# Professor

Otávio Reis – Web Developer

# Entendendo o que é o Git e sua importância

Criado em 2005

Sistema de versionamento de código distribuído

Criado pelo Linus Torvalds

Concorrente: BitKeeper,

CVS – Sistema de versionamento antigo.

GitHub – (Microsoft) – Repositório online

GitLab – concorrente do GitHub.

# Comandos básicos para um bom desenvolvimento no terminal

GUI x CLI

GUI (Graphic User Interface) = Interface gráfica do usuário

CLI (Comand Line Interface) = Interface comando do usuário

bach

## Windows

- cd

- dir

- mkdir

- del/rmdir

- cls

- rmdir nomepasta /s /q

## Unix

- cd

- ls

Mkdir

rm -rf

- clear ou (Ctrl + L)

- rm -rf (o r é de recursivo)

cd / - vai para a raiz do sistema

cd Windows – Estando na raiz, estaremos entrando no diretório Windows

cd.. – volta um nível acima.

TAB – auto completa

Seta para cima – reescreve os últimos comandos.

Silence on sucess - Silêncio no sucesso – Quando dá certo, não mostra nada.

echo hello - exibe hello no terminal

> redirecionador de fluxo.

echo > hello.txt

echo hello > hello.txt

Recursivo, verifica se dentro da pasta tem mais arquivos e f (force) força a exclusão.

Flag’s – variantes – complementos

Windows sub-system for Linux ?

# Ressaltando as principais diferenças entre os sistemas operacionais

Instalação do git

# Tópicos fundamentais para entender o funcionamento do Git

Sha1 – algoritmo de encriptação de 40 bits

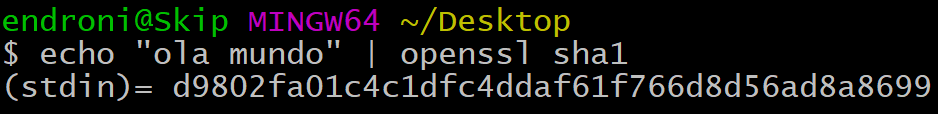
SHA1

A sigla SHA significa Secure Hash Algorithm ( 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.

Exemplo:

Echo “ola mundo” | openssl sha1



Exemplo 1:

openssl sha1 texto.txt

texto.txt = é o nome do arquivo.

Objetos fundamentais

Sistema distribuído

Segurança

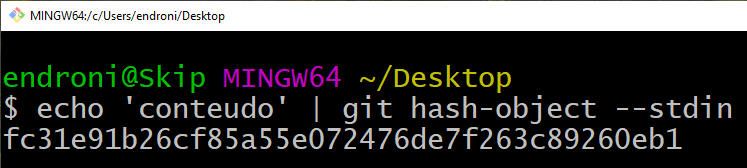
# Objetos internos do Git

BLOBS

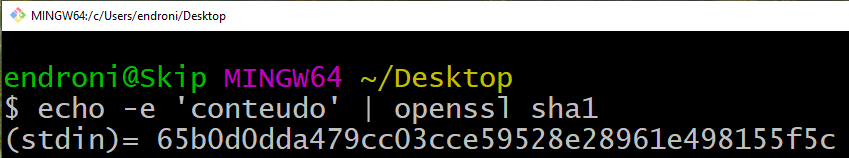
TREES

COMMITS

echo 'conteudo' | git hash-object –stdin



echo -e 'conteudo' | openssl sha1



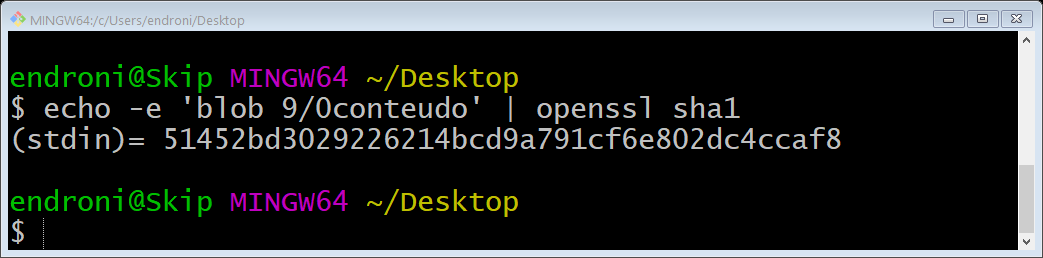
Blob = bolhas

É onde os arquivos ficam armazenados. Os blob’s possuem metadados. Ou seja. Tipo do objeto, tamanho do arquivo, tamanho da string, etc.

O blob representa o arquivo.



echo -e 'blob 9/0conteudo' | openssl sha1



TREE = Arvores



As tree’s armazenam blob’s.

As tree’s armazenam metadados.

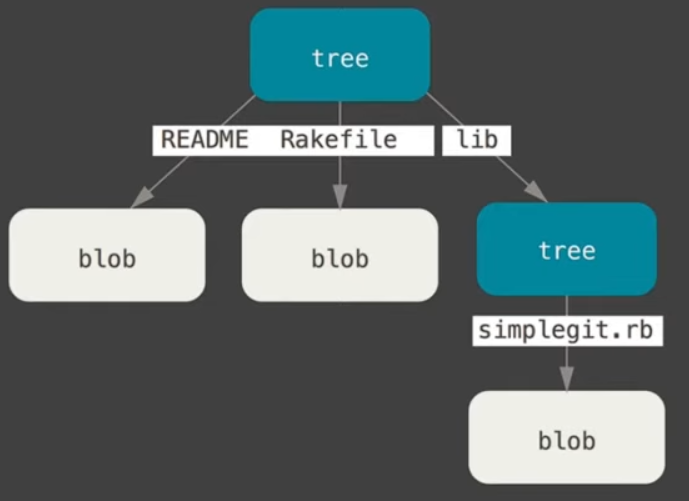
As tree’s apontam para um blob que tem um sha1 e guarda o nome do arquivo.

A árvore é responsável por montar toda a estrutura de onde estão armazenados os arquivos. Pode apontar para outras árvores ou outros blob’s.

A árvore é recursiva.

A árvore também possui um sha1 dos seus metadados.

O blob é o tipo básico.



Commit

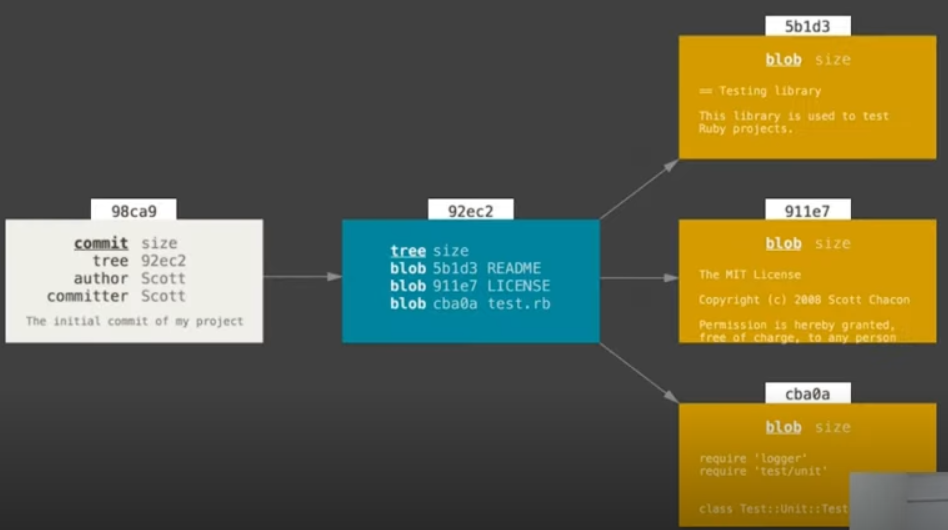


timestamp = data hora de quando foi criado.

O SHA1 desse commit é o hash de toda essa informação.

O commit significa uma alteração, ele contem dados que identificam esse status.

O commit é único para cada autor.



Resumo: O blob é o arquivo. A árvore é um conjunto de blob’s e ou outras árvores. O commit reúne todo o conjunto de blob’s, árvores, data, hora e uma mensagem que o identifica. O commit representa um estado único em que todos esses arquivos se juntaram.

O git é um sistema distribuído Seguro, porque é muito difícil burlar todo esse mecanismo.

# Primeiros comandos com Git

**git init**

**git add**

**git commit**

**Criando um repositório**

Dentro do C:

mkdir workspace

cd workspace

mkdir livro-receitas

git init

ls -a = mostra arquivos ocultos

**Configurações da primeira vez que se utiliza o git**

git config --global user.email “meuEmailDoGitHub”

git config --global user.name “nomeDoUsuarioDoGitHub”

# Markdown

Extensão .md

Programa sugerido Typora

Ajuda>Markdown references = Temos ajuda para uso

# Título Principal

## Sub título do Título principal

\*\*Negrito\*\*

\_itálico\_

:símbolo

- Lista

\* Lista

>>>>>> Comentários

# commit

git add \*

git commit -m “initial commit”

Retorna algumas informações como:

Máster - Branch

Os primeiros números do sha1

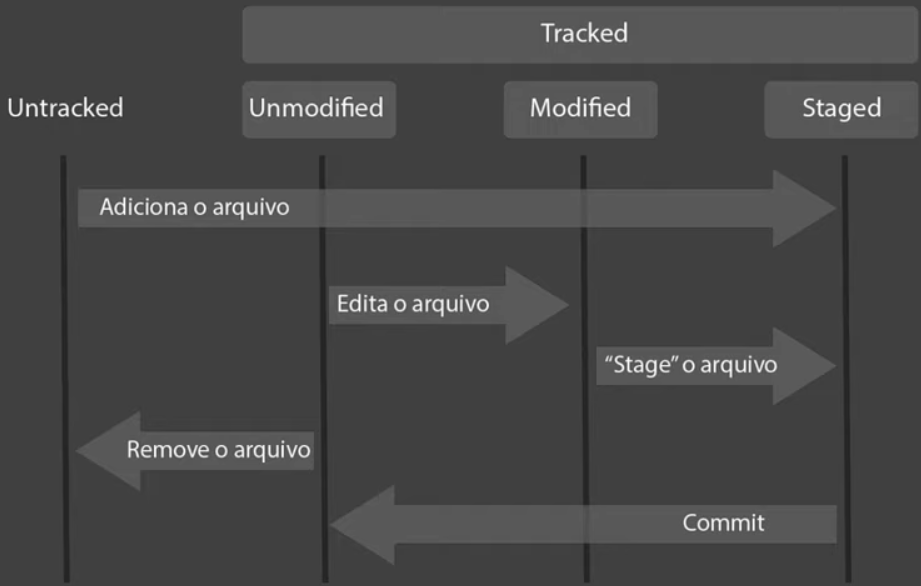
Descrição do commit

Quantos arquivos adicionados

# Ciclo de Vida dos Arquivos no GIT

O git init inicializa um conceito do git chamado repositório. Ou seja, a criação de um repositório dentro do git

**Tracked ou Untracked**

****

Untracked são os arquivos que o git ainda não tem ciência.

Tracked são os arquivos que o git já tem ciência deles.

Unmodified são os arquivos que ainda não foram modificados.

Modified são os arquivos que sofreram modificações.

**Staged** é um conceito chave:

Em um teatro, as pessoas que trabalham no **backStaged** são as pessoas que cuidam do palco, maquiagem, figurino, etc e as pessoas que estão se preparando para se apresentar.

**Staged** são os arquivos que estão se preparando para um outro tipo de agrupamento.

Antes de comitados, os arquivos estão “atrás do palco se preparando para se apresentarem”. Quando commitamos, os arquivos apresentam.



Quando rodamos o comando < git add . >



Adicionamos os arquivos para o staged, ou seja, deixamos eles no prontos para serem commitados.

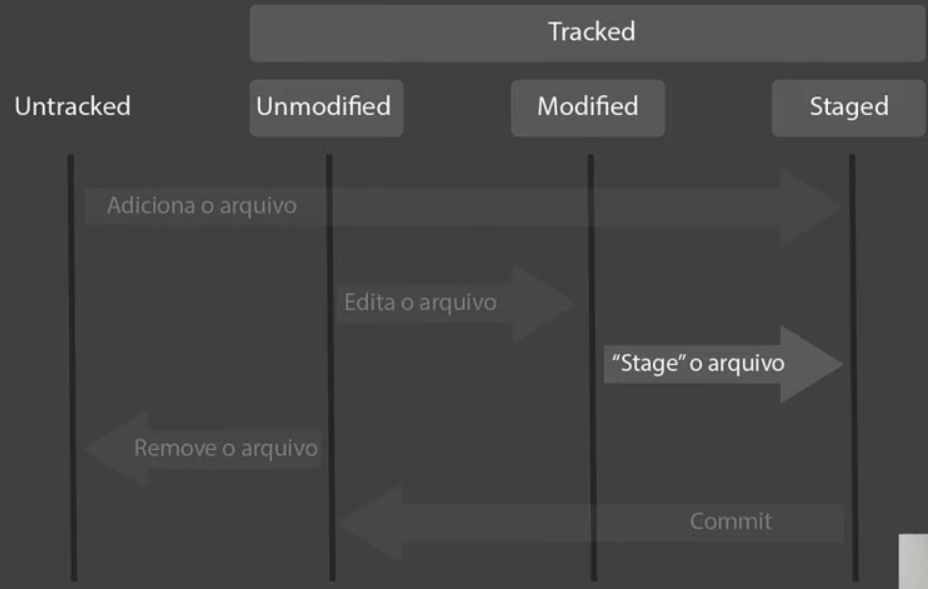
Arquivos Unmodified, são aqueles arquivos que não sofreram nenhuma alteração.



Quando modificamos um arquivo, automaticamente eles são identificados como Modified.

O git sabe que eles foram modificados, quando o valor de sha1 deles é alterado.

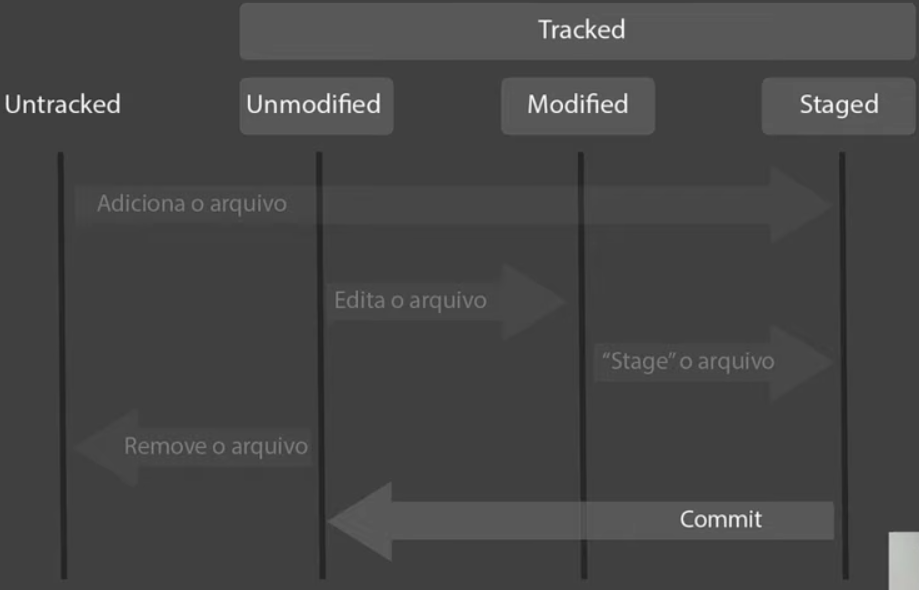
Quando um arquivo Modified sofre um < git add > ele também é movido para o Staged, ou seja, está pronto para ser commitado.



Se removermos um arquivo que estava no estado Unmodified, ele simplesmente volta para o stado Untracked, (não reconhecido pelo git).



Os arquivos que estão no modo Staged, estão prontos para serem commitados.



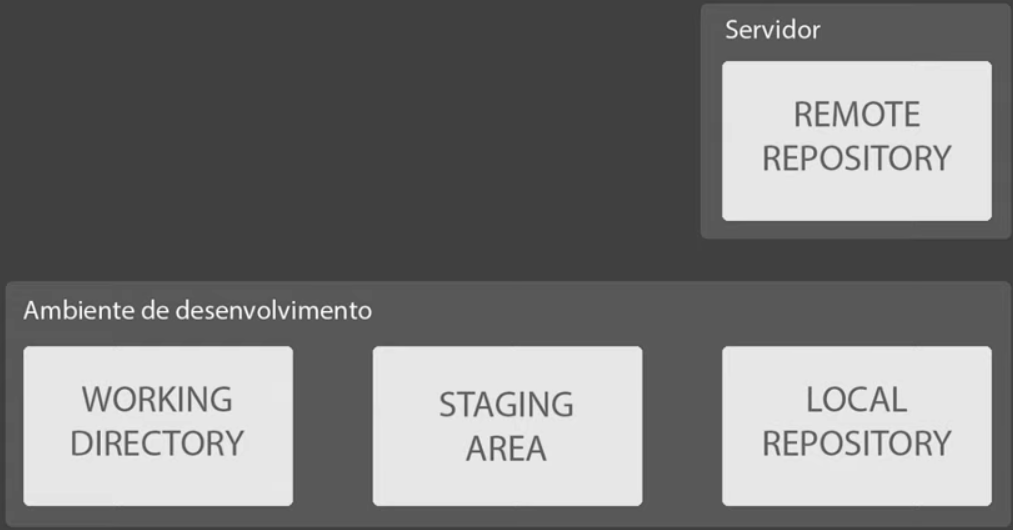
O commit retorna todos os arquivos para o estado Untracked, fechando assim o ciclo cíclico.

O commit faz um snapshot. Tira uma foto do estado dos arquivos.

**Repositório**

Quando rodamos o < git init > iniciamos um repositório.

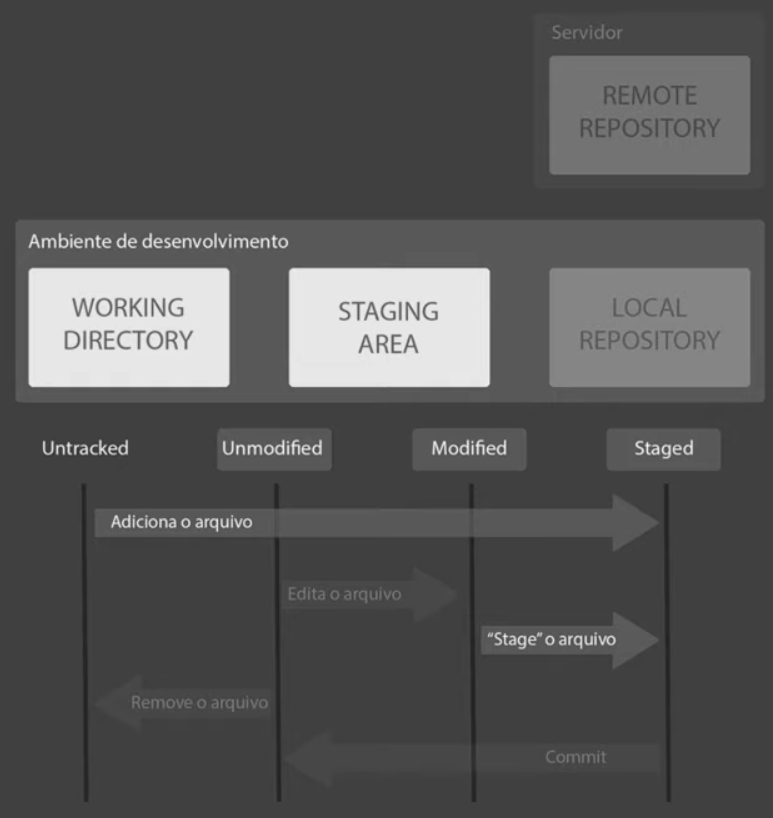
No nosso Ambiente de desenvolvimento.



Quando damos um commit, os arquivos presentes passam a compor um nosso repositório local.

O repositório local pode ser empurrado para o repositório remoto (GitHub).

Estados dos arquivos.

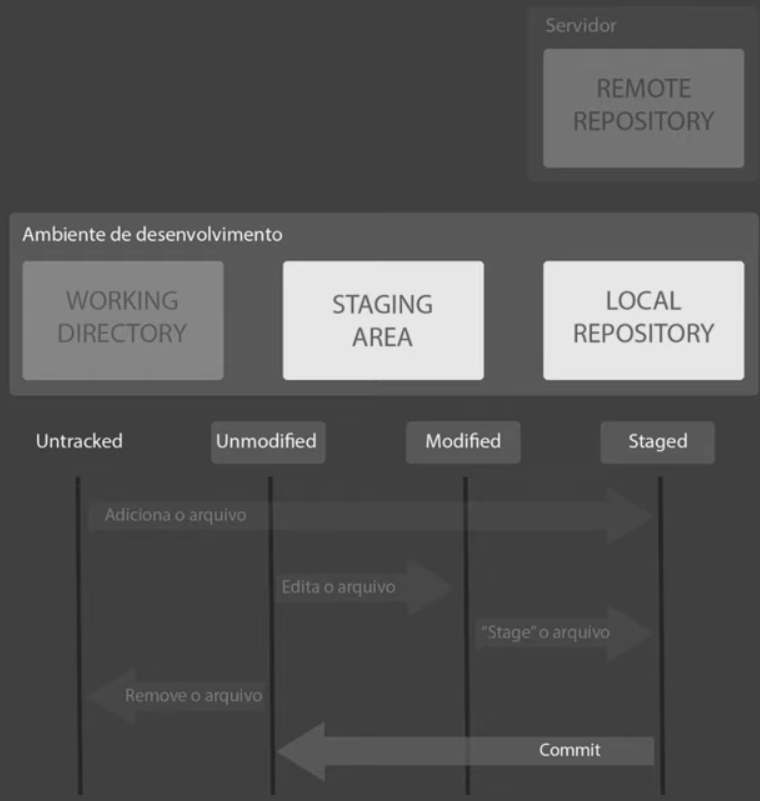


Quando damos um < git add > em um arquivo desconhecido (Untracked), ele é movido para Staged.

Quando temo um arquivo modificado (Modified) e damos um < git add > , também estamos movendo ele para o Staged.

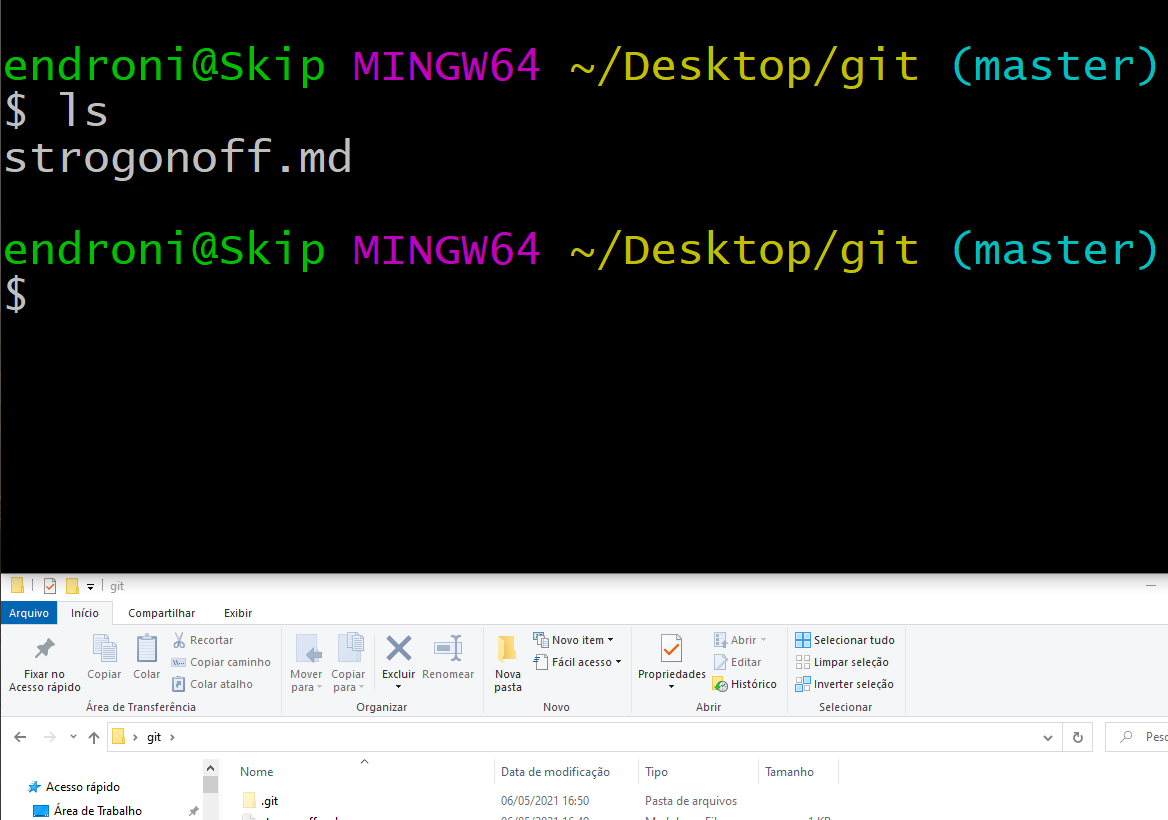
Com disso, os arquivos ficam transitando dentro dos ambientes de desenvolvimento WORKING DIRECTORY e STAGING AREA.

Nós fazemos os arquivos saírem de “traz do palco” quando realizamos um commit.

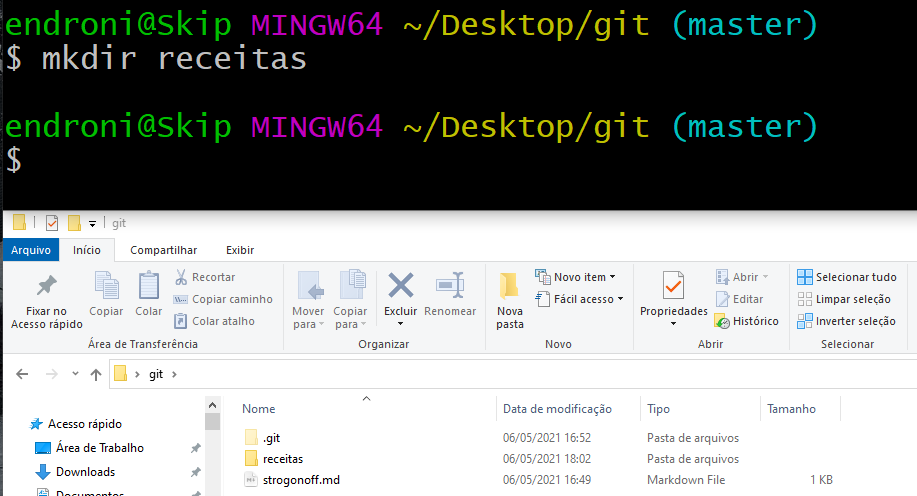


Na prática

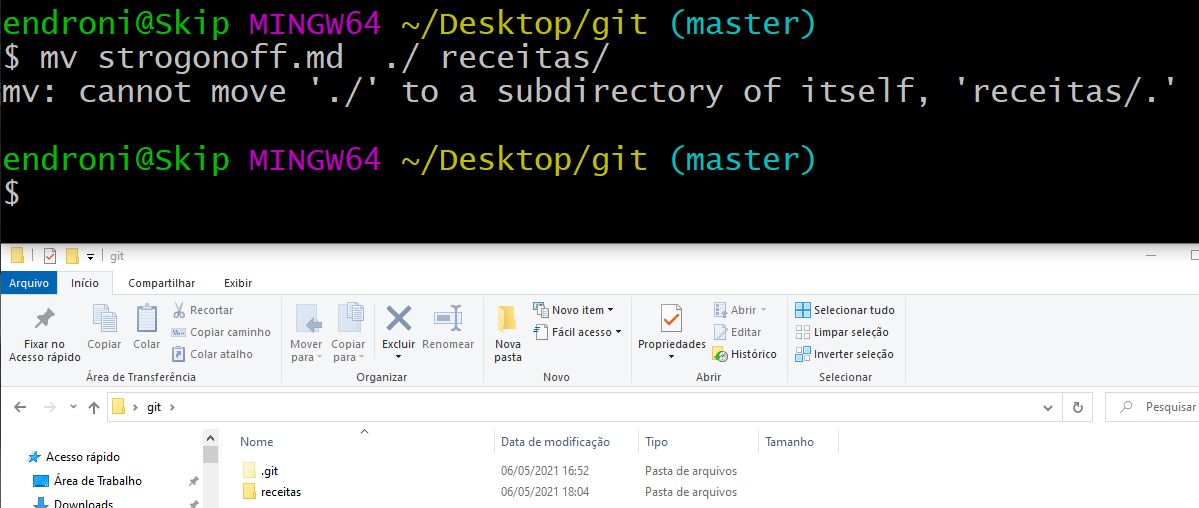
ls – nos mostra os arquivos existentes.



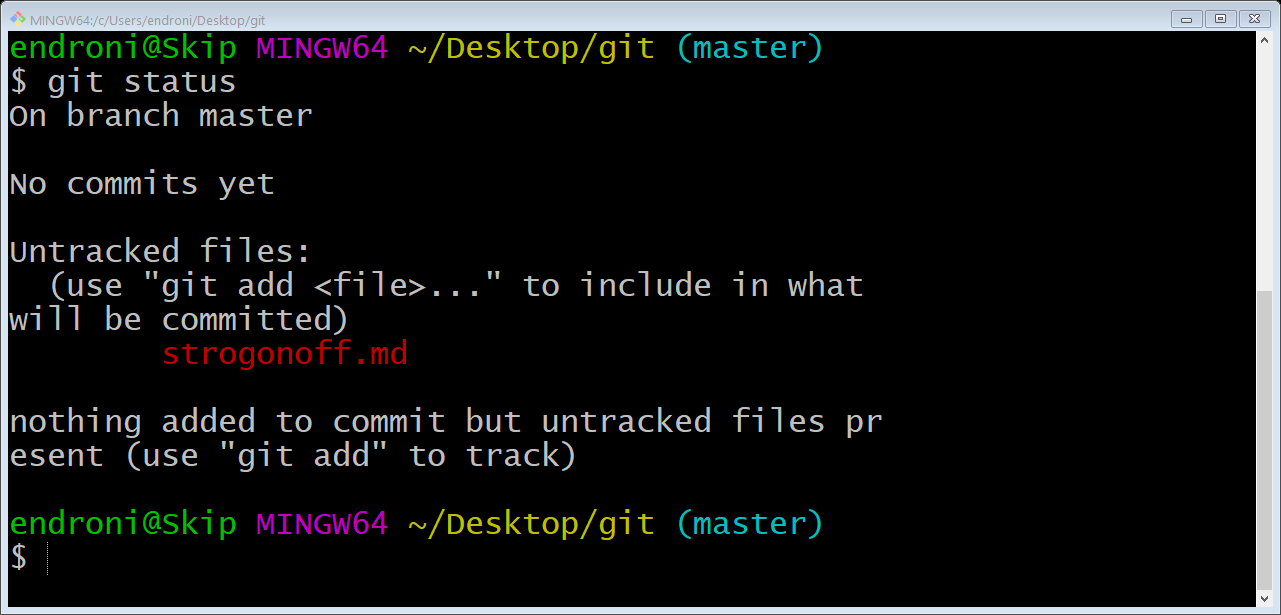
Criando um diretório



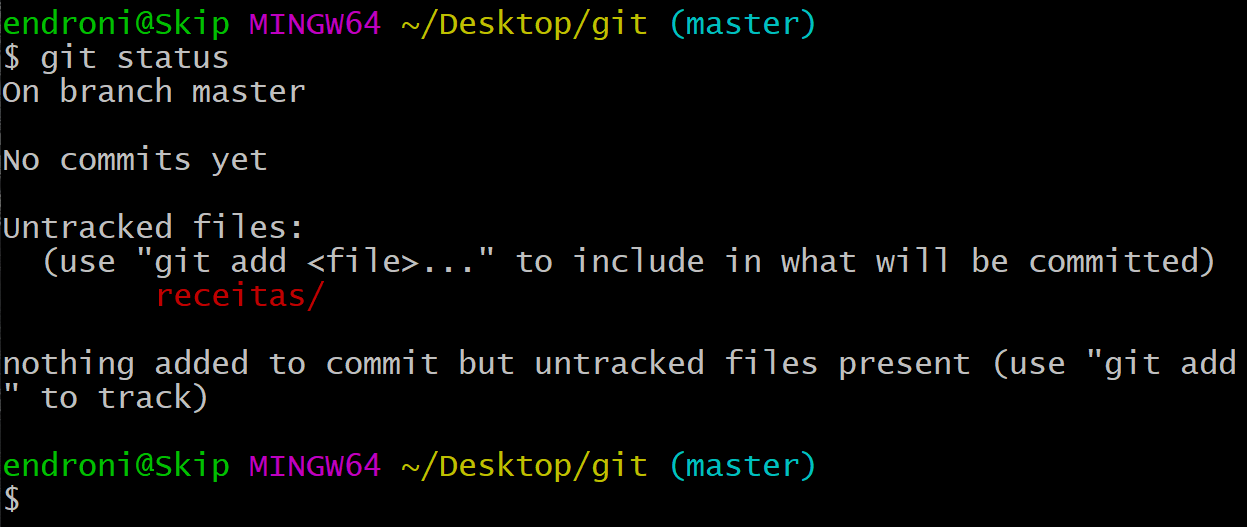
Movendo o arquivo para dentro do diretório



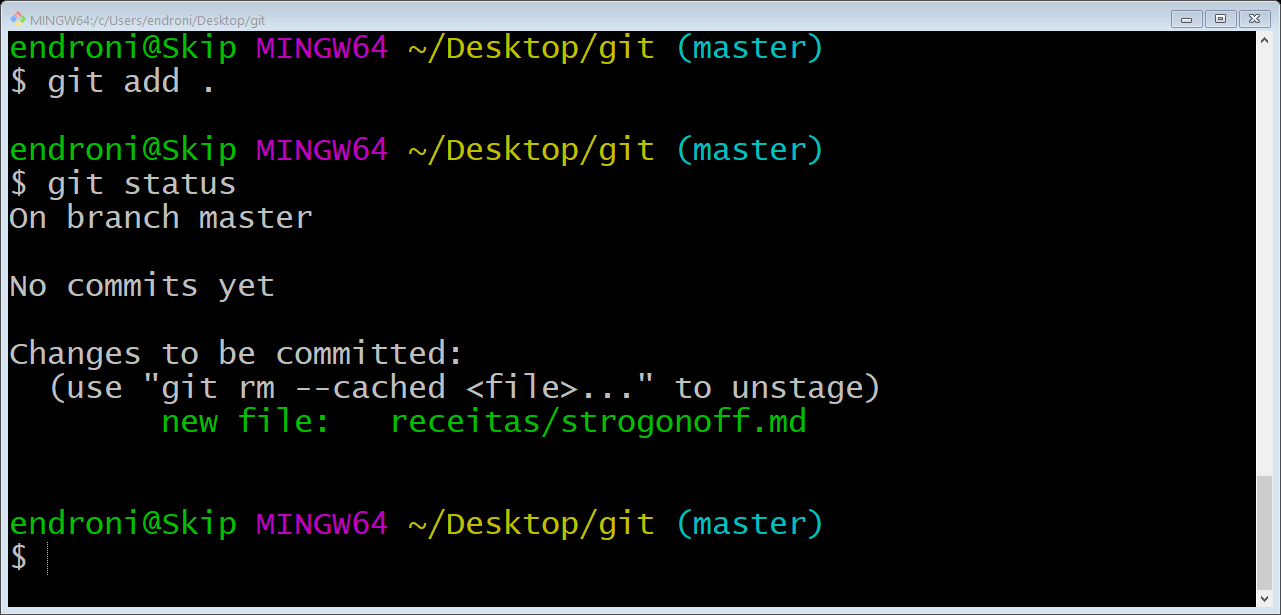
git status = nos informa como estão os arquivos.



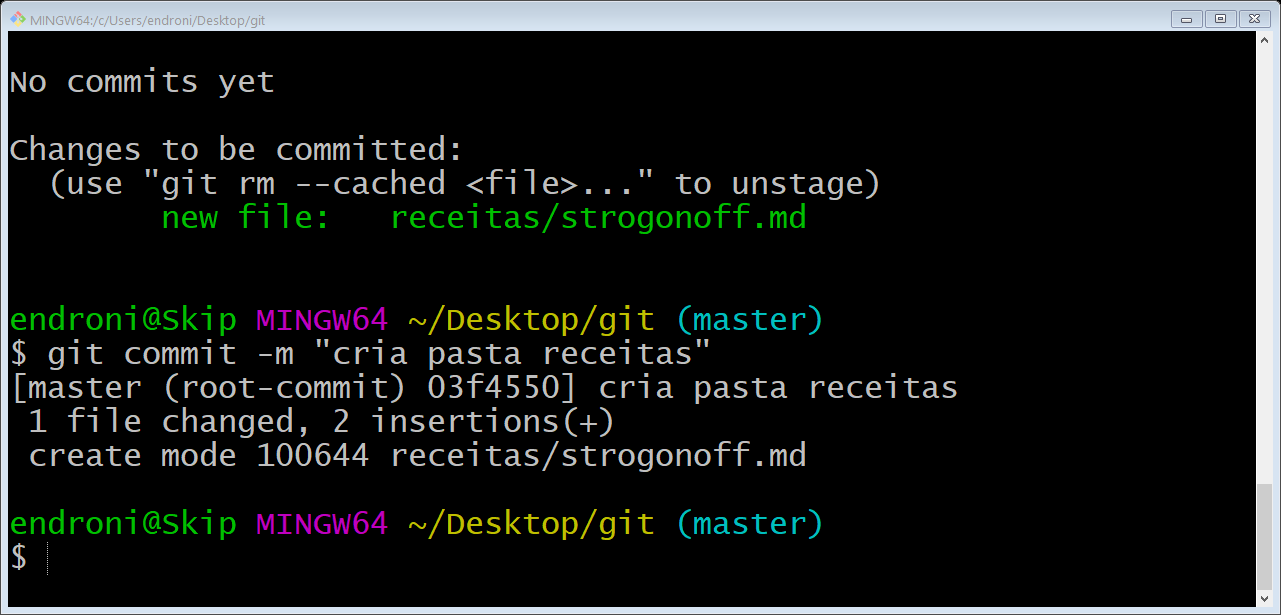
Depois de criar a pasta receitas e mover o arquivo



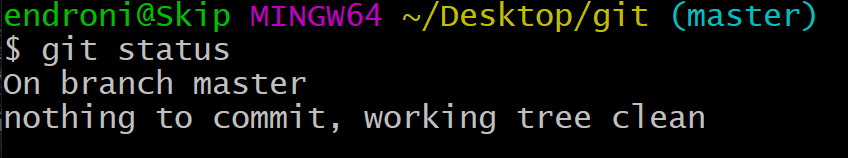
changes to be committed = mudanças para serem commitadas.



Criando commit



git status



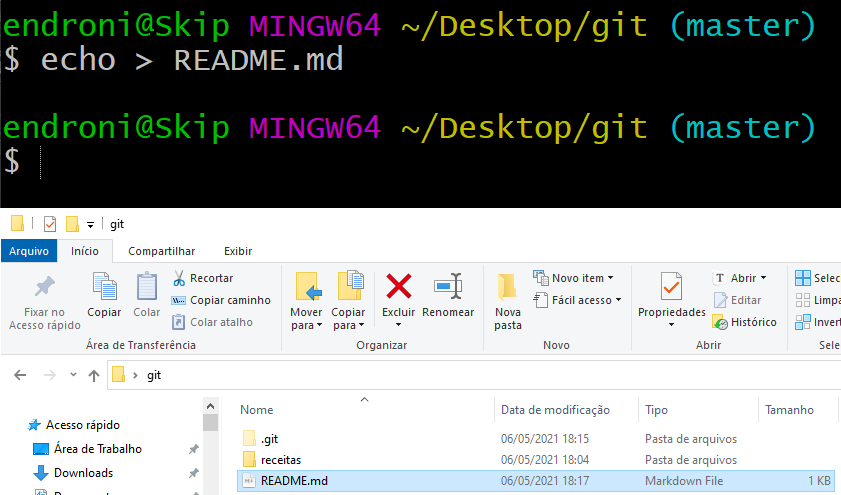
on branch master = Na branch máster

nothing to commit = nada para commitar

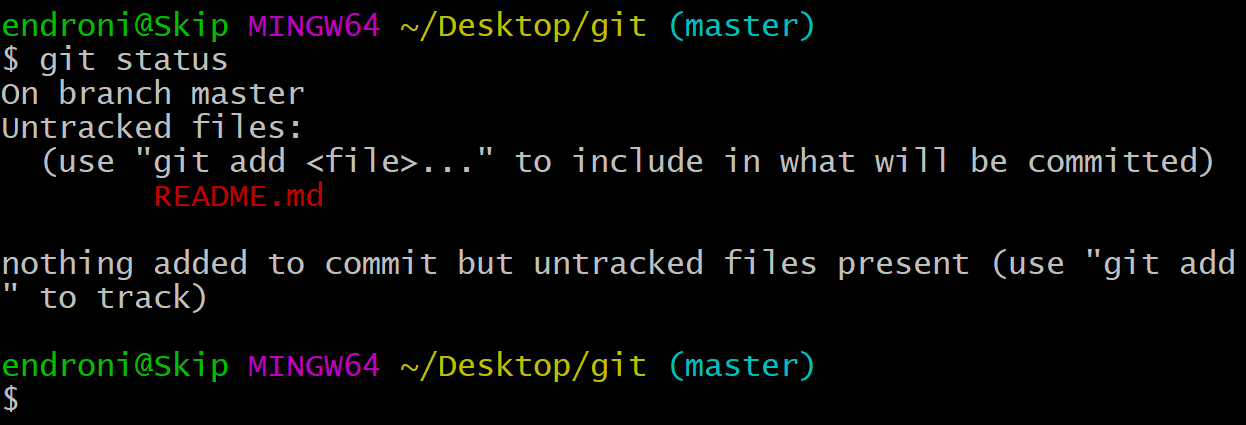
working tree clean = a árvore de trabalho está limpa.

**Criando README.md**

echo > README.md

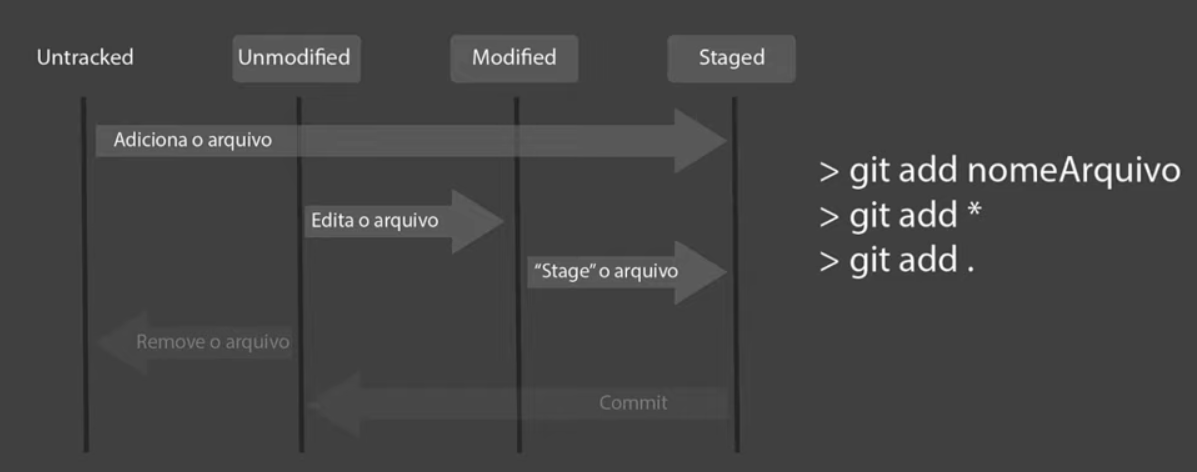


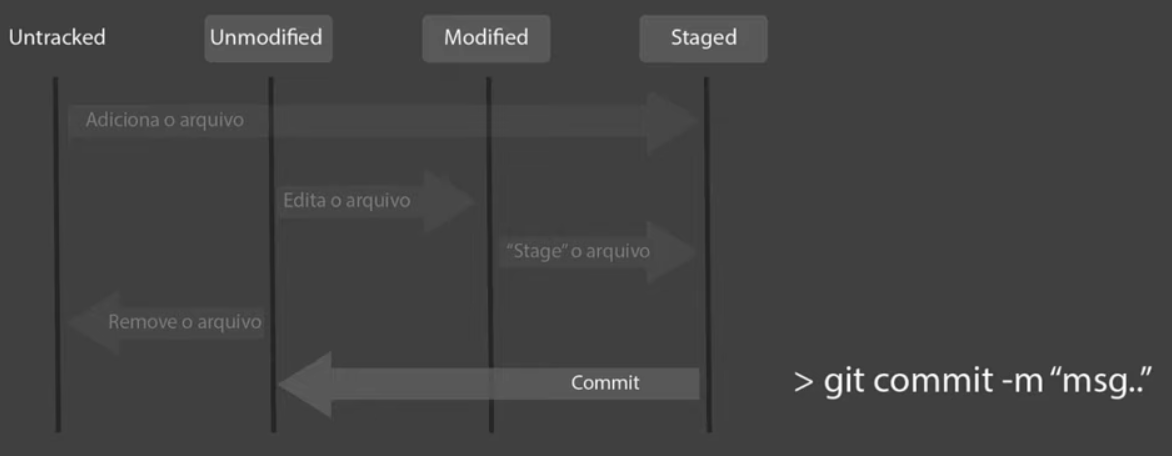
git status

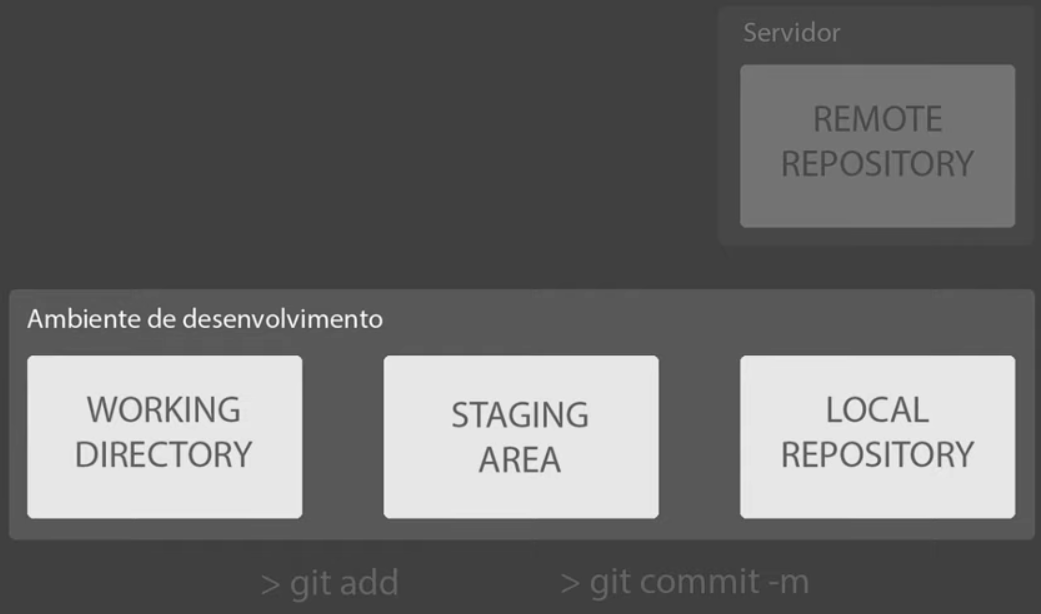


A apresentação do projeto.

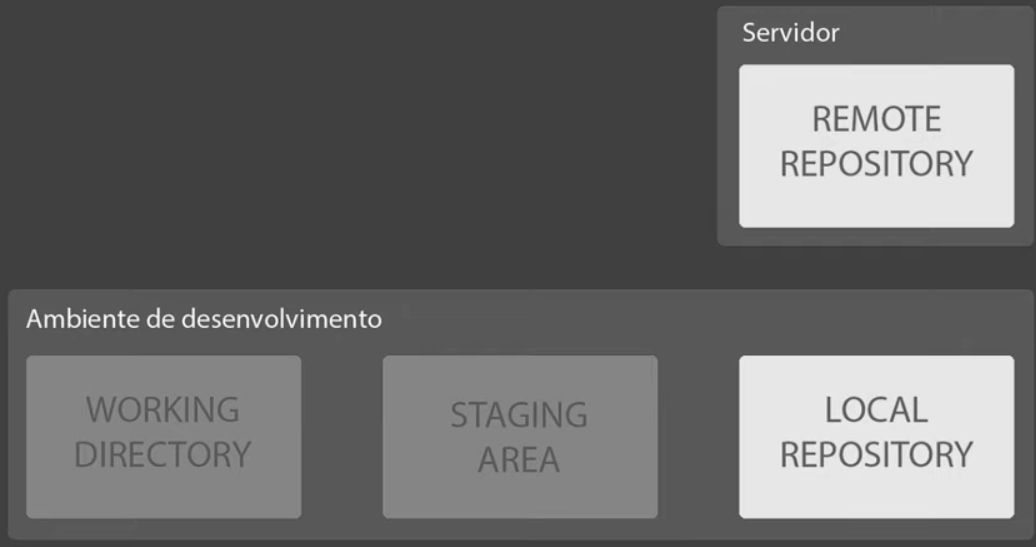
**Resumo**







Próximo:



# Trabalhando com o GitHub

git config --list

Traz alista de todas as configurações do git

q – sai do list

git config –global –unset user.email

git config –global –unset user.name

Os dois comando acima excluem a configuração de e-mail e nome.

git config --global user.email “emailDoGitHub.com.br”

git config --global user.name “nomeDoGitHub”

Obs. Não tem como alterar os commits caso tenha configurado o usuário errado.

**Apontar o repositório local para o repositório no GitHub**

git remote add origin <urlDoMeuRepositório>

Empurrando para o servidor remoto, o GitHub

git push -u origin master

origin – é um alias, um apelido para a url do repositório.

git remote -v

Lista os repositórios cadastrados.

O número que aparece próximo do número de commits é o início do número sha1. Ao clicar nele, podemos ver mais detalhes sobre o commit.

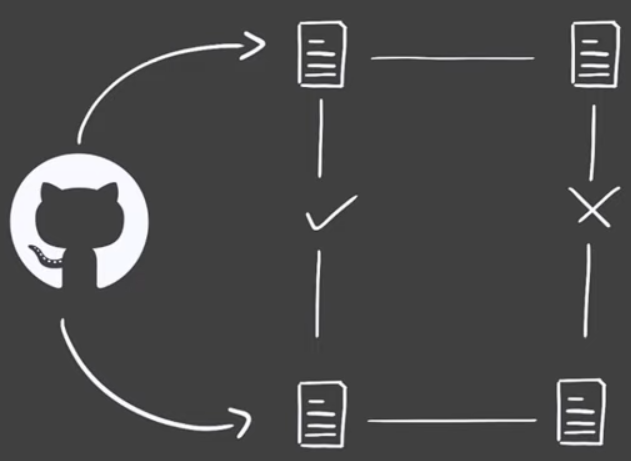
Podemos ver também o commit parent (parente), geralmente um commit é linkado a outro commit.

Na página principal do repositório, ao clicar nos commits, podemos verificar a lista dos commits. Ao clicar no número sha1 desse commit, podemos ter acesso às BLOB’s (os arquivos) desse commit.

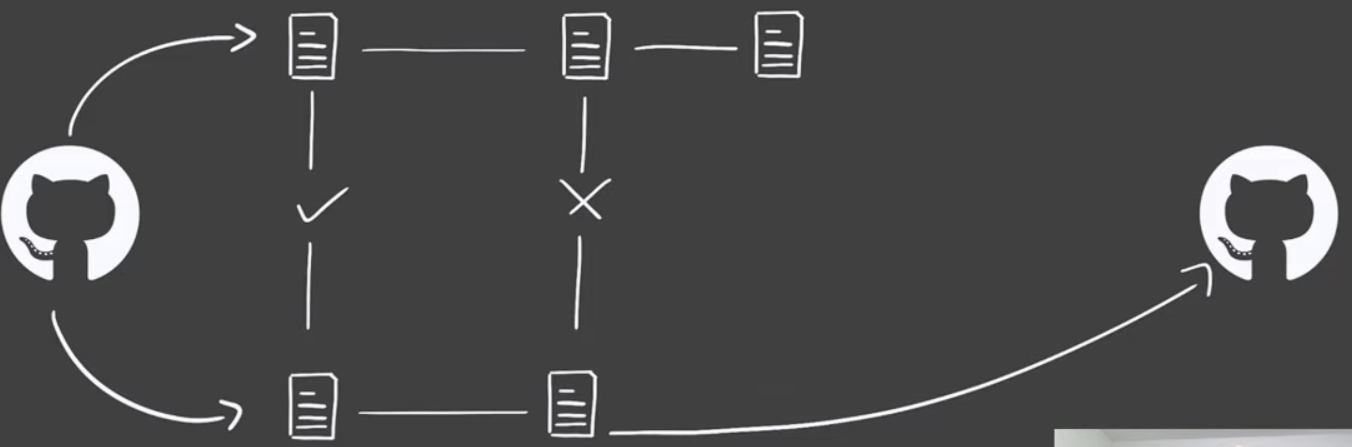
Vai montando a stack da branch. – O que significa o termo **stack**?

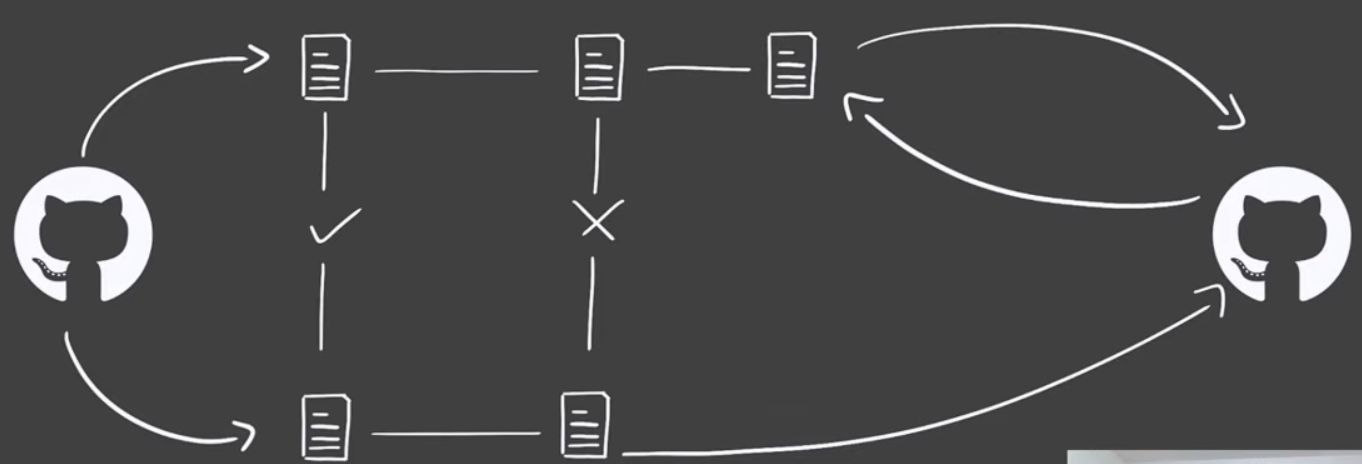
# Resolvendo conflitos

git clone – clonar um repositório que está no GitHub

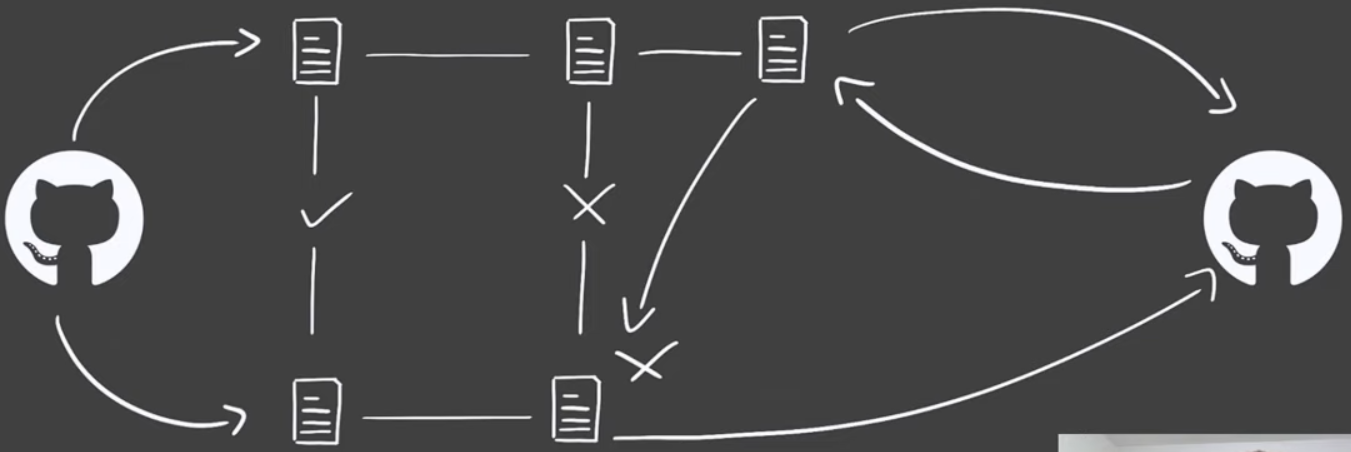


Ocorre problema quando existe alterações na mesma linha por repositórios diferentes.

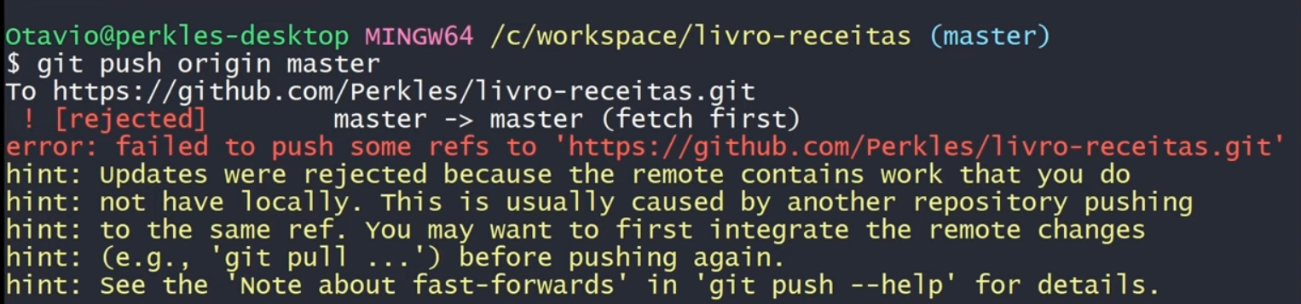




Conflito de merge



Rejeitado.



pull – puxar

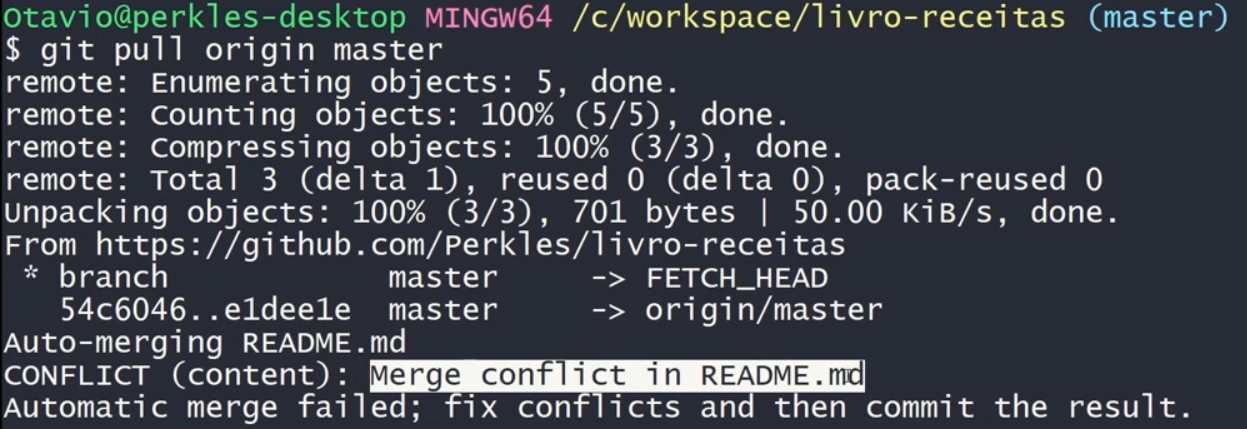
git pull origin master

Quando damos pull e existe conflito de merge, o git mostra qual é o mais recente e qual é a minha versão.

O mais recente fica em cima, e a minha versão fica embaixo.

**Para corrigir**, dê pull, apague as linhas conflitantes, faça a correção, realize o commit e depois o push.

Conflito de merge



<<<< HEAD

O que estiver aqui é a alteração mais recente.

=====

O que estiver aqui é o que estamos tentando editar

**Clonando um repositório:**

git clone <link do repositório>

Ao clonar, estamos baixando a pasta .git que contem todos os históricos de commits daquele repositório.