Comandos uteis GIT

# Padrão do GIT

git init => inicia a pasta local como git

git add . => adiciona todos os arquivos que estão na pasta local para serem upados no diretório

git commit -m “comentário” => faz um comentário sobre os arquivos que estão sendo upados

git remote add origin [git@github.com:SEU\_USUARIO/SEU\_REPOSITORIO](mailto:git@github.com:SEU_USUARIO/SEU_REPOSITORIO) => conecta a pasta com o repositório on-line

git push origin main => sobe os arquivos on-line

# Inicializar uma pasta no pc como git

Entrar na pasta e digitar o código git init

# Vinculando um projeto local a um repositório no git

git remote add origin ‘url do repositório no git hub’

# Comentando projetos

git commit -m “comentário do projeto”

# Adicionando arquivos locais ao repositório no git

git add *nome do arquivo*

git add . para adicionar todos os arquivos da pasta ao repositório

# Subindo os arquivos para o repositório

git push origin main

# Vincular a pasta criada com um projeto já existente

git remote add ‘*apelido’ url*

Onde ‘*apelido’*  é o nome que damos a origem onde o projeto está hospedado

E *url* é o endereço do repositório do git

# Listar repositórios do projeto

git remote -v

terão duas saídas uma de envios de commis (push) e a de baixar (fetch)

# Listando nome do repositório

git remote

# Removendo um repositório

git remote remove ‘apelido’

onde ‘apelido’ e o nome do repositório que deseja remover

# Alterando a url de um projeto

git remote set-url origin https: ‘nova url’

# renomeando o apelido de um repositório

git remote rename ‘apelido’ ‘novo-apelido’

# Configurando o usuário global da maquina

Para que o git sempre reconheça o usuário da máquina conectada ao repositório

git -config --global user.email "seuemailaqui@example.com"

git config --global user.name "seu nome aqui"

# Commit

Commit é como se fosse uma gravação do código até o período que o código foi escrito, é um ponto no tempo de como estava o código no momento em que o commit foi feito.

# Criando uma Branch

Uma Branch é como uma linha do tempo separada dentro de um desenvolvimento de projeto, imagine que temos a linha principal “master” uma Branch é uma cópia dessa linha principal até aquele ponto no tempo onde o código pode ser alterado sem afetar a linha do tempo principal, após todos os teste essa Branch pode ser mesclada (merge) com a linha do tempo principal aplicando as mudanças feitas

git branch -M main => onde main é o nome da branch

# Gerenciando as versões do projeto

É uma ferramenta essencial para gerenciar alterações no controle de versão Git. Ele fornece uma visão geral do estado atual do repositório, indicando quais arquivos foram modificados, adicionados ou excluídos desde o último commit.

git status

# Baixando o repositório on-line para a maquina local

git clone ‘url do repositório on-line’

# Visualizando os historicos dos commits

git log

para sair digitar ‘q’

git log –oneline => exibe apenas uma linha de cada commit, o que foi feito

git log --graph => mostra um gráfico das branches

git log --pretty ou git log --format me permite escolher como quero ver o log

# Baixando as informações alteradas do repositório

git pull origin main, onde origin é o nome do repositorio

# Recuperando um commit no tempo

Para recuperar um estado de um commit no tempo, ou seja, uma versão do projeto, precisamos localizar o código do commit da alteração com o comando:

git log

copiar o código do commit

e utilizar o código:

git revert *‘codigo do commit’*

Esse revert não desfaz as alterações no código, ele refaz um novo commit registrando um novo commit

Após o revert utilizar o código

git push origin main

# Apagando um commit

Um commit que ainda n foi enviado ao repositório on-line pode ser apagado através do comando:

git reset -hard *‘codigo do commit anterior ao que queremos reverter’*

# Alterando último commit

Se quiser alterar a mensagem feita no **último** commit, basta utilizar o código

git commit --amend -m ‘nova mensagem no commit’

# Omitindo arquivos no git hub

Como não subir todos os arquivos para dentro do repositório on-line

Criar um arquivo dentro do diretório local com o nome

.gitignore

E colocar o nome do arquivo, pasta, coleção de arquivos que deseja omitir

temp/ '=> ignora uma pasta'

arquivo\_senhas.txt '=> ignora um arquivo expecífico'

\*.css '=> ignora uma cadeia de arquivos do tipo css'

Podemos criar um arquivo .gitignore de forma “automática”, pois algumas linguagens possuem arquivos padrão que não são enviados ao repositório o line através do sit [gitignore.io](https://www.toptal.com/developers/gitignore/)

# Verificando o que foi alterado

git show ‘hash do commit’

o Hash do commit é uma parte do código de identificação de cada commit, conseguimos velo através do código

git log – oneline são os primeiros caracteres em destaque

# Verificando se há commits a serem upados

git status

verifica se há commits feitos que ainda não subiram para o repositório on line, e

git diff mostra qual é a diferença entre o último commit feito e salvo no diretório on-line e o arquivo local

se quiser verificar a diferença em mais de um commit

git diff *‘hash mais antigo’* ..*’hash mais novo’*

# Branch – o que é?

Branch é uma linha de trabalho, é uma ramificação do código

git branch

<https://git-school.github.io/visualizing-git/> para visualizar a linha do tempo do git

* Para renomear uma branch
  + git branch -m ‘nome atual’ ‘novo nome’
* Criando uma nova branch
  + git branch ‘nomedabranch’
* Mudando entre as branches
  + git swith ‘nome da branch’
  + git checkout ‘nome da branch’
* Criando uma branch e se movendo para ela
  + git swith -c ‘nome-da-branch’

# Unindo as branches

Você deve estar na branch principal ‘main’ e utilizar o c

git merge ‘nome da nova branch’ => mescla as branches em uma mesma linha

* forcando um merge commit
  + git merge --no-ff ‘nome da branch’

# Deletando uma branch

git branch -d ‘nome da branch’

Para deletar a branch também do repositório online

git push origin :’nome-da-branch-deletada’

# Adicionando a alteração da branch no commit mais recente da main

git rebase *‘branch principal’*

é como se as alterações da branch fossem movidas pra apos da versão mais recente da main, verificando commit a commit se não há inconsistência

A branch adicional é movida pra depois do último commit da branch principal.

# Pausando uma tarefa não finalizada

* git stash
  + guarda uma alteração sem realizar o commit e desfaz o que eu alterei no projeto principal
* git stash pop
  + recupera o último estado da pilha alteração que foi guardado anteriormente
  + podemos adicionar o índice da stash como git stash pop 1, que será aplicado e removido da lista de stashs
* git stash apply *‘indice do stash’ – numeric*
  + *aplica o indice da stash, mas não a remove da lista de stashs*
* git stash list
  + lista os stachs
* git stash clear
  + limpa os stachs feitos
* git stash push -m *‘lista de alteracaoes da stash’*
  + identa as alterações, como se fosse um commit pra detalhar o que foi feito
* git stash drop
  + deleta o último item da stash sem aplica-lo, para deletar uma stash específica basta adicionar o índice ao final do codigo

# Desfazendo alterações que estão prontas pro commit

git restore . => o ‘.’ Significa que todos os arquivos serão restaurados Restaura o último commit feito.

* git restore –staged
  + restaura o que está na staging área após o git add, são alterações que já foram feitas, mas não commitadas.
* git restore --source=*’rash identificador do commit’ ‘nome-do-arquivo a recuperar o estado deste commit’*
  + *posso fazer esse restore ver o que foi feito e restaurar de volta o arquivo com git restore ‘nome-do-arquivo’, é como se eu tivesse conferido as alterações e retornado ao último estado do arquivo*

# Lançando uma versão

O projeto está pronto para ser compartilhado, é como se fosse um checkpoint no tempo

Uma versão para se caso perdemos alguma coisa podemos recuperar essa versão completamente funcional

Utilizamos o comando

git tag *‘nome da versão’*