**Comandos e explicações do curso Git e Github**

http://devhints.io/git-log - site que contém os comandos de git log

**: x** - salva e atualizar a branch master

**clear** - limpa a tela do terminal

**ls** - mostra os arquivos salvos na pasta de trabalho

**git init** - inicializa um repositório na pasta escolhida

**git status** - mostra todas as modificações a serem "comitadas" e estão sendo monitorados

**git add** <file> - adiciona o arquivo com o nome digitado

**git add** . - adiciona todos os arquivos dentro da pasta

**git commit** -m "Mensagem contendo as informações de alteração (Usar mensagem intuitiva)", comita os arquivos

na branch master ou escolhida, salva as alterações feitas.

**git log** - mostra informações como hash(especie de id unico para cada commite), ramo onde foi

realizado o commit, mostra a mensagem com a informação de alteração, data, hora, user name, email.

git **config --local user.name "Adicionae o nome"** - configura o usuário responsável

pelas alterações do projeto

**git config --local user.email "Adicionar o email"** - configura o email do usuário responsável

pelas alterações do projeto

**git config --global user.name "Adicionae o nome"** - mesma função do config local mas configura

o usuário para todos os projetos criados e commitados

**git config --global user.email "Adicionar o email"** - mesma função do config local mas configura

o usuário para todos os projetos criados e commitados

**git log --oneline** - mostra cada commite em uma linha mostrando apenas o Hash(ID), a mensagem de comite e

a branch em qual ocorreu o commit

**git log -p** - mostras todas as infos do git log e ainda mostra as linhas que foram alteradas

**git log --pretty="format:H%"** - mostra apenas o hash(ID) do commit

**git log --pretty="format:H% s%"** - mostra o hash resumido e a mensagem de commite

**git log --pretty="format:H% s% %ae"** - adiciona o email do autor no fim das informações de git log

criar arquivo .gitignore - arquivo especial que serve para o git ignorar determinados arquivos, ou seja,

todos os arquivos listados no arquivo .gitignore serão ignorados pelo git e não serão monitorados. é necessário

adicionar ou commitar o .gitignore

**cd** .. - sobe ou volta 1 pasta da posição atual

mkdir nome da pasta - cria uma pasta no diretorio escolhido

git init --bare - indica que este repositorio so irá conter as alterações dos arquivos e não uma cópia

física de cada um deles

git remote - lista todos os repositórios remotos que o meu repositorio conhece

git remote add local endereço ou caminho do diretorio - adiciona um repositorio remoto (Giithub)

git remote -v - mostra o endereço do local mostrando o caminho no qual irá buscar dados e enviar

git clone endereço - irá clonar todos os arquivos do repositorio remoto para o seu local pela primeira vez

git push local master - envia todos os dados do repositorio local para o servidor na branch master

git remote rename origin local - renomea o repositorio de origin para local

git pull local master - baixa os dados do repositorio remoto local para a sua branch master

**Enviando arquivos locais para o Github**

Antes de enviar o arquivo é necessário preparar o repositorio local com os comando git init, add, commit, etc

criar um repositório no github para poder enviar os arquivos do seu pc

escolher a opção https copiar o comando com endereço do repositorio

**git remote add origin endereço/URL** do repositório no github

**git push -u origin master** - sempre que digitar git push e estiver na master irá enviar os dados para a branch

origin

**git push origin master** - enviar os arquivos da branch master local para o repositorio remoto origin

**git push origin master** - Uma melhor opção é usar o comando git push "repositorio remoto" "branch que quero enviar"

irá pedir login e senha do github, caso esteja no linux a senha será solicitada no proprio terminal git bash

**Branches ("ramos")** são utilizados para desenvolver funcionalidades isoladas umas das outras.

A branch master é a branch "padrão" quando você cria um repositório.

git branch - lista todos os branch existentes

git branch titulo - cria um novo branch chamado titulo (subsitutir pelo nome desejado)

git checkout titulo - fará mudar a linha de desenvolvimento para o branch titulo

git checkout master - fará a linha de desenvolvimento voltar para branch master

git checkout -b lista - cria um novo branch com nome lista e muda a linha de desenvolvimento para este

novo branch criado chamado lista

git merge titulo - cria um commit juntando 2 branchs, neste exemplo trouxemos as modificações do branch

titulo para o branch master que é a branch principal de desenvolvimento (neste caso estavamos fazendo mudanças

na branch titulo e precisavamos alterar um bug na brand lista que não fazia parte do nossa parte. Então voltamos

para branch master com git checkout master, fazemos a mudanças necessária na lista e comitamos, após finalizar

caso precise trazer as mudanças feitas na branch titulo executamos git merge titulo que irá fazer a junção da branch

titulo e lista.

git rebase titulo - pega as mudanças feitas na branch titulo e adiciona na branch master fazendo com que ela esteja

atualizada com as mudanças necessárias e depois adiciona o commit da branch master na frente de todos os outros.

Esse recurso nos permite atualizar a branch master com as atualizações das demais brachs sem ter que criar branchs como

foi utilizado no comando git merge, onde após utilizar este comando ele irá criar uma branch de merge juntando as 2 branchs

com suas atualizações.

**git log --graph** - mostra as linhas especificas de desenvolvimento, mostrando as alterações, branchs criados.

**git checkout** **-- <nome do arquivo>** - Desfaz as alterações no arquivo de código antes de adicionar no repositorio

**git reset HEAD <nome do arquivo>** - Desfaz as alterações no arquivo após usar git add, fazendo com que voltemos para

o ambiente de desenvolvimento que estamos trabalhando. Após isso basta utilizar o comando git checkout e a alteração

não estará disponível para commit

**git revert hash do commit (ID mostrado no comando git log)** - desfaz o commit que foi adicionado fazendo com que o código

volta ao estado inicial antes da mudança.

**git stash list** - mostrar a lista com todos os arquivos salvos que podem ser usados

**git stash** - salva os dados do diretorio atual para continuar trabalhando depois

**git stash apply numero da stash** - aplica as alterações salvas na stash mas não apaga da lista

**git stash drop numero da stash** - apaga o arquivo salvo na stash

**git stash pop** - remove a ultima alteração salva no stash e traz pra branch

**git checkout hash do commit(7 primeiras casas ID mostrado no git log)** - desloca, desanexa para o commit especificado

retirando das branchs anteriores de trabalho. Este comando permite que naveguemos pelos diversos commits e merges

do nosso projeto. Com este comando as alterações que são feitas serão descartadas, sendo assim caso precisemos salvar

as alterações que fizermos no código deveremos criar uma nova branch com git checkout -b nome da branch, e então podemos

navegar para a master ou qualquer outra branch salvando as atualizações.

**git diff hash do 1 commit..hash do 2 commit** - Permite visualizar todas as alterações que foram feitas entre esses

dois commits

**git diff** - mostra todas as alterações feitas e que ainda não foram adicionadas para commit

**git tag** - mostra todas as tags disponiveis

**git tag -a nome da tag (exemplo v.0.1.0) -m "Mensagem"** - gera uma versão, um marco no commit, aplicação escolhido

Cria um marco que não pode ser modificado, qualquer modificação necessária irá vir nas proximas atualizações. Serve para

marcar um ponto que não evolui mais, nao muda mais na aplicação. O código ainda poderá ser alterado mas com a tag criamos

a oportunidade de navegar para aquela tag, aquele ponto onde o código estará naquele estado sem modificação. Ao enviar

esta tag para o Github nos geramos uma release, ou seja, poderemos baixar um arquivo compactado com o nosso codigo neste ponto.