**MANUAL DO INICIANTE 4.0-GIT/GITHUB**

**Comandos do banco de dados**

**Comandos citados**

**Nomenclatura criada para teste**

**PRIMEIRO ACESSO**

Primeiro comando ao acessar o GIT é usado para definir o nome do usuário entre as aspas duplas

**GIT CONFIG - -GLOBLA USER.NAME “JD”**

Segundo comando ao acessar o GIT é usado para definir o E-mail entre as aspas duplas

**GIT CONFIG - -GLOBLA USER.EMAIL “JD@EXAMPLE.COM”**

Validando informações das configurações pode ser usado o comando **GIT CONFIG** porém para que tenha dados especifico deve informar o parâmetro em seguida

**GIT CONFIN USER.NAME**

**CRIANDO E ACESSANDO DIRETORIO**

Diretório é onde será adicionado os arquivos com o controle de versão, diretório é uma pasta onde é definida que é um diretório.

**PASTA**

Para acessar uma pasta é usado comando **CD** seguido do nome da pasta o caminho, para o conteúdo da pasta usar o comando **DIR** para volta para pasta anterior usar comando **CD ..**, para criar pasta fazer uso do comando **MKDIR** seguido do nome desejado.

**MKDIR TABELAS**

Para definir um repositório é usado dentro da pasta desejada o comando **INIT** será criado uma pasta oculta dentro das mesmas com nomenclatura (. GIT) e ao acessar a pasta que foi criado um repositório será sinalizado no GIT como MASTER.

**GIT INIT**

**ACESSANDO ARQUIVOS**

Para acessar no GIT pelo VIM utiliza comando VIM nome desejado e sua extensão caso o arquivo não exista será criado no local que foi executado será aberto a tela do vim no GIT onde usa comandos **I** para inserir dados **ESC** para sai do modo INSERT digita “**:** ” mais **W** para salvar alterações no modo dois pontos **Q** para sai da edição podendo usar **WQ** junto.

**Fases do controle de versão**

Comando executados dentro do diretório.

Commit é salvar o que foi feito no repositório.

O comando **GIT STATUS** serve para verificar como está seus arquivos quando sofre uma alteração ou está na área para commitar dentro do seu diretório.

**MODIFY**

O arquivo modificado passa pela primeira área quando é alterado onde usando o comando **GIT STATUS** o arquivo modificador fica em cor vermelho, para verificar o que foi alterado utiliza comando **GIT DIFF** onde ele mostra em vermelho o comando antes de alterar e em verde o comando novo caso deseje reverter o que foi feito nessa etapa utiliza comando **GIT CHECKOUT** em seguida do nome do arquivo e sua extensão (para reverter apenas aquele arquivo).

**STAGED**

Quando o arquivo é modificado e cai na primeira área tem a opção de manda para segunda área(STAGER) antes do commit, utilizando o comando **GIT ADD** com o nome do arquivo e sua extensão (para passa o arquivos especifico para staged) caso deseje voltar para primeira área utilizar comando **GIT RESET HEAD** e nome do arquivo com extensão (para especificar o arquivo que deseja voltar) ou ao invés do nome do arquivo utilizar comando **FULL** para trazer todos que estão nas segunda etapa de volta para primeira, na segunda etapa (STAGED) para verificar o que foi modificado utiliza do comando **GIT DIFF --STAGED** seguindo a mesma lógica do **GIT DIFF.**

**COMMIT**

**GIT COMMIT** seguido do nome do arquivo com extensão (para especificar qual arquivo fazer commit) serve para salvar as alterações em versões, o commit pode ser feito a partir da primeira etapa, para adicionar comentário no commit faz uso do **–M** e a mensagem entre aspas duplas

**GIT COMMIT FILE.SQL -M “V1.1”**

Caso deseje altera o último commit utilizar o comando **- - AMEND** no commit podendo alterar até mesmo a mensagem gravada

**GIT COMMIT - -AMEND –M “NOVA MENSAGEM”.**

**LOG**

para verificar o que foi salvo utiliza comando **GIT LOG** onde vai trazer detalhes dos commit como comentários, autor, chave e etc... caso deseje vê apenas as chaves e o comentário utiliza comando **GIT LOG - -PRETTY:ONELINE** para filtra apenas por commit de um autor utilizar comando **GIT LOG AUTHOR=”NAME”** o **GIT LOG –P** busca todos commit e detalhe do que foi alterado em ordem cronológica e decrescente podendo também definir quantos commit em você deseja buscar definido depois de traço a quantidade **GIT LOG –P -2**, o **GIT SHORTLOG** agrupa o log por autor ordenando em ordem alfabética e trazendo apenas informações do comentário em ordem crescente dentro da organização de cada autor caso deseje apenas a quantidade de commit por autor utilizar comando - -SN **GIT SHORTLOG - -SN**.

**DELETE MR**

Para deletar um arquivo utiliza comando **GIT RM** em seguida do nome do arquivo e sua extensão (para especificar o arquivo que deseja deletar), quando deletado o arquivo fica na segunda área o (STAGED) para que seja feito o commit do delete podendo fazer o commit ou voltar para primeira área com o comando **GIT RESTORE - -STAGED** voltando para primeira área pode ser feito o restore com o comando **GIT RESTORE**.

Deletando commit

Git reset –soft –mixed --hard

**TAG**

As TAGS servem como se fosse uma etiqueta que é colocada nos commit para se referência para criar uma TAG utiliza o comando **GIT TAG** junto com o nome da TAG desejada caso queira adicionar a informações de data hora e autor entre outras informações utiliza a **–A** antes do nome da TAG podendo também ser adicionado comentário após a criação da TAG com o comando **–M**.

**GIT TAG –A V1.0 –M “TESTE DE VERSÃO”**

Para criar uma TAG me um log anterior basta usar mesmo comando com a chave do log após o nome da TAG

**GIT TAG –A V1.0** 4feddd6a5919fc770ab82ef3a612ac4d7b002019 **–M “TESTE DE VERSÃO”**

Para verificar as TAGS disponíveis naquele diretório basta utilizar comando **GIT TAG** caso precise de detalhes sobre a TAG utiliza o comando **GIT SHOW** seguido do nome da TAG onde vai trazer todas informações da TAG e do log o qual está vinculado.

Para voltar um script para versão anterior através da TAG utiliza comando

**GIT CHECKOUT NOME\_DA\_TAG**

Dessa forma o script volta a ser o mesmo do commit que a TAG está vinculada para voltar para última TAG criada só utilizar **GIT CHECKOUT MASTER**.

Para deletar uma TAG utiliza comando **–D** seguido do nome da TAG

**GIT TAG –D V1.0**

**BRANCH**

BRANCH são ramificações da linha de compilação para que seja feito alterações sem mexer com o código existente ele funciona paralelo a linha de compilação original.

Para criar um BRANCH utiliza o comando GIT BRANCH e o nome desejado logo após utiliza o comando GIT CHECKOUT e nome do BRACH assim ele vai está acessando uma cópia de quando o BRANCH foi criado sem alterar o código da linha MASTER.

**GIT BRANCH TESTE**

**GIT CHECKOUT TESTE**

Para trazer as alterações que fora feito commit no BRANCH para linha principal MASTER utilizar comando MENGE assim as alterações que foram feitas serão copiadas para o código MASTER.

**GIT MENGE TESTE**

Para deletar um BRANCH utiliza o comando **–D** seguido do nome do **BRACH**

**GIT BRACH –D TESTE**

rebase

**ACESSANDO DIRETÓRIO REMOTO**

Para criar um repositório compartilhado no servidor local para que seja acesso por outras pessoas na rede utiliza o comando **GIT INIT - -BARE** será criado várias pastas dentro do repositório ao invés de apenas a pasta (. GIT) e ao acessar a pasta que foi criado um repositório será sinalizado no GIT como BARE: MASTER.

A pasta do repositório deve estar compartilhada.

Para acessar um repositório remoto em uma máquina cliente utiliza o comando

**GIT CLONE FILE://// HOST\_NAME ou IP/NOME\_DA\_PASTA\_COMPARTILHADA/NOME\_DO\_DIRETORIO NOME DIRETORIO DESEJADO**

Este comando vai clonar tudo que existir no diretório que foi direcionado para máquina cliente com a nomenclatura do diretório de acordo com que foi descrito no final do código.

Por padrão a conexão remota entre as máquinas recebe o nome de ORIGEN.

Para atualizar os que foi feito em sua máquina para o servidor faz uso do comando **PUSH** definindo a conexão (ORIGEN) e o BRANCH que está saindo as alterações.

**GIT PUSH ORIGEN MASTER**

Para atualizar seu diretório com as informações existente no servidor faz uso do comando **PULL** definindo o nome da conexão e do BRANCH que vai ser atualizado assim o diretório local recebera as atualizações feita no servidor.

**GIT PULL ORIGEN MASTER**

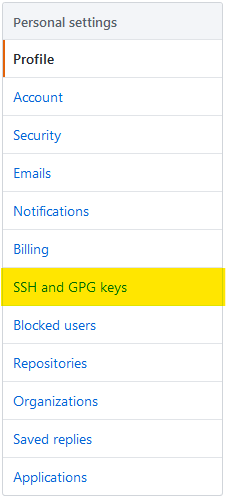
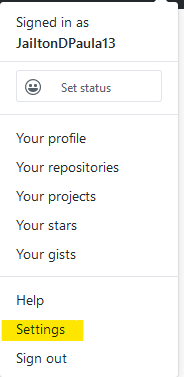
Git fetch origin branch moneclatura

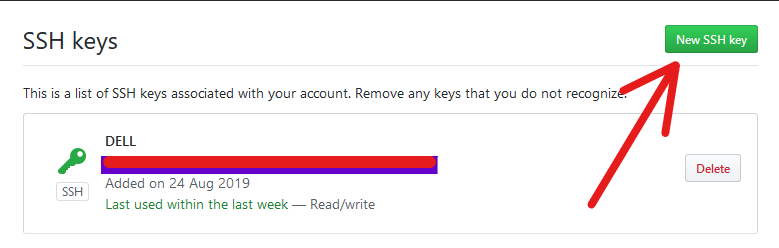
**GITHUB**

GITHUB é um repositório remoto para conectar a ele é necessário criar uma chave de acesso dentro de sua máquina utilizando o comando

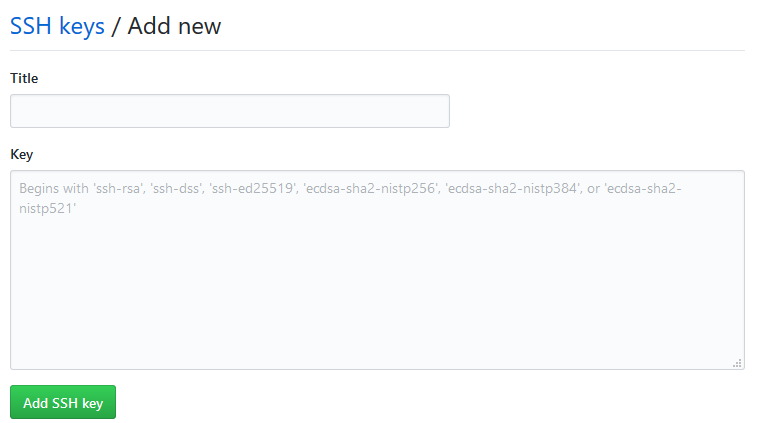
SSH-KEYGEN -T RSA -B 4096 -C “[YOUR\_EMAIL@EXAMPLE.COM](mailto:your_email@example.com)”

Definindo E-mail de acordo com que foi configura no GIT será criado uma pasta dentro do usuário com nomenclatura de (. SHH) onde a chave será criada com nomenclatura de (ID\_RAS.PUB) onde dentro dela vai conter a chave que irá conectar com GITHUB.



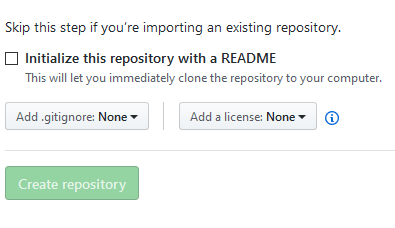


Definindo um titulo e em KEY adicionando uma chave ele irá ter uma conexão com seu GIT local

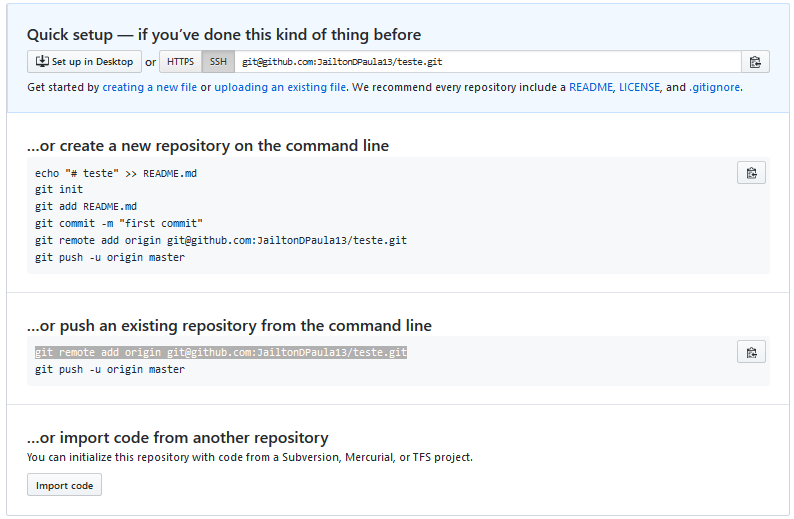


Existir duas maneira de conectar com um repositório no GITHUB uma delas é criar um repositório na maquina e transferindo pro GIT ou outra é criar um repositório no GITHUB ou pegar um já existente.

Para mandar um repositorio do GIT pro GITHUB primeiro tem que criar um repositorio dentro do GITHUB onde ira receber o repositório do GIT, ao criar repositorio deixa a opção (**INITIALIZE THIS REPOSITORY WITH A README**) desmarcada



Será criado um repositório vazio com instruções para conectar



Para importar do repositório do GIT para o GITHUB é preciso ultilizar o comando **GIT REMOTE** segiudo do nome que deseja para conexão e o link do diretótio.

**GIT REMOTE ADD TESTE GIT@GITHUB.COM:JAILTONDPAULA13/TESTE.GIT**

Para conectar um repositório do GITHUB para GIT utiliza o comando **GIT CLONE** seguido do link de conexão e nome que deseja para pasta no diretório do GIT.

**GIT CLONE** [**GIT@GITHUB.COM:JAILTONDPAULA13/TESTE.GIT**](mailto:GIT@GITHUB.COM:JAILTONDPAULA13/TESTE.GIT) **PROJETOS**

Para fazer DOWNLOAD e UPLOAD do diretório no GITHUB funciona da mesa forma que acesso remoto utilizado comando **PUSH** para upload ou **PULL** para download seguido do nome da conexão com o nome do **BRANCH.**

**GIT .IGNORE**