**Introdução ao Git e ao GitHub**

-Git: criado em 2005, sistema de **versionamento** (estratégias para gerenciar as diferentes versões de um código) distribuído. É uma forma de administrar as mudanças que são feitas e garantir mais segurança na transição de uma versão pra outra.

-Benefícios de usar Git e GitHub:

● Controle de versão (git)

● Armazenamento em nuvem (github)

● Trabalho em equipe (pessoas do mundo todo podendo opinar nos seus projetos)

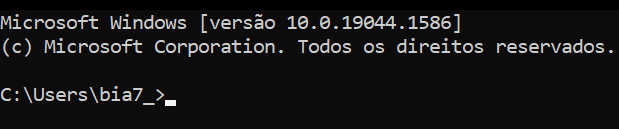
● Melhorar seu código

● Reconhecimento

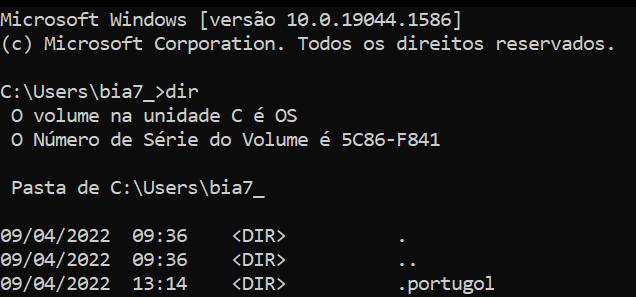
\*Navegação básica no terminal e instalação

-GUI x CLI: GUI são os programas normais, onde o usuário clica e navega por paginas. CLI são os programas que são utilizados através de linhas de comando, o Git é um deles.

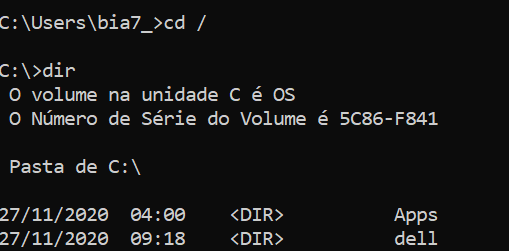
-Para abrir o terminal do Windows: botão Windows → cmd → Prompt de comando

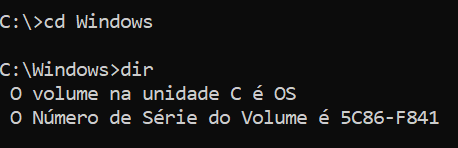


-**dir**: mostra todos os diretórios (pastas) situados dentro dessa pasta “bia7\_”



-**cd /**: leva pra base do diretório C: (antes mesmo do usuário “bia7\_”)



-**cd Windows**: vai pra pasta do Windows, escrevendo dir nela aparecem todas as pastas dentro dessa pasta. 

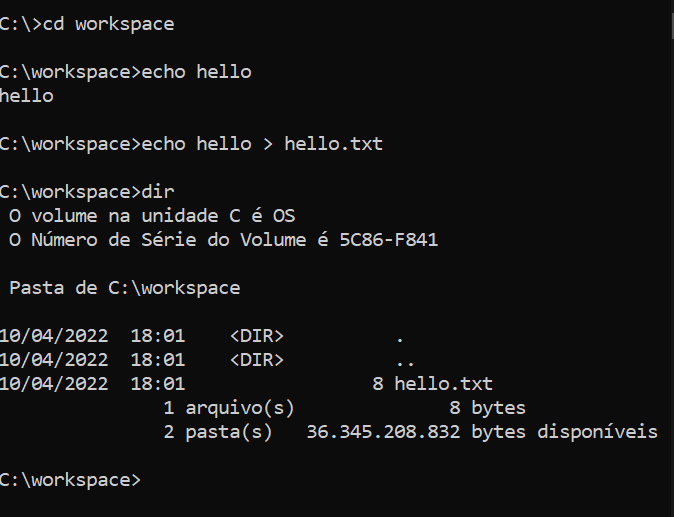
-**cd ..** : volta um passo (ex: no caso do cd Windows, ele volta pra C:\> )

-**cls**: limpa o terminal voltando para o C:\>

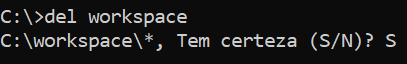
-**TAB**: auto-completa (ex: C:\>cd W **+ TAB** vai completar o W com Windows automaticamente, pois reconhece que dentro dessa pasta C: tem uma pasta Windows)

-**mkdir nome**: cria uma pasta (repositório) com o nome que vc colocar

- **>** : redirecionador de fluxo (ex: echo hello **>** hello.txt vai criar um arquivo de texto dentro da pasta em que vc está se não houver nenhum arquivo .txt com o mesmo nome)



-**del**: deleta os arquivos dentro da pasta em que vc está (pelo comando, parece que está deletando a pasta workspace inteira, mas na vdd deleta só os arquivos dentro dela)



-**rmdir workspace /S /Q**: deleta a pasta workspace e tudo que tem dentro dela



\*Como o Git funciona por baixo dos panos

-SHA1 (Secure Hash Algorithm): conjunto de funções hash criptográficas projetadas pela NSA. A encriptação gera conjunto de caracteres identificador de 40 digitos, que é único para cada arquivo e muda a cada modificação do arquivo.

**\*Objetos internos do Git**

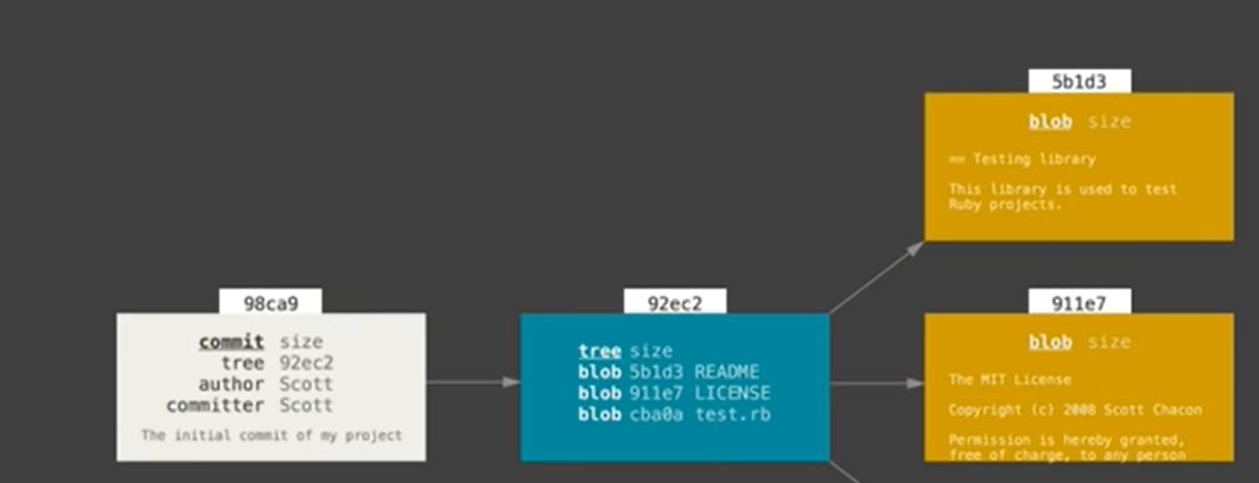
**-Blob: armazena metadados do Git (tipo do objeto, tamanho do arquivo, SHA1 do objeto, etc);**

**-Tree: contem metadados, podem apontar para blobs ou outras trees, armazenam tbm o nome dos arquivos do blob (o blob não armazena). Se mudar uma virgula de um arquivo dentro de um blob, o SHA1 da tree tbm muda;**

**-Commit: aponta para uma tree, pra um parente, para um autor e uma mensagem (que explica o arquivo contido nela).**

O Git é confiável pois com um commit, ele prova que ninguém mexeu naquele arquivo. É impossível alterar um código de um commit sem ser percebido pois tudo fica armazenado.

Exemplo de blobs, trees e commits:



-Pq o Git é tão seguro? R: Suponha que vc tenha seu código hosteado na nuvem. O código que está lá representa o final do seu código, a versão mais atualizada do seu programa. Se tiver 40 pessoas mexendo nele e o mantendo atualizado, tanto a versão mais recente quanto qualquer uma dessas 40 versões distribuídas nas maquinas são confiáveis pois o commit garante que nenhuma outra pessoa modificou.

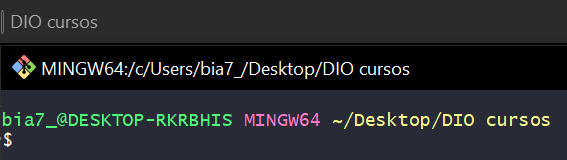
\*Chave SSH e Token

-Chave SSH: forma de estabelecer ligação segura e encriptada entre 2 maquinas (a sua maquina com a maquina do site). Com a SSH o Github entende que sua maquina é confiável e tbm entende que essa chave é só sua.

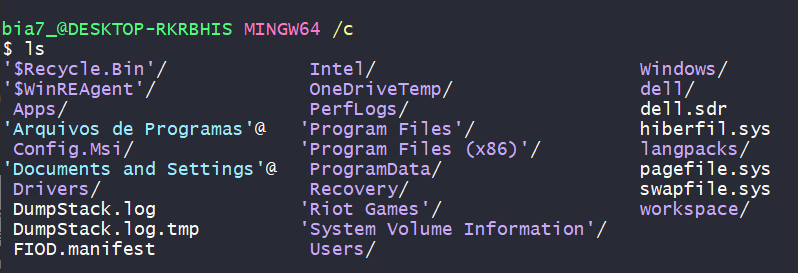
-Token de acesso pessoal: senha pessoal (diferente da senha do site), boa pra usar em pcs que não sejam o seu pois a chave SSH é apenas pra sua maquina pessoal onde só vc usa).

\*Primeiros comandos com o Git

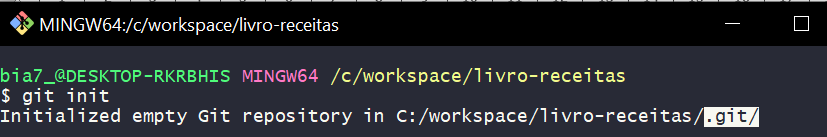
Quando estamos lidando com um terminal, coloca-se sempre o nome do programa na frente pro terminal saber de qual programa puxar as coisas.

Se quiser abrir o Git Bash direto na pasta onde vc está, é só clicar com o botão direito e em “git bash here”, adiantando o trabalho de vc ficar procurando a pasta dentro do Git. Ex do Git aberto direto na pasta “DIO cursos”:  


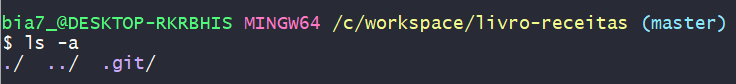
-**ls**: lista tudo que tem na pasta



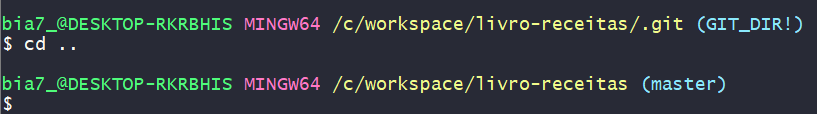
-**git init**: inicializa o git e começa a gerenciar e versionar o código. A pasta .git é oculta (onde fica todo o código do git), por isso ela não aparece se digitarmos **ls**

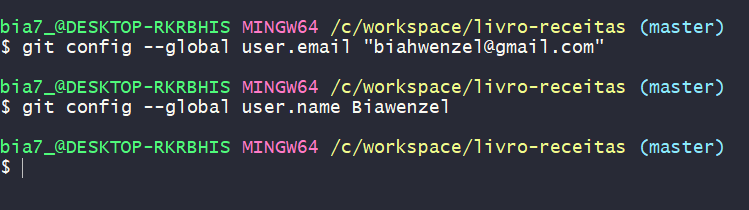


-**ls -a**: comando para listar até as pastas ocultas dentro de pasta.



-**cd ..**: volta um nível na pasta (nesse caso, voltou da .git para a livro-receitas).

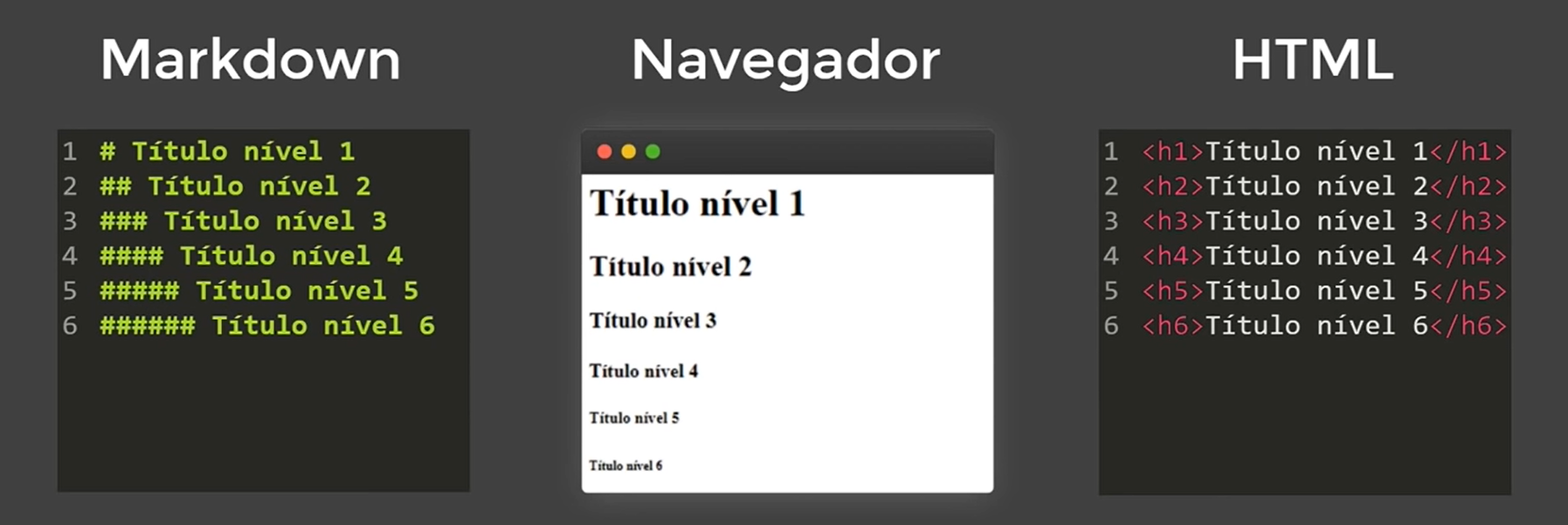


-criando um usuário para “assinar” os commits criados no git:  


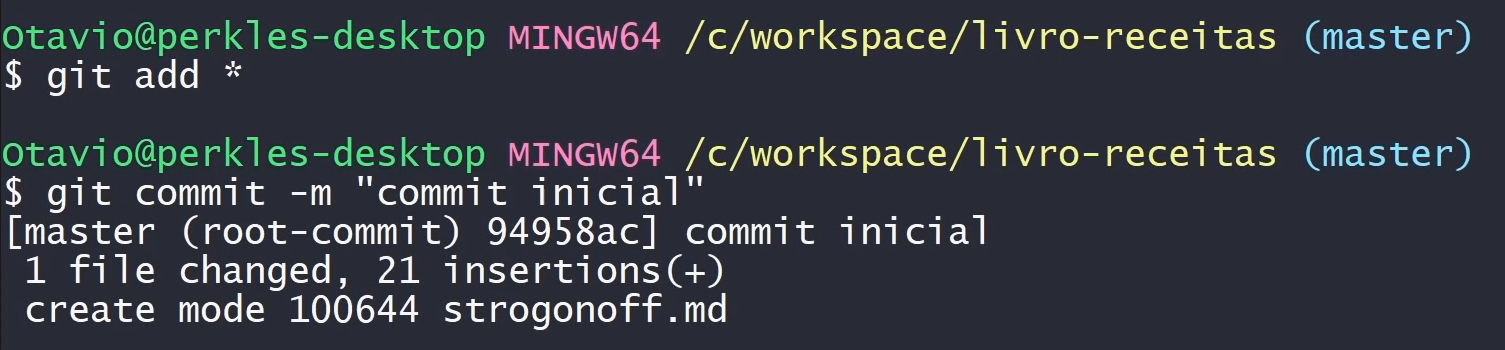
-Adicionando um arquivo:

● Markdown (forma mais humana de escrever um arquivo html). Todo arquivo Markdown tem .md no final

# -> h1, ## -> h2, etc...



-**git commit -m “commit inicial”**: criou a primeira commit nessa pasta e com o nome, sabemos que é a primeira commit.



● 94958ac] : início do SHA1 dessa commit

\*Ciclo de vida dos arquivos no Git

-git init: cria um repositório do git dentro daquela pasta.

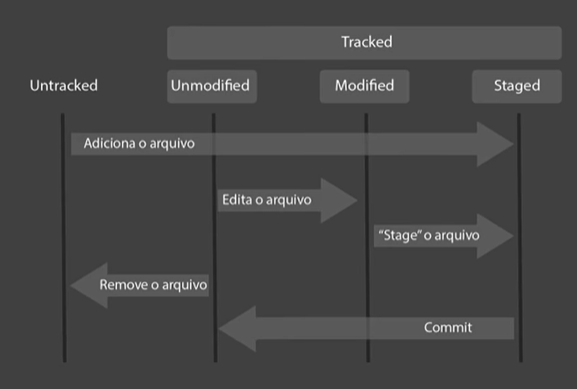
-Tracked: arquivos rastreáveis pelo Git. Podem ser divididos em:  
● Unmodified (não modificados);

● Modified (modificados);

● Staged (onde ficam os arquivos que estão se preparando para fazer parte de outro agrupamento, de um commit).

● O arq passa por um ciclo entre essas 3 fases: se vc faz uma modificação, ele vai pra modified, se add ele a um commit ele vai pra staged, se vc escreve o commit certinho ele volta pra unmodified e assim por diante...

-Untracked: o arq não é rastreável (pois o Git nem sabe da existência dele ainda), mas qnd vc usa o ‘git add’ vc manda ele direto pro ‘staged’ pois ele está “aguardando pra entrar em cena”.



-Repositórios:

● Ambiente de desenvolvimento: tudo que está na nossa maquina;

● Servidor: o Git tem o Github como servidor.

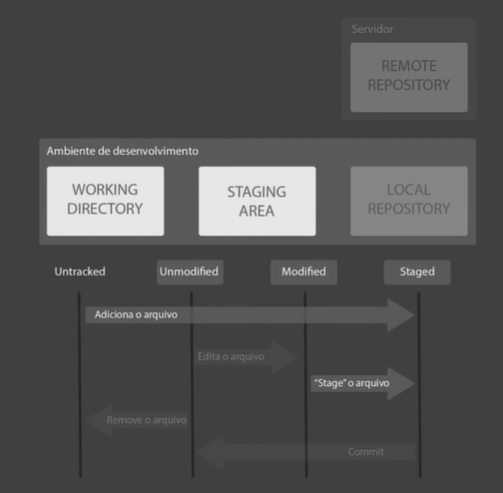
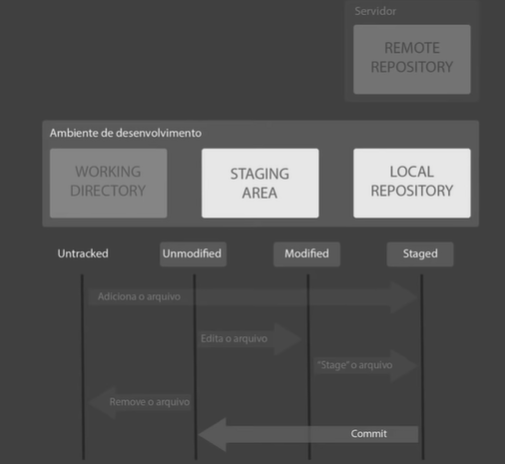
Quando vc modifica alguma coisa na sua maquina, ela não repercute imediatamente na versão que ta no servidor.



● Working directory: seu ambiente, o que estamos manipulando agora (livro-receitas);

● Staging área

● Local repositor: quando vc faz um commit, vc “empurra” o arquivo pra ele. E dele, vc pode “empurrar” os arquivos para um repositório remoto.

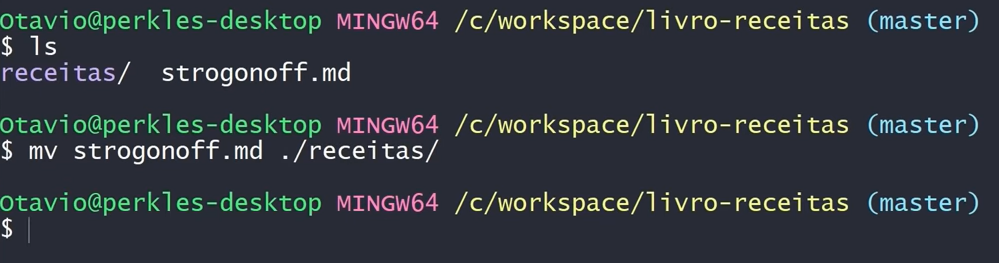
 

● Quando vc add um arquivo untracked com ‘git add’ ele é movido pra ‘staged’. Quando tem um arq ‘modified’ com ‘git add’ ele tb é movido pra ‘staged’. Os arq transitam entre working directory e staging área. (fig 1)

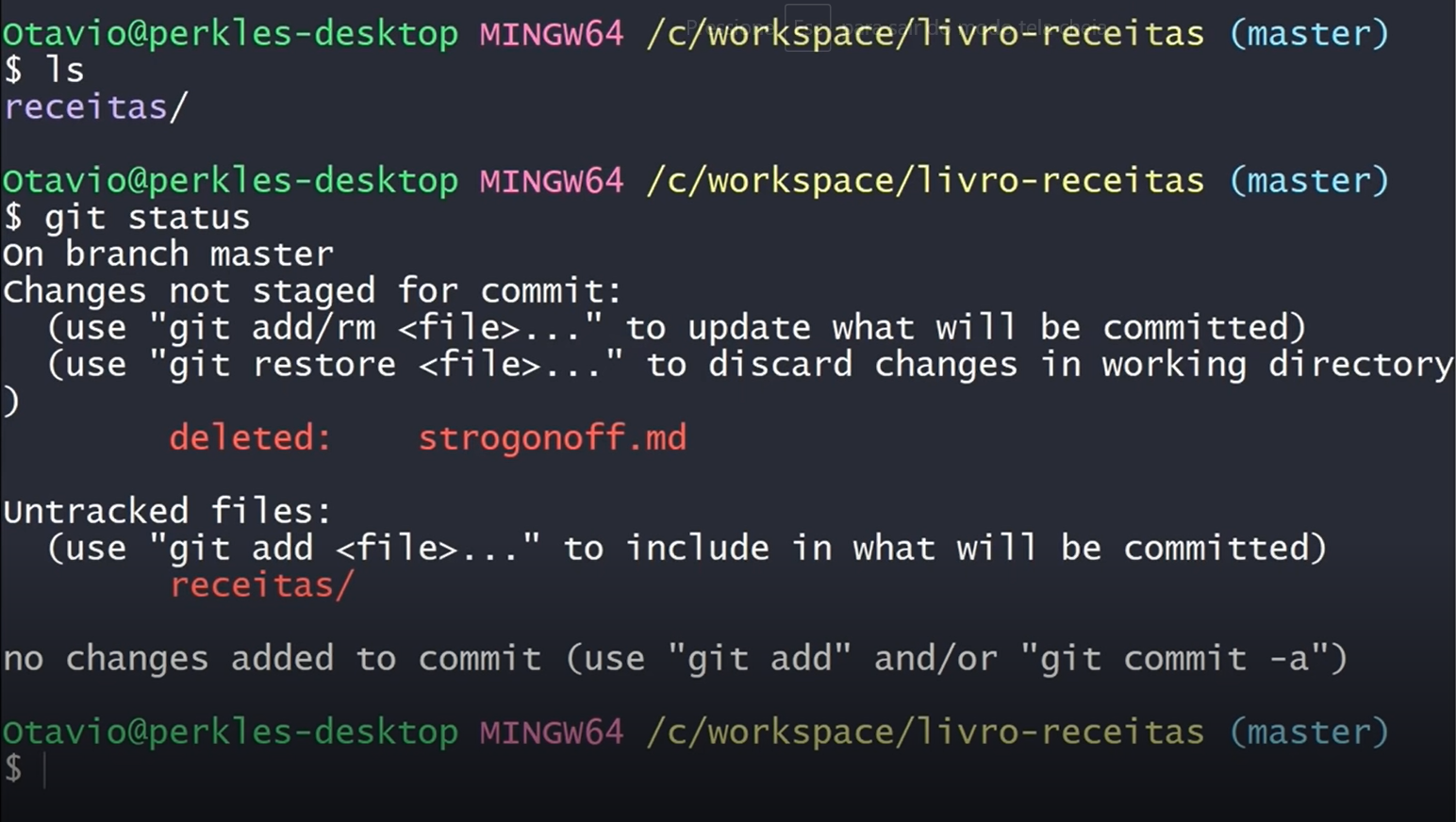
● O arq tb move da staging área quando vc commita ele. Quando vc commita os arq, vc tira eles do staging e manda pro unmodified e pro local repositor. Tudo que estiver no ‘local’ precisa estar commitado, se não não pode ser enviado pro ‘remote’.

-**git status**: monitora os status dos arquivos daquela pasta (se ta staged, modified, etc)

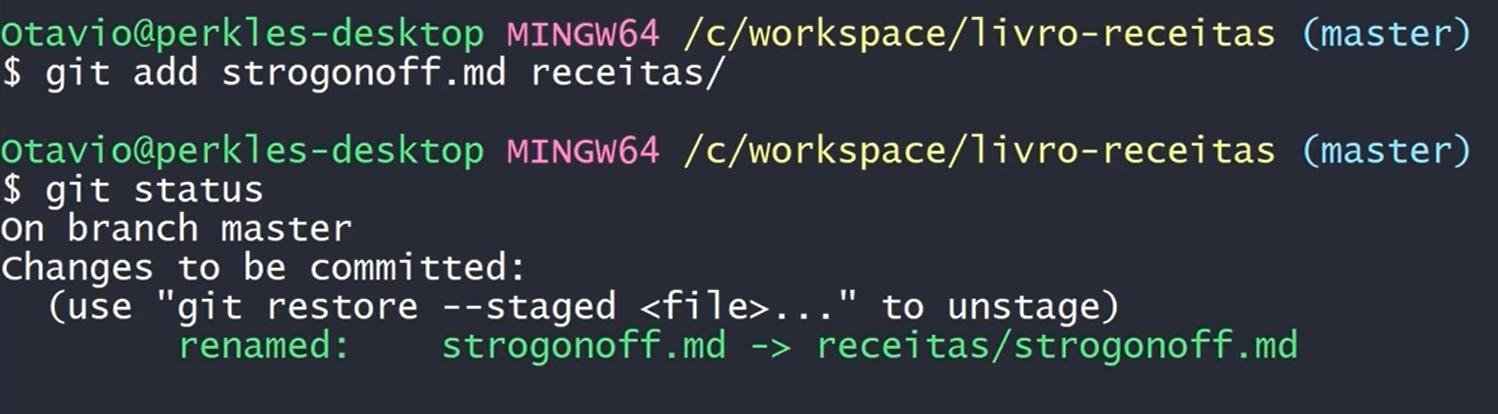
-**mv**: move o arquivo dentro do próprio git bash.



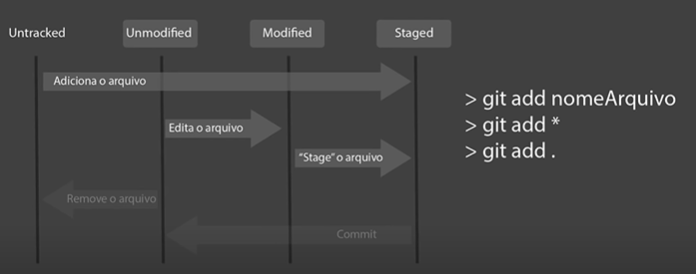
● Como a pasta ‘receitas’ foi criada depois de termos feito o commit com o arq ‘strogonoff’, o git não reconhece a pasta, por isso ela aparece como ‘untracked’



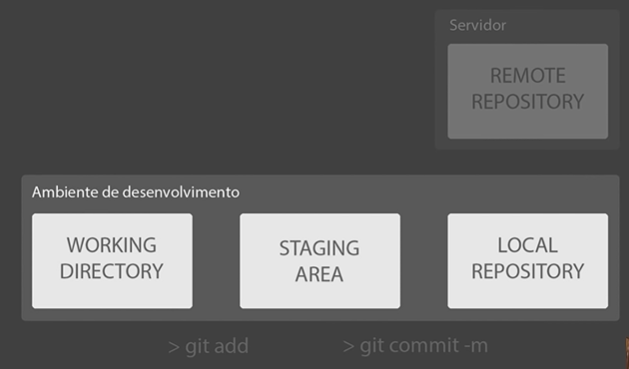
● Se der um “git add <file>...” inclui na área staged para ser commitado.



-**git add \***: pega tudo que foi modificado no ‘working directory’ e manda pro stage pra poder commitar.

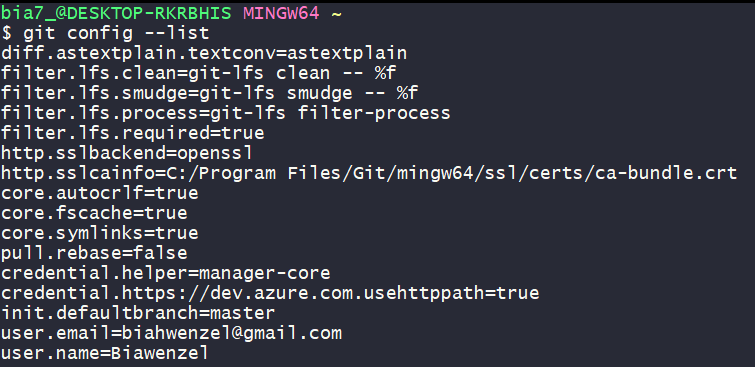






\*Trabalhando com o GitHub

-**git config –list**: lista todas as suas configurações dentro do Git bash (bom que email e name sejam os mesmos do GitHub!)



-**git config –global –unset user.email**: para modificar seu email configurado no Git (para mudar o nome, usa .name, etc...);

-Pra colocar um arquivo da sua maquina num repositório remoto:

1) criar um repositório no GitHub;

2) digitar **git remote add origin <link do repositório>** (obs: ‘origin’ é o apelido desse link, pra não precisar digitar sempre, vc digita o apelido que vc deu, nessa caso, origin)

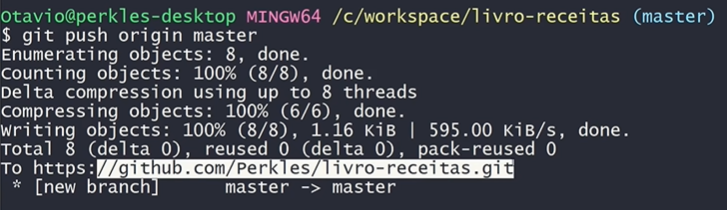


3) **git push origin master** -> enter

4) o Git bash vai pedir sua autenticação do GitHub -> enter



5) seu arquivo já está no repositório online!



Arquivo no GitHub:



-Dá pra criar arquivos dentro da própria plataforma também (GitHub).

\*Como resolver conflitos no GitHub

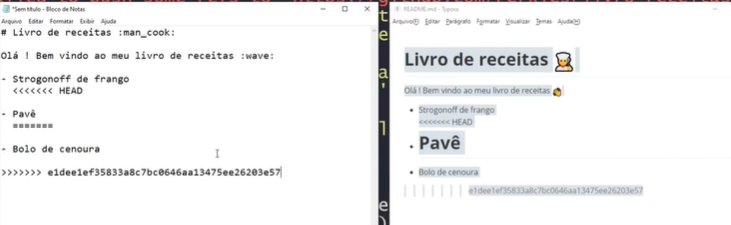
-Supondo que uma pessoa baixe seu arquivo do github pra maquina dela, no primeiro momento vcs tem exatamente o mesmo arquivo, mas cada um de vcs faz modificações diferentes (o código do github é um, da sua maq é outro e da maq da pessoa é outro). Se a pessoa termina a modificação dela e posta o código novo no github, o seu código ficará desatualizado (pois o novo código valido é o da pessoa). Quando vc for colocar seu código no github, vc vai ter que puxar a versão do github, alterar ela (juntar sua modificação com a da pessoa) e aí subir o código novo pro github. É aí que acontece o **conflito (quando existem duas modificações na mesma linha).** O github vai deixar que vc abra o arquivo e vc mesmo resolva o arquivo e devolva o código pro github.

-**git pull origin master**: puxa o conteúdo do github pra sua maquina

-Quando vc puxa o arquivo do github, ele mostra quais alterações vc fez e quais foram feitas por outras pessoas (por isso o conflito) com marcações específicas.

● <<<<<<< HEAD: mostra a alteração mais recente (nesse caso, ‘pavê’)

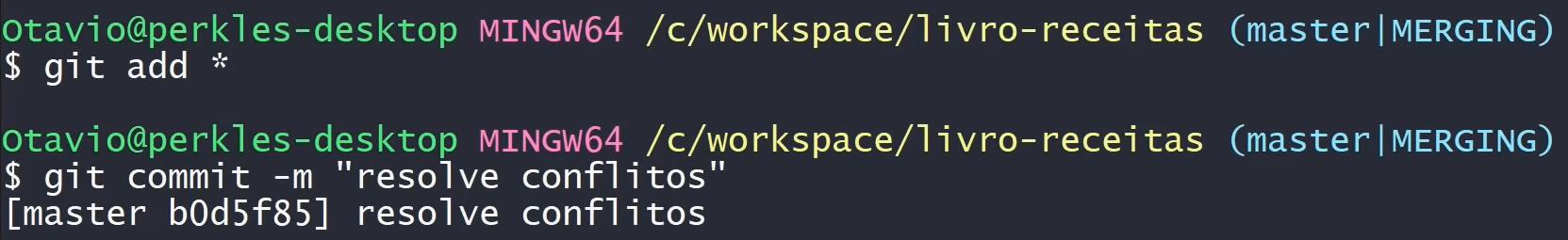
● ======== : mostra a alteração feita por outro usuário (nesse caso, ‘bolo de cenoura’ foi add no mesmo lugar onde vc add ‘pavê’)



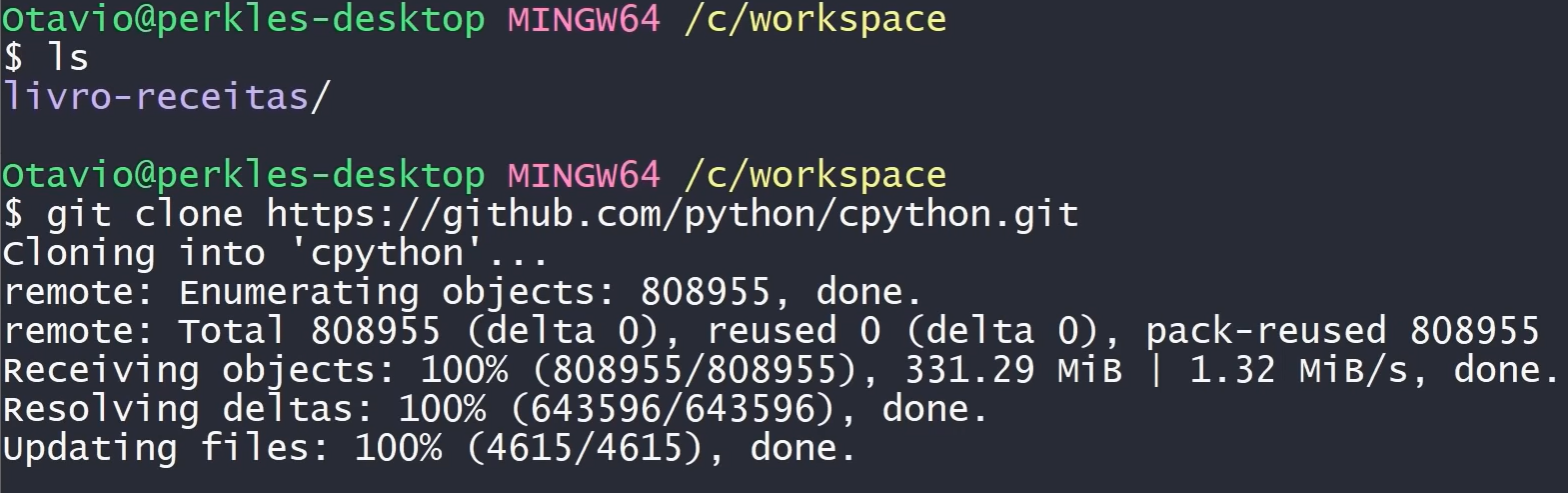
● Se você aceitar a alteração da outra pessoa, basta deixar o bolo de cenoura tbm, mas o importante é que vc está mexendo no arquivo mostrando pro github que vc aceita as modificações e que o arquivo correto agora é esse, com pavê e bolo de cenoura:



-Após fazer as modificações necessárias no arquivo que estava em conflito, tem que criar a commit pra add de novo no site



-Para **copiar um repositório inteiro** do github: **git clone <url do repositório>**



-A diferença entre uma **pasta comum e um repositório**, é que o repositório contem uma pasta .git (oculta) que só conseguimos ver dando **ls -a** dentro do repositório.