Comandos Git

- git init -> inicia um repositório;
- git status -> mostra o estado do nosso repositório, ou seja, quais arquivos foram alterados;
- git add NOMEARQUIVO -> para adicionar um único arquivo no monitoramento do git;
- git add . -> para adicionar todos os arquivos alterados naquele repositório;
- git rm NOMEARQUIVO para remover O MONITORAMENTO do git daquele arquivo;
- git commit -m "mensagem de envio" -> registrando uma mensagem quando do envio de uma alteração para o Git. Resumo descritivo;

- git pull OPÇÕES REPOSITÓRIO REFSPEC -> formato geral do pull
  OPÇÕES são as opções de comando, como --quiet ou --verbose.
  REPOSITÓRIO é o URL do seu repositório. Exemplo: https://github.com/freeCodeCamp/freeCodeCamp.git;
  REFSPEC especifica quais referências devem ser obtidas (fetch) e quais referências locais devem ser atualizadas;

- git pull NOME-REMOTO NOME-BRANCH -> Pull de um branch específico. Traz os dados de um repositório remoto para um local;
  NOME-REMOTO é o nome do seu repositório remoto. Por exemplo: origin;
  NOME-BRANCH é o nome do seu branch. Por exemplo: develop;

- git log -> abre o histórico dos commits, inclusive com o hash individual de cada um deles. Imagine o HASH de um commit como um código de barras;
- git log --oneline -> estrutura o log numa linha;
- git log -p -> visualiza o que foi alterado commit a commit;
- git log --graph -> mostra o log detalhado, inclusive com branchs;

- git config --local -> altera para este projeto;
- git config --global -> altera para todos;
- .gitignore -> arquivo especial onde todos os arquivos cujos nomes estão inseridos no .gitignore o git vai ler e ignorar;

- git remote -> lista todos os repositórios remotos
- git remote add NOME CAMINHO -> adiciona um repositório remoto de nome NOME e indicando o CAMINHO do repositório;
- git remote -v -> mostra os endereços do repositório;

- git clone CAMINHO NOME-DA-PASTA -> traz pela primeira vez todos os dados de um repositório remoto para nosso repositório local com o NOME-DA-PASTA que escolhermos

- git push NOME-REPOSITORIO-LOCAL NOME-DO-REPOSISITORIO-DESTINO (OU RAMO/BRANCH) (Ex.:main ou development ou feature);

- git branch NOME-BRANCH -> para criar uma branch para melhor controlar o versionamento do projeto;
- git branch -d NOME-BRANCH -> remove a branch;
- Obs.: Se tiver algum conflito, ou seja, o branch estiver a frente do branch atual, se essa branch que eu for apagar tiver commits a frente do master/main, então o 'd' deve ser maiúsculo:
- git branch -D NOME-BRANCH

- git checkout NOME-DA-BRANCH -> muda para aquela branch existentes (suas alterações vão pra a "nova");
- git checkout -b NOME-BRANCH -> cria e vai diretamente pra a branch criada;

- git merge NOME-BRANCH-A-SER-JUNTADA -> une o trabalho desenvolvido a partir da branch indicada (sem ser a principal) - dai se cria um commit de merge - para salvar e confirmar devemos digitar :x e ENTER (dois pontos, x , ENTER);

- git rebase NOME-BRANCH-A-SER-JUNTADA -> serve para deixar sua branch sempre atualizada. Ela pega os commits da branch a ser juntada na master, deixando com o log tenha em fluxo todos os commits das branchs, ou seja, ele atualiza a master em uma única linha, sem merges;

- git revert HASH-DE-UM-COMMIT -> desfaz o commit informado. Você pode ver qual o hash no git log;

- git stash -> Você salva as alterações em um local temporário sem gerar um log pra isso, permitindo que, por exemplo, você faça outros trabalhos e depois voltar aquele stash;
- git stash list -> lista o que está salvo;
- git stash apply NUMERO-DA-STASH ->Ttraz ela para o trabalho;
- git stash drop -> para remover;
- git stash pop -> tira a última alteração salva e traz novamente pra trilha e remove do stash;

- git checkout HASH-DO-COMMIT -> retorna o código ao estado daquele hash;

- git diff -> mostra a diferença de um arquivo que estou editando e ainda não foi adicionado pro commit o código já "commitado";

- git diff HASH1..HASH2 -> mostra todas as alterações que foram feitas do HASH1 até (significado dos '..') o HASH2;

- Uma tag marca um ponto na aplicação, não modificável, servindo pra marcar entregas (releases) de um sistema em desenvolvimento;
- git tag -a NOME-TAG-V0.1.0 -m "MENSAGEM" -> o '-a' cria a tag, o '-m' registra a mensagem. Sugiro v0.1.0 de inicio;
- git tag -> mostra todas as tag's disponíveis;
- git push LOCAL-OU-ORIGIN v0.1.0 -> envia a tag criando uma release no Gitlab

- git branch -m master main -> transforma master em main;
- git push -u origin main -> primeiro push;

Git Flow

Master/Main -> só recebe os commits prontos para produção. A partir desses commits a gente gera as tags.

A partir das master existe a branch de desenvolvimento.
Caso encontremos um erro na master, ou seja, em produção, deve ser criado uma branch a partir da master chamada hotfix

Development -> a partir dela serão criadas as branchs de features, ou seja, de funcionalidades novas, que, após "codadas" são mandadas de volta para a branch development. Quando a branch de desenvolvimento tiver todas as features nela a gente cria uma branch a partir dela de release

Features -> onde são desenvolvidas as novas funcionalidades

Hotfix -> onde são corrigidos os erros em produção. Após corrigidos devem ser enviados para a master e para a development

Release -> quando começa o processo de lançar uma nova versão. Nela só são corrigidos os bugs relacionados a essa release. Havendo correções deve ser enviado para a development e para a master

Passo-a-Passo GitFlow

O Mater/Main deve ser sempre o local onde temos o código pronto para produção.
Development é a branch onde fica todo o código em desenvolvimento.

As branchs criadas a partir da development são as features que estão sendo desenvolvidas em questão.
Ex.: git checkout -b feature/nome-da-feature1, git checkout -b feature/nome-da-feature2...

Quando essas features ficarem prontas é preciso fazer o merge pra development:
De dentro da developmente: git merge feature/nome-da-feature1

Se tiver sido encontrado bug no master/main deve-se abrir uma branch hotfix/v0.1.1 para realizar o conserto. Depois faz o checkout pra master/main e faz o merge da hotfix. Lembrar de fazer o merge também para a branch development;

Quando a development ficar pronta a release, deve-se criar uma branch a partir da development chamada release/número-da-versão. *Ex.: git checkout -b release/v0.2.0*
Ela serve para fazer as correções da release que já foi feita, mas veio com bug.

Depois de corrigir os bugs faz o checkout pra main/master e solicita o merge da release/v0.2.0

Com tudo incluído se cria a tag para marcar o fim daquela versão e início da próxima:
git tag -a v0.2.0(nome da tag) -m "mensagem indicando o que foi acrescido na versão"

OBS.: não esquecer de depois de consertar os bugs na branch release... fazer o merge também pra development pra manter o código sempre atualizado.