**Anotações aula 📝**

**Entendendo o que é Git e a sua importância ⭐️**

*Por quê?* Git é importante pois ele controla o históricos de alterações realizados ao longo de um projeto, tendo sempre a referência de código base. É colaborativo e permite flexibilidade, segurança e desempenho no fluxo de trabalho.

* Criado pelo Linus Torvalds de forma colaborativa
* Git e GitHub são tecnologias diferentes, mas complementares
* Existem outras tecnologias mas o Git e o GitHub são amplamente utilizados

Benefícios

1- Controle de Versão

2- Armazenamento em nuvem

3- Trabalho em equipe

4- Melhorar seu código

5- Reconhecimento

**Navegação via command line interface e instalação📜**

O Git foi concebido para ser utilizado via linha de comando (Command Line Interface) e não por interface gráfica (Graphic User Interface), os comandos variam de acordo com o terminal (Windows ou Unix).

**Primeiros comandos no Windows**

1- *dir*: lista os repositórios e arquivos

2 - *cls*: Limpa a tela

3 - *cd*: navega entre os repositórios = *cd nome repositório ou caminho*

4 - *cd..*: volta um nível

5 - *mkdir*: cria repositórios = \*mkdir nome do repositório

6 - *echo*: cria arquivos = *cho nome arquivo > nome arquivo.extensão*

7 - *del*: deleta os **arquivos** dos repositórios = *del nome repositório*

8 - *rmdir*: deletar os **repositórios** = *rmdir nome repositório + /S /Q*

**Entendendo como o Git funciona por baixo dos panos🖥**

**Tópicos fundamentais para entender o funcionamento do Git**

**Conceitos:**

* SHA 1
* Objetos fundamentais
* Sistema distribuído
* Segurança

A sigla SHA significa *[Secure Hash Algorithm](https://en.wikipedia.org/wiki/Secure_Hash_Algorithms)* (Algoritmo de Hash Seguro), é um conjunto de funções [Hash](https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_hash_criptogr%C3%A1fica) criptográficas projetadas pela [NSA](https://www.nsa.gov/) (Agência de Segurança Nacional dos EUA ).

O resultado da encriptação gera um conjunto de caracteres identificador de 40 dígitos **ÚNICOS**. Sendo uma forma curta de representar um arquivo.

Na prática o SHA1 irá encriptar arquivos e qualquer alteração que for realizada dentro de um arquivo ou repositório resultará em uma chave SHA1 diferente, garantindo assim o controle e a integridade do projeto.

Os [Objetos fundamentais](https://git-scm.com/book/en/v2/Git-Internals-Git-Objects) são: BLOBS, TREES e COMMITS

Na prática os arquivos são armazenados dentro dos BLOBS esse objeto contém metadados do Git (tipo, tamanho, \0, conteúdo).

As TREES (árvores) elas armazenam BLOBS e apontam para tipos de BLOBS diferentes, elas também contém metadados (tipo, tamanho, \0, BLOB, SHA1, nome do arquivo ).

Eles apresentam uma relação de efeito dominó ou seja, qualquer alteração realizada em um arquivo do BLOB para o qual a TREE aponta, resulta na alteração do SHA1 do BLOB e da TREE.

A estrutura das TREES como no exemplo abaixo apontam para: arquivos (README, Rakefile, lib) > BLOBS > TREES > aquivos (Simplegit.rb) > BLOBS

**Commit** é o objeto mais importante pois ele agrega todos os objetos, sua estrutura contém (tipo, tamanho, TREE, parente, autor, mensagem, timestamp) é neste momento que se coloca a mensagem que vai dar sentido para a alteração que foi feita no seu arquivo.

Qualquer alteração realizada em um arquivo do BLOB resulta na alteração do SHA1 do BLOB, TREE e do Commit.

Chave SSH é uma forma segura e prática de integrar seu servidor local com o servidor remoto (GitHub), isso permite com que o fluxo de trabalho para a nuvem seja mais rápido sem a necessidade do uso login e senha no GitBash uma vez configurada a chave para a criação da chave é necessário as seguintes etapas:

1- Criação da chave SSH o **GitBash** com os seguinte comandos:

ssh-keygen -t ed25519 -C [user@email.com](mailto:user@email.com)

(a saída será a mensagem de gerando par de chaves publica/privada e o local onde a chave será gerada)

Em seguida press enter e digitar uma senha (guarde essa senha)

2 - Navegar até a pasta em que foi gerada as chaves e usar o comando ls para listar as chaves da pasta

3 - com o comando: *cat* **chavepública**

ou seja é o arquivo.pub ( a saída será uma chave que deverá ser copiada nesse momento)

4 - Criação de nova chave no GitHub no menu (SSH and GPG keys) no seu perfil do GitHub (colocar o nome da chave no Title e colar a chave copiada do GitBash no Key)

5 - No GitBash inicializar a chave com o comando dentro da pasta onde a chave foi gerada :

eval $(ssh-agent -s)

resultado de saída será o agente pid

6 - comando ls para listas as chaves novamente

7 - nessa etapa a chave será entregue para o agente que é o "administrador" da nossa chave com o comando:

ssh-add **chaveprivada**

press enter e ele solicitará a senha digitada no passo 1

**Primeiros comando com Git🌳**

**Iniciando o Git e criando um commit**

**Objetivos**

* Iniciar o GIT
* Iniciar o versionamento
* Criar um commit

Com a pasta de trabalho criada:

*git init* (dentro da pasta que é para ser versionada)

(se utilizado pela primeira vez é necessário configurar usuário e e-mail com os comandos:

git config --global user.email "[user@email.com](mailto:user@email.com) " para configurar e-mail do usuário

git config --global user.name nome)

Com o arquivo criado em editor de texto por exemplo utilizar os comandos:

*git add* \*: para adicionar o arquivo no modo versionamento

\*git commit -m "mensagem explicando as alterações"

**Ciclo de vida dos arquivos no Git🌳**

Passo a passo no ciclo de vida

O ciclo de vida de um projeto se resume na imagem abaixo *untracked* são arquivos que ainda não foram rastreados e *tracked* são arquivos rastreados.

Arquivos **tracked** tem seu ciclo de vida dentro de três estágios:

1 - **unmodified**: são arquivos que não foram modificados

2 - **modified**: são arquivos **unmodified** que foram modificados

3 - **staged**: são arquivos que estão prontos para serem commitados

Na prática os arquivos ficam transitando dentro deste ciclo: cria-se um arquivo *untracked* ele migra direto para **staged** onde é commitado, após isso ele migra para **unmodified** aqui pode-se remover o arquivo nesse caso ele migra para o *untracked* novamente ou o arquivo pode ser editado migrando então para o **modified** nesse estágio ele migra para o **staged** novamente, e recomeça todo o ciclo quando ele é commitado.

**Comandos**

*mkdir*: cria repositórios

*git status*: mostra a situação atual dos seus arquivos

*mv****nomearquivo****./repositório/*: esse comando muda um arquivo para outro repositório desejado

*git restore -- sataged*: tranforma o arquivo para unstaged

*git commit - m "mensagem da alteração realizada"*: commita as novas alterações