Vamos focar no **Git**, um sistema de controle de versão distribuído (DVCS) que **permite que um time trabalhe em um mesmo projeto** ou em um mesmo arquivo e provê ferramentas para contornar a maioria dos problemas que podemos ter nesse tipo de situação.

Os principais pontos abordados serão:

* O que é o Git ?
* Qual a principal diferença para os outros VCS's ?
* Como configurar o Git na sua máquina?
* Como iniciar um repositório local?
* O que é o GitHub ?
* Como subir seu repositório local para o GitHub ?
* O que é Branch , Merge , Commit ?

Você será capaz de:

* Instalar e configurar o Git no seu computador;
* Salvar seus projetos no repositório Git local;
* Controlar as alterações e versões dos seus arquivos;
* Enviar seus projetos para o repositório remoto do GitHub .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CONCEITOS BÁSICOS**

Vídeo 01

O **GIT** é o mecanismo de controle de versão mais popular. Utilizando o Git , você e seu time vão conseguir **colaborar** de forma assíncrona e harmoniosa, ganhando produtividade no momento de **"juntar as partes"** que cada um fez e tendo segurança de que todo o trabalho realizado está salvo. Estar salvo depende de você sempre dar "push" (enviar as alterações para o repositório online) antes de sair com seu computador por aí, né?

O **GITHUB**, ferramenta online/plataforma que **hospeda repositórios Git**, é uma excelente ferramenta para toda a gestão do seu código, tendo recursos para acompanhamento de projeto ágil de software, controle de Issues , quadros Kanban, Pull Requests que permitem um Code Review mais bem feito, entre outras dezenas de integrações que vão facilitar sua vida com desenvolvimento de software.

**Mas o que é mesmo esse Controle de Versão?**

Um sistema de controle de versão (como o próprio nome já diz) tem a finalidade de **gerenciar diferentes versões de um documento**. Com isso ele te oferece uma maneira muito mais inteligente e eficaz de organizar seu projeto, pois é possível acompanhar um **histórico** de desenvolvimento (em commits), **desenvolver paralelamente** e ainda te oferecer outras vantagens. Ou seja, todas as alterações no arquivo estão registradas.

**Como funciona um Controle de Versão?**

Basicamente, **os arquivos** do projeto ficam **armazenados em um repositório** (um servidor em outras palavras) e o histórico de suas versões é salvo nele. Os desenvolvedores podem acessar e resgatar a última versão disponível e fazer uma cópia local, na qual poderão trabalhar em cima dela e continuar o processo de desenvolvimento. A cada alteração feita, é possível enviar novamente ao servidor e atualizar a sua versão a partir outras feitas pelos demais desenvolvedores.

**E se por acaso os desenvolvedores estiverem editando o mesmo arquivo?**

Para evitar problemas de sobreposição, o Sistema de Controle de Versão oferece ferramentas úteis para mesclar o código e evitar conflitos.

Atualmente, os sistemas de controle de versão são classificados **em dois tipos**:

* O **centralizado** trabalha apenas com um servidor central e diversas áreas de trabalho, baseados na arquitetura cliente-servidor.
* O **distribuído** vai mais além. Ele é recomendado para equipes com muitos desenvolvedores e que se encontram em diferentes filiais. Esta versão funciona da seguinte maneira: cada área de trabalho tem seu próprio “servidor”

A **comunicação** entre o servidor principal e as áreas de trabalho funciona com outras **duas operações**, para atualizar e mesclar o projeto, chamadas de **pull** e **push** (puxar e empurrar).

* **PULL:** com esta operação, é possível pegar a versão de outra área de trabalho e mesclar com a sua.
* **PUSH:** com esta operação, temos o processo inverso do pull, ou seja, enviando para outra área a sua versão do projeto.

Vídeo 02

Agora, vamos a **três conceitos importantíssimos** para o seu trabalho:

* **Ramificação** (Branching);
* **Mesclagem** (Merge);
* **Resolução** de conflitos.

**CONCEITOS GERAIS**

* **Branches:** versão independente e editável que se cria de um código
* **Branch Master:** é a ramificação principal do código, onde todo ele é consolidado.
* **Merge:** é quando passamos alterações de uma branch para outra. Mesclar.

**Autenticação - Como funciona a conexão entre local (GIT) e remoto (GITHUB)?**

É aí que entra o processo de **autenticação** do GitHub, que lhe permitirá, ao mesmo tempo, **proteger** suas informações pessoais e **mandar** comandos para o GitHub diretamente pelo seu terminal!

Quando esse processo é feito, você informa ao sistema remoto que é para utilizar as credenciais da sua conta ao executar algum comando do git e, ao mesmo tempo, comprova para o GitHub que você é exatamente quem diz que é.

Existem **duas formas** pelas quais você pode **acessar o GitHub pelo terminal**:

* **HTTPS** (Hypertext Transfer Protocol Secure)
* **SSH** (Secure Shell).

Ambas as formas são válidas, mas possuem algumas diferenças entre si:

**CONCEITOS SSH e HTTPS**

* **SSH ou Secure Shell:** é um protocolo de **criptografia de rede** que serve para transferir dados de forma segura mesmo em redes inseguras. Usando o protocolo SSH, você pode se conectar ao GitHub sem precisar digitar seu nome e chave de acesso pessoal a cada comando executado.
* **HTTPS ou Hypertext Transfer Protocol Secure**: é uma extensão do protocolo de **internet** HTTP que **utiliza certificados digitais para autenticar os dados** e permitir que eles sejam criptografados de forma segura.

Como o SSH nos permite pular a etapa de digitar login e senha do GitHub a cada comando, iremos utilizar o SSH como modelo padrão de autenticação.

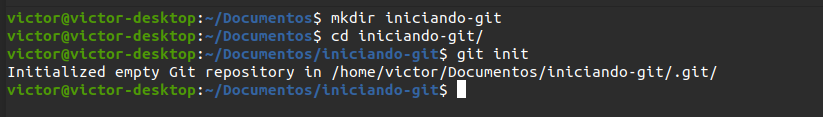
**Do *git init* ao *git push***

Agora você vai aprender a iniciar um repositório Git , fazer seu primeiro commit (registro) e subi-lo para um repositório no GitHub .

1. **CRIANDO UM REPOSITÓRIO LOCAL**

Antes de se criar um repositório é preciso criar uma pasta para ele. Para isso você deve utilizar o comando ***mkdir.***

**Para criar** um repositório você deve digitar o **comando *git init***. É muito importante que esteja dentro da pasta criada para o repositório.

Como seu nome bem diz, esse comando é responsável por **iniciar** um repositório Git dentro da pasta em que foi executado.

**Para verificar se um repositório Git foi de fato iniciado**, você pode executar o **comando *git status***, que retorna o status do repositório.

No contexto de um repositório recém criado onde nenhuma modificação foi feita você receberia a seguinte resposta:

*No ramo master*

*No commits yet*

*nada para enviar (crie/copie arquivos e use "git add" para registrar)*

Vamos Praticar

Para que possamos versionar alterações feitas no código é preciso sempre seguir a seguinte sequência:

* Adicionar ( ***add*** )
* Comitar ( ***commit -m*** "mensagem" )

Os comandos ficam da seguinte forma:

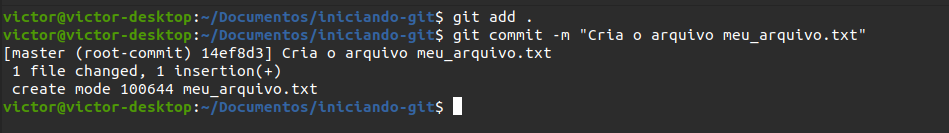
# Adicionar todos os arquivos modificados: ***git add***

# Ou você também pode adicionar arquivos específicos: ***git add meu\_arquivo.txt***

# Então você comita a alteração: ***git commit -m "Mensagem sobre a alteração feita"***

OBS: ao comitar, você deve adicionar também uma mensagem que descreve o que aquela alteração faz

Após adicionar e fazer um commit de suas alterações, você receberá no terminal algo parecido com a próxima imagem:



Criando um repositório no GitHub

PASSO 1: Acesse o GitHub e procure o ícone com um ***sinal +*** na barra superior e, ao clicar nele, busque pela **opção *New repository***.

PASSO 2: Você deve então **adicionar um nome** ao seu repositório, como por exemplo *meu-super-repo* . Após fazer isso, o botão Create repository será habilitado e ao clicar nele seu repositório será criado.

PASSO 3: Como seu repositório estará vazio, o GitHub lhe dará algumas dicas. Você deverá **clicar no botão *SSH*** e então **copiar a URL** gerada. Você irá precisar dela para realizar a conexão entre seu repositório local e seu repositório remoto.

Link gerado: [git@github.com](mailto:git@github.com):VitBrandao/meu-super-repo-.git

PASSO 4: Para conectar os dois repositórios você deverá abrir o seu terminal, acessar o diretório do seu repositório e então executar o seguinte comando:

*git remote add origin* [*git@github.com*](mailto:git@github.com)*:user-github/repo-name.git*

No caso, com meu link, fica assim:

***git remote add origin git@github.com:VitBrandao/meu-super-repo-.git***

OBS: a palavra “origin” é só um apelido, poderia ser qualquer outro.

PASSO 5: Para verificar que tudo funcionou corretamente, execute o comando

***git remote -v*** . O resultado foi:

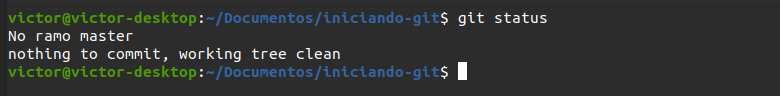
*origin git@github.com:VitBrandao/meu-super-repo-.git (fetch)*

*origin git@github.com:VitBrandao/meu-super-repo-.git (push)*

Sincronizando os repositórios

Os repositórios já estão criados e também já estão conectados, agora é a hora de **enviar as alterações** feitas no repositório **local** para o repositório **remoto**.

Para isso, certifique-se que as alterações já foram adicionadas e comitada utilizando o comando ***git status*** , caso esteja tudo certo, você receberá uma mensagem no terminal parecida com a imagem a seguir:



OBS: Caso tenha alguma alteração que não foi comitada, faça o commit para depois enviar as alterações.

Com tudo comitado é hora de enviar essas alterações para o repositório remoto utilizando o seguinte comando:

***git push origin master***

Com isso, você está **enviando as alterações feitas localmente para a branch principal**, master , do seu repositório remoto, origin .

Se tudo ocorreu conforme o esperado, acesse novamente, ou atualize, a página do seu repositório no GitHub , você então verá que os arquivos e alterações que comitou agora se encontram lá!

**Configuração**

O Git vem com uma ferramenta chamada ***git config*** que permite ver e atribuir variáveis de configuração que controlam todos os aspectos de como o Git aparece e opera.

O comando *git config --global core.editor "code --wait"* torna o **VS Code** o editor principal.

Se digitar *code .gitconfig* o VS Code abre.

Aula ao vivo - Resumo dos comandos

- Criando uma brench (registro)

***git branch form***

Sendo "form" o nome da branch.

- Alternar/Viajar de uma branch para outra

***git checkout form***

- Criando/Registrando um novo commit

***git commit -m "Adiciona imagem do formulário"***

o commit é como bater uma foto/registrar. para garantir, o ideal é **comitar depois de qualquer alteração.**

O commit é como uma “caixinha” onde colocamos coisas. Depois de fechar, registramos.

A criação de uma nova commit não altera as outras.

- Excluindo uma branch

***git branch -d form***

- Unindo duas branches. Vá até uma delas e digite:

***git merge form***

Sendo "form" o nome da segunda a ser mesclada