**Introdução ao Git**

**Entendendo o que é Git e sua importância**

Criador do Git Linus Torvalds, foi o criador do Linux também. O Git é um sistema de versionamento de código. O Git Hub é um repositório e o Git é um software que complementa o Git Hub.

Vantagens de estar aprendendo sobre Git e Git Hub:

1. Controle de Versão
2. Armazenamento em nuvem
3. Trabalho em equipe
4. Melhorar seu código
5. Reconhecimento

**Navegação via command line interface e instalação**

O Git é um CLI (Command line interface) um programa que usa linhas de comando.

O que vamos aprender?

* Mudar de pasta
* Listar as pastas
* Criar pastas/ arquivos
* Deletar pastas/arquivos

**Comando com Windows e Unix (Linux)**

Windows:

* cd (entra em uma pasta)
* cd .. (volta uma pasta)
* dir (lista os diretórios na pasta atual)
* cls (limpa o terminal)
* mkdir (cria um diretório)
* rmdir (exclui um diretório)
  + passamos um nome do diretório logo após o comando e usamos as FLAGS /S /Q
* echo (printa uma mensagem no terminal)
  + echo “mensagem” > mens.txt (cria um arquivo com a mensagem do echo)
* del (exclui apenas arquivos)

Unix (Linux):

* - cd (entra em uma pasta)
* - cd.. (volta uma pasta)
* - clear (limpa a terminal)
* - Is (lista os diretórios na pasta atual)
* - mkdir (cria um diretório)
* - rm – rf (exclui um diretório com outros dentro)
  + FLAG -R é de recursivo e a FLAG -F é de force, para não pedir nenhuma permissão
* echo (printa uma mensagem no terminal)
  + echo “mensagem” > mens.txt (cria um arquivo com a mensagem do echo)

Todos os comandos tem Flags que modificam, acrescentam ou formatam a forma como os comandos são devolvidos para nós.

A tecla TAB ajuda a autocompletar os nomes.

**Para instalar o Git no Linux Ubuntu vamos usar esses comandos**

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

sudo apt update

sudo apt install git

Para ver qual a versão que foi instalada, basta usar o comando

Git - -version

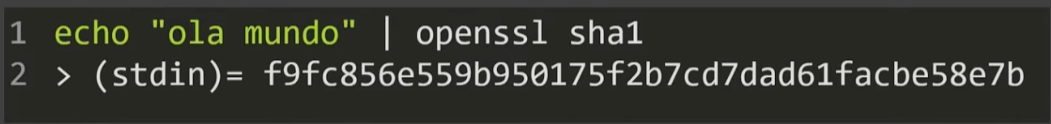
**Entendendo como o Git funciona por baixos dos panos**

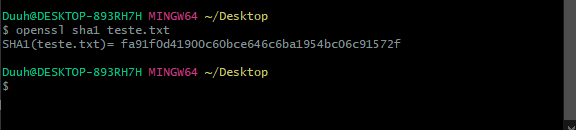
**SHA1**

A sigla SHA significa Secure Hash Algotiyhm (Algoritmo de Hash Seguro), é um conjunto de funções hash criptográficas projetadas pela NSA (Agência de Segurança Nacional dos EUA).

A encriptação gera um conjunto de caracteres identificador de 40 dígitos únicos.

O Git faz uso desse modo de encriptação por ser muito seguro.

É uma forma curta de representar um arquivo.

Para passar um arquivo para o algoritmo de encriptação vamos usar o comando **OPENSSL SHA1** e o nome do arquivo.

Com isso será gerado um identificador único para aquele arquivo da maneira que ele está, caso seja alterado, será gerado outra chave única.

**Objetos internos do Git**

Podemos também criar chaves únicas para objetos no Git, esses objetos são chamados de **BLOBS**, **TREES** e **COMMITS**. Esses são os três tipos básicos de objetos responsáveis pelo versionamento no Git.

1. O objeto **BLOB** vai guardar os arquivos e esse objeto contem meta dados dentro dele. O objeto **BLOB** vai ter o tipo do objeto, um tamanho do arquivo ou string, uma barra ao contraio (\) e o número zero e o conteúdo do arquivo.



1. O segundo objeto é o **TREE**, eles armazenam **BLOBS**. O **BLOB** é o bloco básico de composição, a **TREE** armazenando e apontando para tipos de **BLOBS** diferentes e depois os **COMMITS**.

A estrutura da **TREE** é parecida com a do **BLOB**, porem ele ainda tem o tipo e nome do arquivo e pode apontar para outras **TREES**.

Se o **SHA1** de um **BLOB** mudar e esse **BLOB** estiver dentro de uma **TREE**, consequentemente o **SHA1** dessa **TREE** também vai ser alterado.

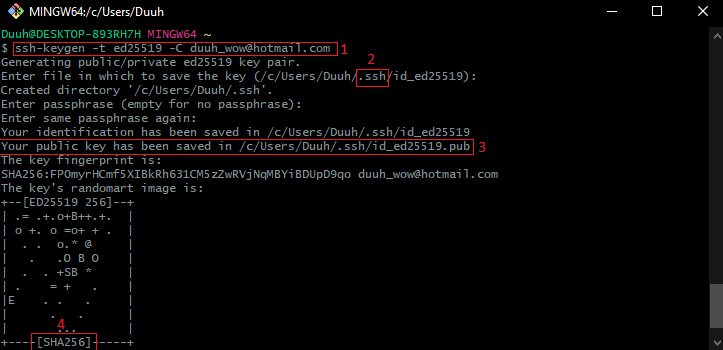
1. O terceiro objeto é o **COMMIT**, o **COMMIT** aponta para uma **TREE**, ele aponta para um parente (**COMMIT** realizado antes dele), ele aponta também para um autor, aponta para uma mensagem e tem um **TIMESTAMP** que é um carimbo de tempo que é a data de quando ele foi criado.

Os **COMMITS** também têm um **SHA1**.

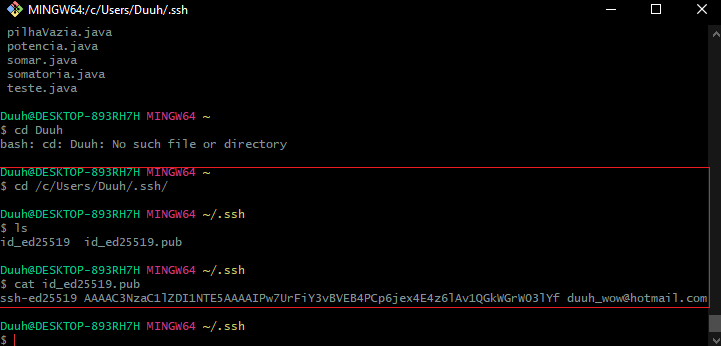
**Chaves SSH e Tokens**

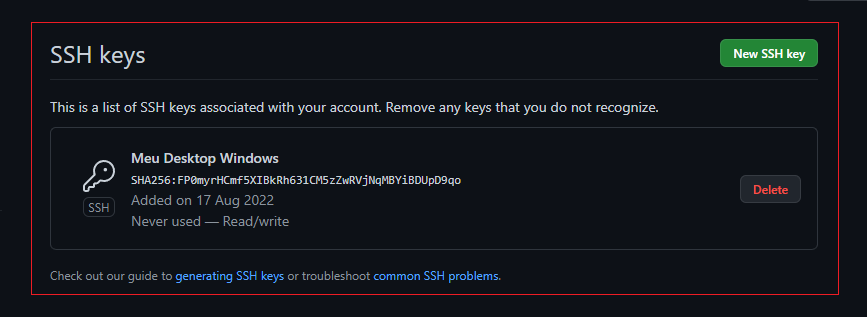
Chave SSH é uma forma de estabelecer uma conexão segura e encriptada entre duas maquinas. Vamos estabelecer essa conexão com o servidor do GitHub e dizer que a nossa maquina é segura. Para isso vamos estabelecer essa conexão com duas maquinas, uma pública e uma privada. Vamos pegar a chave publica e colocar no GitHub e a partir disso o GitHub vai conhecer a nossa maquina e vai ter uma conexão previa.

Para isso, depois de criarmos a conta no GitHub, nós vamos até as configurações no site do GitHub e procurar pela opção SSH and GPG Keys, vamos ter o campo de “titulo” e “Key”. Agora vamos abrir o Git Bash e usar o comando **SSH-KEYGEN -T** e agora vamos passar o tipo de criptográfica da chave junto com o comando anterior, o tipo vai ser ED25519, depois adicionamos o -C e o nosso email que está cadastrado no GitHub



1. Linha de código para estabelecer a conexão com o GitHub
2. Todo diretório que começar com ponto (.) é um diretório oculto
3. Linha que exibe onde a chave pública está armazenada
4. Tipo de criptografia usada.

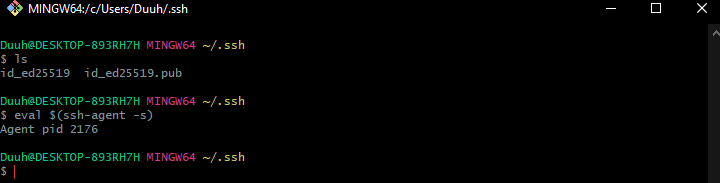
Agora vamos navegar até a pasta onde estão a chaves e vamos verificar se elas estão lá mesmo com o comando LS. Logo em seguida vamos dar um comando CAT e a identificação da chave publica para poder visualizar o conteúdo dela, copiar e colar ela lá naquele campo “Key” no GitHub.



Depois disso vemos essas informações com o novo Titulo da chave, a data e informando que ela nunca foi usada. Porém ainda temos que fazer mais um processo para poder utilizar da melhor forma.

Vamos voltar para o Git Bash e ainda na pasta onde estão as chaves, vamos rodar um programa para lidar com as chaves, esse programa é chamado de **SSH Agent** e para iniciarmos ele vai usar o seguinte comando **EVAL $(SSH-AGENT -S)**

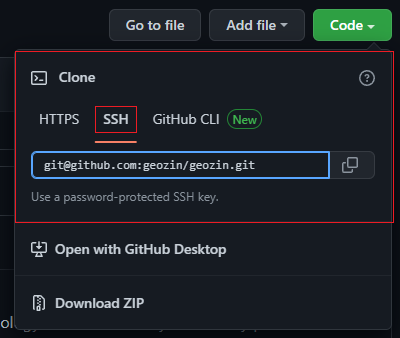
Com isso ele vai gerar um numero do processo que ele está fazendo.



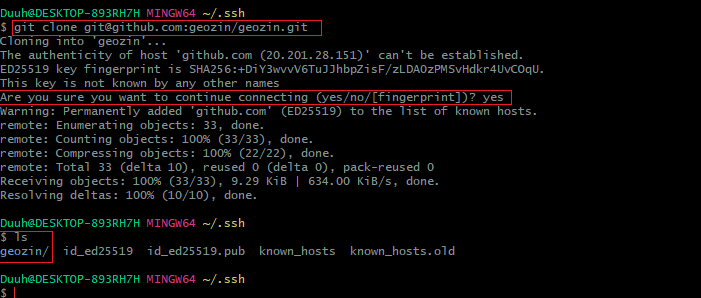
Agora vamos usar o comando **SSH-ADD** e vamos passar para esse comando a nossa chave privada (caso não estivéssemos na pasta onde estão as chaves, precisaríamos passar o caminho para esse comando), com isso toda vez que recebermos uma criptografia com essa chave, ele vai usar a nossa chave privada para descriptografar essa mensagem. Logo em seguida ele vai pedir a **PASSPHRASE** que definimos anteriormente.



Agora vamos clonar um repositório do GitHub. Para isso quando estivermos diante de um repositório vamos clicar no botão verde escrito “**Code**” e vamos ter 3 opções para usar, no nosso caso vai ser a opção SSH, nela vamos copiar o “link” e agora vamos lá para o Git Bash.



Agora no Git Bash vamos usar o comando **GIT CLONE** e passar o link **SSH** que copiamos lá do repositório. Logo em seguida ele vai perguntar se você tem certeza que quer continuar e você pode escolher entre “**yes**” ou “**no**”, no nosso caso vai ser “**yes**” e depois de confirmar ele vai copiar todo o repositório. Se dermos um comando **LS** vemos que temos uma nova pasta com o nome do repositório que acabamos de clonar.

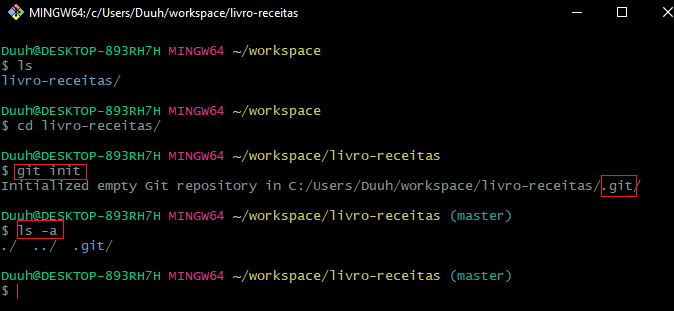


**Primeiros comandos com o Git**

Sempre que formos usar os comandos do Git vamos ter que usar a palavra “**git**” e depois o comando que queremos para poder iniciar a ação.

Vamos criar uma pasta com o comando “**MKDIR**” (no git bash) chamada “workspace” e vamos entrar nela com o comando “**CD**” e nome da pasta. Logo em seguida vamos criar outra pasta chamada “livro-receitas” e entrar nela.

Agora vamos iniciar o Git dentro dela com o comando “**GIT INIT**”, será retornado uma mensagem dizendo que iniciamos o Git em uma pasta vazia, mas se reparamos tem uma pasta chamada “**.git**”, e como vimos anteriormente, as pastas que começam com o ponto (.) estão ocultas e se usarmos o comando “**LS -A**” ele vai retornar todas as pastas ocultas.

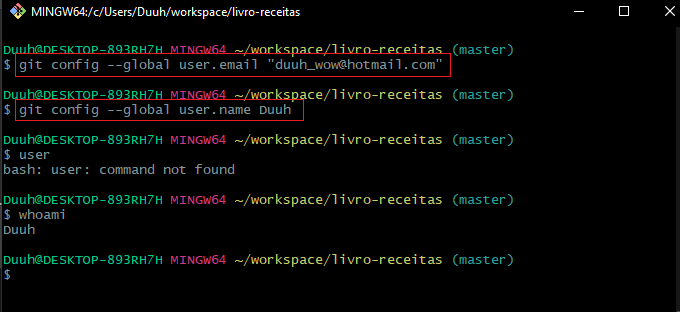


1. Iniciando o Git
2. A pasta oculta
3. Listando a pasta oculta.

Agora vamos configurar um usuário, por que o todo commit precisa de um autor e nesse caso vai ser o que nós vamos configurar.

Primeiro vamos usar o comando “**GIT CONFIG - -GLOBAL USER.EMAIL**“seu email aqui”.

Agora vamos configurar o nome do usuário com o comando “**GIT CONFIG - -GLOBAL USER.NAME** “seu nome aqui”.



**Passo a Passo do ciclo de vida**

Quando usamos o comando “**GIT INIT**”, iniciamos um repositório dentro daquela pasta.

O conceito de **TRACKED** e **UNTRACKED** é o seguinte: **TRACKED** são os arquivos que o Git tem ciência. O **UNTRACKED** são os arquivos que o Git não tem ciência.

Dentro do **TRACKED** temos os arquivos:

* **UNMODIFIED** – arquivos que não foram modificados;
* **MODIFIED** – arquivos que já foram modificados;
* **STAGED** – arquivos que estão prontos para fazer parte de outro agrupamento.

Na aula anterior usamos o comando “**GIT ADD** \*”, para adicionar o arquivo que nós criamos dentro do pasta “Livro-receitas”. Esse arquivo foi da posição **UNTRACKED** direto para o **STAGED**.

Quando fazemos um **COMMIT** desse arquivo que está no **STAGED** ele volta para o **UNMODIFIED** e aguarda novas alterações.

Quando modificamos um arquivo que já está dentro do **TRACKED** ele compara a codificação (**SHA1**) e vê que tem uma modificação e ele sai do **UNMODIFIED** para **MODIFIED**.

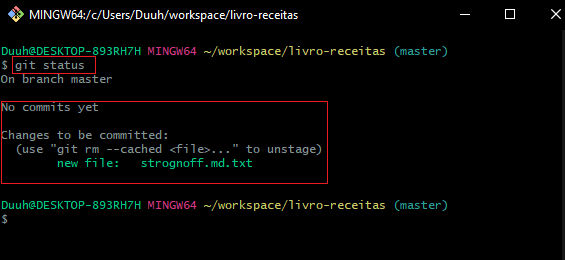
**Entendendo o conceito de repositório**

O Git tem a versão publica dele que é o Git Hub (Servidor), e o Git é o nosso ambiente de desenvolvimento. Quando nós fazemos uma modificação em um arquivo que está no nosso computador, ele vai interferir na versão que está lá no repositório do Git Hub.

Quando fazemos um **COMMIT** de um arquivo, ele sai da **STAGING AREA** e vai para o **LOCAL REPOSITORY** e de lá ele pode ser empurrado para o **REMOTE REPOSITORY** (Git Hub).

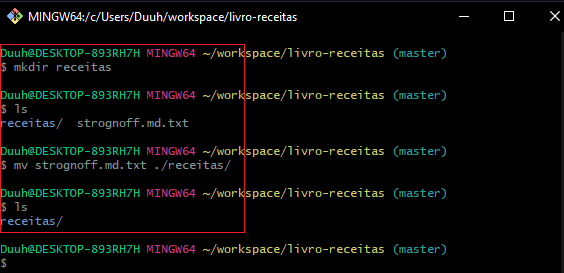
Tudo o que está no **LOCAL REPOSITORY** tem que estar com **COMMIT**, caso contrário não será possível enviar para o **REMOTE REPOSITORY**.

Vamos usar um comando agora no GIT BASH que é muito importante, é o **GIT STATUS**. Esse comando nos ajuda a saber qual a situação dos arquivos naquele repositório local.

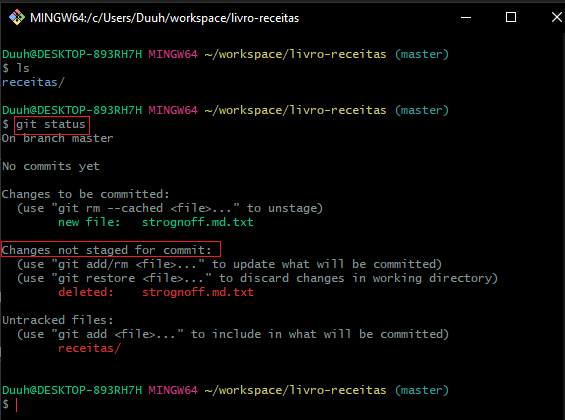


Agora vamos criar um outro repositório chamado “Receitas” e vamos mover esse arquivo para esse novo repositório.

Vamos usar o comando **MV**, o nome do arquivo e depois o ponto, barra e o nome do diretório (./), para dizer que do diretório local vamos enviar o arquivo para o outro diretório escolhido.

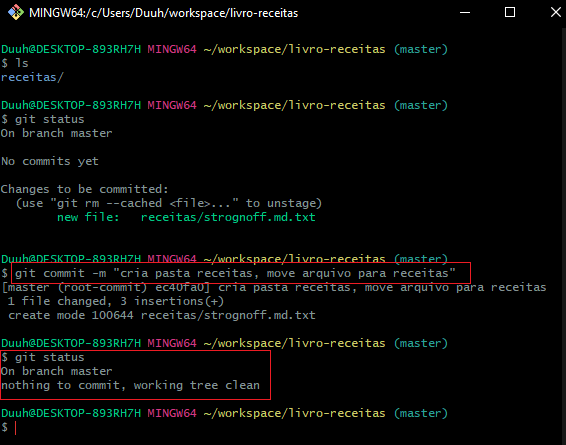


Agora se usarmos o comando **GIT STATUS**, ele vai nos retornar muitas informações e uma delas é “**Changes not staged for commit:**” que quer dizer que as mudanças não foram “**commitadas**” ou seja, não foram enviadas para o repositório local.



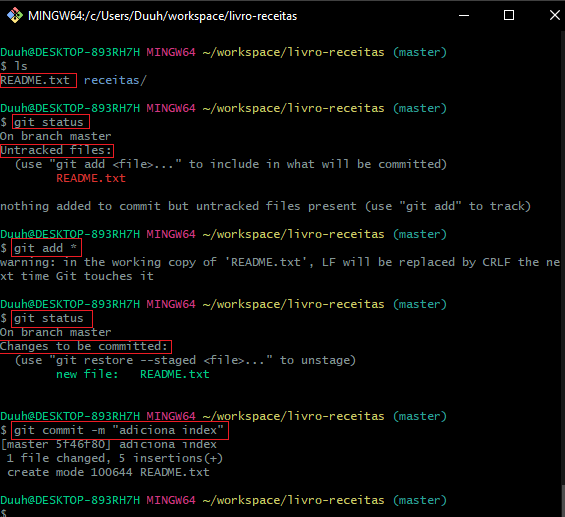
Agora vamos adicionar o arquivo e a pasta com o comando **GIT ADD**, depois o nome do arquivo e nome da pasta.

Agora vamos fazer o **COMMIT**. Primeiro vamos usar o comando **GIT COMMIT -M** (o -m é para dizer que vamos adicionar uma mensagem), depois entre aspas (“ “) vamos adicionar a mensagem.



Se dermos um **GIT STATUS** vamos ver que não temos mais nada no **STAGED**.

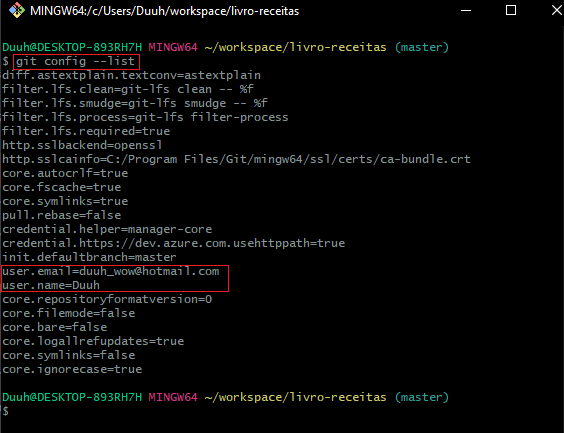
Vamos criar um arquivo chamado **README.txt** com o comando **ECHO**, depois vamos escrever alguma coisa nele. Se dermos um **GIT STATUS**, ele vai aparecer como **UNTRACKED**, que quer dizer que ainda não está sendo rastreado pelo repositório. Vamos usar o comando **GIT ADD \***, para adicionar os arquivos e mudanças deles. E por ultimo vamos criar um **COMMIT**.



**Trabalhando com Git Hub**

Quando criamos uma conta no Git Hub sempre é bom ser igual a que configuramos no Git na parte de configuração de email e nome que usamos lá atras.

Para ver qual email e nome usamos no Git, basta ir até o Git Bash e usar o comando **GIT CONFIG - -LIST**.



Mas digamos que você vá usar outro email e nome de autor para os commits futuros, e queira retirar o autor atual, como vamos fazer isso?

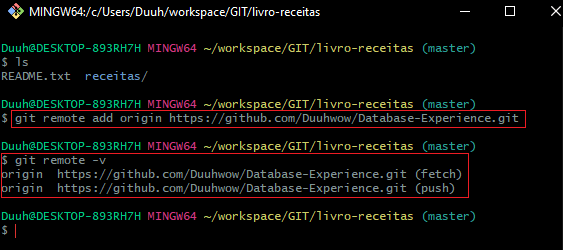
Vamos usar o comando **GIT CONFIG - -GLOBAL - - UNSET USER.EMAIL** e apertar **ENTER** e para **retirar o nome** basta usar o mesmo comando apenas **alterando a parte de email para NAME**.

Caso você tente fazer um **COMMIT** sem um autor, o **GIT** vai pedir para você configurar um novo autor (email e nome) para atribuir ao **COMMIT**.

Agora vamos criar um repositório no servidor do Git Hub e depois de criar ele, vamos ter uma URL desse repositório. Agora vamos no Git Bash e adicionar essa URL ao nosso repositório local.

Vamos usar o comando **GIT REMOTE ADD ORIGIN** e depois a URL que pegamos do Git Hub.

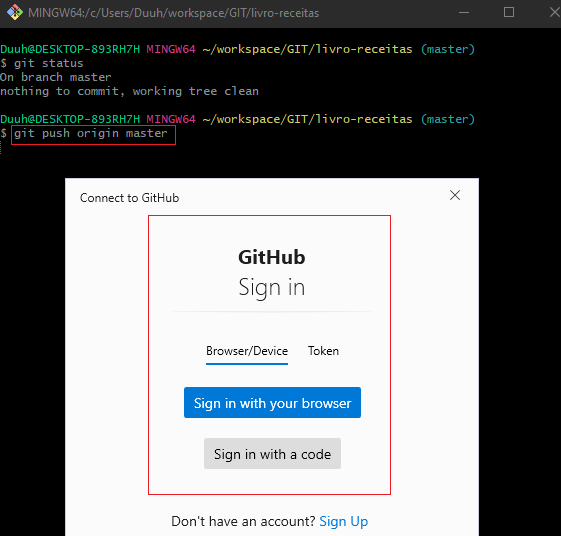
Se não retornar nenhum erro, quer dizer que deu certo. Agora vamos usar o comando **GIT REMOTE -V** e ele vai retornar as listas de repositórios remotos disponíveis.

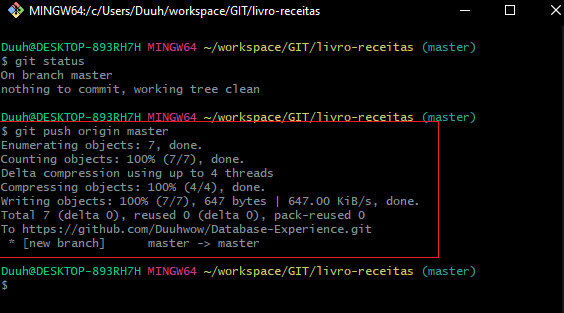


Quando colocamos o apelido **ORIGIN** é nome que damos ao repositório remoto, para que não precisamos digitar a todo momento o link HTTPS do servidor.

Agora vamos empurrar esse nosso repositório local para o repositório no servidor. Para isso vamos usar o comando **GIT PUSH** passar o apelido do repositório que no nosso caso é **ORIGIN** e depois a **Branch** que estamos usando que é a **MASTER**.

Por questão de proteção, o Git Hub vai pedir para que possamos logar na nossa conta e depois de logar ele vai retornar as informações que vão sair do nosso servidor local para o remoto.





Já no site do Git Hub podemos ver o **SHA** resumido e clicando nele vamos ver os **PARENTS** atribuídos a ele, e o histórico de **commits**.

1. **SHA** resumido
2. Histórico de **Commits**.

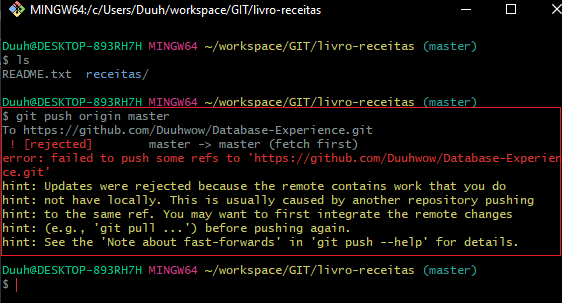


**Resolvendo Conflitos**

Digamos que você e um colega de trabalho pegaram o mesmo arquivo que estava no repositório do Git Hub e começaram a fazer alterações nele, só que o seu colega de trabalho enviou o arquivo para o repositório do Git Hub e agora você está com uma versão desatualizada do arquivo. Quando você tentar empurrar esse arquivo para o Git Hub, ele vai reclamar dizendo que você está com uma versão desatualizada e precisa pegar as alterações que foram feitas e adicionar no seu arquivo para que depois você possa enviar ele para o servidor do Git Hub. Esse erro é conhecido como “**Erro de Merge**”.

Então agora vamos lá no Git Hub e adicionar um arquivo chamado “GIT” só pra deixar o repositório do servidor diferente do nosso repositório local.

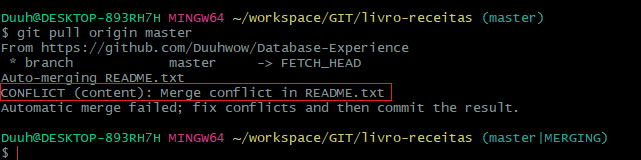
Depois de ter feito isso, vamos adicionar uma informação no nosso arquivo README.txt que está no nosso repositório local e tentar enviar para o nosso repositório no Git Hub e vamos ver o que acontece.

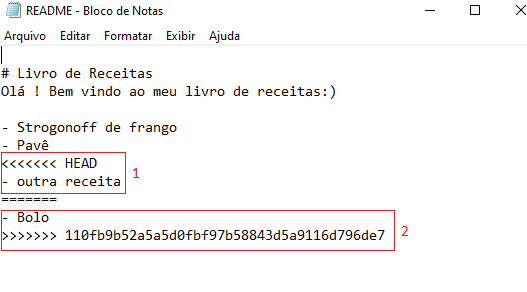


Esse erro se deve ao arquivo que adicionamos lá no Git Hub e deixamos dessincronizado com o nosso repositório local.

Para resolver isso primeiro devemos puxar os arquivos do repositório do Git Hub como comando **GIT PULL ORIGIN MASTER** (quase igual quando enviamos nossos arquivos para lá). E depois de fazermos o commit vamos empurrar de volta para o Git Hub com o comando **GIT PUSH ORIGIN MASTER**.

Agora digamos que tem um arquivo no Git Hub que tem uma modificação exatamente no lugar onde modificamos o nosso arquivo no nosso repositório local. Quando formos tentar empurrar o nosso arquivo ou até mesmo tentar puxar os arquivos do Git Hub para o nosso servidor local, ele vai aparecer um erro de conflito de MERGE.

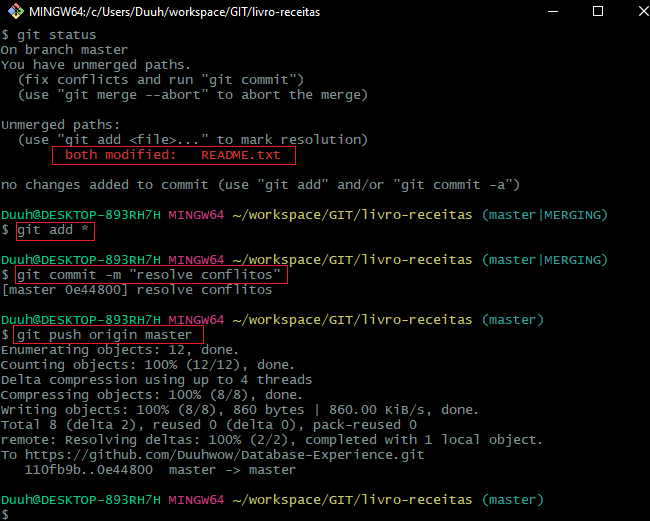


Agora vamos abrir o arquivo em questão para ver o que foi alterado.

No primeiro item é mostrado o que nós alteramos no arquivo no nosso servidor local, já o segundo item mostra o que foi alterado na mesma linha no arquivo que está no servidor remoto do Git Hub.

Então vamos deixar a nossa modificação e excluir a outra (mas nada impede de deixarmos ela também, porem em outra linha). Basta excluir a parte do HEAD, os sinais de igual e a parte do SHA da outra modificação e de pois é só salvar o arquivo.

Agora vamos até o Git Bash e dar um GIT STATUS e vamos ver que temos uma mensagem dizendo que os dois arquivos foram modificados. Isso também é um conflito porque a versão que nós puxamos do Git Hub tinha um arquivo modificado e nós fizemos outra modificação nele. Agora já sabemos qual a versão que vai ficar porque abrimos o arquivo e modificamos ele, agora basta adiciona-lo a área de STAGED, fazer o COMMIT e empurrar ele para o servidor do GIT Hub.



**Clonando um repositório do Git Hub**

Caso a gente queira copiar um repositório do Git Hub podemos ir até o repositório que queremos e no botão verde escrito “CODE” podemos pegar a URL daquele repositório e no nosso Git Bash vamos ir até a pasta que queremos deixar esse repositório e clonar com o comando GIT CLONE e colar a URL que pegamos no Git Hub.

