Aluno: Valmir José de Santana

Anotações Gerais do curso que desejo aplicar.

**Introdução ao Git e ao GitHub**

**Prof.: Otávio Reis**

**Aula 01: Entendendo o que é Git e sua importância**

**- 16/03/2021**

O Git é um sistema de versionamento de código distribuído que foi criado em 2005 por Linus Torvalds, criador do sistema operacional Linux.

Benefícios de aprender a manusear o Git e o GitHub:

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

**Aula 02: Navegação via command line Interface e Instalação**

**- 16/03/2021**

***Navegação***:

GUI (Grafic User Interface) para softwares com interfaces gráficas;

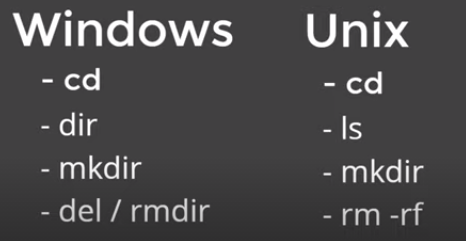
CLI (Command Line Interface) para softwares sem interfaces gráficas.

O Git é um Command Line Interface (CLI). Ou seja, ele não tem uma interface gráfica.

***O que vamos aprender?***

* Mudar de pastas;
* Listar as pastas;
* Criar pastas/arquivos;
* Deletar pastas/arquivos.

Diferenças dos comandos Git de acordo com o terminal do sistema operacional utilizado:



Segue a descrição de alguns comandos e sua função ao acessarmos o prompt de comando do Windows por meio do comando CMD.

Nestas anotações, nos concentraremos nos comandos para o sistema operacional Windows.

***dir*** – Exibe ou lista todos os diretórios na pasta em que o usuário está situado;

***cd (Change Directory)*** – Possibilita que o usuário navegue entre pastas específicas. Ou mude de diretório. Ou seja, entra em uma pasta ou diretório específico. Esse comando é igual para todos os sistemas operacionais.

Nota: Para exibir o conteúdo da pasta assim que ela for localizada, basta utilizar o comando DIR.

***Cd .. (cd + ESPAÇO + dois pontos)*** – Retrocede um nível na pasta. Ou seja, sai de uma pasta ou diretório específico. Esse comando é o mesmo para o sistema Linux.

***cls (Clear)*** – Limpa o terminal (Promp de comando). No Linux a mesma função é feita pelo comando “Clear” ou simplesmente CTRL + L.

**ls -a**: Exibe arquivos ocultos em um repositório.

**Ls**: Lista ou exibe os arquivos de um repositório.

***Tecla TAB*** – Tem a função de autocompletar textos de nomes de arquivos reconhecidos pelo Windows.

***mkdir (Make Directory)*** – Cria uma nova pasta ou diretório. Esse comando é o mesmo para o sistema Linux.

**Echo** – ‘Printa’ ou replica no diretório selecionado a palavra que o usuário digitar logo após esse comando.

Ex.:

c:\workspace>echo hello world

O resultado será: hello world.

***echo*** *+* ***(nome do arquivo)*** *+* ***>*** (sinal de maior que ou Out Put que é um redirecionador de fluxo) – Cria um novo arquivo dentro de um diretório.

Ex.:

c:\workspace>echo hello > hello.txt

O resultado será: hello.txt (Arquivo criado dentro da pasta ‘workspace’.

***del*** – Exclui um arquivo específico dentro de um diretório ou pasta mas mantém a pasta ou diretório.

***↑ (Arrow – Ceta para cima)*** – Navega entre os últimos arquivos criados ou acessados.

***rmdir (Remove Directory)*** – Exclui um diretório ou pasta com todos os arquivos e sub-pastas que ele contém.

**Aula 03: Entendendo como o GIT funciona por baixo dos panos**

**- 22/03/2021**

Para entendermos como o GIT funciona por debaixo dos panos, precisamos saber o que significam os seguintes termos:

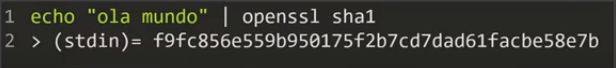
* SHA 1;
* Objetos fundamentais;
* Sistema distribuído;
* Segurança.

**SHA 1**: 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 Estados Unidos). Ou seja, o GIT captura um arquivo (foto, vídeo, áudio etc.) e criptografa (embaralha) esse arquivo. É utilizado para identificar um arquivo.

A encriptação gera um conjunto de caracteres identificador de 40 dígitos. Esse conjunto é único e serve como identificação. Ele é alterado cada vez que realizamos uma modificação no arquivo. Ou seja, a cada modificação realizada, um novo conjunto de caracteres contendo 40 dígitos é gerado.

Podemos dizer que o SHA é uma forma curta de representar um arquivo.

Ex.:



No GIT Bash utilizamos o comando “***openssl sha1 + nome do arquivo com a sua extensão***”. Em seguida, teclamos “Enter”.

**Objetos Fundamentais do GIT**: Esses são Blobs, Trees e Commits. Esses são responsáveis pelo versionamento do código.

**Blob (Bolha):** São os blocos básicos de composição de estruturas no GIT. Elas armazenam metadados do GIT. Esse metadados são informações tais como: Tipo do objeto, tamanho da string, tamanho do arquivo entre outros.

Ex.:



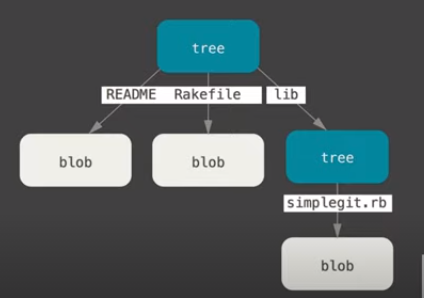
**Tree (Árvores):** Armazenam Blobs. Assim como as blobs, elas também contêm metadados. Diferente das Blobs, no entanto, as Trees guardam o nome do arquivo. Será responsável por montar toda a estrutura de onde está o arquivo. Elas podem apontar para Blobs ou outras Trees.

As blobs estão diretamente relacionadas as trees de modo que se alterarmos algo em uma blob, automaticamente estaremos alterando a estrutura de uma tree.

Ex.:

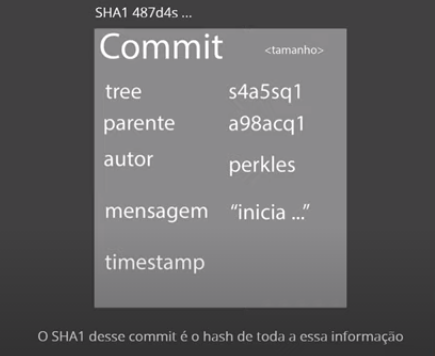


**Estrutura de uma Tree**

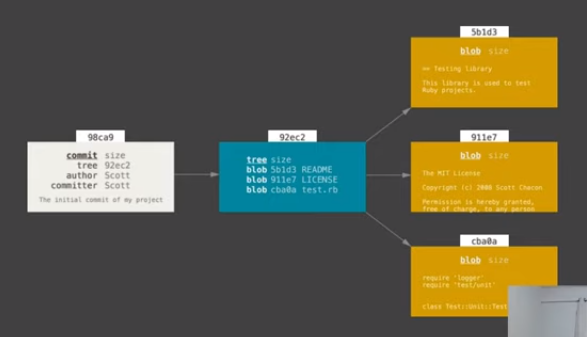


**Commit:** É o objeto mais importante do GIT. Trata-se do objeto que junta tudo (Blobs e Trees). Ela dá sentido a qualquer alteração que for feita no objeto. Além de apontar para um blob, ele aponta também para uma Tree, para um autor, para um parente, para uma mensagem e até mesmo para o último commit ralizado. Eles traçam uma “linha do tempo” de cada alteração realizada no arquivo.

Os commits também possuem encriptação ou SHA1. Como as blobs, trees e commits estão relacionadas, cada nova alteração que for feita no arquivo irá exigir a realização de um novo commit. Como cada commit realizado gera um SHA único, esse procedimento garante a segurança do arquivo mesmo que ele esteja sendo acessado e modificado por diversos autores.



**Esquema de um Commit realizado em um arquivo contendo Blobs e Trees.**



**Sistema distribuído e seguro:** O GIT é considerado um ‘sistema distribuído’ porque contém múltiplas copias de si mesmo em diferentes locais. Dessa forma, mesmo que o código original esteja hospedado em um host ou na nuvem, várias pessoas (mainteners) podem utilizar, alterar e guardar a versão original e alterações realizadas nesse código.

**Aula 04: Primeiros comandos com GIT**

**- 22/03/2021**

Objetivos da aula:

* Iniciar o GIT;
* Iniciar o versionamento;
* Criar um Commit.

Os comandos que iremos explorar são:

* Git init;
* Git add;
* Git commit.

A flag (bandeira) “**ls -a**” lista arquivos ocultos criados pelo GIT. Arquivos ocultos são arquivos gerenciais onde ficam armazenados todos os códigos do GIT e onde ele versiona os objetos que estão sendo manipulados.

**Git init**: Inicia ou cria um novo arquivo no GIT com a extensão “.git”.

**Git add + \* (Asterisco)**: Adiciona todas as alterações feitas em um arquivo no repositório GIT para ser comitada;

**Git commit -m + Descrição do commit realizado colocada entre aspas**: commita ou salva no repositório do GIT a última alteração feita no arquivo.

***OBS.: Antes de executar esse comando é imprescindível executar o comando mostrado acima.***

**Nota:** Para limpar a tela do bash do Git basta teclar ***CTRL + L***.

**Aula 05: Ciclo de vida dos arquivos no GIT**

**- 22/03/2021**

**Git init**: Além de iniciar ou criar um novo arquivo no GIT com a extensão “.git”, inicializa um conceito do GIT chamado “Repositório”. Em outras palavras, quando utilizamos o comando git init estamos, de fato, criando um repositório do GIT dentro da pasta do sistema que escolhemos.

***Tracked*** (Monitorados ou rastreados): Arquivos reconhecidos pelo GIT. Se subdividem em:

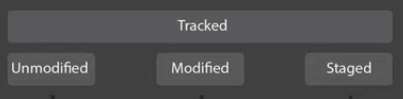
Unmodified (Não modificado): Arquivos que ainda não foram modificados. Ou seja, não sofreram alteração.

Modified (Modificado): Arquivos que anteriormente eram unmodified e sofreram alteração.

Staged (Encenado): Arquivos que estão prontos para serem alterados.

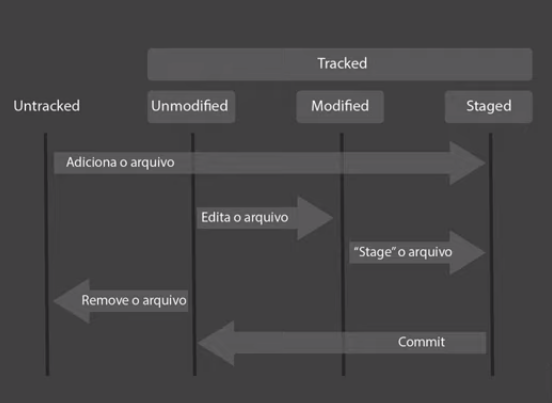
***Os arquivos classificados como Tracked (unmodified, modified e staged) atuam de forma cíclica dentro do repositório do GIT. Ou seja, um arquivo na posição staged, uma vez que for comitado, retorna para a posição unmodified. Após ele sofrer qualquer alteração, o SHA dele é alterado e ele passa da posição unmodified para modified. Se for rodado o git add nesse arquivo, ele passará para a posição staged daí aguardando para ser comitado. Após ser realizado o commit, o arquivo volta para a posição unmodified a fim de iniciar o ciclo novamente***.

*Subdivisões dos arquivos “Tracked”*



***Untracked*** (Não monitorado ou não rastreado): Arquivos não reconhecidos pelo GIT.

**OBS.:** Caso um arquivo na posição unmodified seja removido, ele volta para a posição untracked.



**Distribuição dos Servidores no GIT**



Com base no esquema apresentado acima, podemos dizer que quando realizamos um commit no GIT movemos um arquivo da área de Staging para o repositório local.

**Git status** – Monitora a situação em que um arquivo se encontra no repositório do GIT. Ou seja, se ele está Unmodified, modified ou staged.

**Aula 06: Introdução GitHub**

**- 22/03/2021**

**Git config --list :** Comando para que o GIT apresente todas as configurações pessoais que foram realizadas.

**Git config –global –unset user.email** : Altera as configurações de E-mail registradas no GIT.

**Git config –global –unset user.nickname** : Altera as configurações de usuário registradas no GIT.

**Q** : Sai das configurações pessoais no GIT.

**Git remote add origin + NOME ou URL da Origem**: Adiciona um repositório remoto ao repositório local do GIT.

**Git remote -v**: lista todos os repositórios remotos cadastrados em seu repositório local.

**Git push + NOME ou URL da Origem**: Literalmente ‘empurra’ ou grava as últimas alterações feitas no repositório local para o repositório remoto.

**OBS.:** No caso do comando acima, pode-se utilizar também o comando **Git push origin master.**

**Git pull origin master**: ‘Puxa’ ou grava as últimas alterações feitas no repositório remoto para o repositório local. Faz o processo inverso ao do comando apresentado anteriormente.

**Git clone + NOME ou URL da Origem**: Faz um clone ou cópia de um repositório remoto em seu repositório local.

***OBS.: Antes de executar os comandos em azul destacados acima é imprescindível executar os comandos “git add \*” e “git commit -m”. Ver descrição detalhada desses dois comandos em aula anterior.***

**Resolvendo problemas de conflito no Git Hub**

Normalmente problemas de conflitos acontecem quando ocorrem edições na simultâneas na mesma linha do código.

**Conflito de Merging**: Ocorre quando há duas alterações ao mesmo tempo na mesma linha de código.