

Entendendo o que é Git e sua importância

Git é uma ferramenta para gerenciar código, permite o controle de versão. O GitHub permite o armazenamento em nuvem do código, de forma que outras pessoas podem acessar os arquivos. Utilizando Git e GitHub em conjunto tem como vantagem obter reconhecimento conforme os programadores compartilham seus códigos, permite também o trabalho em equipe e o melhoramento de seu código conforme uma ou mais pessoas trabalhem sobre um determinado repositório.

O Git também é um sistema distribuído, pois conforme foi dito anteriormente várias pessoas podem contribuir com um repositório, além disso é seguro devido ao fato da utilização de funções hash criptografadas denominada SHA1, projetada pela NSA. A encriptação gera conjunto de caracteres identificador de 40 dígitos, é uma forma curta de representar um arquivo.

O git possui 3 objetos internos, e todos eles possuem o seu determinado SHA1, qualquer alteração em algum arquivo é refletido através da mudança nos dígitos do SHA1, ou seja é uma forma unívoca de representação. Os objetos são Blobs, Trees e Commits.

Uma Blob é um metadado do git que contém um tamanho, o caractere \0 e o conteúdo propriamente dito. Como exemplo a figura 1 mostra o SHA1 de um objeto que é apenas a string “conteúdo” utilizando o bash do git, observa-se também que o mesmo SHA1 é criado ao passar para o comando específico a blob.

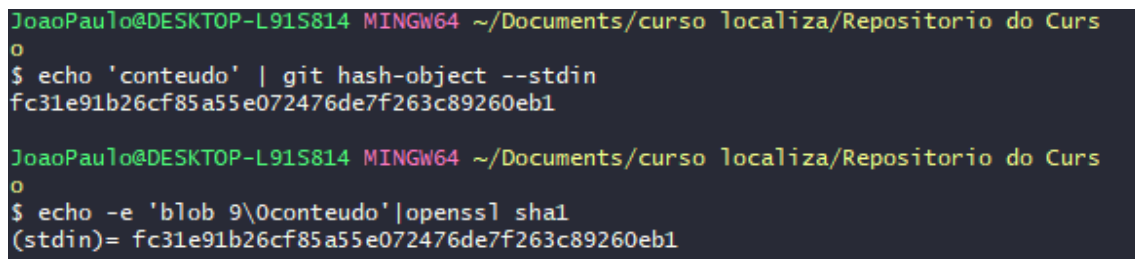
A terminal window with a dark background and light green text. The prompt is 'JoaoPaulo@DESKTOP-L91S814 MINGW64 ~/Documents/curso localiza/Repositorio do Curso'. The first command is '\$ echo 'conteudo' | git hash-object --stdin', which outputs 'fc31e91b26cf85a55e072476de7f263c89260eb1'. The second command is '\$ echo -e 'blob 9\0conteudo'|openssl sha1 (stdin)=', which outputs 'fc31e91b26cf85a55e072476de7f263c89260eb1'.

Figura 1: SHA1 de ‘blob 9\0conteudo’.

A tree também é um objeto interno do Git que possui um SHA1, um tamanho, o caractere \0 e aponta para uma blob, ou seja é também um metadado que guarda o nome do arquivo que essa tree aponta. A Tree é uma árvore e ao apontar para as blobs é montada uma estrutura que mostra como estão organizados os arquivos, da mesma forma que é feito por exemplo na estrutura de diretórios, pastas e arquivos. Uma Tree também pode apontar para outra Tree, ao modificar qualquer arquivo na estrutura recursivamente os SHA1 de todas as Trees na estrutura são modificados.

Um Commit também possui um SHA1 e metadados, Através dele todas as mudanças feitas podem ser rastreadas, já que um Commit aponta para a árvore e para um Commit parente (último Commit feito antes dele) e também aponta para o autor e a mensagem descritiva daquele Commit e por fim também aponta para o Timestamp, que é a data e hora da criação do Commit.

A figura 2 mostra uma Blob, uma Tree, um Commit e um exemplo de como estes três objetos podem ser organizados [1].

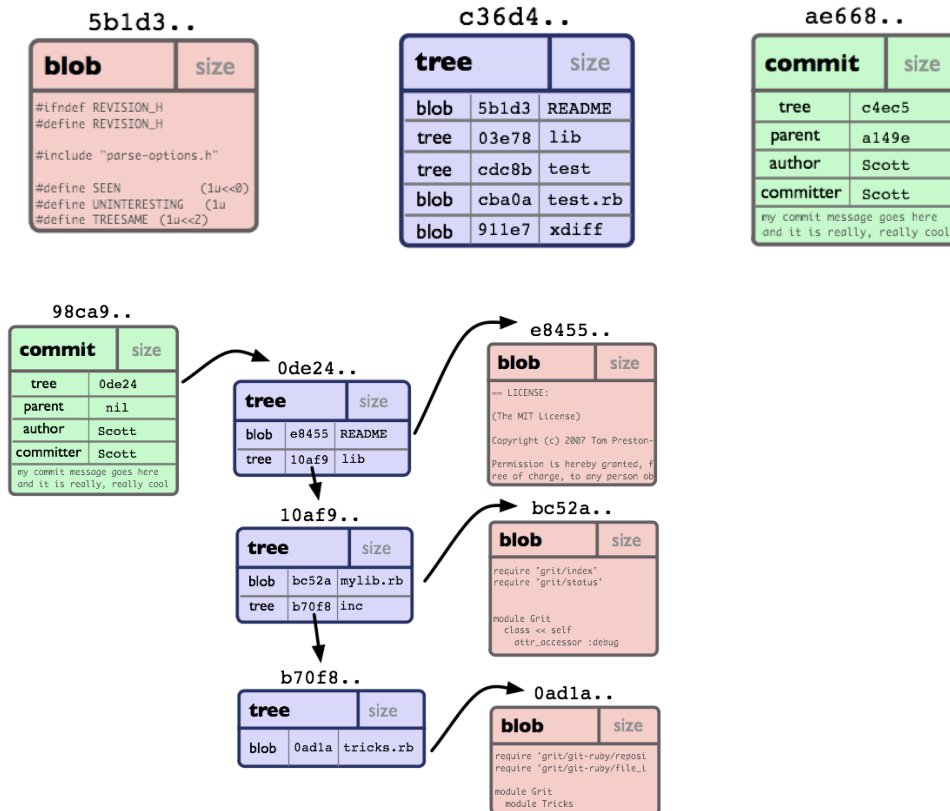


Figura 2: Blob, Tree e Commit.

Ciclo de Vida dos Arquivos no Git

O comando `git init` cria a pasta oculta `.git`, ou seja, é inicializado um repositório dentro de um diretório. As mudanças feitas nos arquivos ou arquivos que foram criados o git ainda não sabe da existência deles, logo a sua situação é dita como sendo `untracked`, ao utilizar o comando `git add *`, todos os arquivos `untracked` vão para a situação `Staged`, ou seja os arquivos estão preparados para poder fazer parte de um agrupamento denominado `commit`, um arquivo que o git têm ciência de sua existência, mas foi modificado também pode ser preparado para ir para a situação de `Staged` através do comando `git add *`, assim que o `commit` é feito é salvo todas as alterações e os arquivos vão para o estado `Unmodified`. Os arquivos podem ser removidos ou editados de forma que o ciclo recomeça. O git consegue saber se algo foi modificado através da comparação do SHA1. A figura 3 exemplifica todos estes conceitos.

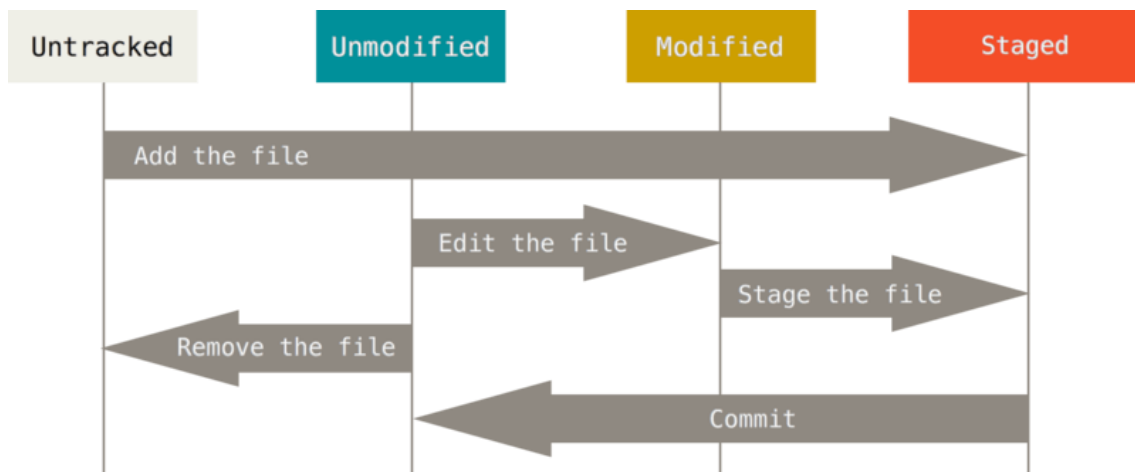


Figura 3: Ciclo de vida dos arquivos no Git [2].

O ambiente de desenvolvimento na máquina local é composta do Working Directory, Staging Area e Local Repository. Este último é composto apenas por commits, e pode ser enviado para um Remote Repository como por exemplo o GitHub de forma que os arquivos são compartilhados na nuvem. Para empurrar os arquivos do repositório local para o repositório remoto o comando `git push origin master` pode ser usado, em que `origin` é um apelido para a URL do repositório criado no GitHub e a `master` é uma branch, em alguns casos podemos ter a branch `main` também. Caso uma outra pessoa tenha feito modificações na mesma linha pode haver um conflito, para resolver o conflito as mudanças que estão no git mas não foram refletidas no repositório local pode ser feita através do comando `git pull origin master`, e por fim os conflitos são resolvidos manualmente, e quando estiver tudo pronto as mudanças podem ser salvas no git novamente através da seguinte sequência de comandos: `git add *`, `git commit -m "adicione a mensagem do commit"`, `git push origin master`.

Para salvar um repositório remoto na máquina local, basta copiar por exemplo o link gerado através do método HTTP ou através da chave SSH e usar o comando `git clone` seguido do link.

Por fim alguns outros comando úteis do Bash são `cd` para mudar de diretório, `ls` para listar os arquivos do diretório, `mkdir` para criar um novo diretório, e `rm -rf` para remover um diretório e todo o seu conteúdo.

Referências

- [1] The Git Object Model. http://shafiul.github.io/gitbook/1_the_git_object_model.html. Acesso em 25/01/2022.
- [2] Ciclo de vida Git. <https://www.dio.me/articles/ciclo-de-vida-git>. Acesso em 25/01/2022.