**Introdução ao Git**

Criado por Linus Torvalds

Kernel – estrutura interior (núcleo) de um sistema operacional

**Software é feito de forma colaborativa!**

**Benefícios:**

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

**Git = command line interface**

**Comandos básicos**

**Windows**

* **cd** = levar para alguma pasta específica. Se usar apenas a /, levará à base
* **dir** = listar diretórios e pastas
* **mkdir** = criar pasta
* **rmdir** = remove o diretório. Ex: rmdir workspace /S /Q
* **TAB =** ele completa o nome. Ex: cd Wi + tab = cd Windows
* **Echo “frase ou palavra”** = retornar uma frase. Ex: echo Hello
* **echo >** = redirecionador de fluxo. Vai pegar o que foi escrito pelo echo e enviar para algum arquivo específico. Se não existir, ele irá criar um. Ex: echo Hello > hello.txt
* **cd ..** = volta um repositório
* **cls** = limpar a tela do prompt de comando (Windows)
* **del** “nome da pasta/arquivo” = deletar os **arquivos** da pasta
* **↑ =** histórico dos comandos usados
* **pwd** = comando para mostrar o diretório completo
* **ls =** lista o conteúdo
* **ctrl + l =** limpar pasta (git)
* **mv =** mv “arquivo” ./”diretório”

**Unix**

* - **cd** = levar para alguma pasta específica. Se usar apenas a /, levará à base
* - **ls** = listar diretórios e pastas
* **mkdir** = criar pasta
* **rm** -**rf** = remove o diretório. Ex: rm -rf workspace/
* **echo** = retornar uma frase. Ex: echo Hello
* **echo >** = redirecionador de fluxo. Vai pegar o que foi escrito pelo echo e enviar para algum arquivo específico. Se não existir, ele irá criar um. Ex: echo Hello > hello.txt
* **cd ..** = volta um repositório
* **clear** = limpar a tela do prompt de comando (unix)

**Funcionamento do Git**

* SHA1 – criptografia 40 dígitos
* Objetos fundamentais
* Sistema distribuído
* Segurança

**SHA1**

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 identificadores de 40 dígitos – é único**

É uma forma curta de representar um arquivo.

**Ex**: dentro da página do desktop, tem um arquivo hello.txt.

**Dentro do git bash:** openssl sha1 hello.txt

Resultado: dc15a75dd1ac4d01102f743a35c2e8fdcd848a3c

**Ao alterar ou adicionar qualquer caractere dentro do arquivo txt e usar o comando novamente:** 53f1e27574a20cb85d11b2810e69c2a901ced7ff **= resultado diferente**

**Objetos Fundamentais**

* **BLOBS**
* **TREES**
* **COMMITS**

**BLOOBS** – aponta para diretórios com arquivos e para arquivos diretamente. **Representa os arquivos**

Ele contém metadados do Git, como o tipo do objeto, o tamanho da string, tamanho do arquivo, entre outros. Guarda o SHA1 do arquivo (conjunto dos 40 caracteres)

Ex: echo ‘conteudo’ | git hash-object –stdin

**TREES** – aponta para blobs ou para outras árvores. **Representa as pastas**

Armazenam blobs e aponta para tipos de blobs diferentes.

Também contém metadados. Guarda o nome do arquivo.

É responsável por montar toda a estrutura de onde estão os arquivos.

Podem apontar tanto para blobs quanto para outras árvores (diretórios).

**tree**

READMI Rakefile lib

Blob blob tree

simplegit.rb

blob

Tree aponta para blobs (bolhas) ou outras árvores.

Blob é um objeto que encapsula esse comportamento de diretórios e usado para apontar diretórios que contém arquivos e também só para os arquivos mesmo.

**COMMIT** – é o objeto que junta tudo e dá sentido à alteração que se está fazendo. Garante a segurança de que o arquivo não foi modificado (se foi modificado, o SHA1 muda)

É um sistema distribuído seguro

**Aponta para:**

* Árvore
* Parente (último commit realizado antes dele)
* Autor
* Mensagem
* Data e hora da criação
* Histórico

**Chaves SSH e Tokens**

**Chave SSH**

* Forma de estabelecer uma conexão segura e encriptada entre duas máquinas
* Sempre haverá uma chave pública e uma chave privada
* Pega uma chave pública (nossa máquina) e enviará para o github, fazendo com que ele reconheça nossa máquina

**pwd** = comando para mostrar o diretório completo

**ls =** lista o conteúdo

**Token de acesso pessoal**

Rever aula, se necessário

**Primeiros comandos com o Git**

* Iniciar o Git – **git init**
* Iniciar o versionamento/mover arquivos – **git add**
* Criar um commit – **git commit**
* Verificar o status dos arquivos – **git status**

**Criando um repositório:**

$ mkdir livro-receitas

$ ls

$ ls -a (o **-a** mostra arquivos ocultos da pasta)

$ cd .git

$ ls

HEAD config description hooks/ info/ objects/ refs/

Nano@DESKTOP-TGQMC66 MINGW64 /c/workspace/livro-receitas (master)

$ git config --global user.email "renanleote@hotmail.com"

Nano@DESKTOP-TGQMC66 MINGW64 /c/workspace/livro-receitas (master)

$ git config --global user.name Nano

Criação de um user e vínculo de e-mail

**Adicionando um arquivo:**

typora ou ghostwriter:

# Alguma frase – **ficará em uma fonte grande**

\*\*negrito\*\* - **ficará em negrito**

\_italico\_ - **ficará em itálico**

### alguma frase – **quanto mais #, menor será o título**

- frase – **criará um marcador obs:** vai um espaço antes e depois do –

**Ciclo de vida dos arquivos (dentro do git)**

**Git init** – inicializa um repositório / criando um repositório no git dentro daquela pasta

**Tracked:**

* Unmodified – arquivo ainda não modificado
* Modified – arquivo unmodified que foi modificado
* Staged – onde ficam os arquivos que estão se preparando para fazer parte de um outro tipo de agrupamento. Área que está esperando uma ação/fazer parte de um commit

Git add \*: **move** o arquivo **untracked** e o **modified** direto para o **staged**

Ao alterar algum arquivo, irá mudar de unmodified para modified através da leitura do **SHA1**. O git irá reconhecer essa mudança

Ao deletar um arquivo unmodified, ele voltará a ser untracked

O commit retorna todos os arquivos para unmodified.

**Ciclo:** retorna para unmodified quando “commita”, faz uma alteração ele volta pra modified e se você adiciona ele para algum commit, ele vai para um stage, escreve o commit e ele volta para o unmodified

**Untracked** – git não tem ciência desses arquivos. Não está sendo trackeado

**O que é um repositório:**

Servidor:

* Remote repository – a que está no git

Ambiente de desenvolvimento:

* Working directory
* Staging area
* Local repository – **commits** (pode ser enviado para o remote)

Os arquivos vão ficar **alterando** entre o **diretório de trabalho e a area de stage**, à medida que vai se alterando os arquivos. Quando se **faz um commit**, ele passa a **integrar o seu repositório local**, que pode ser “**empurrado**” para um **repositório remoto**.

Nano@DESKTOP-TGQMC66 MINGW64 /c/workspace/livro-receitas (master)

$ mkdir receitas

Nano@DESKTOP-TGQMC66 MINGW64 /c/workspace/livro-receitas (master)

$ ls

receitas/ strogonoff.md

Nano@DESKTOP-TGQMC66 MINGW64 /c/workspace/livro-receitas (master)

$ mv strogonoff.md ./receitas/

* **git add nomeArquivo** – adiciona um arquivo no diretório
* **git add \*** - transforma o arquivo untracked e modified para stage area
* **git add .** - transforma o arquivo untracked e modified para stage area
* **git commit -m “msg”** – pegamos todos os arquivos que estavam em staged, criamos uma mensagem para dar significância a eles e criou o objeto commit
* **git config –list** – trás a lista de todas as configurações do git, como username e e-mail
* **git config –unset user.email** – desvincular o e-mail vinculado ao git
* **git config –unset user.nickname** – desvincular o nickname vinculado ao git
* **git config user.email “email@usuario”** – vincular um novo e-mail
* **git config --global user.email “email@usuario”** – vincular um novo e-mail
* **git config --global user.name “nomeusuario”** – vincular um novo nome de usuário
* **gut push origin master (ou main)** – empurrar os arquivos para o github
* **git pull origin master (ou main)** – trazer os arquivos do github para nossa máquina
* **git remote -v** – aponta o link do diretório remoto