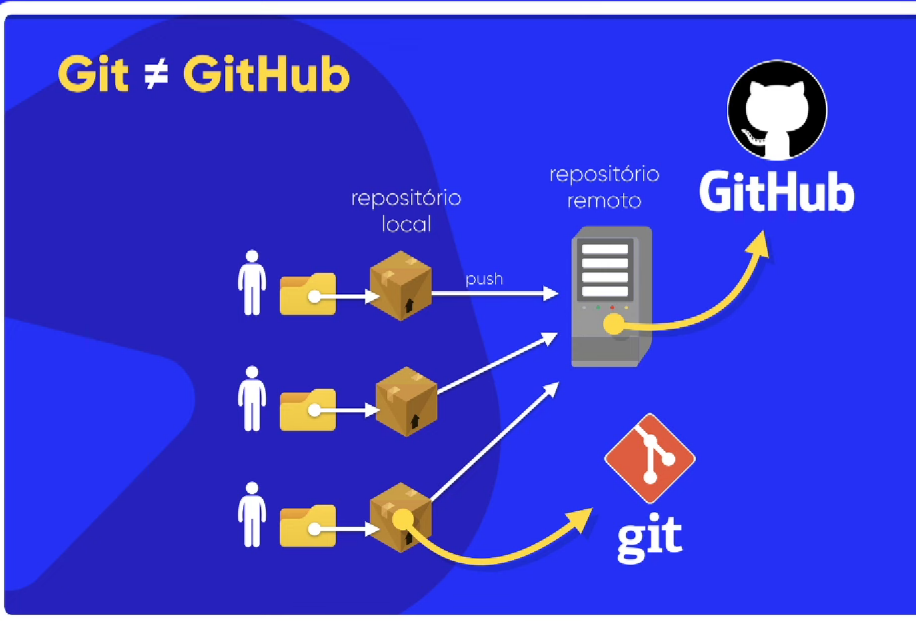
**Aula 01**

**Git e GitHub são coisas diferentes.**

Git é um software de controle de versão. E o GitHub é uma plataforma de redes sociais para programadores.

**Explicando Git:**

Git é um VSC “software de controle de versão”, resumindo em uma palavra, versionamento/versões, e programador sempre lida com diferentes versões de algo, não só quem programa mais também quem utiliza softwares.

**Versionamento**

Existem 2 tipos de versionamento: o centralizado/linear, que é a forma de guardar arquivos em servidor, de forma centralizada, onde você divide seus arquivos diretamente com as outras pessoas relacionadas a um projeto. Existe também o distribuído, onde você tem sua partição de arquivos ao mesmo tempo que consegue dividir eles com os demais.

A imagem mostra um exemplo de versionamento distribuído.

**Principais Vantagens de usar esse tipo de software:**

* Controle de histórico
* Trabalho em equipe
* Ramificação de projeto
* Segurança
* Organização

**Aula 02**

GitHub, já sabemos que é uma rede social de programados, o uso maioral dela é guardar códigos, mostrar códigos prontos para aprimorar chances de emprego, funcionar diretamente como repositório remoto para empresas que usam GitHub.

**Vantagens:**

* Repositório ilimitado
* Hospedagem de código-fonte
* Características de rede social
* GitHub Pages integrado
* Colaboração
* Forks

Só existe o GitHub? Além dele temos:

* GitHub
* GitLab
* Bitbucket
* PHABRICATOR
* Gogs
* Kallithea

**Aula 03**

Falando um pouco da história...

**CVS: SVN:**  **BitKeeper**

1985 2000 2000

Centralizado Centralizado Distribuído

Open Source Open Source Proprietário

Mais Popular Ativo até hoje Versão comunidade

Alguns problemas CVS-Like CVS-free

**BitKeeper BitKeeper** **Git**

2004 2005 Distribuido

Source Puller Nova licença Open Source

Engenharia Reversa Acesso a metadados feito em 10 dias

Recursos destravados Só na versão comercial Perfomance

O quer dizer **Git**?

Três letras do alfabeto

Teimoso, cabeça-dura, pensa que sempre está certo

Global information tracker

Goddamn idiotic truckload of sh#t

Aqui claramente são brincadeiras que o criador usava para se referir.

O **Git** tem esse nome por que era o que tinha.

E o GitHub?

**GitHub GitHub**

2008 2018

Proprietário Adquirido pela Microsoft

Hospedagem de código US$ 7.5 BI 🫥

Baseado em Git Operação independente

Em 2011: Ultrapassou **SourceForge** 2020: GitHub compra a npm

2018: Maior ataque dos **DDoS** da história

**Aula 04**

O que preciso instalar?

Navegador -> já utilizo o Chrome, então vou me manter nele.

Editor de códigos -> Já tenho o Visual Code, não tenho conhecimento, mas irei me manter nele.

**Git** e **GitHub Desktop** -> tenho obrigação de ter hehe

Bom, aqui foi só isso kkkkk, de aproveitamento dessa aula foi apenas algumas alterações no Visual Code que eu só fui instalando algumas extensões recomendadas até o momento.

**Aula 05**

Aula totalmente pratica, até o momento consegui criar meu primeiro repositório 🫡, me pareceu passos simples, primeiro a pasta é criada de forma local “Git” e depois tenho a opção de jogar essa pasta para o GitHub de forma pública ou de forma privada.

Pelo Visual Code, é possível gerenciar o README, esse arquivo vai possibilizar personalizar o seu repositório do GitHub, adicionar um comentário, fazer uma anotação ou deixar explicado como o seu repositório funciona e o que tem nele.

Anotações de como estou entendendo:

O **commit** serve para aplicar alterações no Git.

**Push** serve para mandar ou puxar essas alterações para o GitHub.

**Pull** pega o que está no GitHub “que foi alterado de forma Remota” e trazer para o Git, local.

Isso tudo vai interagir com meu repositório Local, com o GitHub Desktop, GitHub remoto “acesso pelo site” e pelo visual Code 🤯.

Alguns Atalhos:

CRTL + Enter= commitar depois de digitar o titulo

CTRL + P= Push logo após commitar

Aula Top hehe 🧠

**Aula 06**

Instalando o GitHub no Linux (não uso, mas não é por isso que irei deixar de assistir a aula.

Basicamente o mesmo processo usado para o Windows, vi a aula completa por comprometimento ao conteúdo, não pretendo usar Linux, caso precise já tenho uma Base, e já sei em que aula devo voltar caso precise de ajuda.

Pra ser sincero, revisão da aula 05 no Linux, pra quem tem Linux é uma aula fundamental, pra quem já viu o processo no Windows pode até pular.

**Aula 07**

Clonando repositório:

Clonagem de repositório é algo bem simples de ser feito e muito útil, consigo ir lá no perfil público de uma pessoa, como o Gustavo Guanabara, e clonar os arquivos públicos que essa pessoa disponibiliza, quando um repositório é clonado todos os arquivos que estão lá vem pro dispositivo de forma local, é como fazer downloads de arquivos na internet ou fazer um **pull** dos seus arquivos que estão na nuvem.

Inclusive já clonei o repositório de HTML e CSS do guanabara por que vai ser minha próxima fonte de estudos.

**Aula 08**

Versionando Projetos

Tudo que foi feito durante a aula eu já tinha feito no intervalo da aula anterior para essa, peguei exercícios e anotações que foram criados durante o estudo de algoritmos e consegui commitar e fazer o push.

É falado sobre Gitignore, posso selecionar um arquivo para não commitar, ele vai fazer a leitura se tem mais arquivos iguais e jogar todos esses arquivos para pasta gitignore, tudo que estiver nesta pasta não vai versionar nem adicionar no GitHub.

Até o momento está bem claro como é feito o funcionamento entre Git e GitHub, versionamento e também sobre o nome dos comandos usados.

Ele reexplica sobre a parte de clonar, puxar esses arquivos que está de na Nuvem do GitHub para o Dispositivo Local, deixando claro que clonar não é jogar o repositório de outra pessoa, e exatamente como foi explicado, ele baixa de forma local.

Se quero puxar o projeto pro perfil existe o Fork, ele menciona mais não entrou a fundo ainda.

**Aula 09**

Aula sobre Issues

Issues é um problema que o publico do GitHub tem, não conseguiu identificar e resolver, e assim publica no GitHub para procurar pessoas que pode ajuda a resolver aquele problema, aquele Issues.

Essa lacuna só aparece se o problema ainda não foi resolvido, então se alguém publica um Issues, esse é resolvido, ele deixa de ser um.

Na aba de Issues temos a aba Open” abertos”, e Closed” fechados”, os abertos ainda não foram resolvidos e os fechados foram abertos e resolvidos deixando de ser Issues, nas verdades tem inúmeros motivos para fechar kkk.

Dicas do Guanabara: Sempre que tiver um problema, não crie diretamente uma Issue, procure no GitHub, dentro do projeto, se alguém teve a mesma questão, depois de procurar e confirmar que não vai criar uma “issue duplicada”, aí sim pode criar a sua e procurar por ajuda.

É ensinado a criar e reponder Issues, fechar Issues, criar e responder com imagens e adicionar alguns efeitos de formatação.

Coisa mais simples que podem ser feitas usando você mesmo a rede social, e prestando atenção a cada detalhe.

É possível criar assinatura padrão, criar uma mensagem que vai ser aplicada com poucos clicks ao invés de sempre ficar digitando a mesma coisa.

**Aula 10**

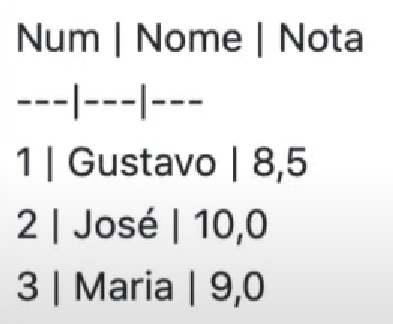
Ele vai falar sobre outra linguagem de marcação, já que não é só o HTML que existe.

É falado sobre o arquivo README, com ênfase em destacar a extensão “md”, que simboliza Markdown, o nome da outra linguagem de marcação, que é como se fosse o HTML só que mais simples.

A linguagem foi criada pra ser mais fácil de ser lida de ser compreendida, essa aula vai ser foca em entender a base da linguagem.

**Vamos para anotações sobre Markdown:**

Esses símbolos devem ser adicionados antes e depois do texto que vai ser destacado.

* (\*\* ou \_\_) negrito.
* (\* ou \_) itálico.
* (~~) Tachado.
* (#) Título 1.
* (##) título 2.
* (###) título 3
* (--- ou \*\*\*) Linhas na vertical.
* Lista numérica basta adicionar o número no início e (.) na frente.
* Lista demarcada basta colocar (\*) antes de iniciar a linha.
* Lista de tarefas (- []) para tarefas não feitas e (- [x]) para tarefas concluídas.
* Quando uma imagem é adicionada dentro da caixa de texto, tudo que estiver entre os colchetes é a descrição da imagem, então pode ser alterado.
* Para criar links funciona assim: [acesse meu GitHub] (<https://github.com/Luide-da-Paz>).
* É possível criar tabelas de acordo com a imagem ->
* Outra forma de destacar algo, dessa comandos é

com o uso de crases (‘‘), no início e no fim do comando.

* Lembrando que posso atribuir imagens, prints e outros

que podem ajudar.

* Para marcar pessoas que tem GitHub basta iniciar com @.

**Aula 11**

De início cita o curso de HTML5 e CSS3 que é o que pretendo ter foco total durante os próximos meses.

A aula vai ser redirecionada a segurança no GitHub.

**Dicas para ter senha forte:**

* Pelo menos 8 caracteres
* Letras e números
* Maiúsculas e minúsculas
* Símbolos
* Evitar nomes e palavras comuns
* Evitar padrões
* Nunca compartilhe uma senha

Ele explica sobre a autenticação de dois fatores e explica a importância de ter uma conta bem protegida, vou fazer a ativação da funcionalidade, mas é algo padrão que tem que ser feito em todo lugar e conta que você guarda algo de importância, inclusive depois dessa aula decidi fazer um repositório de logins, já que é muito difícil memorizar todas a senhas de tantas contas que se pode ter atualmente e claro que esse repositório tem que ser privado e criado com maior cuidado possível.

Importante listar que o guanabara não recomenda essa autenticação de por SMS por diversos motivos, não vou listar aqui esses motivos.

Pelas configurações do GitHub é possível visualizar se outras pessoas estão tentando fazer login de outro local na sua conta.

**Aula 12**

Branches ou Ramificações:

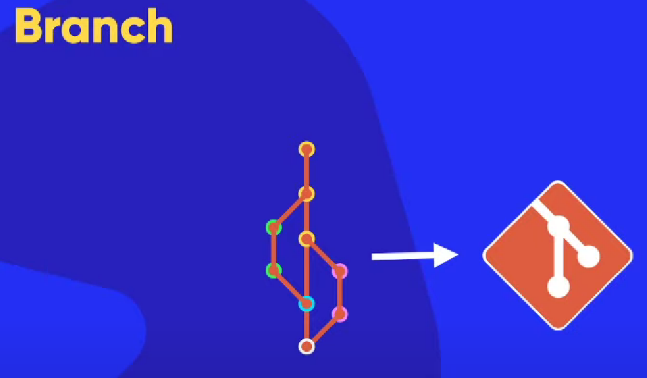
Branch -> Ramo -> arvore -> Ramo de Produção ou Ramo principal

Branch são ramificações, um exemplo: Alguém da sua equipe criou uma parte do projeto e você encontrou um erro, você vai corrigir e cria uma Branch, ou seja, uma ramificação daquele projeto, e como já falado e ainda não de forma especifica, como Git é possível ver todas as ramificação e alterações feitas em cada uma delas para facilitar a linha de projeto e por isso o git é uma ferramenta tão importante nesse ramo de programação.

Primeira versão criada de um software que você commit é chamada de Branch master, lembrando que Git é o versionamento local e GitHub é a forma de versionamento remoto, na nuvem ou como tenha entendido, é forma de compartilhar seu trabalho ou de guardar para acessar em outros dispositivos, cada vez que eu usar o commit eu crio uma versão nova, um versionamento novo, até o momento só tínhamos feito o commit no ramo mestre, ou seja em cima da primeira versão.

Ele pede que não façamos isso de forma profissional, por que se eu altero algo no meu projeto principal e faço uma “gagada” sem ver, o projeto acaba indo água abaixo, e aqui entra algo que eu queria entender desde o início, como criar as outras ramificações sem prejudicar o software passado.

O certo é cria uma nova Branch, um novo ramo, ir usando o commit nessa nova Branch, e quando finalizado se der tudo certo eu posso unir essa Branch que estava fora com a principal, o nome desse feito é merge, ele joga Branch no ramo principal, deu pra entender de forma teórica sim, quero pratica 🤯.

É possível que Branch criada não fique legal, e como a ramificação foi feita “universo paralelo” e se der falha, ignora. Caso haja conflitos o GitHub avisa, vou deixar o gráfico que o Guanabara criou para explicação de Branch, ramos:

Vamos para parte pratica.

As Branches podem ser criadas no canto superior do GitHub Desktop, assim que crio uma Branch posso definir de onde ela vai vir (ramo principal ou master e etc), e automaticamente essa Branch é adicionada de forma local, ou seja, não preciso fazer o commit dela porque o GitHub já fazer, e depois posso o push para o GitHub, para a área remota, ficou bem claro a explicação de Branch agora.

Quanto alterno entre as ramificações no GitHub Desktop, ele já faz as alterações diretamente no visual code, é possível unir essas ramificações? Talvez, não fazer ainda.

Falando sobre unir os conteúdos, no caso o merge, para fazê-lo é necessário estar na ramificação principal, master ou main.

Fica meio confuso ne entender por ter exemplos pratico de HTML e CSS e como não estudei ainda, o cérebro da bug, entendi como funciona, mais para ter 100% de compreensão desse conteúdo é necessário ter o básico de de html e css.

Basicamente, 1 ramificação principal, ela vai ser a base de todo o projeto, 1 alternância dela para conteúdo, vai servir para fazer a parte de HTML, e 1 alternância dela para Design que vai servir para parte de CSS, depois de concluir essas ramificações alternativas, é possível trazer para master/main, lembrando que essa transição de ramificação para a principal é chamada de merge, isso vai facilitar o trabalho, essa compreensão que tive até o momento.

**Aula 13**

Antes de prosseguir com a aula 13, decidi reler todas as anotações feitas das aulas anteriores, isso porque durante as aulas vi que existe muitos termos e comando com nomes em inglês que ainda não domino, vou deixar todos esses termos e comando anotados aqui, e explicados de acordo com minhas anotações:

* Git: software de versionamento Local.
* GitHub: software de versionamento remoto e rede social de programadores.
* Repositório: local onde é organizado todos os arquivos do projeto.
* README: arquivo gerado no repositório para personalizar o design e informações.
* Commit: aplica todas as alterações feitas no Git.
* Push: pega o commit e puxa para o GitHub.
* Pull: pega oque foi alterado de forma remota e traz para o repositório local.
* Clonar: traz os arquivos de um usuário publico para uma pasta local.
* GitIgnore: posso selecionar um arquivo para não fazer o commit ou push, bom ter em mente que ele procurar se existe outros arquivos com mesmo nome e extensão e manda embora também.
* Fokr: puxa o repositório de outra pessoa para o seu GitHub.
* Issues: é uma aba dentro do GitHub que serve para o público criar interações de um problema, você cria um para pedir ajuda em um problema especifico, ou entrar para procurar se alguém deve o mesmo problema e como solucioná-lo.
* MarkDown: linguagem de marcação usada dentro do Github.
* Branches: Ramificações que posso usar para trabalhar em equipe, ou criar versões diferentes de um arquivo.
* Merge: transição de ramificação para Branch mestre.

Indo para o conteúdo da Aula 13 agora: