diff --name-only**

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git diff --name-only
```

E para descobrirmos o que foi alterado apenas em um arquivo, usamos o comando: **git diff nome_do_arquivo_e_sua_extensão**. Veja no exemplo do professor:

```
MacBook-Pro-de-Boniekky:modulogit boniekky$ git diff style.css
diff --git a/style.css b/style.css
index e69de29..f9616a0 100644
--- a/style.css
+++ b/style.css
@@ -0,0 +1 @@
+adicionar uma nova linha
\ No newline at end of file
```

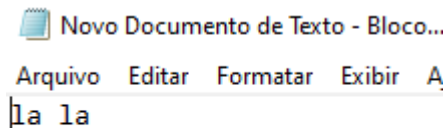
Para desfazer a alteração que ainda não foi commitada, apenas em arquivo

```
modulogit --bash -- 80x24
MacBook-Pro-de-Boniekky:modulogit boniekky$ git checkout HEAD -- style.css
MacBook-Pro-de-Boniekky:modulogit boniekky$
```

git checkout HEAD -- nome_do_arquivo.

A palavra chave **HEAD** se trata do branch que estamos.

Vamos para um exemplo prático:



Novo Documento de Texto - Bloco...

Arquivo Editar Formatar Exibir A

la la

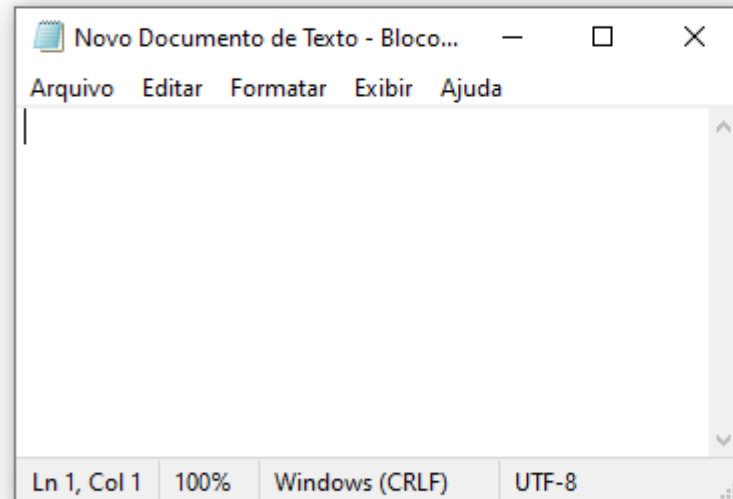
Nós já salvamos essa alteração, mas ainda não foi commitado

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Novo Documento de Texto.txt

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

SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git diff Novo\ Documento\ de\ Texto.txt
diff --git a/Novo Documento de Texto.txt b/Novo Documento de Texto.txt
index e69de29..d014ec7 100644
--- a/Novo Documento de Texto.txt
+++ b/Novo Documento de Texto.txt
@@ -0,0 +1 @@
+la la
\ No newline at end of file
```

Desfizemos a alteração que foi salvo no computador porém ainda não foi commitado.



ado 0 bytes

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git checkout HEAD -- Novo\ Documento\ de\ Texto.txt
```

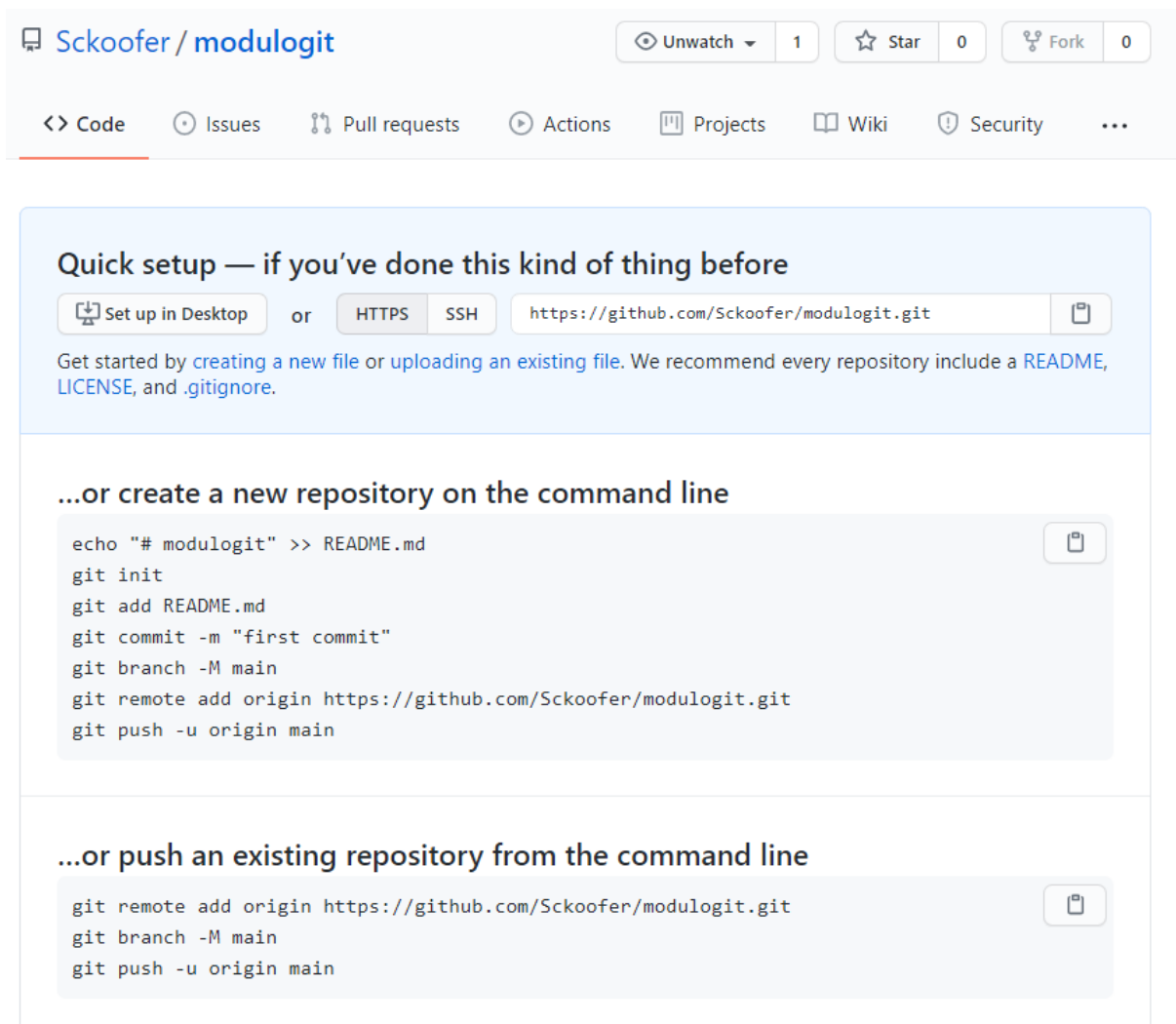
01:27

Repositório local é dentro do computador, repositório remoto é como se fosse um site. Vários programadores podem trabalhar no mesmo projeto em diferentes lugares do mundo.

Vamos transferir os arquivos do repositório local para o repositório remoto, no github. Faremos isso usando o git.

Criamos nosso repositório no nosso github, não optamos por colocar o arquivo README.md

O resultado foi a tela abaixo



The screenshot shows the GitHub interface for a repository named 'modulogit' by user 'Sckoofer'. At the top, there are buttons for 'Unwatch' (1), 'Star' (0), and 'Fork' (0). Below these are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and a menu icon. The main content area has a light blue header with the text 'Quick setup — if you've done this kind of thing before'. Below this header, there are buttons for 'Set up in Desktop', 'HTTPS', and 'SSH', followed by the repository URL 'https://github.com/Sckoofer/modulogit.git'. A paragraph of text follows: 'Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).' Below this is a section titled '...or create a new repository on the command line' with a list of git commands in a code block. At the bottom is another section titled '...or push an existing repository from the command line' with a list of git commands in a code block.

Sckoofer / modulogit

Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security ...

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

Set up in Desktop or HTTPS SSH https://github.com/Sckoofer/modulogit.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 "# modulogit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Sckoofer/modulogit.git
git push -u origin main
```

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

```
git remote add origin https://github.com/Sckoofer/modulogit.git
git branch -M main
git push -u origin main
```


01:15

Antes de passar nosso projeto que está no git, para o github Precisamos criar uma chave que será gerada. Para isso acontecer, mesmo email do github deverá ser usado nos parâmetros do comando: **ssh-keygen -t rsa -b 4096 -C "sckoofer@gmail.com"**

Isso vai gerar a pasta oculta **ssh**

Vai pedir uma senha também, mas não precisa colocar uma, tecla enter.

Vai pedir para confirmar a senha, mas não precisa, tecla enter.

Então será gerado a chave pública e a privada, de acesso:

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ ssh-keygen -t rsa -b 4096 -C "sckoofer@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/SckooferWin10/.ssh/id_rsa):
Created directory '/c/Users/SckooferWin10/.ssh'.
Enter passphrase (empty for no passphrase): ✖
Enter same passphrase again: ✖
Your identification has been saved in /c/Users/SckooferWin10/.ssh/id_rsa
Your public key has been saved in /c/Users/SckooferWin10/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:XdEBIbOqTAPrh8HY7bzWATZfGhn1WoyG04u5jiv1d7U sckoofer@gmail.com
The key's randomart image is:
+---[RSA 4096]-----+
|          .+ ++.. |
|          .o B .. |
|          oo= =   |
|      + ++ oB.=   ..|
|      . =.++S+=   .E|
|      . B ++o .   |
|      o B.o.      |
|      ..*.        |
|      .o .        |
+---[SHA256]-----+
```

Para entrar nesse arquivo criado, onde estão as chaves geradas, precisa acessar o arquivo que foi passado na imagem acima **C:\Users\SckooferWin10\.ssh**

SckooferWin10 > .ssh					Pesquisar .ssh	
Nome		Data de modificação	Tipo	Tamanho		
id_rsa		23/06/2021 11:49	Arquivo	4 KB		
id_rsa.pub		23/06/2021 11:49	Arquivo PUB	1 KB		

Aqui estão as duas chaves criadas.

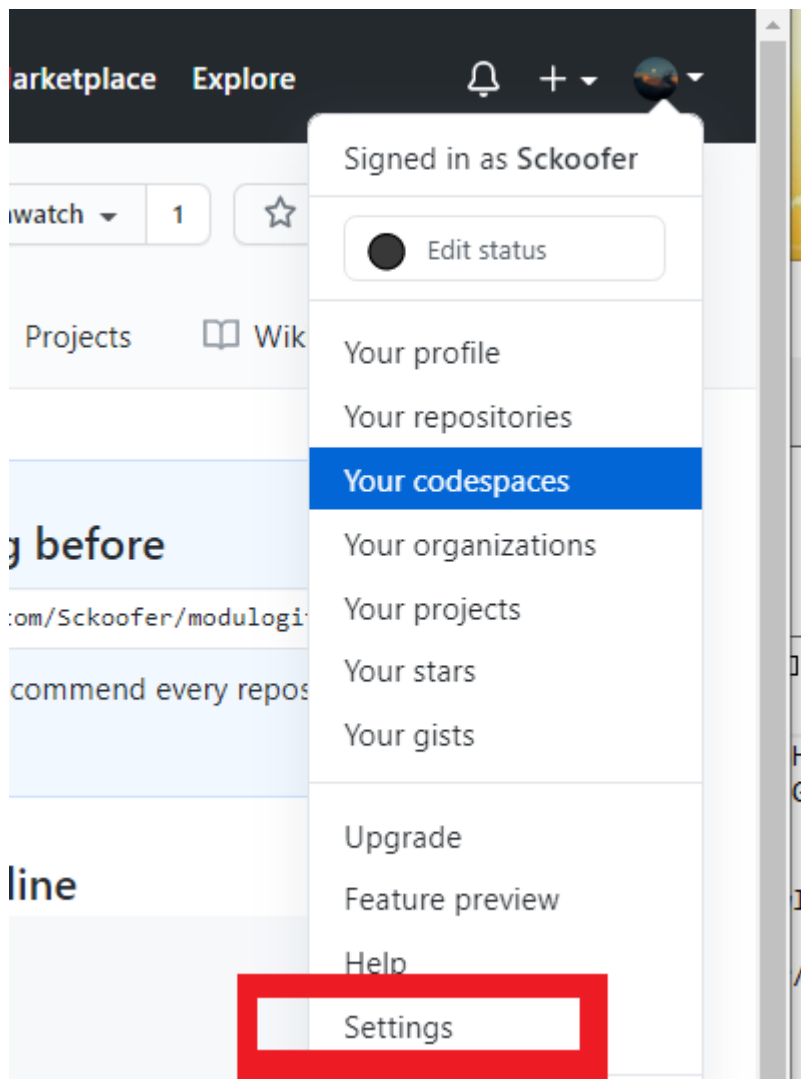
Vamos abrir o arquivo **id_rsa.pub**



A screenshot of a Notepad window titled "id_rsa.pub - Bloco de Notas". The window contains a long string of text representing an SSH public key. The text is as follows:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCMNkHLZ/TKiGV+pWYKoH11vHXW/GK5Z67154hRhs2Xz0BK/RtP+z1U4npUcR3D/VTW5HXk
+xZFwOCZyX3h2vvC3zReThdhABcxHnzyEuvuiXuULY0C/9J6xC0hP1hb/TA5fzvDmWLyLitnSAciwBvGtMkvE2FefRT5U59m00j/nXnWvaGA1VC
5bjXpjpWlrX1+nYZzKqXCi67Ksf5wn1pM1D5Y4JUkXh98w3Js8dgCkwt94nXFXDjR6s81UEFPszVPdv1k9mZw6BwUzMK3fBMB9pyTe
+1THNoJg8HWJLnPMNnyvh+yCfeDrjzS1W5zsM6DQPjzr7wIp/bN7IMXELQecUo9UZxDv6I790/odFvKmmS1AUyx0LiJg//KgqtA
+NguPC58QbrxwkQJz0k2MVgOGqe6gpE1hZ0mQPNG7hTyTq40riUIShIyUy+gEEQ503zBnUfMpMqj8NDxXe+PQAMyqY5+3j3hgrPCLv8w1V
+KdR
+xV8aKhoa9wv2j1XM0ff/q91ydJ7qBb/AzUa5kFeKDXu95Art2e41NK1CFW/JpKapHgKow44Ht1a9ywSgTeeA1XTxVMAZ/UpoM5cqdyKr/t0AP
9/v0UwTsQ7gskvlgToJyUpHpbH91KFQ5sNhZ0UziZiLxwDiapAXjBVY0r+AA1E5bafCAiYJsRxVD4mH/Pxw== skoofer@gmail.com
```

Selecionamos e copiamos essa chave para usar ela no github. A usamos a opção settings.



Security & analysis	
Emails	
Notifications	
Scheduled reminders	
SSH and GPG keys	
Repositories	
Packages	
Organizations	

Bio

Robert

You can @

URL

Twitter u

Compan

fer

ional account [Switch to another account](#)

[Go to your personal profile](#)

s

SSH keys

[New SSH key](#)

There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

[New GPG key](#)

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

SSH keys / Add new

Title

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Na área **Key** você cola a chave copiada e dá um título para identificar a fonte de acesso.

SSH keys / Add new

Title

Chave usando SckooferWin10 Desktop

Key

```
039m0Cj/nXnWvaGAtvC5bJxpjpwXt+tmZZRqXc107Ks13wntpiwTD31490KXt96w
3Js8dgCkwt94nXFXDjR6s81UEFPszVPdvlk9mZw6BwUzMK3fBMB9pyTe+ITHNoJg8
HWJLnPMNnyvh+yCfeDrjzSIW5zsM6DQPjzr7wlp/bN7IMXELQecUo9UZxDv6I79O/
odFvKmms1AUyxOLiJg//KgqtA+NguPC58QbrxwkQJzOk2MVgOGqe6gpElhZOmQ
PNG7hTyTq4OriUISHlyUy+gEEOQSO3zBnUfMpMqj8NDxXe+PQAMyqY5+3j3hgrP
CLv8wIV+KdR+xV8aKhoa9wv2jIXM0ff/q91ydJ7qBb/AzUa5kFeKDXu95ARt2e41NK1
CFW/JpKApHgKow44Ht1a9ywSgTeeAIXTxVMAZ/UpoM5cqdyKr/t0AP9/vOUwTsQ7
gskvlGToJyUphpbH9IKfQ5sNhZ0UziZiLxwDiapAXjBVYOr+AA1E5bafCAiYJsRxVD4m
H/Pxw== sckoofer@gmail.com
```

Add SSH key



Sckoofer

Your personal account

[Switch to another account](#)


[Go to your personal profile](#)

Account settings
Profile
Account
Appearance
Account security
Billing & plans
Security log
Security & analysis

SSH keys

[New SSH key](#)

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



Chave usando SckooferWin10 Desktop
SHA256:XdE8IbOqTAPrh8HY7bzWATZfGhn1WoyG04u5jiV1d7U
Added on 23 Jun 2021
Never used — Read/write

Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

A primeira parte foi realizada com sucesso!

Vamos acessar o github e encontrar lá o repositório que criamos agora a pouco.
E é essa a mensagem que recebemos agora assim que acessamos ele. É praticamente um passo a passo ensinando como lidar com o repositório usando linha de comando.

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



Set up in Desktop

or

HTTPS

SSH

<https://github.com/Sckoofer/modulogit.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 "# modulogit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Sckoofer/modulogit.git
git push -u origin main
```



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

```
git remote add origin https://github.com/Sckoofer/modulogit.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

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

 Set up in Desktop

or

HTTPS

SSH



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

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

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



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

```
git remote add origin https://github.com/Sckoofer/modulogit.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](#)

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git remote add origin https://github.com/Sckoofer/modulogit.git
```

O repositório foi adicionado

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git remote
origin
```

```
SckooferWin10@DESKTOP-BGGT4IT MINGW64 ~/Desktop/Blá (master)
$ git remote -v
origin https://github.com/Sckoofer/modulogit.git (fetch)
origin https://github.com/Sckoofer/modulogit.git (push)
```

1:44

Comandos:

git push -u origin master

Envia o repositório local **master** para o repositório remoto **origin**

Meu tutorial para mim mesmo de como mandar um arquivo para meu próprio repositório no github. Isso só deverá funcionar pelo meu computador porque ele já tem acesso graças às chaves de acesso.

- 1) Crie um arquivo
- 2) Clique com o botão direito e abra o **git bash**
- 3) Para inicializar o monitoramento Digite: **git init**
- 4) Crie e salve um arquivo com informações, para poder mandar para o github
- 5) Faça o commit desse arquivo com o comando: **git commit -am "Comentário"**
- 6) Para saber se o commit funcionou use o comando: **git log** para encontrar o commit na lista de commits realizados
- 7) Adicione um "arquivo invisível" com o comando:
git remote add nome_do_arquivo endereço SSH do repositório (a gente chama o arquivo de origin, mas não é obrigatório)

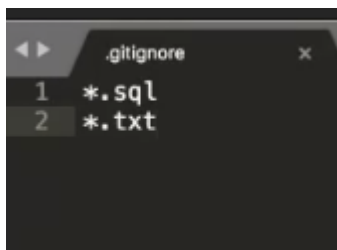
Agora você vai enviar a branch que você quer:

- 8) **git push nome_da_branch_do_git_hub
nome_da_branch_que_estamos_com_ela_aberta_agora_no_git**

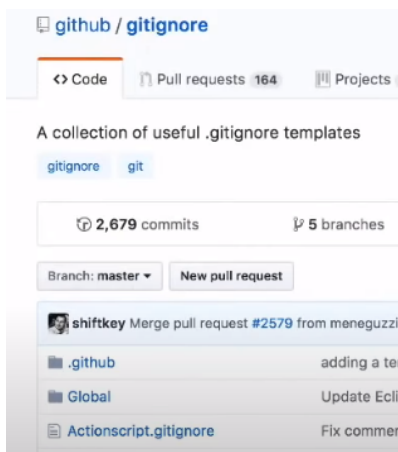
Tem que ter a chave

1:57

O arquivo **.gitignore** serve para você escrever os nomes dos arquivos que você não quer que fiquem dentro do repositório



Todos os arquivos que terminam com .sql e *.txt serão ignorados pelo git, não serão enviados para o repositório remoto.



Caso você não saiba o que colocar no .gitignore, o github tem um repositório só com nomes de arquivos que deveriam estar no .gitignore.

2:01:07

git revert, é parecido com o git reset, mas ele não perde os commits posteriores daquele commit que você está acessando. O professor disse que é o “salvador das sextas feiras”, porque dá pra resolver o problema naquele momento, sem perder os commits

git revert --no-edit hash_do_commit não modifica nada de verdade

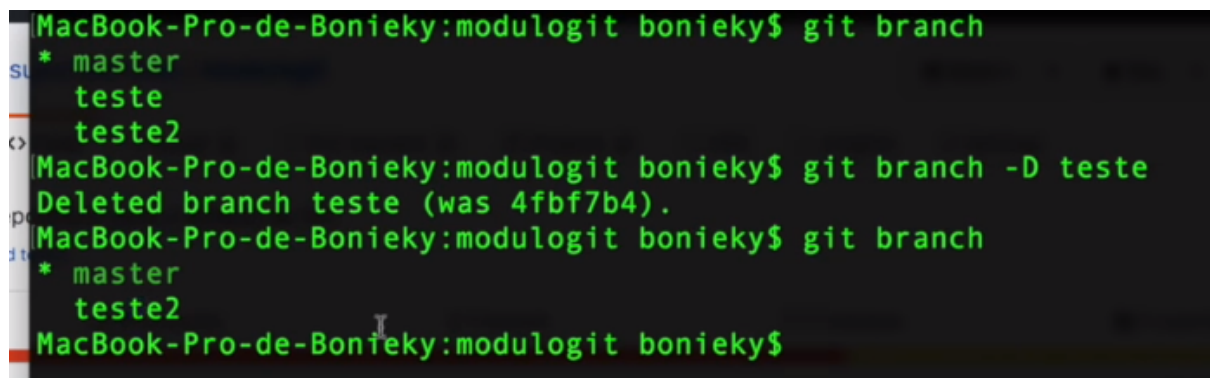
2:07:12

Deletar branches remotos

git push nome_da_branch : nome_da_branch

Deletar branches locais

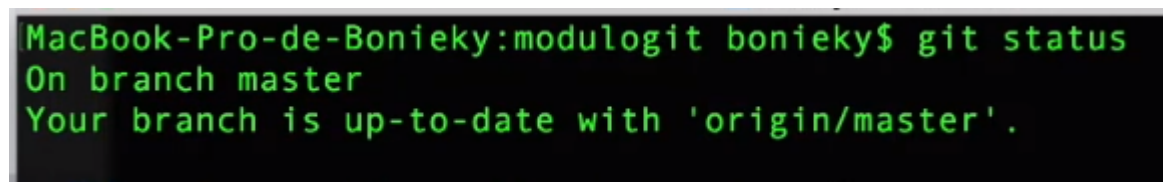
git branch -D nome_da_branch_local



```
MacBook-Pro-de-Boniekky:modulogit boniekky$ git branch
* master
  teste
  teste2
MacBook-Pro-de-Boniekky:modulogit boniekky$ git branch -D teste
Deleted branch teste (was 4fbf7b4).
MacBook-Pro-de-Boniekky:modulogit boniekky$ git branch
* master
  teste2
MacBook-Pro-de-Boniekky:modulogit boniekky$
```

2:13

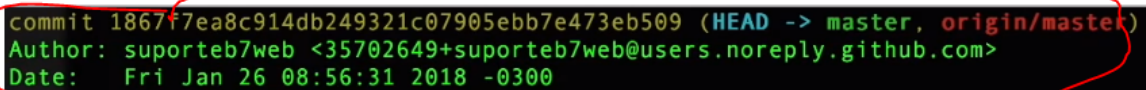
push é quando enviamos algo do repositório local para o repositório remoto



```
MacBook-Pro-de-Boniekky:modulogit boniekky$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
```

A imagem acima significa que o branch do repositório local e do repositório remoto estão Atualizados com o mesmo conteúdo, não tem nada para acrescentar nem decrementar. Mas isso nem sempre é verdade, pois muitas alterações não aparecem no repositório local, alterações essas que estão no repositório remoto (por exemplo, github).

Para isso fazemos o **git pull nome_da_branch_remota nome_da_branch_local** para trazer as informações das alterações da branch remota para a branch local, você estará trazendo informações para seu computador, ele traz o histórico de commits também.



```
commit 1867f7ea8c914db249321c07905ebb7e473eb509 (HEAD -> master, origin/master)
Author: suporteb7web <35702649+suporteb7web@users.noreply.github.com>
Date: Fri Jan 26 08:56:31 2018 -0300
```

Criando o teste.js

```
commit 10c4ce337603bc03a8a4238a6810a72374ed3c57
Author: Bonieky Lacerda <suporte@b7web.com.br>
Date: Fri Jan 26 08:26:55 2018 -0300
```

Revert "Nova funcionalidade X"

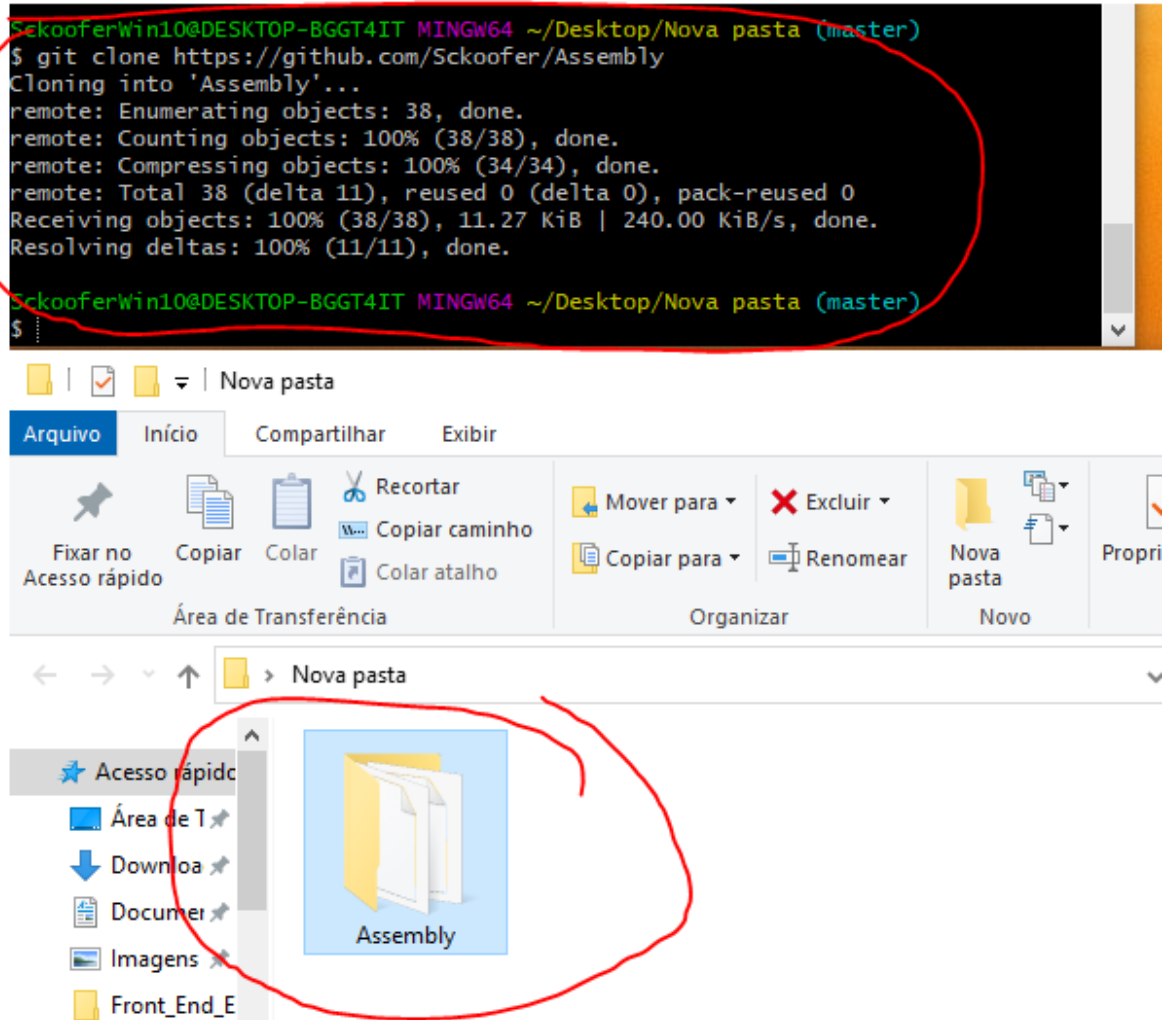
This reverts commit 7bb237921f14cc0b9f3e207e9ea69de86fda9e07.

A imagem acima em vermelho mostra uma alteração feita por outro usuário (aquele nome é fictício, criado pelo github para definir um usuário)

Fica a dica, antes de fazer um commit, você sempre faz um **git pull** para atualizar as informações do repositório local com base no repositório remoto.

2:21:40

Vamos clonar projetos, para isso usamos o comando: **git clone**
endereço_https_do_projeto



Dá pra acessar o log de todos os commits realizados no projeto.

2:26

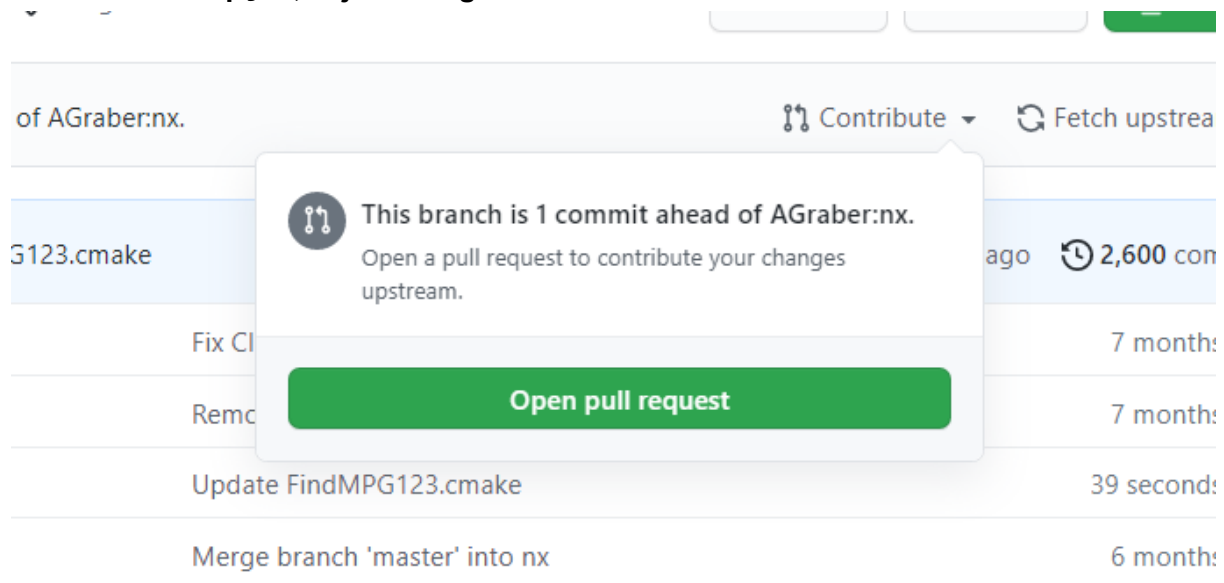
Para saber qual é o nome do servidor remoto: **git remote -v**

Como contribuir para projetos que não são nossos?

Vamos para o github, fazemos o fork do projeto para dentro do nosso github.

Usamos o git clone para realizar a mesma tarefa e trazemos o fork do nosso repositório para dentro da nossa máquina, fazemos as alterações que queremos no projeto e então fazemos o push para nosso repositório remoto.

Para oferecer nossas alterações feitas no fork para o dono do outro projeto (estamos fazendo agora no github) nós vamos lá para o repositório original onde está o projeto e clicamos na opção, veja a imagem abaixo:



Eu alterei o fork que fiz, tem a opção de contribuir para o projeto original se eu clicar em Open pull request

- 1) fazemos o fork
- 2) fazemos o clone para o local
- 3) fazemos as alterações que queremos
- 4) fazemos o push para o nosso repositório remoto
- 5) clicamos em uma opção como a de cima para oferecer às nossas alterações para o dono do projeto original (pull request). Se o dono gostar das alterações ele aceita elas.