**CONTEÚDO DO CURSO GITHUB**

**SESSÃO 1 – ENTENDO SOBRE O GITHUB**

**- Introdução**

**- Controle de Versão**

**- História do Git**

**- O que é o GitHub** (é um local na web, na nuvem que vai armazenar os projetos versionados pelo git) O Git não o mesmo que o GitHub (Git é o sistema de controle de versão – GitHub é um local onde você coloca seus projetos em Git)

**SESSÃO 2 – CONFIGURANDO O GIT**

**- Instalando o Git**

**- Configuração inicial do Git** (Guarda informações em 3 lugares “git config sistema” “git config usuário” “git config projetos”)

\* git config --global user.name “Luan Eduardo”

\* git config --global user.email “[luandurcartola@gmail.com](mailto:luandurcartola@gmail.com)”

\* git config --global core.editor (comando do editor)

**SESSÃO 3 – ESSENCIAL DO GIT**

**- Inicializando repositório**

\* Criar uma pasta ( mkdir git-couse )

\* Entrar na pasta ( cd git-couse/ )

\* Inicializar repositório ( git init)

\* Se digitar (ls – la) vai falar que tem um diretório chamado git

\* Para visualizar diretório ( cd .git/)

\* Voltar na pasta (cd..)

**- Usando editor do terminal**

\* Para editar o arquivo ( vi Readme.md )

\* Ao digitar letra “i” para entrar no modo (Insert)

\* Após escrever basta apertar teclas:

esc/ para sair modo

; / iniciar algum comando

w/ escrever e salvar

q/ sair

**- O ciclo de vida dos status de seus arquivos**

Untracked – arquivo adicionado, mas não foi visto pelo git

Unmodified – após ser adicionado mas não foi modificado

Modified – arquivo editado mas não foi salvo

Staged – momento em que o arquivo fica sendo avisado para levar os arquivos a ser (Unmodified) ele está pronto para ser salvo.

\*Serve para reportar como está o repositório no momento (git status)

\* Para criar um commit (git commit -m “mensagem”) commit é pegar os arquivos e cria uma imagem

\* Para adicionar arquivo ( git add “nome do arquivo” )

**- Visualizando logs**

\* ( git log ) mostra rest do commit, quem foi o autor da modificação, a data e uma mensagem.

\* (git shortlog) mostra em ordem alfabética, quantos commits e quais eles foram.

\* ( git log --graph) mostra em forma gráfica o que está acontecendo

**- Visualizando o diff**

\* (git diff) Mostra a modificação “ usa-lo sempre antes de fazer algum commit”

\* (git show) mostra a mudança que fez no começo

**- Desfazendo coisas**

\* (git checkout Readme.md) Ele retorna o arquivo para antes da edição.

\*( git reset HEAD Readme.md) mostra o arquivo que foi tirado

\*( git reset - -soft - -mixed - - hard) cada um tem a sua função.

**SESSÃO 4 – REPOSITÓRIOS REMOTOS**

**- Criando um repositório no Gitgub**

\* Repositório remoto é aquele que fica em outro servidor.

**- Criando e adicionando uma chave SSH**

\* SSH = Protocolo que autentica um usuário remoto ao servidor.

\*(ssh-keygen “gmail”) Após abrir o terminal

\* (cd ~/ .ssh/) Esse é o diretório onde fica as chaves do SSH

\* Para usar a chave utilizar ( cat id\_rsa.pub)

**- Ligando repositório local a um remoto**

\*(git remote add origin [git@github.com:gmail/github-couser.git](mailto:git@github.com:gmail/github-couser.git)) Para adicionar um repositório remoto.

**- Enviando mudanças para um repositório remoto**

\* (git push origin máster) Ir no terminal e colocar comando para fazer o push e enviar.

**- Clonando repositórios remotos**

\* (g clone “repositório”/ github-couser-clone) Comando para clonar no novo repositório.

**- Fazendo fork de um projeto**

\*Fork pega um projeto que não é seu e faz uma cópia dele para você.

**SESSÃO 5 – RAMIFICAÇÃO (BRANCH)**

**- O que é um branch e por que usar ?**

\*Branch é um ponteiro móvel que leva a um commit.

\*Vantagens: pode modificar sem alterar o local principal/ facilmente “desligável”/Múltiplas pessoas trabalhando/ Evita conflitos.

**- Criando um branch**

\*(git checkout – b testing) para criar e entrar

**-Movendo e deletando branches**

\* Para entrar em um branch (git branch (nome))

\* Para voltar ( git branch máster)

\* Para apagar (git branch –D (nome))

**- Entendendo o Merge**

\*Sempre que criar um Merge, criará também um commit

Merge Pro – Operação não destrutiva

Merge Contra – Commit extra/ Histórico poluído

**- Entendendo o Rebase**

\* Rebase pega o commit e move para frente (C3 para frente do C4)

Rebase Pro – Evita commits extras/Histórico linear

Rebase Contra – Perde ordem cronológica

**- Merge e Rebase na prática**

\*Crie uma pasta

\*Inicialize o repositório

\*Crie outro arquivo e commit ele.

\*Crie outro arquivo, no caso o terceiro.

\*Usando o (merge branch ‘test’) vai adicionar

\*(R) Criar um arquivo buzz

\* Usar sempre o Rebase quando puder

**SESSÃO 6 – EXTRAS**

**- Criando o .gitignore**

\* Ele serve para ignorar especificamente, não traquear alguns arquivos.

\* Criar arquivo ( git .gitignore)

\* Dentro do gitignore, você pode especificar oque deseja

\* Para editar (vi.gitignore)

**- Git stash é lindo**

\*(git stash apply) Ele vai aplicar as mudanças que tinha guardado

\*(git stash list) Mostra a lista de todos os stashs que foi feito.

\*(git stash clear) Limpa tudo que estiver no stash

**- Alias para que te quero**

\*Criar atalhos dos comandos

\*(git config - - global alias.s status) Criou um atalho com o comando status

**- Versionando com Tags**

\*(git tag –a 1.0.0 - -m “Readme finalizado”) Tag criada e adicionada

\*(git push origin máster - - tags) Para subir pro Git

**- Salvando sua sexta com Git Revert**

\*(git revert) Ele reverte o commit

\* Pra que usar o Revert ? Não perder as mudanças que você fez.

**- Apagando tags e branches remotos**

\*Para apagar no local (g tag –d 1.0.1) mas isso não sobe para o repositório

\*Para apagar as tags ( git push origin ; (a tag que deseja apagar))