**Introdução Git e GitHub**

**Por quê?**

Porque ao construirmos projetos várias versões surgem devido a quantidade de informações que são levantadas e alteradas ao longo de sua execução. Desta forma há a necessidade de um versionamento do mesmo, onde várias versões de um mesmo projeto são criadas e caso erros aconteçam ao longo do caminho, podem ser revertidos ao estado anterior sem comprometer todo o projeto. Por isso a necessidade de sistemas de versionamento de código como o GIT

**GIT**

Criado por Linus Torvalds, também criador do sistema LINUX.

**GitHub**

Empresa da Microsoft, Git e GitHub são coisas distintas. Git é um software de gerenciamento de versões. GitHub é um repositório online.

**Benefícios no uso de Git e GitHub**

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

Comando DIR (Windows) e LS(Unix) servem para listar os diretórios existentes.

CD (change directory) utilizado para alterar o diretório.

cd nome da pasta – para entrar na pasta

cd .. – para retroceder um nível

cls (Windows) e clear (unix) para limpar o terminal.

mkdir nome da pasta – usado para criar uma pasta

echo conteudo > nome conteúdo.txt – para criar um arquivo

del nome da pasta – deleta todo o conteúdo dentro dela, restringe-se apenas a deletar arquivos

rmdir nome do diretório /S /Q (WINDOWS) – deleta todo o diretório

rm -rf nome do diretório/ (UNIX) - deleta todo o diretório

Como o GIT Funciona por baixo dos panos

SHA1

É um algoritmo de encriptação (SHA – Secure Hash Algorithm), constituído por funções hash criptográficas projetadas pela NSA (Agência de Segurança Nacional dos EUA). Através desse algoritmo de encriptação há uma saída de um conjunto de caracteres indentificador de 40 dígitos que é único. É uma forma curta de representar um arquivo. Ele utiliza a mesma chave de encriptação para as mesmas quantidades de caracteres presentes, ou seja, se alterarmos o arquivo original e pedirmos para ser encriptado uma chave será gerada, se alterarmos um ponto dentro do arquivo ao pedirmos para encriptar novamente outra chave será geradas, se retornarmos a alteração e pedirmos uma nova encriptação a primeira chave será retornada novamente.

*1 echo “ola mundo” | openss1 sha1*

*2 > (stdin) = f9fc856e559b950175f2b7cd7dad61facbe58e7b*

Objetos dentro do GIT

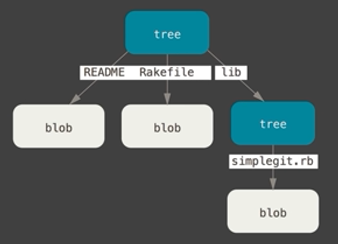
BLOBS – a estrutura do bloob possui o tipo do objeto, o tamanho da string/arquivo, \0 e o conteúdo do arquivo. Não guarda o nome do arquivo, somente o SHA dele. São os “arquivos”.



TREES – armazenam os blobs, contém também metadados e apontam para um blob. Guardam também o nome do arquivo além do SHA dele, diferente do Blob. Elas podem apontar tanto para Blobs (“arquivos”) quanto para outras árvores, isso faz sentido por diretórios poderem conter outros diretórios dentro de si.

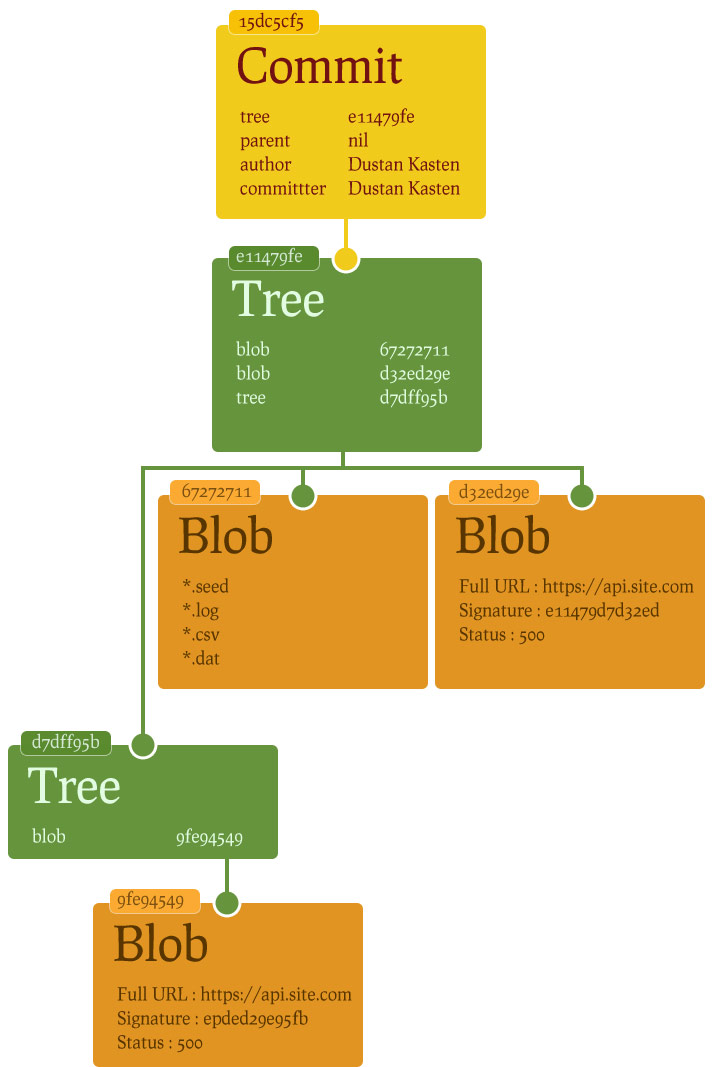


Em resumo até aqui, as bolhas/blobs contém o SHA1 do arquivo, as arvores/trees apontam para essas blobs e nessa tree há um SHA1 encriptado nos metadados da árvore. Dessa forma caso houver alteração de um vírgula sequer em algum arquivo no qual essa árvore está apontando, quando ela gerar o SHA1 desses metadados o SHA1 da bolha terá mudado o que irá também refletir no SHA1 da árvore. Então uma coisa sempre está atrelada a outra. Então ao alterar um arquivo, toda a estrutura é mudada, desta forma toda a interpretação/leitura e encriptação da árvore é alterada.

Então a tree é um objeto que encapsula esse comportamento de diretórios e é utilizada para apontar para diretórios que contenham arquivos e por consequência apontam para arquivos também.

COMMITS – é o objeto mais importante de todos. Ele aponta para a árvore, para um parente (último COMMIT realizado antes dele), para o autor, para a mensagem (pode-se escrever uma mensagem explicando que alteração foi feita e porque). Possui também um timestamp que é uma espécie de carimbo de tempo, que leva a data e hora de criação do COMMIT.

O commit também possui um SHA1 portanto qualquer alteração de uma blob irá alterar uma árvore que pode ou não apontar para outras árvores, isso também irá alterar o SHA1 do commit. Isso que torna o GIT tão confiável já que há registro de toda alteração que ocorrer além do autor da alteração.

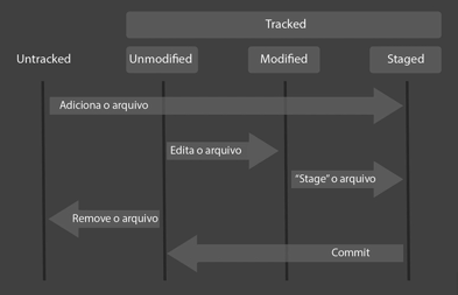


Sistema Distribuído Seguro

Git é um sistema distribuído. Vamos supor que há um repositório hospedado em um servidor na nuvem e o código que vive lá representa o estado final do código do sistema/software sendo a versão mais recente. Nesse repositório há 40 pessoas que contribuem nele, então na máquina de 40 pessoas há também uma versão desse código, pelo fato dos commits serem tão difíceis de serem alterados, tanto a versão do servidor quanto as que estão nas máquinas desses contribuidores também são versões confiáveis. Portanto se ocorrer um problema na nuvem e o código deixar de existir o mesmo deve ocorrer com os códigos dessas 40 máquinas porque as versões deles também são extremamente seguras. Por isso o GIT é um sistema distribuído seguro.

Primeiros Comandos GIT

- git init – iniciar diretório. Inicializa repositório

 Untracked – arquivos que o GIT não tem conhecimento que existem, que acabaram de ser criados. Ou que são removidos do GIT.

Tracked – arquivos que o GIT conhece a existência e rastreia.

Unmodified – arquivo que não sofreu modificação e passa para o estado de modified automaticamente após uma edição.

Modified – arquivo que foi modificado. Quando adicionado a um COMMIT vai para Staged para aguardar COMMIT.

Staged – arquivos que estão se preparando para fazer parte de um outro tipo de agrupamento conhecido como COMMIT. Ou seja, todas as modificações se tornam um “envelope” com mensagem, autor, data, etc. e retornam para Unmodified para todo o processo ser iniciado ou não novamente.

- git add – utilizado para mover arquivos do untracked ou modified direto para staged, então movemos os arquivos para área que vai entrar em ação que será o grupo commit.

>git add nomeArquivo

>git add \*

>git add.

- git commit – pegamos toda a área do staged (“backstage antes de entrar no palco para atuar”). Então “envelopamos” em uma mensagem todos os arquivos que estavam na área de staged dando significância a eles e cria-se esse objeto chamado commit.

>git commit -m “msg...”

Então para exemplificar, no ambiente de desenvolvimento, pegou-se o repositório de trabalho rodou o GIT ADD que agrupou todos aqueles arquivos untracked e modified e moveu-os para o Staging Area.

Utilizando o comando GIT COMMIT -M e a mensagem os arquivos da Staging Area foram inseridos em um objeto COMMIT que foi disponibilizado no repositório local.

Trabalhando com GitHub

Para empurrar o diretório local para o GitHub utiliza-se o seguinte comando.

git remote add origin link repositório do GitHub – utilizado para criar a origem

git push origin master – utilizado para empurrar arquivos para o repositório remoto.

Resolvendo Conflitos

Ocorrem com frequência em sistemas distribuídos. Quando possuímos um código no GitHub e na máquina local idênticos sem alterações temos então um versionamento de código de forma distribuída. Suponhamos que outra pessoa faça um clone do código disponível no GitHub, neste momento ainda não houve alteração portando ambos possuem o mesmo código presente no GitHub de forma sincronizada. Em seguida ambos abrem o código em suas respectivas máquinas, portanto houveram alterações diferentes no código partindo de duas máquinas distintas, portanto o código agora está dessincronizado. Suponha que a pessoa que clonou o código empurre a alteração e você permanece alterando o código. Tanto você quando a pessoa que clonou o código alterou uma mesma linha. Agora então o código atualizado é o da máquina que clonou o código e o código do GitHub, uma vez que ela empurrou/push o código pro GitHub.

Ao submeter o código para o GitHub ele retornará avisando que houve uma alteração no código e irá pedir que antes seja puxado o código do GitHub. Nesse momento surge o conflito de merge, onde houve duas alterações na mesma linha em locais diferentes. Então é solicitado que o conflito seja resolvido manualmente, para depois ser carregado ao GitHub. Então puxamos primeiro.

git pull origin master – para puxar o código alterado do GitHub

Desta forma resolvemos os conflitos, baixando o arquivo do repositório remoto, comparando com o arquivo do repositório local e por fim commitando as alterações a serem salvas no repositório remoto sincronizando assim com o repositório local.

Clonar Repositório do GitHub

Para clonar um repositório no diretório local basta utilizar o código abaixo.

git clone link do repositório no GitHub