**Git e Github – por Otávio Reis**

São sistemas independentes porém complementares

**Git** – Criado em 2005, pelo mesmo criador do Linux – é um sistema de versionamento de código distribuído. OpenSource

Ele cria versões do código e permite que pessoas do mundo todo possam alterar até mesmo a mesma linha;

Trabalha CLI – trabalha por linha de comando

|  |  |  |
| --- | --- | --- |
| SistemaOperacional | Windows - kernel | Unix – bash (Linux/mac) |
| NavegaMuda de Pasta | Cd | Cd |
| MostraPasta (lista de diretórios da pasta aberta) | Dir | Ls |
| CriaPasta | Mkdir nomepasta | Mkdir nomepasta |
| DeletarPasta | Del nomearquivo/rmdir nomepasta /s /q | Rm –rf nomepasta |
| LimparTerminal | Cls | Clear ou ctrl+L |
| AutoCompleta | Tab |  |
| CriaArquivo | Echo nome >nome.extensao | Echo nome >nome.extensao |
| Mover Arquivo |  | Mv nmArq ./nmPastaNova/ |
| Voltar uma Pasta | Cd.. | Cd .. |

**SHA1** – Algoritmo de Hash Seguro, pega o arquivo e embaralha ele de uma forma específica, gerando um conjunto de caracteres de 40 dígitos único, encriptografando. E esse processo de gerar uma chave de 40 dígitos, é feito todas as vezes que o arquivo é alterado e a chave só se repete se a alteração for desfeita (retornando ao último estado) e assim identifica os arquivos de forma rápida. Para ver isso na prática, faça:

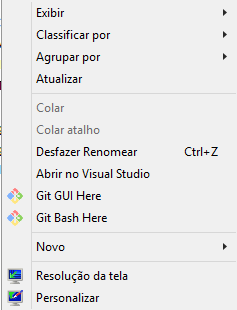
Na pasta de diretório de seus trabalhos, clique com botão direito do mouse e selecione **GIT BASH HERE** e terá acesso ao terminal Git, coloque nesse diretório um arquivo.txt e então digite o comando – openssl sha1 nomearquivo.txt e > enter.

O terminal devolveu a chave $ **b0367708439cf09a1b10ef42c4f2903b8ac84730**, após, abra o arquivo e acrescente um caracter qualquer, salve e repita o comando.

O terminal devolve uma nova chave **4b7ee9f22e8cca0f773b1c4c85c307602e776f87.**

Agora, retire aquele último caracter do arquivo, salve e repita o comando, verá que o terminal devolveu a primeira chave.

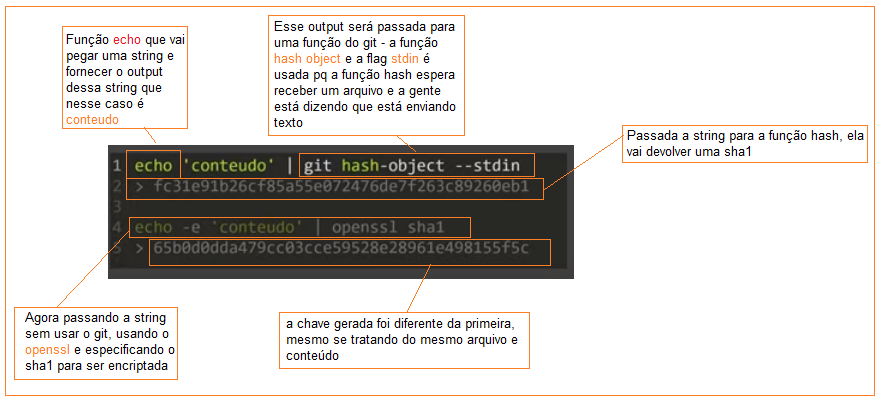
Essa é uma forma inteligente do GIT garantir que o arquivo não sofreu ou sofreu alterações e assim, garantir o conteúdo do arquivo.





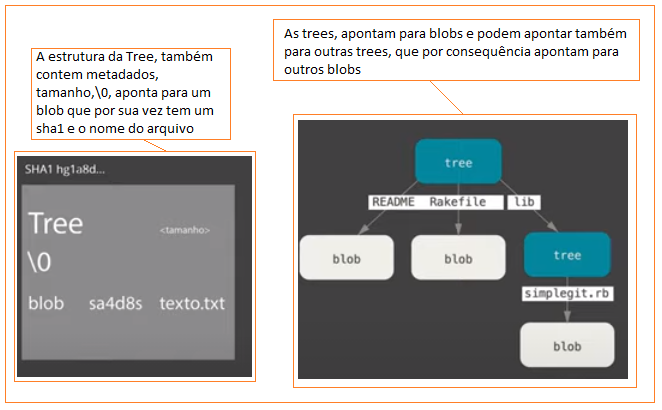
Objetos básicos do Git responsáveis pelo versionamento do código: BLOBS, TREES e COMMITS

**O BLOB**, guarda os arquivos e nele contém metadados, então o objeto Blob tem o tipo do objeto, o tamanho do conteúdo, um \0 e o conteúdo do arquivo. Veja os exemplos:

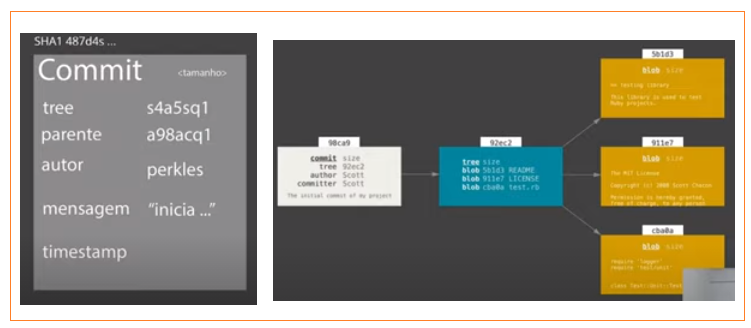




**As Trees**, são responsáveis por montar a estrutura onde estão os arquivos e podem apontar tanto pra blobs outras trees. Isso porque os diretórios dentro de um sistema operacional, podem conter outros diretórios. As trees armazenam blobs, sendo o blob o bloco básico de composição e a tree armazenando e apontando para tipos de blobs diferentes e commites. Logo, se existe um sha1 para o blob e outro para a tree e se houver alteração no arquivo, muda também os sha1 desses objetos.



**O Commit**, é o objeto que vai juntar tudo e dar sentido para essa alteração que está sendo feita. O commit aponta para uma tree, aponta para um parente (ou seja o último commit realizado antes dele), aponta para um autor e aponta para uma mensagem e o autor e a mensagem fazem parte dessa ideia de sentido, tem um timestamp, tem tamanho, possuem sha1 e é reação em cadeia, ou seja, alterado o arquivo, será mudado o sha1 do arquivo, do blob, da tree, do commit, logo, uma vez alterado o arquivo, altera também toda a estrutura daquele commit em questão. Portanto, o commit é único para cada autor.



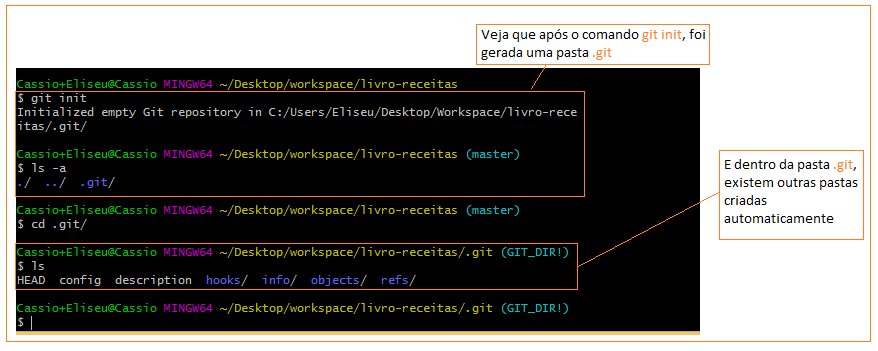
Então, imaginando que você tenha um código num repositório do GIT na nuvem, logo, esse código representa a versão mais recente do seu repositório. E nesse repositório existam 30 pessoas que contribuem nele, que possuem uma versão desse código em suas respectivas máquinas, são todas versões confiáveis por conta de toda essa estrutura do GIT.

Agora, dando inicio aos comandos do git:

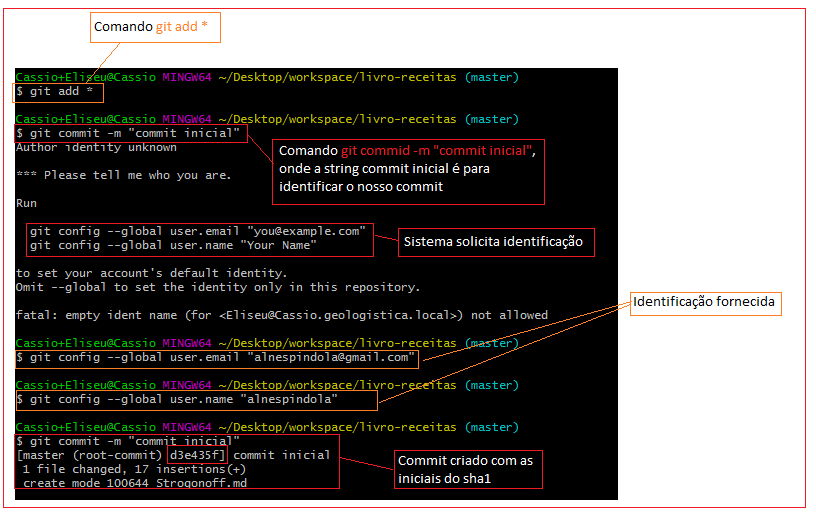
Abra o **git bash here** na pasta de seu diretório e inicie o git com o comando **GIT INIT** para que ele possa gerenciar os versionamentos e iniciar o repositório do GIT;

Em seguida, digite **ls –a** para ter visualização da pasta oculta **.git** criada na inicialização.

Abra a pasta .git com o comando **cd .git/** e veja que foram criadas algumas estruturas dentro dessa pasta



**Criação do commit, utilizando git add, credenciais e git commit**



Vamos entender internamente como funciona:

**Git init** – além de criar a pasta .git com os arquivos iniciais, esse comando inicializa um conceito do git chamado de repositório. Cria então um repositório no diretório.

Logo, dentro de **TRACKED** (arquivos que o git tem ciência deles e que o Git rastreia), pode existir a seguinte subdivisão:

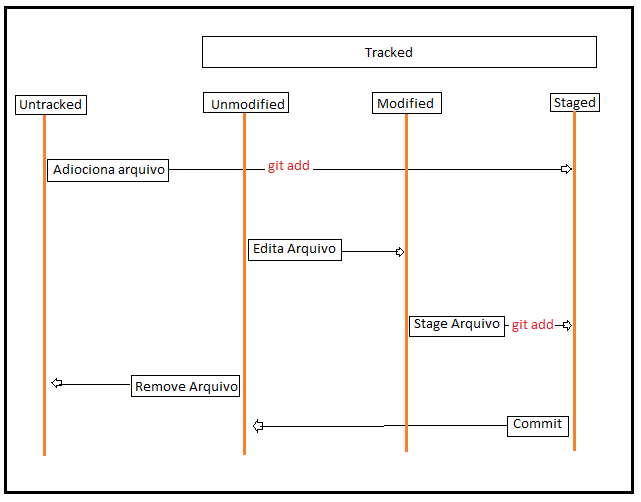
**Untracked** – Arquivos que o git ainda não tem ciência deles. Quando usamos o git add, ele move o arquivo direto para staged.

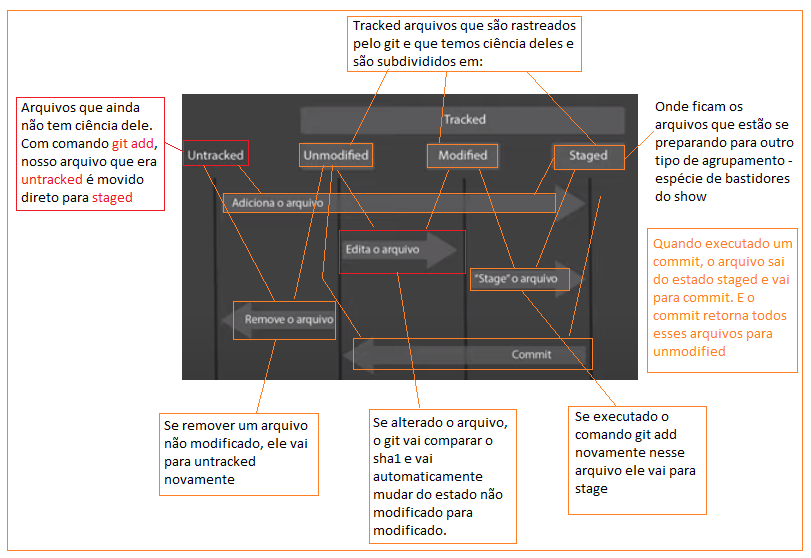
**Unmodiefied** – Arquivo que ainda não foi modificado, se for alterado, o git vai comparar o sha1 dos arquivos e vai mudar direto para modified. Já se esse arquivo ainda sem nenhuma modificação for removido, ele irá direto para untracked.

**Modified** – Arquivo que sofreu modificação. Se rodar o comando git add aqui novamente, esse arquivo vai para staged

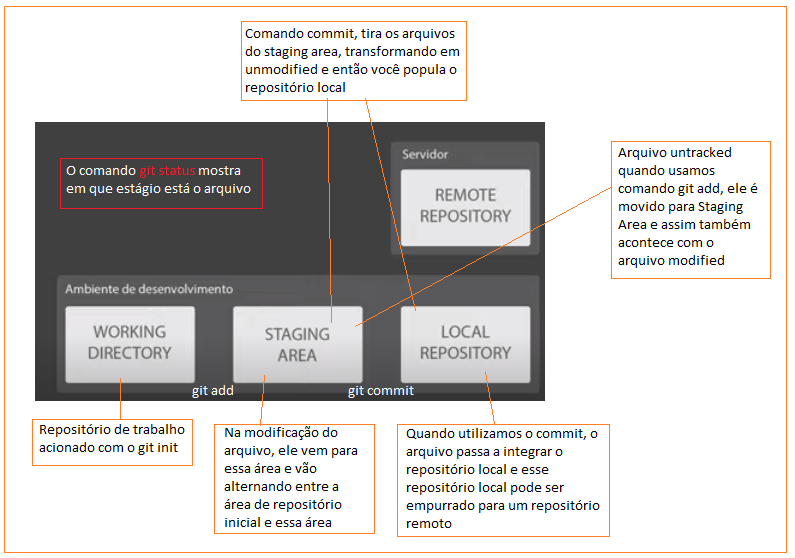
**Staged** – Espécie de palco onde ficam os arquivos que estão se preparando para poder fazer parte de outro agrupamento. Arquivos que aguardam um commit e quando executado o comando **git commit –m “msg texto”**, o arquivo deixa de ser staged e vai para commit (muda de estágio) e o commit, manda os arquivos para unmodified (para começar o ciclo novamente)

Veja a figura:



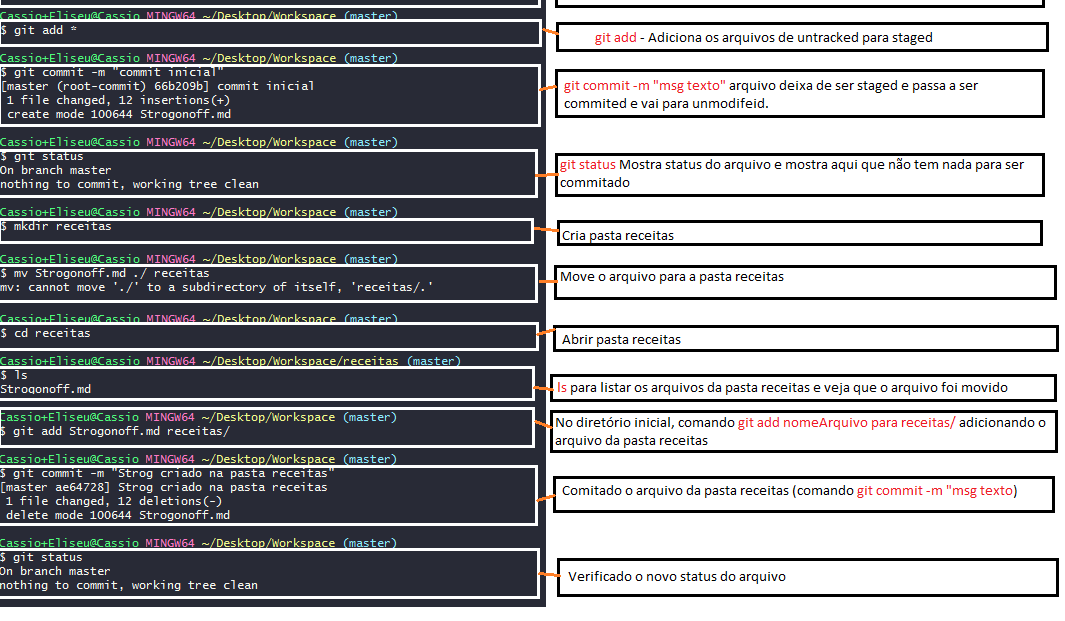


O comando **git status** mostra os estágios do arquivo, e estará satisfatório quando retornar a mensagem **nothing to commit**



Se movido o arquivo ou criado uma pasta após o commit, deve ser repetido o processo (git add e todos os outros) para que o ciclo aconteça.

Amostragem de todos os comandos:



Quando houver nova pasta, execute o comando abaixo:



**GitHub** – Microsoft, armazena seus versionamentos – repositório de arquivo/código.

Após criar conta no site: <https://github.com>

No git, é possível verificar as configurações de permissão com o comando **git config –list (**esse comando mostra tudo o que está também oculto de configuração do git**)** e será mostrado o e-mail e nome de usuário que configurou e para setar essa configuração, use o comando **git config –global –unset user.name ou user.email** conforme o que deseja alterar. Esse comando, deleta as configurações de e-mail e nome feitas anteriormente no git.

git config --global credential.provider generic  (força que as credenciais de login do github volte para o padrão)

O comando abaixo, indica no git, o caminho para o repositório remoto no github

git remote add origin <https://github.com/alnespindola/Teste.git>

Para ver a lista de repositórios cadastrados, use no git o comando: **Git remote –v (**Mostra o caminho do seu push/pull e fetch**)**

Verifique então se não existe nenhuma pendência no repositório com o comando **git status**

**Git show-ref** = Mostra as referencias locais do seu repositorio no caso o importante é estar no refs/heads/main

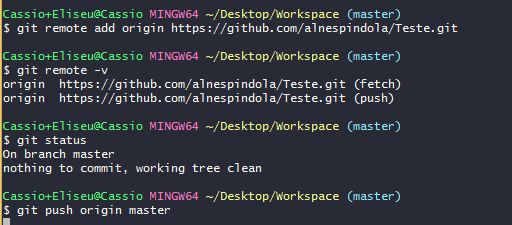
git ls-remote = Mostra as referencias remota do seu repositorio no caso tem que bater com o local ter o refs/heads/main

git branch –l = lista se seu branch esta no main (acho que entendeu que é para dar refs/heads/main acima xd

git branch -m main        **(**Garante que o branch principal do seu repositorio fisico seja o main e não origin(acontece as vezes)**)**

Para enviar o código para o GitHub, use o comando **Git push origin máster or git push –u origin main** , e então será pedido suas credenciais para o GitHub

Se o push ficar eterno, veja o artigo: <https://web.digitalinnovation.one/articles/problemas-com-push-eterno-papai-te-ajuda?back=%2Farticles&page=1&order=oldest>



**Situação Problema**

Quando você colocou o seu arquivo no GitHub, ele ficou disponível para todos e então, vc em sua máquina local e uma outra pessoa em qualquer lugar do mundo, pegou seu código e sem saber, vocês fizeram uma edição na mesma linha do código. A pessoa, finalizou antes de você e disponibilizou a nova versão dela no GitHub. Então, quando você for dar um empurrar o seu código para o GitHub, ele vai “dizer: - olha, já tem uma edição feita nesse arquivo e nessa linha. E agora você vai pegar esse arquivo que tenho aqui e você vai resolver manualmente esse conflito e me dizer quem deverá ser mantido como arquivo mais atual. O seu ou da tal pessoa”. Essa situação, é chamada de conflito de merge.

Então, feita uma alteração no arquivo, você vai add, depois commitar e o git vai devolver uma mensagem de erro, alegando que existe outro arquivo mais atual que o seu.

Então, é hora de puxar o arquivo remoto mais recente para a maquina local **git pull origin master** e esse comando vai baixar o repositório remoto e tentar integrar com seu arquivo local. Entao, vc vai abrir o arquivo que puxou e vai decidir qual alteração deve permanecer. Feito isso, volte no git e verifique o status, o git vai mostrar que teve alterações, então é só seguir o trâmite com git add e git commit e para finalizar, envie para o github usando git push origin máster.

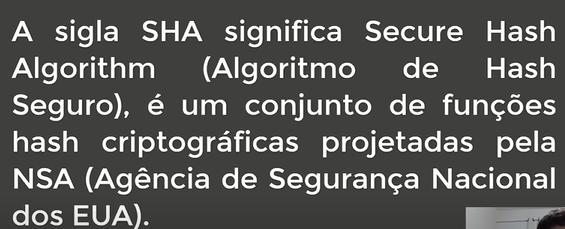
**Baixar repositórios do GitHub**

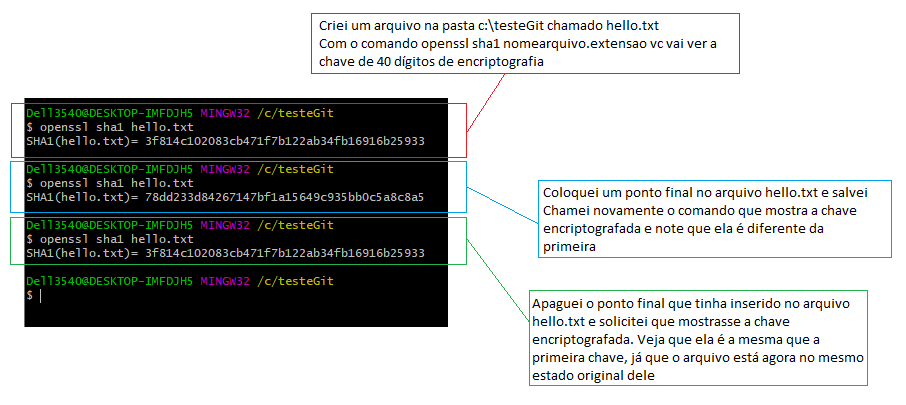
Selecione o link do repositório desejado e o comando git clone nomedaURL

**Git e Git Hub**

Imagine que vc precisa criar um arquivo sobre um determinado assunto, então vc cria um arquivo texto, chamado rascunho e envia para seu editor e ele te sugere uma série de alterações, logo vc vai nomear o rascunho como arquivo inicial como rascunho.old e o arquivo com as sugestões como rascunhodoeditor, e esse arquivo do editor, vai para o editor chefe que faz novas alterações e assim sucessivamente... Logo o Git e git hub tem a ideia de versionamento controlado e compartilhamento do código. Onde pessoas do mundo podem trabalhar num mesmo projeto, e ele garante guardar as versões de cada alteração realizada

O GIT trabalha com CLI – Command Line Interface (importante aprender comandos)

Então o git vai pegar seu arquivo, seja um doc, uma foto e etc e embaralhar muito bem. A saída dessa encriptação gera um conjunto de caracteres de 40 dígitos únicos. Logo, no arquivo.old do nosso exemplo, recebeu um conjunto de 40 dígitos e quando geramos o rascunhodoeditor mesmo que a partir do rascunho.old, um outro conjunto de caracteres de 40 dígitos será gerado. Imagina que vc criou um arquivo e salvou com nome rascunho, então foi gerada uma chave de 40 dígitos única. Se vc abrir esse mesmo arquivo e inserir uma vírgula e salvar, uma nova chave de 40 digitos será gerada. E se vc remover a vírgula e salvar, ele vai novamente te dar a chave que gerou a primeira vez antes da inserção da vírgula.

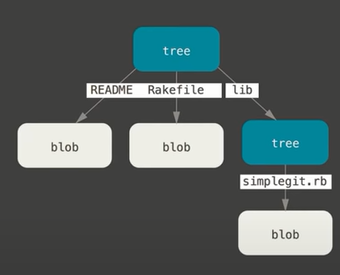


E através dessa chave, vc controla todas as alterações do arquivo.

O git inicialmente, gera um objeto chamado blob – que guarda metadados com o tamanho, tipo e tem a estrutura blob 9\0 conteúdo, onde 9 é o tamanho e conteúdo é o que tem no arquivo.



Em seguida, existem as trees, árvores que armazenam blobs. Essa tree tbm possui metadatos que aponta para determinado blob, o sha1, tem tamanho e nome do arquivo. As árvores monta a estrutura e mostra onde está o arquivo e podem ser apontada para blobs ou outras trees, assim como um diretório pode estar apontando para outro diretório.

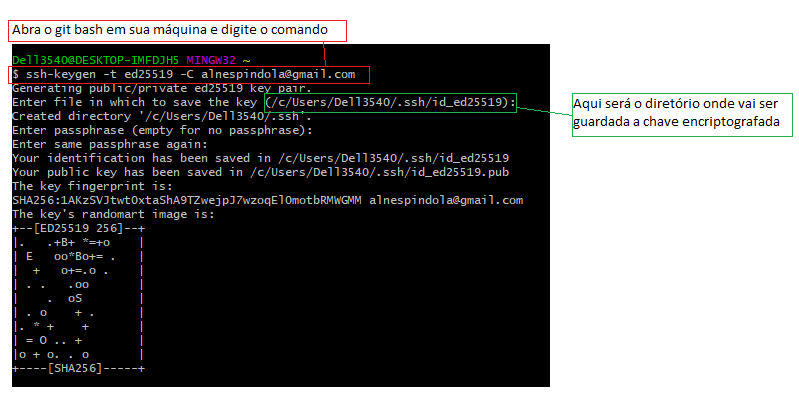


E por último, existe o objeto commit que vai juntar tudo e dar sentido ao que está sendo feito. O commit aponta para uma tree, aponta para um parente (um commit realizado antes desse), aponta para um autor e uma mensagem e isso significa uma alteração. Carrega também data/hora da criação e também possui uma encriptação. Logo se alterado um dado dentro de uma blob, vai gerar um sha1 desse blob que altera por sua vez a tree e que altera o commit. Processo em cadeia – montando uma linha do tempo em que as alterações foram realizadas.

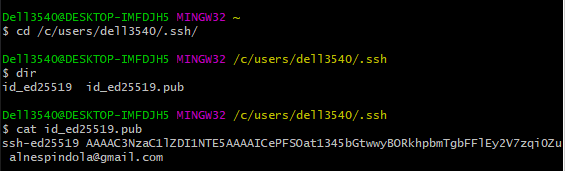
Metaforicamente, os comitês são como estruturas de pastas, onde o commit é o diretório principal, as arvores são subpastas e os blobs são os arquivos.

O GIT é um sistema distribuído, onde o código que está na nuvem, é a versão mais recente que foi realizada. E se N pessoas alteram esse código, cada uma dessas alterações terá um sha1 único e isso garante um versionamento confiável.

O Git pede autenticação através da chave ssh – tornando a sua máquina um conexão confiável para o github – terá uma chave pública e uma privada e assim o github vai reconhecer a sua máquina e autenticar – faça isso usando o git bash

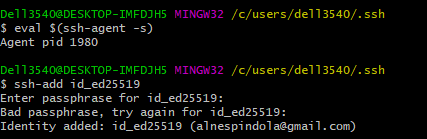


Se navegar no diretório .ssh, verá que foram criados dois arquivos

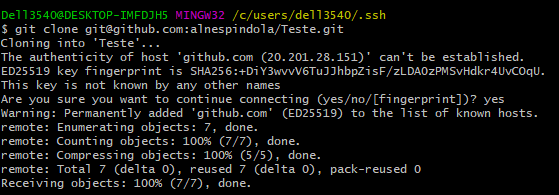


E vamos visualizar a chave pública e divulgar no github

Agora vamos inicializar o ssh-agent que é encarregado de validar essas chaves:



Gerada a chave SSH, já não é possível clonar o diretório através da url, terá que usar a chave fornecida pelo ssh, então no git hub, dentro do diretório, clique em código e ssh e copie a linha fornecida e execute o comando:





E então o diretório já foi clonado.

Pode ser feito através de token também, porém deve ser uma máquina de confiança pq o token vai ficar na sua máquina, podendo gerar uma data de inspiração para o token

Instruções para gerar o token no final do vídeo desse link [DIO | Codifique o seu futuro global agora](https://web.dio.me/course/introducao-ao-git-e-ao-github/learning/7410b862-1989-421a-a48d-500db5857f53?back=/track/inter-frontend-developer)

**Desafio de Projeto Git/Github**

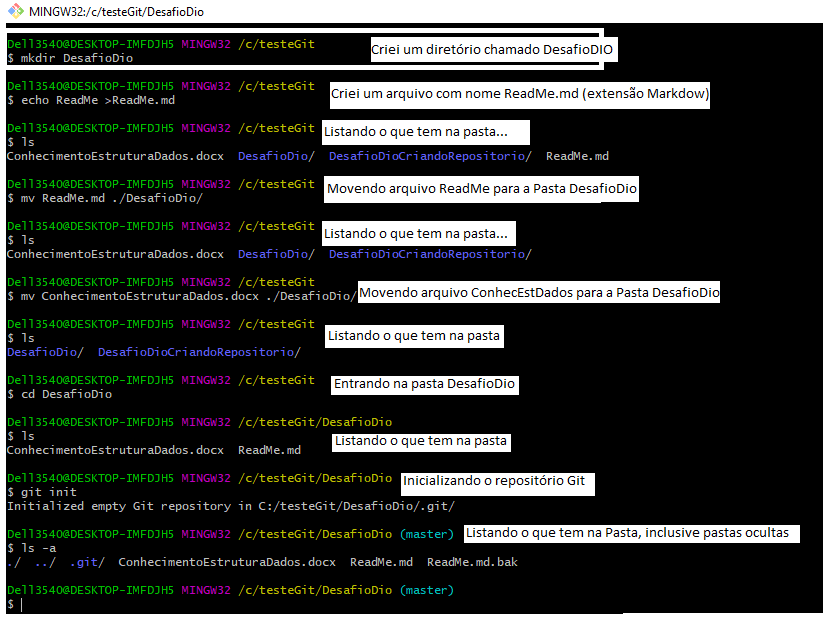
Para criar um novo repositório, vá até sua conta do git e crie new, dê um nome ao seu repositório, em seguida uma descrição, coloque ele como público e Add readme file (esse é uma espécie de considerações iniciais da página, a homepage do seu repositório e nele vc pode colocar informações que julgar importantes, links e etc. Clique create.

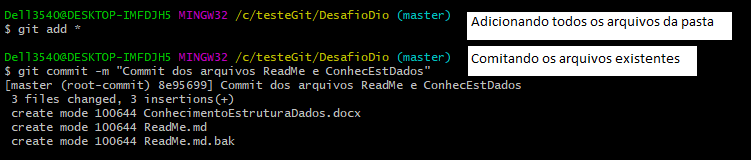
Na sua pasta de projeto, botão direito do mouse, git bash here, digite o comando **git clone** e cole o link que copiou na página do git (em code) – feito isso, vc pode criar seus arquivos e acrescentar na pasta de projeto e para subir esses arquivos para o git, inicialmente **digite git add. Nomearquivo** ou **git add -A nomearquivo**, E em seguida git commit -m “frasedesejada”, agora digite o comando git push origin main ( e esse comando leva os arquivos para seu repositório em núvem)

Para apagar diretório, em configurações, repositório, delete.

**Agora vamos a criação de todo um processo do zero:**

No browser, já logado no github, crie um repositório

Abra o Git Bash na sua máquina e digite os comandos:

Se nesse momento você não estiver usando a autenticação da chave SSH (leia página 11 e 12 desse arquivo), o sistema vai pedir suas credenciais (leia quadro na página 5). Autenticação feita, copie a url do repositório desejado:

