**COMANDOS DO GIT**

**CHAVE SSH**

**ssh-keygen -t rsa -b 4096 -C** [seu-email@seu-servidor.com](mailto:seu-email@seu-servidor.com)

Digitar a senha 2 vezes, se quiser coloca-la

**ls -al ~/.ssh =** verfica a existência de uma chave ssh

**cd ~/.ssh/ =** mostra a localização da chave ssh. Digitar um ls para mostra o id.rsa.pub

**clip < ~/.ssh/id\_rsa.pub ou cat id\_rsa.pub =** copia a chave para adicior no github

**ssh -T git@github.com** – verifica se está logado no github

**LISTAGEM**

**ll ou ls -la =** lista os arquivos do diretório

**git init –bare =** inicia um novo repositório git de forma local

**LOG**

**git log --name-status =** mostra o log dos commits, juntamente com o status dos arquivos que foram modificados, adicionados ou deletados

**git log --pretty=oneline =** mostra os commits em linha

**git log --abrev-commit =** mostra os commits abreviados

**git log --pretty=oneline --abbrev-commit =** mostra o commit abreviado com uma linha

**git log --stat** = mostra alguns dados estatísticos

**git log -p =** mostra as alterações de cada commit e por quem foi alterado

**git log -p -3=** mostra as alterações dos 3 últimos commits

**git log --pretty=format:"%h - %an, %ar : %s" =** mostra a hash abreviada , o autor-name, o tempo ou data e o título do commit

**DIFF - SHOW**

**git diff --staged =** mostra o que foi modificado no nível de staged

**git diff <numero commit> <numero commit> =** compara 2 commits que podem ser com números abreviados

**git diff -w =** mostra as alterações apenas de texto e outros sem os espaços

**git show <numero\_do\_commit> =** mostra um diff com as modificações, após ser commitado.

**COMMIT**

**git commit -am “mensagem” =** adiciona o arquivo no staged e já faz o commit

**git commit -m "mensagem" --amend =** modifica o texto do último commit e também substitui o mesmo. Tomar cuidado de não ter feito **push** antes, page evitar conflitos

**git commit -m "mensagem** - **fix #número-da-issue** " = Resolve problemas de ima issue já comitando.

**PUSH**

**git push origin <branch> --delete** ou **git push origin :branch =** faz o push deletando a branch do github

**MERGE**

**git merge --abort =** o git tenta abortar um merge que entra em conflito

**git mergetool =** abre a ferramenta configurada pra fazer o merge

**REMOÇÃO**

**git rm arquivo =** deleta o arquivo na staged area

**git rm -rf node\_modules/ --cached =** remove o node\_modules da staged area quando adicionado acidentalmente, sem deletar o diretório real

**rm node\_modules -rf =** remove a node-modules do working-directory, juntamente com o seu diretório

**CONFIG**

**git config --list =** mostra configurações globais e padrões do git

**git config -l =** mostra as configurações principais do git

**git config --global user.name ou user.email =** confira o nome e email do usuário no git

**git config --global core.editor =** mostra o editor configurado para fazer commits

**git config --global core.editor "code --wait" =** abre o VS code para edição, ao fazer um git commit

**git config --global -e** = modifica manualmente as configurações do git

**git config --global merge.tool meld =** muda o editor de conflitos para o meld

**git config --global difftool.meld.path "/c/Program Files (x86)/meld/meld.exe" =** faz o diff no meld

**git config --global credential.helper cache =** configura a senha para que as credenciais de senha sejam gravadas sempre que fizer push utilizando HTTPS

**git config --global --unset <configuração-global> =** remove qualquer configuração global passada após o unset, ex: user.name, core.editor e outros.

**git config --global --unset credential.helper =** remove a configuração do credential para voltar a digitar a senha

**git config --global credential.helper 'cache --timeout=3600' =**  configura o credential para remover a senha salva a cada 3600s = 1h

**git config --global alias.<letra> <comando> =** Abrevia um comando no git. **Ex: git config --global alias.s status** – bastaria digitar git s no bach.

**MERGETOOL**

**git config --global mergetool.meld.path "/c/Program Files (x86)/meld/meld.exe" =** configura o meld no Windows

**echo "\*.orig" >> .gitignore =** acrescenta no gitignore, os arquivos gerados automaticamente pelo meld

**git mergetool** = abre o editor de merge configurado.

**BRANCH**

**git branch -m <nome-da-branch>** = renomeia a branch

**git checkout --orphan <nome-da-branch>** = Cria uma branch nova sem pegar dados do histórico de outras branchs, ou seja, vazia.

**MERGE**

**git rm -rf =** limpa todos os arquivos da arvore do git. Vantajoso para trabalhar numa branch vazia, onde precisa começar os arquivos do zero

**REPOSITÓRIO - REMOTE - CLONE**

**git init --bare =** cria um repositório remoto local (principal)

**pwd =** retorna o caminho completo de um diretório no computador

**git remote add <origin> <url do repository> =** adiciona um repositório remoto no projeto

**git remove -v =** mostra o repositório de origem (fetch) e a origem (push)

**git clone <caminho do repository> =** clona um repositório existente

**git remote rm <origin> =** remove um repositório remoto

**git remote set-url <origin> <url> =** altera a url remota de um repositório

**git remote add <repositor> <url do repository > =** utilizado para trabalhar em equipe em um segundo repositório, cujo seria o principal. Exemplo: **git remote add upstream git@github.com:lefialho/git-ninja.git**

**CHECKOUT – RESET- REVERT**

**git checkout .gitignore OU <outro arquivo> =** retira o mesmo da working directory, voltando as alterações ainda não comitadas

**git checkout <numero-do-commit> =** Volta o commit no tempo e permite alterações para ser mergeada com outra branch. É obrigatório criar uma branch a partir deste

**git checkout <branch> -b <nome-da-branch> =** cria uma nova branch baseada na branch específica. Exemplo, se está na branch master e precisa criar uma uma nova branch a partir da dev. Pode ser usada também com um commit voltado no tempo, colocando o número do commit ao invés do nome da branch

**git reset HEAD <arquivo> =** Remove o arquivo do staged área

**git reset –soft <numero\_do\_commit> =** Volta o commit, mas continua no staged e pronto para ser comitado

**git reset --soft –mixed <numero\_do\_commit> =** volta o commit e os arquivos para antes do staged

**git reset –hard <numero\_do\_commit> =** mata o commit e volta tudo que foi modificado para o estado inicial

**git revert <numero\_do\_commit>** = reverte o commit desfazendo as mudanças, porém mantem o mesmo na árvore do git.

**STASH**

**git stash =** congela o working directory com as alterações que ainda não foram comitadas, para trabalhar em outra branch e poder retornar na mesma quando achar necessário

**git stash list =** lista os commits e a alterações que estava sendo feita antes do git stash retornando um **stash@{numero}**

**git stash apply =** retorna o working directory ao estado das últimas alterações, para finalmente serem comitadas

**git stash clear =** limpa tudo que está no stash

**git stash drop stash@{numero} =** remove o stash

git stash pop = aplica o ultimo stash e dropa o penúltimo

**TAG**

**git tag -a 1.0.0 -m “mensagem\_de\_texto”** = cria uma tag com uma versão do projeto para ser baixada no github

**git tag =** mostra todas as tags geradas

**git push origin master --tags =** sobe as tags para o github criando um novo release

**git tag – d <numero\_da\_tag> =** apaga a tag do repositório local

**git push origin :<numero\_da\_Tag> =** apaga a tag do repositório no github

**FETCH**

**git fetch <repository> =** atualiza a memória do git, mostrando as alterações em alguma branch do repositório apontado. Exemplo: **git fech upstream**

**git merge repositoty/branch =** pega os dados mais recentes do repositório apontado. Semelhante ao **git pull que faz o fetch e o merge junto**. Exemplo: **git merge upstream/master**

**git checkout repositor/branch -b <branch> =** pega os dados do repositório remoto no github, baseado numa branch específica e já cria uma nova branch com o mesmo nome. **Exemplo: git checkout upstream/dev -b dev**