GIT e GITHUB

Git – Software criado em 2005 pelo Linus Tovards, é um sistema de versionamento de código distribuído, ajuda a criar e monitorar diferentes versões de código dentro de uma máquina.

GitHub – repositório online, onde se armazena os códigos. É uma tecnologia complementar do Git, então elas não são iguais.

Benefícios do aprendizado dessas duas tecnologias:

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

**COMANDOS BÁSICOS PARA UM BOM DESEMPENHO NO TERMINAL**

A maioria dos programas possuem forma gráfica (GUI – *Grafic User Interface*), é responsivo aos comandos do usuário.

O design do GIT é voltado para comando (linha de comando), ou seja, é CLI (*Command Line Interface*).

Aprender primeiro:

- Mudar de pastas

- Listar as pastas

- Criar pastas/arquivos

- Deletar pastas/arquivos

Primeiros Comandos:

Iniciar – procurar por CMD para abrir o Prompt de Comando.

**Listar as pastas – dir (Windows) e ls (Linux ou Mac)**

Lista de diretórios contidos na pasta no qual a gente está situado

Todos os comandos utilizados possuem variâncias, ou seja, possuem “flags” que são complementos que passa para os comandos, onde eles acrescentam, modificam ou formatam a forma como esses comandos são devolvidos para gente.

**Mudar de pastas – cd (Windows e Linux)**

cd / “nome da pasta” (Ir até uma pasta)

cd.. (Retroceder um nível da navegação)

cls (Limpar o terminal no Windows) ou clear ou Ctrl +L (Limpar terminal no Linux)

Tab – auto completar, só escrever a primeira letra e apertar a tecla Tab que ele completa.

**Criar pastas/arquivos – mkdir para criar um diretório (Windows e Linux)**

Comando echo para criar arquivo e ele printa uma frase ou um texto

**Deletar pastas/arquivos**

No Windows tem diferença entre deletar arquivos e deletar repositórios.

**del** restringe a apenas em deletar arquivos

**rmdir “nome da pasta” /S /Q (Deleta a pasta com todos os arquivos no Windows)**

**rm -rf “nome da pasta”(Linux)**

**INSTALAÇÃO DO GIT**

Site: git-scm.com

Ao instalar, aparece uma tela e aperta Next, verificar se as opções “*Git Bash Here”* e “*Git GUI Here”* estão marcadas, escolher um editor padrão (Vim), marcar “Let Git decide”, marcar “Git from the command line and also from 3rd-party software”, “Use bundled OpenSSH”, “Checkout Windows-style, commit Unix-style line endings”, “Use MinTTY (the default terminal of MSYS2)”, ”Default (fast-forward or merge)”, “Git Credential Manager Core”, “Enable file system caching” e “Enable symbolic links” e prosseguir com a instalação.

**TÓPICOS FUNDAMENTAIS PARA ENTENDER O FUNCIONAMENTO DO GIT**

**SHA1 (*Secure Hash Algorithm*)**

É um algoritmo de criptação, ou seja, é um conjunto de funções *hash* criptográficas projetas pela NSA (Agência de Segurança Nacional). É uma forma curta de representar um arquivo.

A saída desses dados encriptados gera um conjunto de caracteres de 40 dígitos que serve para identificação. Muito eficiente para identificar os arquivos de forma segura e rápida.

Roda não apenas para os arquivos, mas também para objetos internos do Git.

Utilização do Git:

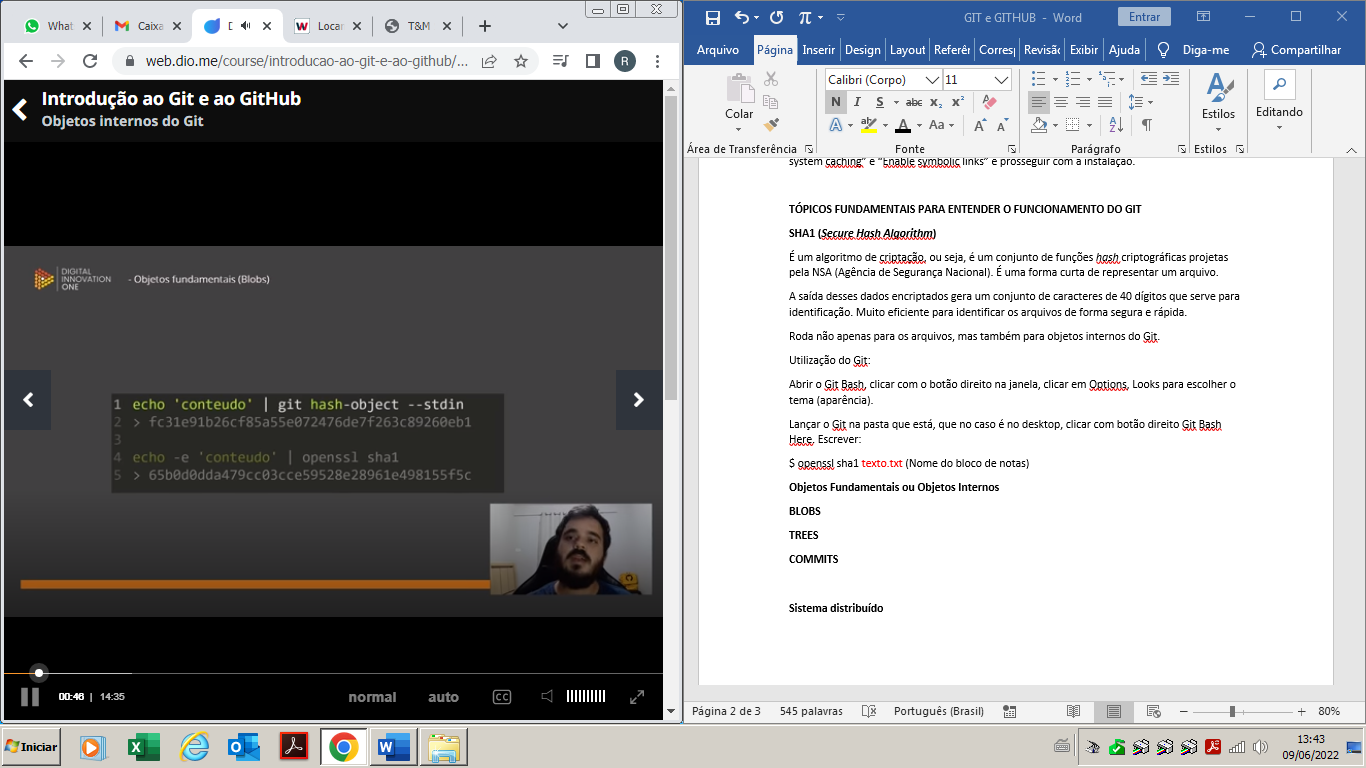
Abrir o Git Bash, clicar com o botão direito na janela, clicar em Options, Looks para escolher o tema (aparência).

Lançar o Git na pasta que está, que no caso é no desktop, clicar com botão direito Git Bash Here. Escrever:

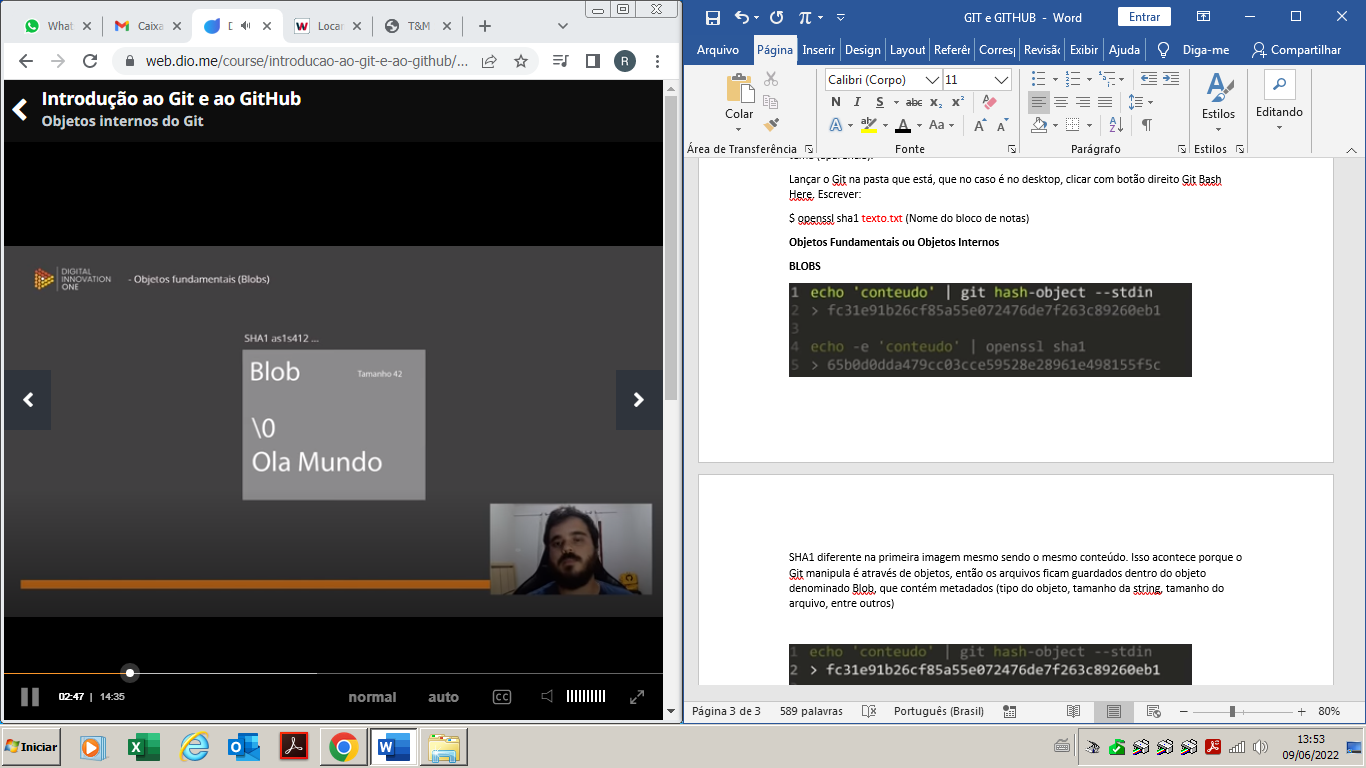
$ openssl sha1 texto.txt (Nome do bloco de notas)

**Objetos Fundamentais ou Objetos Internos**

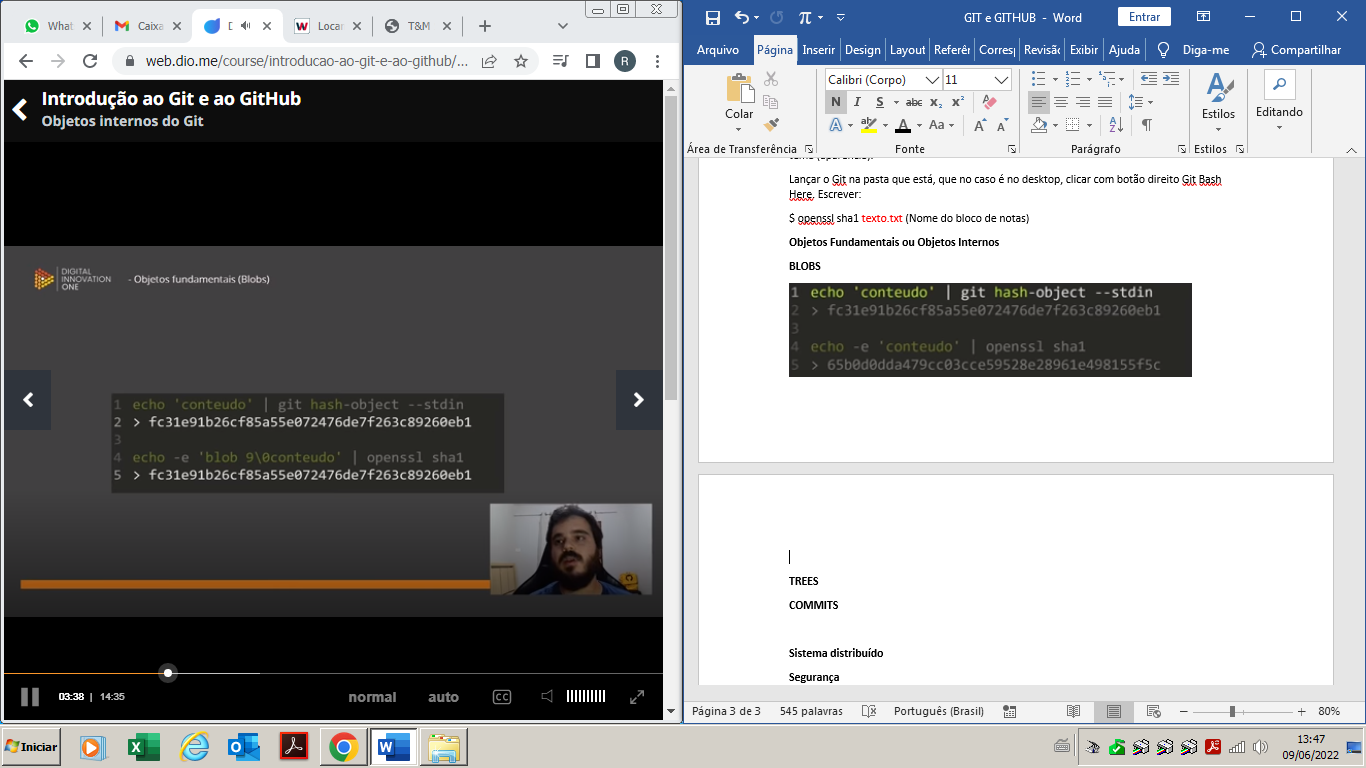
**BLOBS**



SHA1 diferente na primeira imagem mesmo sendo o mesmo conteúdo. Isso acontece porque o Git manipula é através de objetos, então os arquivos ficam guardados dentro do objeto denominado Blob, que contém metadados (tipo do objeto, tamanho da string, tamanho do arquivo, entre outros)



A partir disso, temos:



**TREES**

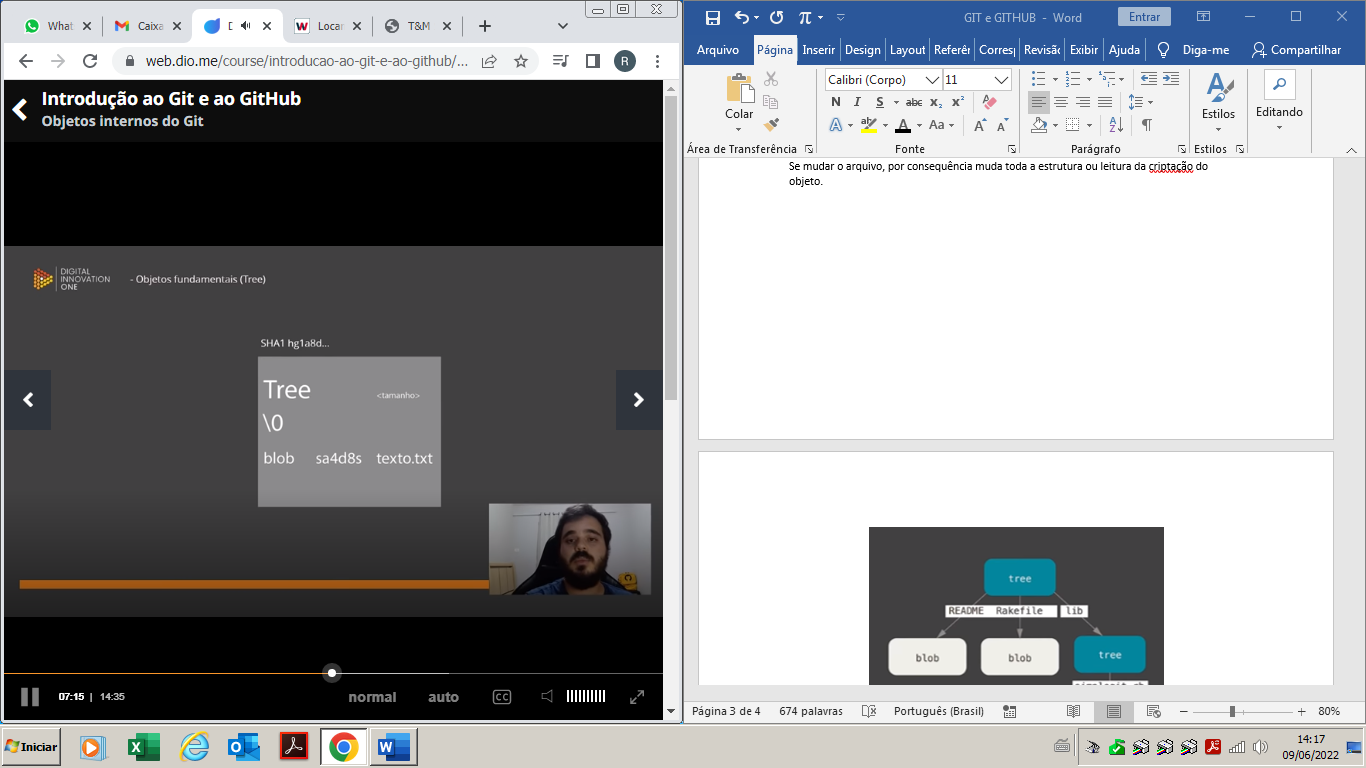
Armazenam Blobs, contém metadados e aponta para um Blob, que por sua vez tem sua SHA1, além disso a Tree guarda o nome do arquivo. É responsável por montar toda a estrutura de onde estão localizados os arquivos.

Podem apontar para outra tree ou bloob.

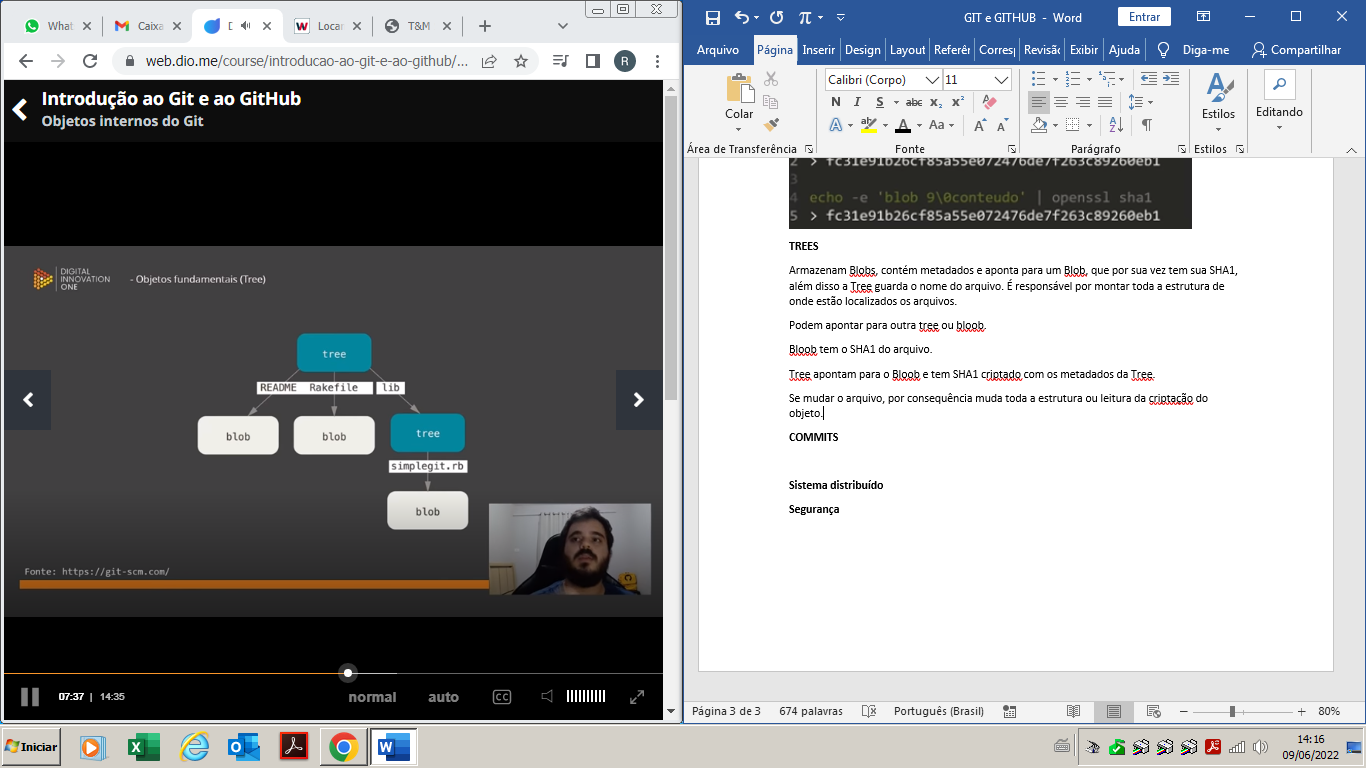
Bloob tem o SHA1 do arquivo.

Tree apontam para o Bloob e tem SHA1 criptado com os metadados da Tree.

Se mudar o arquivo, por consequência muda toda a estrutura ou leitura da criptação do objeto.



Outra estrutura do Tree:



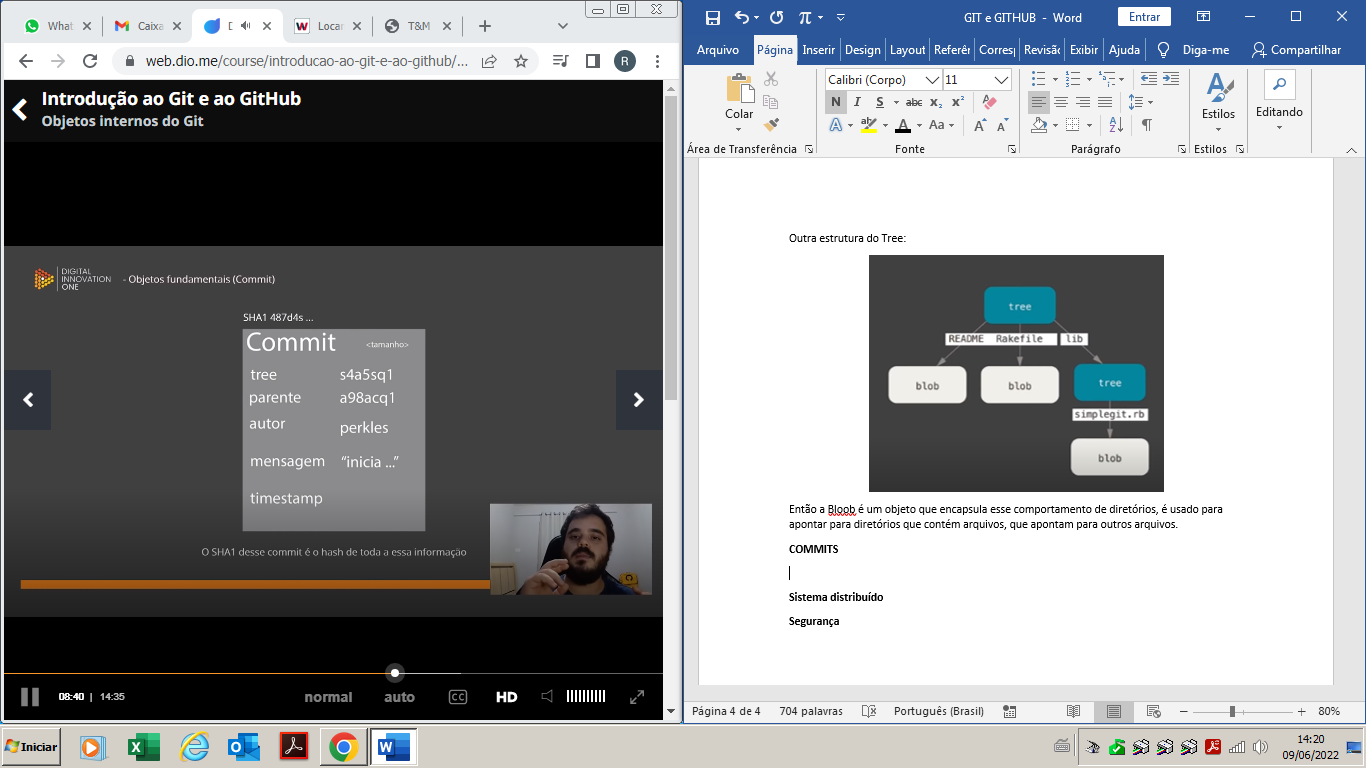
As Trees retêm informações de quais arquivos que ela está apontando (bloobs ou trees), que consequentemente pode apontar para outros arquivos.

**COMMITS**

Objeto que junta tudo, vai dar sentido pra alteração que está fazendo. Possui um SHA1, se alterar um dado na Bloob, altera a Tree e a Commit.

Se muda qualquer coisa no arquivo, altera tudo. Por isso o Git é confiável, quando se tem um commit, garante-se que ninguém alterou o commit, ou seja, é uma forma segura de demonstrar que aquele commit representa o que você quer dizer com o código, porque não teve interferência de outra pessoa.

Ele é o único pra cada autor e tem todos os metadados.



**Sistema distribuído e Segurança**

Pelo fato dos commits serem difíceis de serem alterados, tanto a versão recente quanto as outras que estão distribuídas e mantém o sistema também são confiáveis. Caso aconteça algum problema na nuvem da Github, precisa acontecer em todas as versões por conta da estrutura que o Git mantém e de como foi projetado.

**CHAVES SSH E TOKENS**

Quando coloca o código no GitHub (plataforma em nuvem) precisa se autenticar (nome e senha), porém foram feitas as mudanças para deixar mais seguro, ou seja, é necessário fazer outros processos para se autenticar.

**Chave SSH** é uma forma de estabelecer uma conexão segura e encriptada entre duas máquinas, no caso o servidor do GitHub e a máquina local, estabelecendo uma conexão através de duas chaves: pública e privada.

A chave pública coloca na GitHub, onde reconhecerá a máquina local, ou seja, todos os depositórios que tiver na máquina por processo do SSH serão reconhecidas pela assinatura da máquina, tendo uma conexão.

**Token de acesso pessoal** – sempre que fazer commit vai usar o token de acesso pessoal.

**PRIMEIROS COMANDOS COM O GIT**

- git init (inicia o repositório git)

- git add (mover arquivos e dar início ao versionamento e conhecer os primeiros comandos)

- git commit (criar commit)

Todo comando do git, leva o git na frente e o comando após.

**CRIANDO UM REPOSITÓRIO**

Flag “-a” mostra pastas ocultas

**CICLO DE VIDA DOS ARQUIVOS**

**GIT INIT**

**Criando um repositório no git dentro de uma pasta ou diretório.**

**Tracked ou Untracked;**

Dentro de Tracked (arquivos rastreados pelo git) são subdivididos em 3 fases diferentes:

- Unmodified (arquivo não foi modificado)

- Modified (arquivo modificado)

- Staged (arquivos se preparando para fazer parte de um outro tipo de agrupamento)

Untracked (arquivos que o git não tem ciência).

Git tem a versão no servidor e na máquina, em que as alterações não repercurte no repositório remoto (servidor) a não ser que execute um conjunto de código específicos.

Dois ambientes: Ambiente de desenvolvimento e Servidor

No ambiente de desenvolvimento tem o repositório de trabalho e a área de staging, os arquivos ficam alterando entre o repositório de trabalho e a área de staging à medida que vai adicionando ou modificando novos arquivos do repositório.

Quando faz o commit, passa a integrar o seu repositório local que pode ser empurrado para o repositório remoto.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de computador com jogo

Descrição gerada automaticamente

Quando se usa git add, está sendo movido os arquivos untracked direto para staged e os arquivos modificados movem para staged.

Quando usa o git commit – m “msg”, pega os arquivos do staged e envelopou eles em uma mensagem, dando significância a eles, criando o objeto commit.

**RESOLVENDO CONFLITOS**

**Conflito de Merge – github tenta juntar duas alterações na mesma linha de um código, porém o github não altera nada, ele dá a opção do autor fazer manualmente resolva o conflito antes de empurrar para o Github.**