3-Entendendo o que é Git e sua importância

Introdução ao GIT e ao GITHUB

PQ Aprender GIT ?

Em teoria é dar uma outra interação ao seu projeto, fica exposto a comunidade e versionar de forma segura

GIT CRIADO EM 2005, É UM SOFTWARE DE VERSIONAMENTO DE CÓDIGO CRIADO PELO LINOS TORVALDS

O SOFTWARE É COLABORATIVA, NÃO É TRABALHO DE UM HOMEM SÓ

GITHUB “GUARDA O GIT” ELES NÃO SÃO IGUAIS, APENAS COMPLEMENTARES

FOCO DO CURSO

1. CONTROLE DE VERSÃO
2. ARMAZENAMENTO NA NIVEM
3. TRABALHO EM EQUIPE
4. MELHORAR SEU CODIGO
5. RECONHECIMENTO

COMANDOS BASICOS PARA UM BOM DESEMPENHO NO TERMINAL

1. Navegação básica no terminal
2. Instalação do git

GUI X CLI

GRAPHIC USER INTERFACE X COMAND LINE INTERFACE

GIT É CLI, MAS TEM PROGRAMAS QUE O TORNAM GUI

OQUE VMS APRENDER DE COMANDOS ?

* MUDAR DE PASTAS
* LISTAR PASTAS
* CRIAR PASTAS/ ARQUIVOS
* DELETAR PASTAS/ARQUIVOS

Win Linux

cd (navegar entre as pastas ) cd

cd.. (volta a pasta anterior ) cd..

cls ( limpa a tela do terminal) clear ou CTRL+L

TAB ( ele autocompleta oque vc ta escrevendo) TAB

dir (mostra oque tem na pasta) ls

mkdir (cria uma pasta) mkdir

echo ( determina a saída de volta ao terminal)

“>” (direciona a ação, usando com o eccho ele pode criar um arquivo)

del ( criado apenas para apagar arquivos ( se usado em pasta, vai deletar tudo dentro da pasta))

rmdir ( apaga a pasta em definitivo com as flags /s /q) rm ( com as flags -rf)

“”””””””Silence on sucess””””””””” se o terminal não retornou nada, deu certo

INSTALANDO O GIT

TÓPICOS FUNDAMENTAIS PARA ENTENDER O FUNCIONAMENTO DO GIT

Entendendo como o git funciona por baixo dos panos

* SHA1
* Objetos fundamentais
* Sistema distribuído
* Segurança

SHA1

A SIGLA SHA SIGNIFICA SECURE HASH ALGORITHM (Algoritmo de hash seguro) desenvolvido pela NSA (agencia de segurança nacional dos eua)

Segue o padrão de hash, criptografa tudo em um conjunto de 40 digitos

COMANDO GIT BASH

Openssl sha1 (criptografa o arquivo em hash) ( com isso vc consegue saber se o arquivo foi de alguma forma alterado)

USADO PARA IDENTIFICAR ARQUIVOS

hash-object

OBJETOS INTERNOS DO GIT

* BLOBS
* TREES
* COMMIT

BLOBS

OS ARQUIVOS ELES FICAM DENTRO DO OBJETO CHAMADO “BLOBS” ele possui metadado dentro dele

Ele vai ter o :

Tipo

Tamanho (arquivo ou string)

\0

E só então o conteúdo

Fechando assim o conteúdo de uma blob

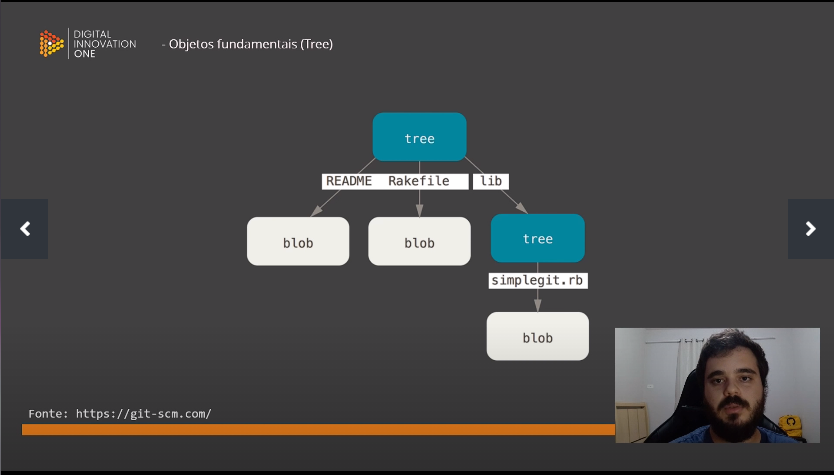
POR ESSE MOTIVO O HASH DO SHA1 DA DIFERENÇA DO HASH DO BLOB, MAS SE PASSADO OS METADADOS DO BLOB JUNTO DA STRING ( OU ARQUIVO) O HASH SERÁ O MSM

O git vai guardar o hash, mas vai conter os metadados listado acima

TREE

As tree armazena e aponta BLOBS e tbm contem metadados além de guardar o nome do arquivo

A arvore é responsável por montar a estrutura dos arquivos, localização, elas podem apontar blobs e outras arvores e tbm possui seu próprio sha1, que se muda qualquer coisa seja na blob ou em outra arvore, ela altera



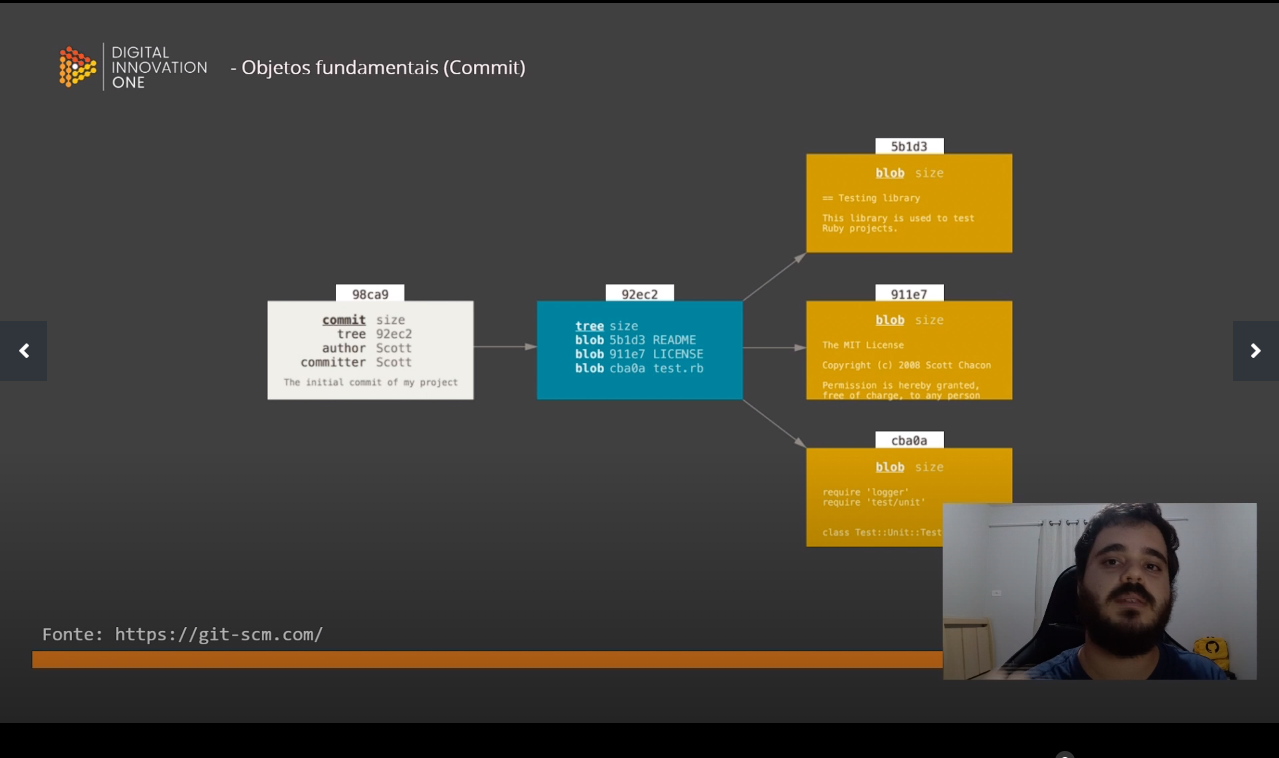
COMITT

Ele junta tudo, da sentido a alteração, ele aponta para arvore, para um parente ( ultimo commit usado antes dele) aponta para um autor e para mensagem

Tudo está ligado a alteração, e essa mensagem passa a dar sentido ao que mudou

timestamp “ carimbo de tempo” leva a hr e a data de quando foi criado

Os commit tbm possui o seu sha1, ou seja, qualquer coisa que foi alterado dentro vai alterar o sha1 gdo commit



Pq é seguro ?

Imagina seu código na nuvem, e é a versão final do seu código e digamos que tem 30 pessoas na equipe

E nas 30 maquinas e no servidor terá o msm commit, logo se em um desses lugares acontecer algo com o programa e ser deletado ou algo do tipo, terá ainda mais 30 lugares com a versão que estava rodando, não se perde

INICIANDO O GIT E CRIANDO UM COMMIT

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

git init (inicia o git dentro da pasta possibilitando o versionamento do código ) (cria uma pasta oculta, que é gerencial do próprio git criando um repositorio) (usando o ls -a mostra as pastas ocultas)

git add nomeArquivo ( com o nome do arquivo o adiciona para joga-lo em unmodified e poder se tornar commit,)

git add \* ( adiciona tudo que tem na pasta adiciona para joga-lo em unmodified e poder se tornar commit)

git add. ( adiciona todas as modificações do repositório local para staging área)

git commit

git status ( comando para mostrar se o arquivo esta unmodified, modified, se há algo para ser commitado ou não )

git config --global user.email “email aqui”

git config --global user.name nomeaqui

git congi --global list ( lista todas as config setadas, username, email e etc)

git config --global --inset user.email (apaga email setado )

git config --global --inset user.nickname(apaga o nome setado )

git remote add origin( “origin” é apelido) url( o próprio github gera, imagem na pagina 12 para ajudar)

git remote -v ( mostra a lista de repositórios que está cadastrado)

git push origin master ( é literalmente o comando que empurra o commit do repositório local para o repositório remoto)

git pull origin master ( puxar o comando do git pra maquina o git já vai tentar juntar os 2 )

git clone url(url do código do github) ( para poder clonar o código pro repositório local)

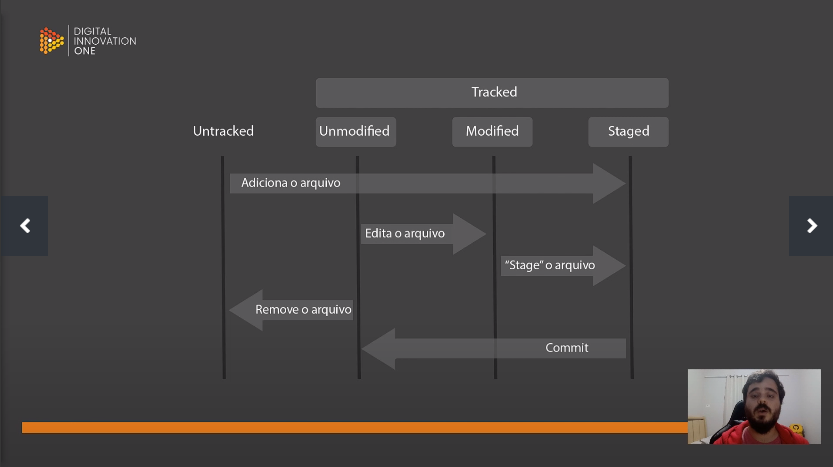
markdown é uma forma humanizada de escrever html

Passo a passo no ciclo de vida

Tracked e untracked

Dentro dos arquivos que são rastreados pelo git

São 3 estagios diferentes, sendo eles Unmodified, Modified e Staged



Unmodified

Arquivo que não foi alterado quando altera o arquivo ele passa a ser modified ( o git faz uma comparação do sha1, se houve alteração no sha1 o arquivo foi alterado) se o arquivo não foi alterado e é deletado, ele volta pro Untracked

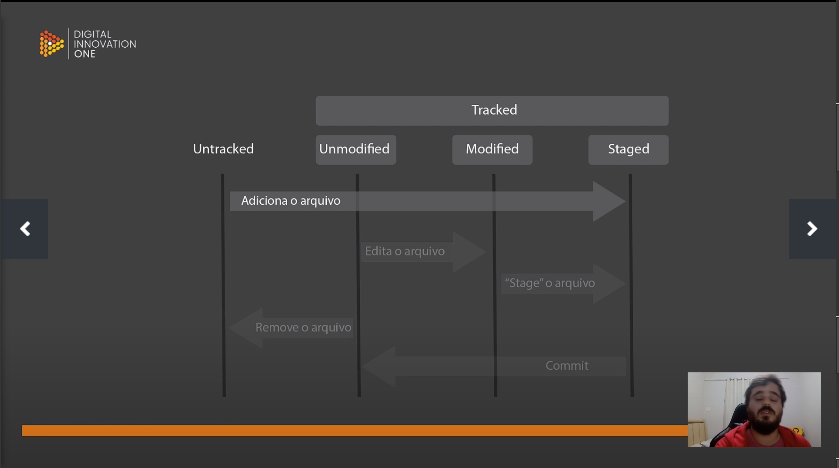
Modified

Arquivo que foi alterado

Staged

É o arquivo que já está pronto para ir pra próxima parte, um outro tipo de agrupamento, vai para o commit do commit volta para unmodified para que o ciclo seja resetado

GIT ADD



Quando usamos, tínhamos um arquivo untracked, quando usamos o git add, ele moveu o arquivo, diretamente para o Staged

EXEMPLO DE REPOSITÓRIOS

2 ambientes

O de dev

E o do servidor

Se vc altera o programa no seu ambiente de dev, vc não altera o do servidor

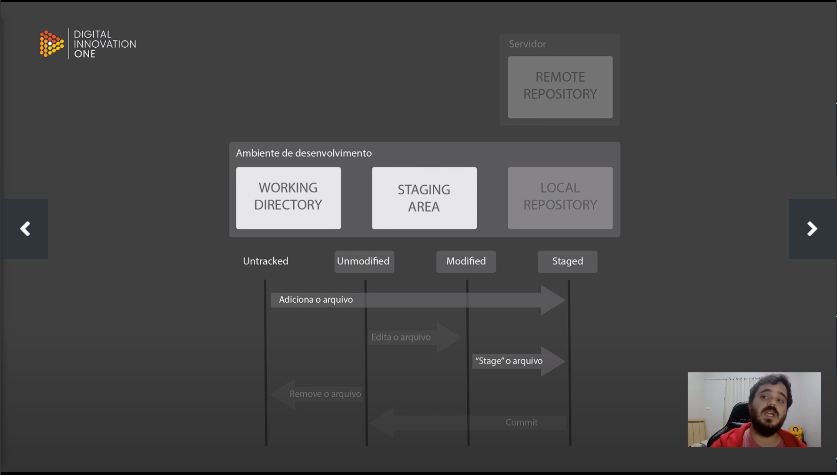
No ambiente de dev tem o:

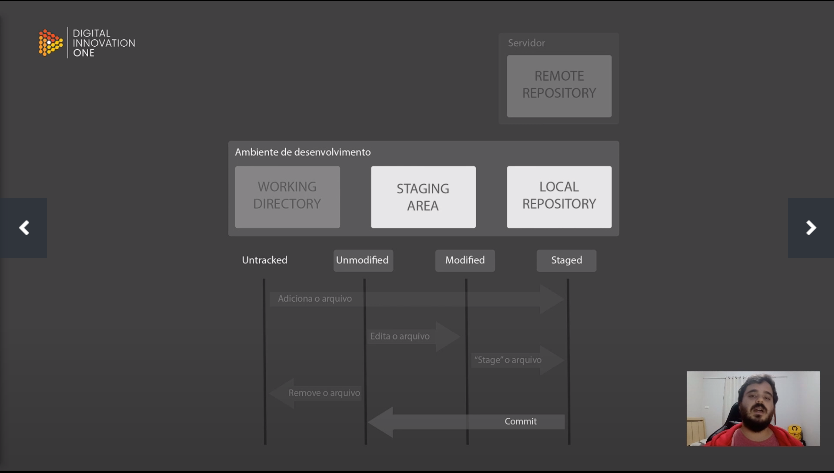
Working DIrectory ( repositório de trabalho - unmodified)

Staging área

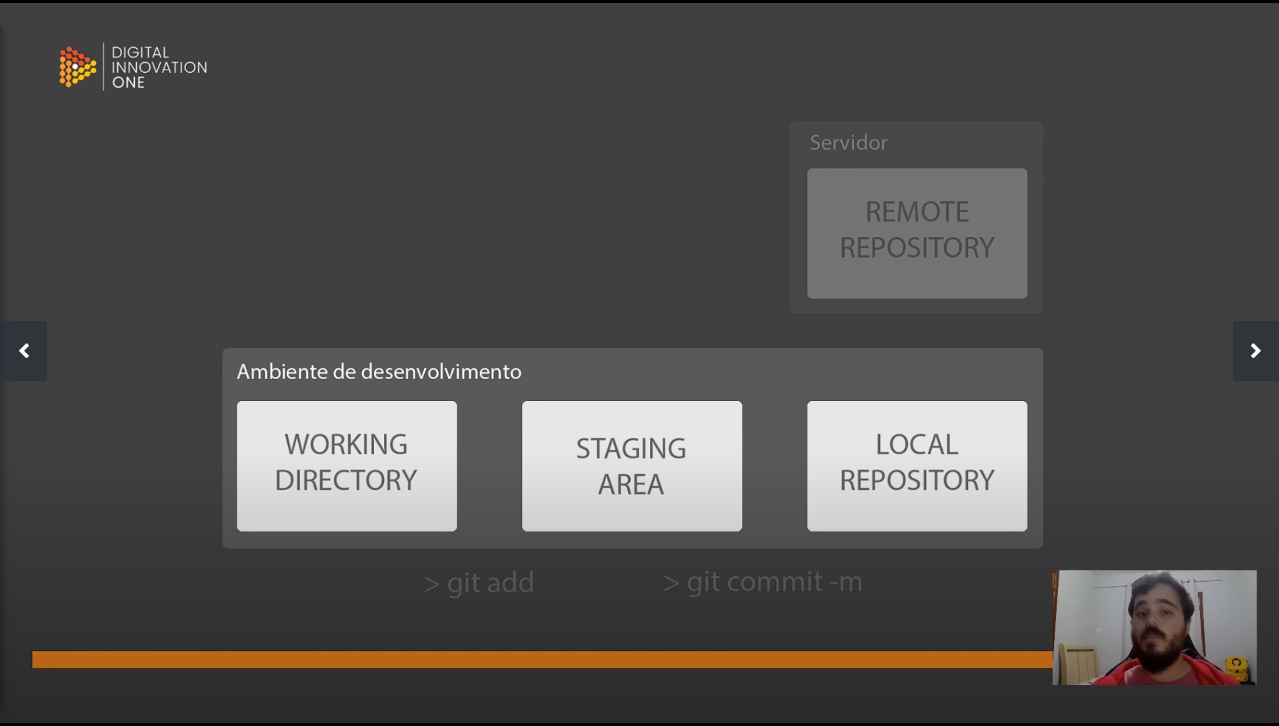
Repositório local ( só vai ser composto unicamente por commit ) ( único estagio que pode mandar o commit para um ambiente remoto ( servidor ))

Os arquivos vai ficar indo e vindo do repositório de trabalho pra staging e quando der um commit ele passa a integrar o repositório local





O caminho é

  
> git add > git commit -m

Trabalhando com o GitHub

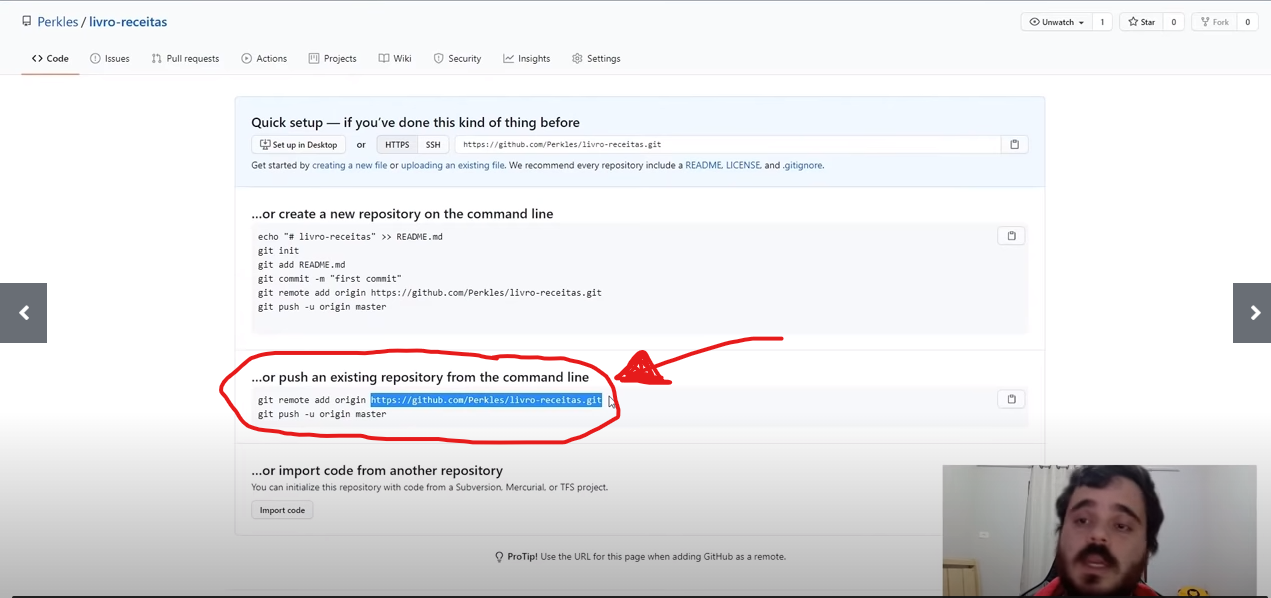
* Criar conta
* Criar repositório remoto
* Mandando o corrido do repositório local para o repositório remoto

Alterar o usuário ou email no git, altera tbm o sha1

De preferencia sempre manter o email e user do git e do github iguais, pq ele busca dentro da plataforma o email linkado

ARQUIVOS README

Conta a história daquele repositório



Resolvendo conflitos

É comum dar conflitos, veremos alguns deles

Situação hipotética:

2 pessoas vão alterar o msm arquivo do repositório do github, automaticamente está na msm versão até que um deles altere; Quando um deles altera o código na msm linha que o outro e empurra para o github, automaticamente o versão de quem não alterou estará desatualizada e quando for empurrar para o github, ele vai devolver um erro “ vc precisa antes de empurrar, baixar a versão atualizada e só então adicionar suas alterações” e isso é um “conflito de mush”

Há algumas maneiras de baixar códito do github

Open with Codespaces

Open with GitHub Desktop

Donwloa dZIP

E Clonando direto para o terminal através da url

GIT CLONE  
Pegar o repositório para fazer alterações

A diferença é que quando vc clona, tem a pasta .git ( significa que tem repositório podendo ser conferido com o git remote -v)