Comandos do GIT

GIT BÁSICO

$ git init (inicia um repositório em uma pasta)

$ git status (mostra o status de um repositório em uma pasta)

$ touch .gitignore (cria um arquivo gitignore para o git ignorar pasta e arquivos)

CONFIGURATIONS

$ git config user.name "Bruno Iglesias" (configura o nome de usuario para o repositório)

$ git config user.email "bruno.iglesias.eng@gmail.com" (configura o email de usuario)

$ git config --global user.name "Bruno Iglesias" (configura o nome de usuario para todos os repositórios)

$ git config --global user.email "bruno.iglesias.eng@gmail.com" (configura o email de usuario para todos os repositórios)

$ git config core.pager cat (configura para mostrar todo o histórico do repositório local sem travar o terminal)

$ git config core.pager less (configura para mostrar todo o histórico do repositório local travando o terminal)

$ git config --global core.pager cat (configura para mostrar todo o histórico sem travar o terminal)

$ git config credential.helper store (configura para armazenar o usuario e senha do github para não pedir toda vez que for realizar um push)

$ git config --unset credential.helper (remove o usuario e senha armazenado com o comando anterior)

$ git add .gitignore (adiciona um arquivo para controle de versão, mudando o estado de untracked para tracked)

$ git add . (adiciona todos os arquivos para controle de versão, mudando o estado de untracked para tracked)

$ git commit -m "Meu primeiro commit" (muda o estado do arquivo para staged, cria uma 'fotografia' do seu projeto)

GIT LOG

$ git log (mostra todo o histórico do repositório - lista os commits)

$ git log -2 (mostra apenas o histórico dos dois ultimos commits)

$ git log --oneline (mostra o histórico com apenas o inicio do código e mensagens)

$ git log --oneline -2 (mostra o histórico com apenas o inicio do código e mensagens dos dois ultimos commits)

$ git log --before="2023-06-01" (mostra o histórico dos commits antes da data especificada)

$ git log --after="2023-06-01" (mostra o histórico dos commits após a data especificada)

$ git log --author="Bruno Iglesias" (mostra o histórico dos commits de um autor)

$ git checkout [id_commit] (navega no projeto para um ponto anterior do tempo)

$ git checkout master (navega no projeto para o ponto mais atual se antes foi usado o comando anterior)

$ git mv nome_antigo novo_nome (renomeia um arquivo do repositório - melhor prática)

$ git remove nome_arquivo (exclui um arquivo do repositório - melhor prática)

$ git diff --staged (compara o arquivo em staged com o do último commit realizado)

$ git diff [id_commit] (compara as diferenças entre o commit especificado no id e o estado atual do arquivo)

$ git diff [id_commit_antigo]..[id_commit_novo] (compara as diferenças entre um commit até outro commit)

$ git commit --amend -m "adicionando uma nova mensagem" (corrige a mensagem de um commit anterior, se houver um novo arquivo em staged, o comando pode ser utilizando para incluir o novo arquivo no commit anterior).

$ git restore --staged [nome_arquivo] (retira o arquivo de staged - não monitorado)

$ git checkout [nome_arquivo] (restaura o arquivo para o estado do ultimo commit)

$ git reset HEAD --hard (restaura todos os arquivos para o estado do ultimo commit)

$ git reset HEAD^ --hard (descarta o último commit)

GIT INTERMEDIÁRIO

$ git branch (consulta as branchs do projeto)

$ git branch [nome_branch] (cria uma branch no projeto)

$ git checkout -b [nome_branch] (cria e muda para outra branch do projeto)

$ git checkout [nome_branch] (muda para outra branch do projeto)

$ git branch -D [nome_branch] (deleta uma branch no projeto)

$ git merge [nome_branch] (faz o merge da branch especificada com a branch atual adicionando os commits ao final da branch principal)

$ git rebase [nome_branch] (faz o merge da branch especificada com a branch atual mas encaixando os comits da outra branch)

$ git clone [repositório] (clona uma branch)

$ git fetch [repositório] (clona uma repositorio sem fazer o merge, após baixar os arquivos é comum usar o git rebase para adicionar os arquivos na branch)

$ git pull [repositório] (clona uma repositorio e depois faz o rebase, pode ser usado para atualizar seu repositorio local com as modificações contidas no repositorio centralizador)

$ git init --bare (inicia um repositório bare "centralizador" em uma pasta)

$ git push (envia seus commits feitos em um repositorio clonado para o repositório central)

$ git tag v1.0 (cria uma marcação no estado da aplicação, pode ser usado para marcar um release ou uma versão)

$ git push origin v1.0 (enviar a tag feita em um repositorio clonado para o repositório central)

$ git checkout v1.0 (muda a aplicação para o estado da tag, não é possivel realizar commits)

$ git switch -c correcoes-v1.0 (cria uma branch a partir de uma tag para fazer novos commits)

Trabalhando com GitHub

$ git remote add origin [endereço].git (registra um repositório bare remoto)

$ git push -u origin master (envia os commits do repositório local para um repositório bare remoto)

$ git remote -v (mostra se há algum repositório remoto registrado no repositorio local)

$ git clone [repositório].git (clona um repositório bare remoto)

$ git pull origin master (atualiza um repositório para obter o estado do bare repositorio)

$ git checkout -- [nome_arquivo] (volta ao estado anterior do arquivo geralmente é usando o arquivo é modificado mas ainda não está tracked -- staged)

$ git checkout HEAD -- [nome_arquivo] (volta ao estado anterior do arquivo geralmente é usando o arquivo é modificado mas já está tracked -- staged)

$ git revert [id_commit] (cria um novo commit desfazendo as modificações do último especificado)

$ git reset HEAD~1 (remove o último commit realizado sem apagar os arquivos modificados)

$ git remote set-url origin https://iglesias@github.com/[repositório](configura o repositorio local para autenticar com determinado usuario)

$ git fetch origin pull/3/head:correcao_i2 (baixa um pull request de id 3 e cria uma branch chamada correcao_i2)