**Curso de Git & Github**

O que é o Git? É um sistema com finalidade de gerenciar diferentes versões de um documento/projeto.

O que é o Github? É um serviço de web compartilhado para projetos que utilizam o Git para versionamento, estilo uma rede social.

1. **Instalando o Git**

Página para Download: <https://git-scm.com/>

* 1. Configurações iniciais do Git
* O Git guarda suas informações em 3 lugares, Git config do sistema, do usuário e do projeto do repositório.
  1. **Configurações iniciais de conta:**
* A primeira coisa que você deve fazer ao instalar Git é configurar seu nome de
* Usuário e endereço de e-mail. Isto é importante porque cada commit usa esta
* Informação, e ela é carimbada de forma imutável nos commits que você

Começa a criar.

* Para configurar seu nome de usuário:

git config --global user.name “Leandro Marcelino Souza”

* Para configurar seu email:

git config --global user.email [“leandro.souza.marcelino@gmail.com](mailto:“leandro.souza.marcelino@gmail.com)”

* validando configurações

git config --list

Para remover/alterar as configurações de usuário para depois criar outro

* Git config –global –unset user.email
* Git config –global –unset user.name
  1. **Configurando o editor principal do Git**

Git config –global core.editor sub - sublime ou

Git config –global core.editor vim - Vim ou

Git config –global core.editor emacs – emacs ou

Git config –global core.editor 'code --wait' – Vs Code ou

* Se nada for definido, o Git assume o Vim como padrão.
* Para executar o edito padrão: git config --global -e
  1. **Testar as configurações setadas:**
* Para que sejam verificadas as configurações, é possível listá-las. Para isso, utilize o
* Comando abaixo:

git config user.name

git config user.email

git config --list

* 1. **Criando pasta do projeto**

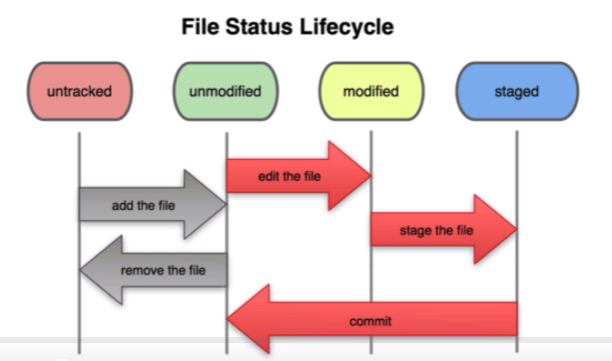
Mkdir nome\_da\_pasta

Cd nome\_da\_pasta

* 1. **Inicializar o repositório criado no git**
* Dentro da pasta do projeto, execute o comando abaixo:

Git init

* 1. **Ciclo de vida de um arquivo no Git**



**Untracked (não rastreado) / Unmodified (não modificado) / Modified (modificado) / staged (encenado)**

* 1. **Verificar o status do projeto, reportar em qual repositório estou**

git status

Ao criar o arquivo no diretório pela primeira vez, ele fica com **untracked(não rastreado)** ao verificar o status( git status).

* Para adicionar o arquivo para commitar (commit) temos que seguir os passos do próximo tópico.
  1. **Commitar arquivos/alterações no repositório local:**
* Vários arquivos ao mesmo tempo, adicionando e depois commitando: git add .

git commit -m “mensagem do commit”

* Adicionar e commitar ao mesmo tempo:

git commit -a -m “mensagem do commit”

* Um arquivo por vez:

git add . meu\_arquivo.txt

git commit meu\_arquivo.txt -m “mensagem do commit”

* 1. **Verificar histórico de commits:**

git log --pretty=oneline ou git log

git log --decorate

git log –-author=”nome”

git shortlog

git shortlog –sn

git log –-graph – mostra a parte gráfica

git show + a “hash” – mostra a parte adicionada no commit

* 1. **Visualizando o diff**
* Verificar as mudanças antes de enviar o commit.
* Ele mostra as modificações.

git diff

git diff –name-only – somente o nome do arquivo que foi modificado.

* 1. **Reverter arquivos**
* Reverter um arquivo ao seu estado original:

git checkout <file>

* E importante entender que o git checkout -- <file> e um a comando perigoso. Qualquer modificação que você fez no arquivo se foi — O Git apenas substitui o arquivo pela última versão (mais recente) que sofreu

commit. Nunca use este comando a não ser que você saiba com certeza que não quer salvar as modificações do arquivo.

* Reverter todas as alterações da branch:

git checkout -f ou git reset –HARD

* **git reset --soft** - vai pegar as modificações e vai matar o commit feito, porém o arquivo vai estar em staged com a modificação pronta para ser comitada(commit) de novo.

**git reset --soft** + a hash

* **git reset --mixed** - vai matar o commit, e volta os arquivos para antes do staged, para modified.

**git reset --mixed** + a hash

* **git reset --hard** - ele vai ignorar a existência do commit e tudo que foi feito nele.

**git reset –hard** + a hash

**Github**

**2.1. Criar conta no Github**

<https://github.com/>

2.2. **Criar um novo repositório**

[**https://github.com/new**](https://github.com/new)

**2.3. Gerar chave SSH no GitHub**

**Criando a chave ssh**

ssh-keygen -t ed25519 -C [leandro.souza.marcelino@gmail.com](mailto:leandro.souza.marcelino@gmail.com)

**Navegando até a pasta das chaves ssh e copiando a mesma com o comando cat**

NyckMel@NYCK-MEL MINGW64 ~/.ssh

$ cat id\_ed25519.pub

ssh-ed25519 aqui aparecera a chave e o seu email leandro.souza.marcelino@gmail.com

**Iniciando o SSH**

eval $(ssh-agent -s)

<https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

**2.4 ligando repositório local a um remoto**

* Adicionar um repositório remoto

git remote add origin [git@github.com:LeandroMarcelino/github-course.git](mailto:git@github.com:LeandroMarcelino/github-course.git)

* Mostrar o repositório

git remote

* Mostrar mais informações

git remote –v

* Envia os arquivos e modificação para o repositório remoto

git push -u origin master

**2.4. Enviando mudanças para um repositório remoto**

git push origin máster/main

**2.5. Clonar repositório**

git clone gi[t@github.com:LeandroMarcelino/github-course.git](mailto:t@github.com:LeandroMarcelino/github-course.git%20G) Github-course-clone

**Branch**

**3.**  **O que é um branch e por que usar?**

* É um ponteiro móvel que leva a um commit.

**3.1 Criando um Branch**

* Criando o branch

git checkout –b testing

* Verificando os branch criados

git branch

**3.2. Movendo e deletando branches**

* Listar branches

Git branch

* Mudar de branch

Git checkout <nome do branch>

* Apagando brench

Git branch –D <nome do branch>

**Gitignore**

**4. Gitignore - especifica arquivos intencionalmente não rastreados para ignorar**

<https://git-scm.com/docs/gitignore>

https://github.com/github/gitignore

* Criando o arquivo gitignore, você pode criar com qualquer editor de texto.

**vim .gitignore**

* **Ignorando a extensão .json e xlsx dentro do arquivo, gitignore**

**\*.json**

**\*.xlsx**

**Stash**

**5. Git stash**

* O git stash cria um rascunho da última modificação feita no arquivo, com isso você pode trabalhar em outros pontos, sem perder o que já foi feito.

git stash

git status

* Quando você quiser salvar as alterações salvas no arquivo, basta executar o comando abaixo:

git stash apply

* Para verificar a lista de stash, basta executar o comando abaixo

git stash list

* Para limpar tudo o que estiver no Stash, basta executar o comando abaixo:

git stash clear

**Alias**

**6. Criando Alias**

* Criando atalho para o comando status

Git config –global alias.s status

**Versionamento com Tag**

**7. Criando versionamento com tags**

* Criando a tag

git tag –a 1.0.0 –m “Readme finalizado”

* Subindo a tag para o Github

git push origin master --tag

**Git Revert**

**8. Git Revert, para reverter uma alteração**

* Com ele você pode voltar para uma configuração anterior (como se fosse um back up), assim, você tem mais tempo para analisar o código em caso de erro.
* Modificando o arquivo

vim Readme.md

* Salvando/commitando as informações alteradas

git commit -am "Add new line"

* Verificando os logs

git log

* Verificando as alterações

git show + a “hash”

git show 94fd3419455184e49444461ba2799ed0c354a23a

* Revertendo as alterações

git revert + a “hash”

git revert 94fd3419455184e49444461ba2799ed0c354a23a

**Apagando tags e branches  
remotos**

**9. Apagando tag local e remota**

* Verificando as tags

Git tag

* Deletando tag local

Git tag –d <tag>

* Deletando tag remota

Git push origin :<tag>

Git push origin:1.0.0

**dio-takeblip-aplicando-conceitos-git-pratica**

Repositório para a Mentória do Digital Innovation One

* Inicializar um repositório no git
* Adicionar arquivos ao nosso projeto
* Fazer o envio para a base do controle de versão
* Ignorar arquivos do Git
* Trabalhar com branches
* Merge com a master
* Inicializar um repositório no Github
* Conhecendo a página do Github
* Subir nossos arquivos local
* Clonar um repositório
* Adicionar um README.md

**Códigos**

-git init

- Git add <nome do arquivo>, com este comando adiciono arquivos

que estão criados no meu workspace para o staging area

- git commit -m 'mensagem de envio'

- .gitignore

- git branch, git checkout, git branch -D list

- git stash / git stash drop stash@{n}

- git stash pop / git stash list

- git merge

- git version

- git log

- git rm -rf node\_modules/ --cached

-recursivo e forçada

- git config --global core.editor 'code --wait'

- git branch -m novo nome

- git branch -D nome da branch

- git push origin <nome da branch>

- git merge --abort

- git reset HEAD

- git push -u origin

- git remote add origin https://github.com/mhnakashima/dio-takeblip-aplicando-conceitos-git-pratica.git

- git branch -M main

Isso significa que, ao executar o comando, a branch atual será renomeada para main, mesmo que essa branch já exista (efeito causado pelo --force).

- git push origin branch --delete

- git push -u origin main

**Dicas úteis**

* Pull sempre que for começar a trabalhar
* Sem commits na master
* Geralmente temos uma branch chamada DEV que receberá todas as features em desenvolvimento.
* Usar uma branch por feature
* Use nomes descritivos para commitar seus arquivos com a nomenclatura fix / feat