**Introdução**

Command Line Interface (CLI) X Graphical User Interface (GUI)

O git é CLI.

Windows: cd; dir; mkdir; del/rmdir; cls; tab (autocompleta); echo

Linux: cd; ls; mkdir; rm -rf; clear (ou CTRL+L)

**Windows:**

mkdir workspace (cria pasta workspace)

echo hello > hello.txt (cria arquivo hello.txt e coloca a string hello dentro)

del workspace (vai deletar os arquivos dentro da pasta workspace, mas não a pasta em si)

rmdir workspace /S /Q (deleta a pasta workspace e seus arquivos)

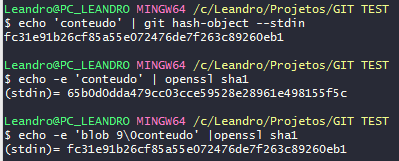
**GIT por baixo dos panos**

SHA1: Secure Hash Algorithm – criada pela NSA para encriptação. Gera um conjunto de caracteres de 40 dígitos baseado no arquivo que está sendo encriptado. É utilizado pelo GIT para identificar os arquivos. Ao comparar arquivos, se os 40 caracteres são iguais, não houve alteração; do contrário, houve alteração e se estaria diante de uma versão diferente do arquivo.

**Objetos internos do GIT**

São 3 objetos internos principais: blobs, trees e commits.

**Blobs**: O GIT inclui nos arquivos metadados, de forma que a encriptação com o SHA1 levará em conta esses metadados. Veja na figura abaixo, que o hash gerado é idêntico entre o git e o openssl somente quando colocamos expressamente os metadados na string que usamos no openssl.



**Trees**: São estruturas que armazenam/apontam para blobs e para outras árvores, também contendo metadados próprios e hash próprio. Se um arquivo de uma blob indicada pela tree for alterado, o hash da blob será alterado e o hash da tree que para ela aponta também será alterado.

**Commits**: São estruturas que contêm referências a árvores, parente (versão anterior), autor, mensagem e timestamp. O commit igualmente possui hash próprio e sofre alterações decorrentes das modificações aplicadas às informações a que se refere.

O Git é um sistema distribuído seguro porque é sempre possível verificar, através do hash, se dois arquivos são idênticos ou se são diferentes, o que permite dizer se um arquivo original foi alterado. Isso permite um controle de versões de arquivos, o que viabiliza que pessoas diferentes utilizem e modifiquem um arquivo sem causar dúvidas na identificação das versões.

**Chaves SSH e Tokens**

No Github, a autenticação deixou de ser por mera senha. Agora há duas possibilidades: chaves SSH e Tokens.

A chave SSH é uma forma de estabelecer uma conexão segura entre duas máquinas (sua máquina e a do Github).

Vamos fazer um procedimento no CLI para obter essa chave que vamos salvar no github:

ssh-keygen –t ed25519 –C [lebarifouse@gmail.com](mailto:lebarifouse@gmail.com)[[1]](#footnote-1)

Se perguntar o arquivo no qual salvar a chave, mantenha o padrão (basta dar enter)

Colocar uma senha

Então, aparecerá onde a chave privada e a chave pública estão salvas.

Navegar até a pasta com o cd e depois usar o comando cat id\_ed25519.pub, de forma a mostrar no terminal a chave pública.

Copiar a chave pública para o github: Entrar no github > settings > SSH and GPG Keys > new SSH key.

Voltar no CLI, na pasta onde estão as chaves. Então, vamos inicializar o SSH Agent com o seguinte comando: eval $(ssh-agent –s)

Em seguida, passar para o SSH a chave privada, digitando o comando a seguir e a senha: ssh-add id\_ed25519

Vamos agora fazer a outra opção, que é a do token de acesso pessoal. Ele se assemelha mais à utilização de senha, pois toda vez que for fazer um commit você terá que copiar e colar o token.

Entrar no github > settings > developer settings > personal acess tokens > marcar repo, colocar uma “nota” (“meu token”, por exemplo) e clicar em gerar token. Quando gerar o token, copiar e salvar em algum local de acesso fácil, mas seguro.

**GIT – PRIMEIROS COMANDOS**

No terminal (GIT BASH), vamos inserir os seguintes comandos:

mkdir livro-receitas (criar diretório para o projeto)

cd livro-receitas/ (adentrar o diretório)

git init (inicializar o git – vai criar os arquivos gerenciais próprios do git e inserir em uma pasta oculta)

ls -a (visualizar as pastas, inclusive ocultas)

git config --global user.email “lebarifouse@gmail.com”

git config --global user.name barifouse

Criar um arquivo markdown com uma receita na pasta criada. Em seguida, voltar ao terminal:

git add \* (adiciona os arquivos ao índice do git)

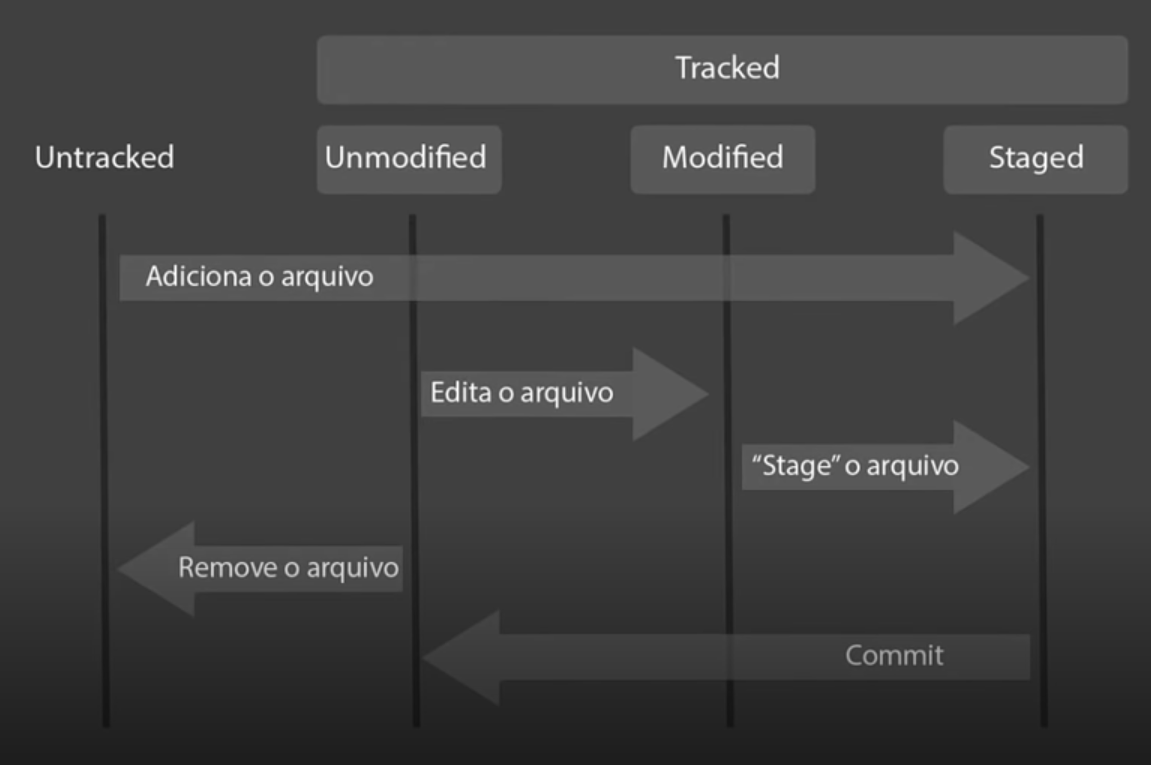
git commit -m “commit inicial” (Salva as alterações no repositório)

**GIT – CICLO DE VIDA DOS ARQUIVOS NO GIT**

O git init inicializa um repositório dentro da pasta de onde ele foi executado.

O git add transforma um arquivo de *untracked* para *staged* ou um arquivo de *modified* para *staged*.

O git commit salva as alterações, transformando um arquivo *staged* em *unmodified*.





O comando git status permite consultar os estados dos arquivos, isto é, em que estado do ciclo de vida eles estão.

Vamos criar uma pasta com as receitas e mover o arquivo já existente para essa pasta:

mkdir receitas

mv strogonoff.md receitas

git status (vai mostrar o strogonoff.md como deletado, mas é porque não consta mais na pasta onde ele estava antes)

git add strogonoff.md receitas/

git status (mostrará as mudanças feitas e que estão *staged* para poderem ser objeto de *comitt*)

git commit -m “cria pasta receitas, move arquivo para receitas”

Vamos criar o README.md (vai ser nosso índice).

ls receitas/

echo > README.md

git status (mostra que o README está untracked)

Vamos adicionar texto em markdown no README e depois vamos inserir os seguintes comandos:

git add README.md

git commit -m “adiciona o índice”

**GITHUB**

Verificar configurações no git, remover os dados do usuário e atribuir igual ao que você vai usar no GITHUB:

git config --list

git config --global --unset user.name

git config --global --unset user.email

git config --global user.email “lebarifouse@gmail.com”

git config --global user.name barifouse

No site do GITHUB, criar um repositório é bem intuitivo e, após a criação, o próprio GITHUB te ensina a fazer o envio do repositório local para o remoto. Seguem as instruções fornecidas pelo GITHUB para nosso projeto criado:

git remote add origin git@github.com:lebarifouse/livro-receitas.git (origin é um *alias*)

git branch -M main

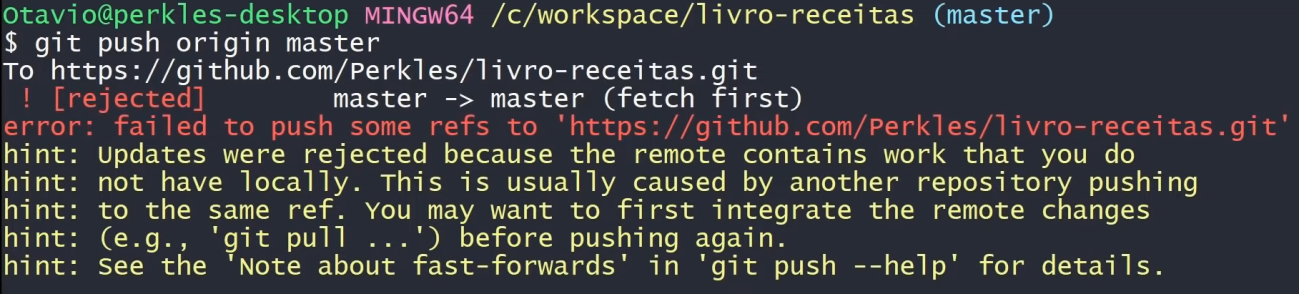
git push -u origin main

É possível usar o comando abaixo para verificar como está o repositório remoto:

git remote -v

**RESOLVENDO CONFLITOS NO GITHUB**

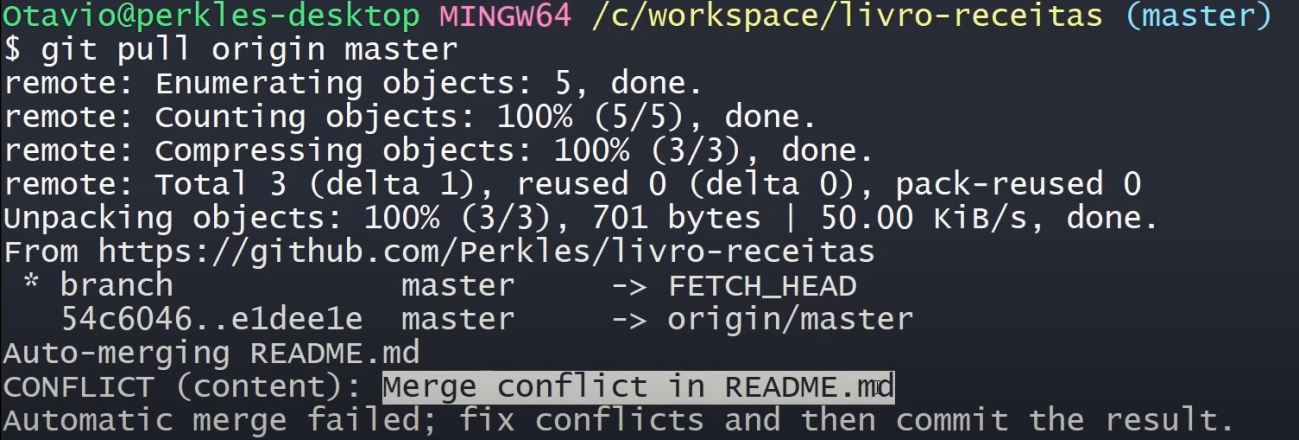
Ao tentar fazer o push de um arquivo que está desatualizado em relação ao arquivo que está no github, é obtida uma mensagem de erro.



Então, vamos puxar para o repositório local aquele arquivo que está no github com:

git pull origin master

Se não for possível fazer o merge dos dois arquivos, teremos um alerta:



Será necessário resolver os conflitos manualmente no código antes de fazer um novo *commit*. Após ajustar o código, é só aplicar o add e o commit:

git add \*

git commit –m “resolve conflitos”

git push origin máster

**CLONAR REPOSITÓRIOS GITHUB**

Basta usar a URL fornecida pelo github no comando clone:

git clone https://github.com/python/cpython.git

1. E-mail que estiver cadastrado no github. [↑](#footnote-ref-1)