

Entendendo como o Git funciona por baixo dos panos

4 Tópicos fundamentais para entender o funcionamento do Git

SHA1

O que é?

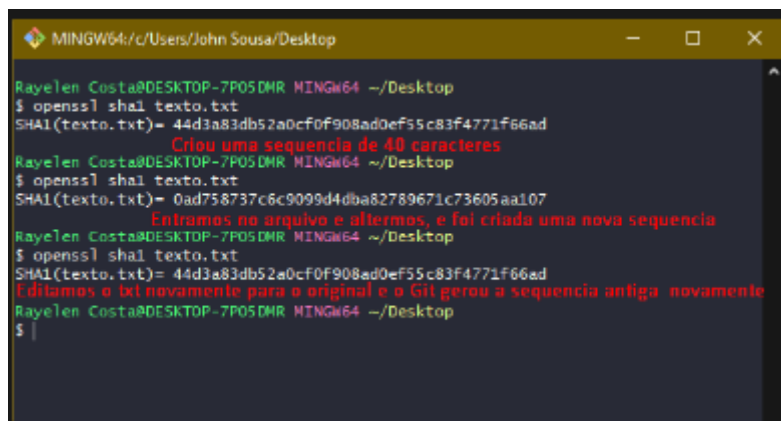
A sigla SHA significa Secure Hash Algorithm (Algoritmo de Hash Seguro), é um conjunto de funções hash criptográficas projetadas pela NSA (Agência de Segurança Nacional dos EUA).

Por que é Relevante para Nós?

A encriptação gera um conjunto de caracteres identificador de 40 dígitos único. O Git efetua uso dessa encriptação para poder identificar os arquivos de uma forma segura e de forma rápida.

Exemplo Prático:

- Para unificar os comandos tanto para quem usa Linux e Windows, usaremos o GIT BASH.
- Em vez de abriremos o GIT Bash pelo pesquisador, pois, ele abrirá em uma pasta totalmente aleatória. Basta entrar no local desejado (no nosso caso área de trabalho), clicar o botão direito e clicar em Git Bash Here.



```
MINGW64: c:/Users/John Sousa/Desktop
Raylen Costa@DESKTOP-7P05DMR MINGW64 ~/Desktop
$ openssl sha1 texto.txt
SHA1(texto.txt)= 44d3a83db52a0cf0f908ad0ef55c83f4771f66ad
Criou uma sequência de 40 caracteres
Raylen Costa@DESKTOP-7P05DMR MINGW64 ~/Desktop
$ openssl sha1 texto.txt
SHA1(texto.txt)= 0ad758737c6c9099d4dba82789671c73605aa107
Entramos no arquivo e alteramos, e foi criada uma nova sequência
Raylen Costa@DESKTOP-7P05DMR MINGW64 ~/Desktop
$ openssl sha1 texto.txt
SHA1(texto.txt)= 44d3a83db52a0cf0f908ad0ef55c83f4771f66ad
Editamos o txt novamente para o original e o Git gerou a sequência antiga novamente
Raylen Costa@DESKTOP-7P05DMR MINGW64 ~/Desktop
$
```

- Podemos observar que isso é uma forma muito inteligente e eficiente do GIT de garantir e identificar que os arquivos sofreram modificação.
- Identificar também que naquele arquivo, variando de uma versão para outra ele sofreu ou não modificações.
- Garantindo assim que dentro do arquivo tenha exatamente o conteúdo correto.

Objetos internos do Git

3 Tipos Básicos de Objetos do GIT

- Responsáveis pelo versionamento do nosso código.

BLOBS

Usaremos a função `hash-object` e a flag `--stdin` (serve para informar que estamos mandando texto em vez de um arquivo, pois o `hash-object` espera receber arquivos).

```
1 echo 'conteudo' | git hash-object --stdin
2 > fc31e91b26cf85a55e072476de7f263c89260eb1
3
4 echo -e 'conteudo' | openssl sha1
5 > 65b0d0dda479cc03cce59528e28961e498155f5c
```

```
1 echo 'conteudo' | git hash-object --stdin
2 > fc31e91b26cf85a55e072476de7f263c89260eb1
3
4 echo -e 'blob 9\0conteudo' | openssl sha1
5 > fc31e91b26cf85a55e072476de7f263c89260eb1
```

Por que ao enviar por funções diferentes o GIT retornou uma sequência de caracteres diferente?

- Pois, do jeito que o GIT lida e manipula eles são por objetos.
- Então, os arquivos ficam guardados dentro desse objeto chamado “blob” e esse objeto contém metadados dentro dele. Ou seja, o objeto “blob” terá o tipo de objeto, o tamanho dessa string ou desse arquivo, uma “\0” e o conteúdo de fato desse arquivo.

O que aprendemos?

Que o GIT irá guardar esses arquivos fazendo o “sha1” dele, gerando a encriptação dele, mas que ele também armazena metadados nesses objetos.

Então nesse primeiro objeto “blob” ele contém metadados do GIT, que são o tipo de objetos, o tamanho da string, o tamanho do arquivo, entre outros.

O “blob” só guarda o “sha1” do arquivo, ele ainda encapsula o comportamento de diretórios.

TREES

As “tree” armazenam “blobs”, ou seja, uma crescente.

Onde o “blob” sendo o bloco básico de composição a “tree” armazenando e apontando para o tipo de “blobs” diferentes e um outro tipo de estrutura de dados, que são os “commits”.

Então a “tree” também tem metadados e o “\0”, que por sua vez tem um “sha1” e a árvore guarda também o nome desse arquivo.

A “tree” será responsável por montar toda a estrutura e onde está localizado os arquivos.

Podendo apontar tanto para o “blob” ou outras “trees”. Da mesma forma que no computador um diretório pode ter dentro de si outro diretório.

A ‘tree’ tem um “sha1” desse metadado.

BLOBS (tem o sha1 do arquivo) → ←TREE (aponta para os blobs).

COMMITTS

É o objeto que juntará tudo, que dará sentido para alteração que você está fazendo.

O “commit” aponta:

- TREE
- “PAREDE”: Aponta para o último “commit” realizado antes dele
- AUTOR
- MENSAGEM

Ele também tem um “timestamp” sendo um carimbo de tempo, data e hora de quando foi criado.

Possuem também um “sha1” dos seus metadados, ou seja, se alteramos uma blob ele gerará um sha1 daquela blob que por sua vez tem uma tree apontando-lhe que terá seus metadados alterados por causa da alteração feita. E o commit aponta para uma tree que por sua vez pode apontar para outras trees. Ou seja, se alteramos qualquer coisa refletirá em tudo.

O commit é único para cada autor.

Por que é um Sistema Distribuído Seguro?

Pelo fato dos commits serem tão difícil de ser alterados, tanto a versão mais recente que está na máquina do servidor quanto qualquer versão distribuída também são versões confiáveis.

Chave SSH e Token

Quando você joga seu arquivo no GitHub você precisa se autenticar que são a Chave SSH e Token.

Chave SSH

E uma forma de estabelecer uma conexão segura entre duas máquinas.

Como pegar as chaves?

- Entra no GitHub → Settings (Definições) → SSH and GPG Keys → SSH Keys
- Abre o GIT Bash → Comandos:
 - `ssh-keygen -t ed25519 (tipo de criptografia) -c "seu e-mail"`
 - Depois da entrada > coloca uma senha "061464"
- Entra na pasta .ssh → Digite esse comando para vermos a chave → Cópia a Chave.
 - `id_ed25519` = Chave Privada
 - `id_ed25519.pub` = Chave Pública
- No GitHub em SSH Keys → Dê um nome a máquina e coloque a chave que pegamos no GIT Bash → Add SSH Key
- Agora precisamos inicializar o SSH Agent:
 - Digite no Git Bash → `eval $(ssh-agent -s)`
 - Depois executa o comando → `ssh-add id_ed25519`

Token Pessoal de Acesso

Se assemelha mais ao processo antigo de informar o usuário e senha.

- Entra no Github → Settings → Developer Settings → Personal Access Tokens → Generate New Token
- Dê um nome para a máquina e uma data de expiração pro Token → Marcar opção Repo → Generate Token