* **O que é controle de versão?**

Pense na seguinte situação...

Estamos programando, começamos a salvar arquivos em algum local/diretório, entretanto durante o salvamento desses arquivos pode acontecer de salvar em cima de arquivos que você tenha feita anteriormente e você acaba perdendo alguns arquivos por acidente



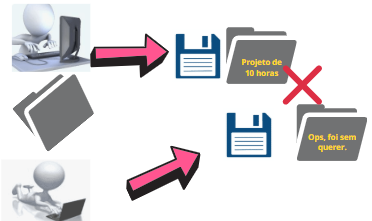
Ou por medo de perder esses arquivos você duplica esses arquivos e fica com versões diferentes do mesmo arquivo gerando uma bagunça enorme de arquivos.



Ou ainda….

Você está desenvolvendo um programa e precisa compartilhar com um outro programador o arquivo que você está desenvolvendo, como resolveria isso? A primeira opção seria disponibilizar em uma pasta compartilhada.

Agora imagina a situação, ambos estão trabalhando em um arquivo, você edita e salva o outro programador também edita e salva por cima jogando fora o trabalho antes salvo por você.



Por essa razão a importância do software de versionamento (VCS), a ideia de controle de versão é salvar essas alterações de arquivos, o sistema salva versões editadas do arquivo e se você precisar pode recuperar alguma versão anterior do trabalho, o seu backup. O software de controle de versão mantém registro de todas as modificações no código em uma base de dados e permite ainda essa interação com outros desenvolvedores, é uma foto de determinado estado do projeto.

Essa parte ainda estou montando, a intenção e pesquisar os softwares de cada tipo

|  |  |
| --- | --- |
| VCS Centralizado ou Linear | VCS Distribuído |
|  | Quando commitar não preciso estar conectada ao repositório central, vou no repositório local através do software de versionamento intalado em meu computador, ou seja um versionamento local . Quando as informações precisam commitar são enviadas através de push para meu repositório remoto. |

>>>>>>

*commit*,: cada alteração salva é chamada de *commit*, por exemplo, a frase "preciso *commitar* as alterações" significa mais ou menos "salvar as alterações", porém *commitar* não é exatamente salvar, pois o arquivo já está salvo localmente na memória da máquina, o *commit* insere apenas aquele estágio do arquivo na estrutura de versionamento do git.

COMMIT -- encerra a transação SALVANDO permanentemente todas as alterações (updates,deletes,inserts) realizadas durante a transação.

>>>>>

**Vantagens**

Controle de histórico

Quem fez e qual foi a alteração

Trabalho em equipe

Ramificação do projeto( front-end, desing, back-end)

Segurança, cada um só pode ver o que foi designado a ele

Organização dos arquivos, economia de espaço e trabalho

* O que é o Git?



Software de controle de versão desenvolvido por Linux Torvalds devido ao *ódio com paixão* (palavras do próprio Torvalds) por ele adquirido pelo CVS.

Sua principal funcionalidade e para o desenvolvimento de softwares, mas pode ser usado para qualquer tipo de arquivo. Inicialmente Torvalds o criou para o desenvolvimento do Kernel Linux porém acabou tendo aplicações em diversos projetos.

Trata-se de um software livre

Ele trabalha da seguinte forma: supondo que estamos desenvolvendo projeto integrador em HTML e já digitamos 300 linhas desse projeto e commitamos, fiz mais 100 linhas de HTML, ele não irá salvar 300 + 100 linhas, irá salvar comente as 100 novas linhas digitadas ou seja a diferença de uma arquivo par o outro. Se em algum momento descobri que fiz besteira e não deveria ter digitado aquelaa 100 linhas posso voltar o meu projeto para as 300 linhas.

**Curiosidade:**

Quando questionado sobre o nome dado ao software Torvalds respondeu:

“Eu sou um desgraçado egocêntrico, então batizo todos os meus projetos com meu nome. Primeiro [Linux](https://pt.wikipedia.org/wiki/Linux), agora Git.”

Trata-se de uma gíria em [inglês britânico](https://pt.wikipedia.org/wiki/Ingl%C3%AAs_brit%C3%A2nico) para *cabeça dura.*

Fonte: <https://pt.wikipedia.org/wiki/Git>

* O que é GitHub?



É um local para guardar seus arquivos, semelhante uma plataforma de redes sociais como insta ou facebook, lá você pode guardar seus projetos e disponibilizar posteriormente seus códigos, é uma rede social de programadores, você pode colaborar, testar os códigos dos outros entre outros, é uma comunidade de programadores com belíssima interface gráfica.

* O que é GitLab?



GitLab é similar ao GitHub, mas o GitLab permite que os desenvolvedores armazenem o código em seus próprios servidores, ao invés de servidores de terceiros além de ser opensource.

Você pode baixar e instalar o GitLab em seu servidor (posso manter meus repositórios gits centralizados em minha rede privada), na sua empresa onde todos da empresa podem utilizam um interface gráfica centralizada .

>>>>>

**Repositório**, ou repo, é um diretório onde os arquivos do seu projeto ficam armazenados

>>>>>

* Explicação de como criar uma conta em um dos serviços.
* **Como é a instalação do git no Linux.**

Fonte: <https://www.hostinger.com.br/tutoriais/tutorial-do-git-basics-introducao/>

Para usuários Debian/Ubuntu (apt-get):

1. Abra o terminal e execute os seguintes comandos:
2. Sudo apt-get update

Sudo apt-get install git

1. Verifique se a instalação ocorreu com sucesso usando git --version.
2. Execute os seguintes comandos no terminal para configurar seu e-mail e nome de usuário que serão associados à sua conta GIT:
3. git config --global user.name "João Silva"

git config --global user.email "exemplo@seuemail.com.br"

Nota: Lembre-se de substituir **João Silva** e **exemplo@seuemail.com.br** com seus dados. Qualquer *commit* criado posteriormente será associado à esses dados.

**Fedora (yum/dnf):**

Você pode baixar pacotes do GIT usando yum e dnf.

1. Abra o terminal e execute os seguintes comandos:
2. Sudo dnf install git

Sudo yum install git

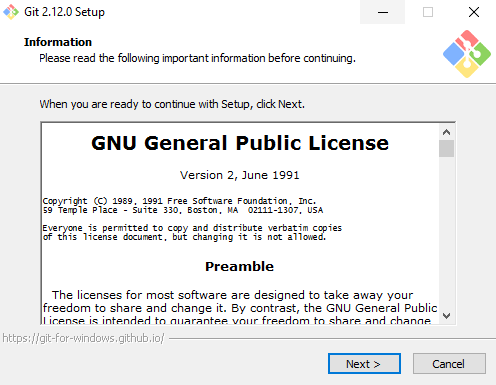
1. Verifique se a instalação ocorreu com sucesso usando git --version.
2. Execute os seguintes comandos no terminal para configurar seu e-mail e nome de usuário que serão associados à sua conta GIT:
3. git config --global user.name "João Silva"

git config --global user.email "exemplo@seuemail.com.br"

Nota: Lembre-se de substituir **João Silva** e **exemplo@seuemail.com.br** com seus dados. Qualquer *commit* criado posteriormente será associado à esses dados.

* **Como é a instalação do git no Windows.**

Fonte: <https://www.hostinger.com.br/tutoriais/tutorial-do-git-basics-introducao/>

1. Acesse [**o site oficial**](https://git-for-windows.github.io/) e faça o download do instalador do GIT para Windows.
2. Depois de baixado, clique duas vezes no arquivo para iniciar o assistente de instalação. Basta seguir as instruções na tela, clicando em **Next**. Ao término, clique em **Finish** para concluir com êxito a instalação.  
   
3. Abra o prompt de comando e digite os seguintes comandos no terminal:

git config --global user.name "João Silva"

git config --global user.email "exemplo@seuemail.com.br"

Nota: Lembre de substituir **João Silva** e **exemplo@seuemail.com.br** com seus dados. Qualquer commit criado posteriormente será associado à esses dados.

### ****Criar/configurar/verificar um repositório****

Um repositório é o maior bem de qualquer projeto controlado por versão. Para transformar qualquer diretório em um repositório GIT, o simples comando git init <directory> pode ser utilizado. Uma pasta chamada .git também deve começar a existir no diretório em que o comando foi executado.

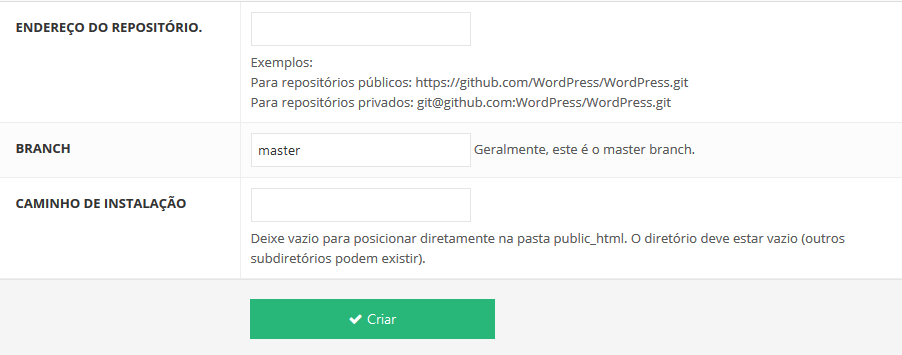
Por outro lado, se você já tem um diretório e deseja verificar (clone-lo), você pode usar o comando git clone. Se você estiver tentando verificar um repositório local, use o seguinte comando:

git clone /path/to/local/repository

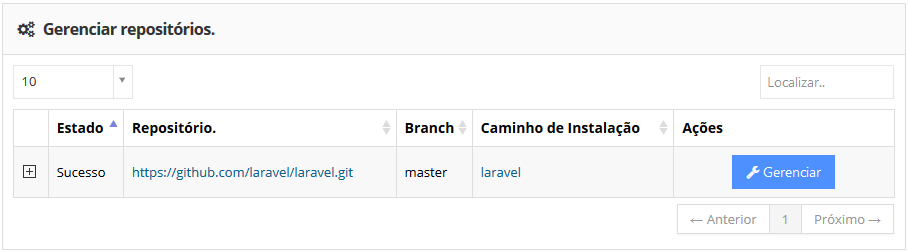
Se você pretende verificar um repositório armazenado remotamente, use:

git clone user.name@host:/path/to/remote/repository

Se você tem uma conta na Hostinger, você pode facilmente clonar e gerenciar repositórios via **Painel de Controle**>**GIT**. Por exemplo, se você quer clonar um repositório GIT, basta digitar seu endereço, escolher um ramo e instalar o caminho, e clicar no botão criar.



Após a conclusão da criação, você poderá gerenciar seu repositório na mesma seção.



## **Fluxo de trabalho**

Agora que o repositório está pronto, vamos falar sobre a estrutura que é mantida pelo GIT. Cada repositório local consiste em três árvores: o **diretório de trabalho** que contém os arquivos reais; O **índice** que desempenha o papel de uma área de teste e o **HEAD** que é um ponteiro para o último commit feito pelo usuário. Então, é assim que o fluxo de trabalho pode ser explicado: o usuário adiciona um arquivo ou alterações do diretório de trabalho para o índice (a área de teste) e uma vez revistos, o arquivo ou as alterações são finalmente comprometidos com o **HEAD**.

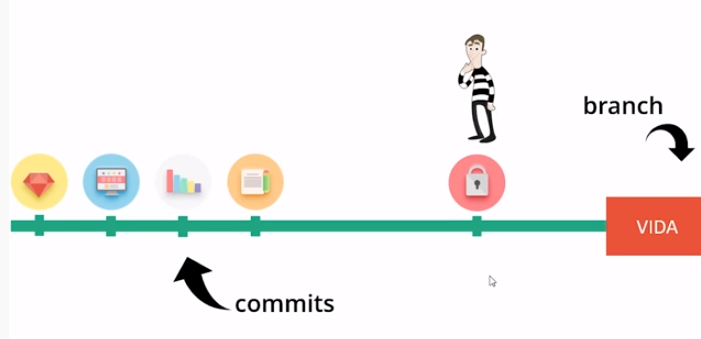
* Quais os principais comandos do git?
  + O que é git pull?
  + O que são branches?

Para explicação do branches me apropriei da ideia de:

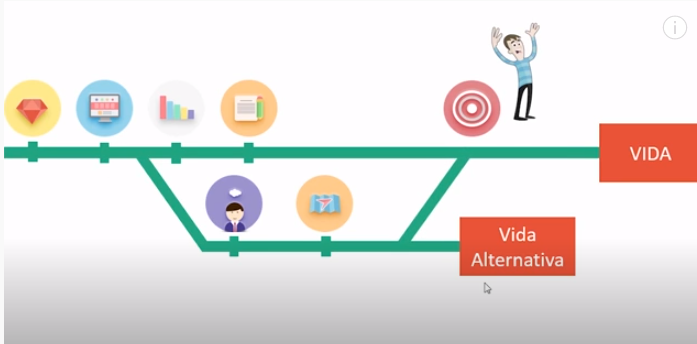
Fonte: <https://www.youtube.com/watch?v=LJqQ_U4sq7Y>

Veja o exemplo do nosso amigo abaixo e sua linha do tempo. Cada commits registra uma parte da sua linha do tempo, cada evento e uma foto de um determinado estado de sua vida.

Essa linha do tempo é chamada de branch, que significa ramo ou braço.

>>>>

Não seria nada mal se nossa vida também tivesse um git pois poderíamos criar uma branch alternativas com novas commits mais satisfatórias (dando um jeito de aquelas coisas que a gente nunca deveria ter feito).



>>>>>

Ramificação, no controle de versão e no gerenciamento de configuração de software, é a duplicação de um objeto sob controle de versão, para que modificações possam ocorrer em paralelo ao longo de várias ramificações

>>>>

### Como criar um branch local

### Fonte: <https://receitasdecodigo.com.br/devops/git-o-que-e-um-branch>

Estando na pasta do projeto (master), podemos criar um novo branch local usando o comando:

git branch branch-001

### Acessando o branch criado

Após criar o branch é preciso fazer o checkout para acessar o branch:

git checkout branch-001

Ou ainda pode criar um novo *branch* e acessar usando o comando:

git checkout -b branch-001

Para verificar o novo *branch* digite:

git status

* Como arrumar uma mensagem errada em um commit

.

* Explicação sobre como um grupo de faculdade pode usar git para colaborar em um projeto

O exemplo é essa atividade que desenvolvemos, o mesmo pode ser salvo no github e cada pessoa pode entrar no projeto e colaborar com partes do mesmo, no HTML por exemplo cada um do grupo pode ficar responsável por uma parte, um aluno responsável pelo css e outro pelo código em html, podemos até recebe colaboração de outros grupos em nossa atividade, ou disponibilizar os códigos para o professor.