**Estudo Autodidata Git e GitHub**



Aula: 1

Tema: O que é Git? O que é versionamento?

**Git e github, pra que serve?**

Git ≠ GitHub

Git: Git é um software de controle de versão  
Github: Plataforma de Rede Social para Programadores

**O que é Git?**

O git é um VCS (Software de controle de versão), o git é uma ferramenta de versionamento de software, versionamento de código.   
A palavre “VERSÃO” se baseia na ideia de que um programador trabalha em MUITAS VERSÕES da mesma coisa.



Com o pedido do desenvolvimento de um site você começa o projeto em uma pasta (site- cliente), com o decorre do projeto você vai criando outros arquivos com base no arquivo original, e isso vai cirando novas VERSÕES do mesmo projeto.

Seguindo com o projeto você vai querer fazer um backup (de preferencia no drive por ser mais seguro e não ter rico de corromper como em um pen drive ou em um HD).

Só que essa metodologia pode gerar problemas e complicações, por exemplo se o projeto é feito de maneira grupal. Imagina um grupo de pessoas salvando várias e várias versões do mesmo site em vários arquivos .zip diferente.



O que seria preciso para solucionar esse problema? Um VCS (um software de controle de versão), vulgo Git.

O versionamento vai atualizar seu código com o decorrer do projeto, como uma linha do tempo que deixa você gestionar as atualizações recentes e passadas do seu código.

**Tipos de sistemas de controle de versão**

**Versionamento Centralizado/Linear**

Nos sistemas de controle de versão centralizados ou lineares, há um único repositório central onde todas as versões do projeto são armazenadas. Os usuários precisam se conectar a esse repositório central para realizar operações como commit, update ou checkout. Isso cria um "ponto único de verdade", mas também pode causar gargalos, especialmente se o repositório estiver inacessível.

Exemplos de VCS Centralizados:

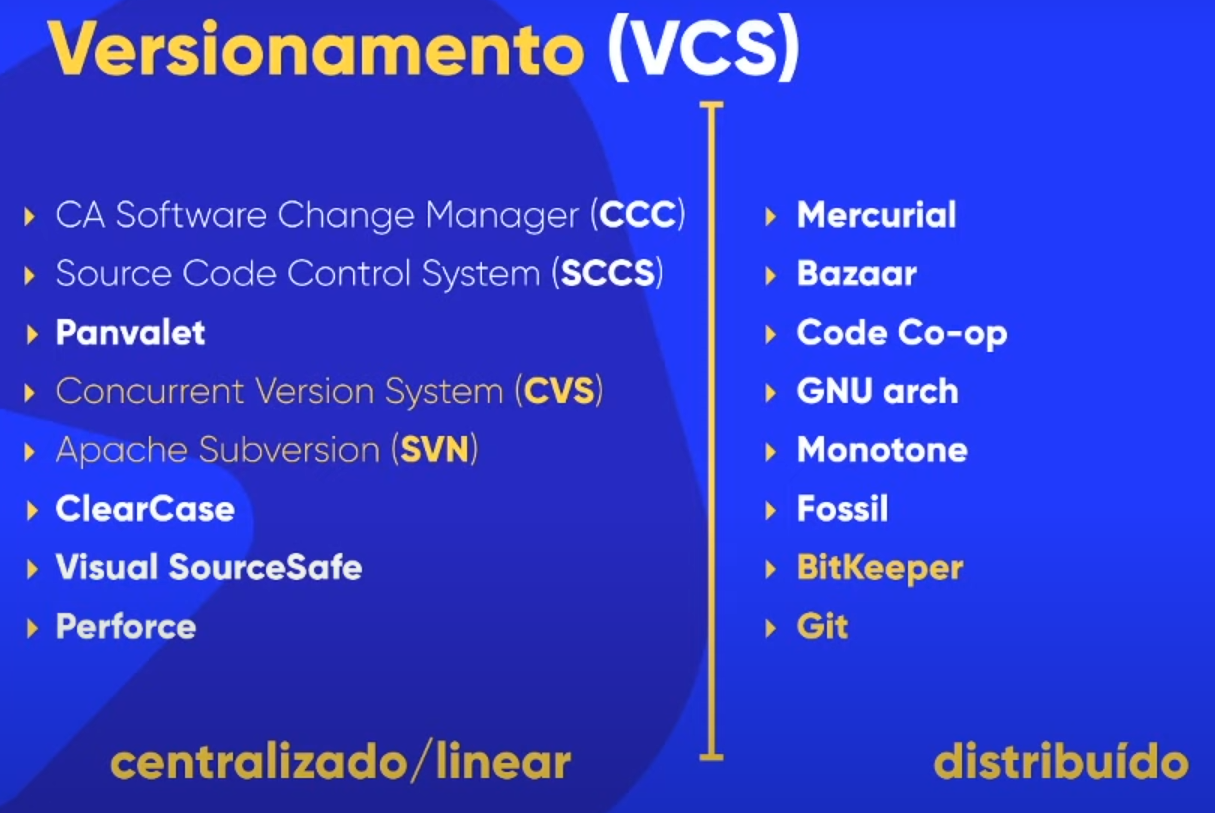
* CA Software Change Manager (CCC): Ferramenta robusta usada por grandes empresas para gerenciar mudanças em softwares complexos. É centralizada e permite controle rigoroso sobre o fluxo de trabalho.
* Source Code Control System (SCCS): Um dos sistemas mais antigos, usados para controlar versões de código-fonte. Ele é linear e mantém um histórico sequencial das mudanças.
* Panvalet: Focado em gerenciar fontes de aplicativos e dados, muito usado em ambientes de mainframe.
* Concurrent Version System (CVS): Um sistema popular nos anos 90, que permitia múltiplos desenvolvedores trabalharem ao mesmo tempo em diferentes partes de um projeto, mas com um único repositório central.
* Apache Subversion (SVN): Uma evolução do CVS, muito popular ainda hoje, que melhora o controle de branches e tags. Subversion também é centralizado e requer que os desenvolvedores enviem suas alterações a um repositório central.
* ClearCase: Uma ferramenta da IBM usada em ambientes corporativos, oferecendo controle de versões e gestão de mudanças com um modelo centralizado.
* Visual SourceSafe: Ferramenta da Microsoft que oferecia um VCS simples para pequenos times de desenvolvimento.
* Perforce: Um VCS centralizado, mas bastante usado por equipes grandes devido à sua performance em projetos de larga escala.

**Versionamento Distribuído**

Nos sistemas de controle de versão distribuídos, cada desenvolvedor tem uma cópia completa do repositório em seu computador. Isso significa que os desenvolvedores podem trabalhar offline e sincronizar as alterações com os demais quando estiverem conectados. Esses sistemas oferecem mais flexibilidade e permitem múltiplos fluxos de trabalho simultâneos.

Exemplos de VCS Distribuídos:

* Mercurial: Sistema de controle de versão distribuído, fácil de usar e com performance elevada, ideal para projetos grandes. Usa um modelo peer-to-peer, onde cada desenvolvedor tem um repositório completo.
* Bazaar: Similar ao Mercurial, foca em simplicidade e flexibilidade, permitindo tanto modelos centralizados quanto distribuídos.
* Code Co-op: Um sistema distribuído que pode sincronizar repositórios por e-mail ou outros métodos, útil em ambientes com conectividade limitada.
* GNU arch: Ferramenta livre e distribuída que foi usada para gerenciar projetos de software livre, mas perdeu popularidade com o tempo.
* Monotone: Outro sistema de controle distribuído, focado em segurança e integridade dos dados.
* Fossil: Um sistema distribuído que inclui não apenas controle de versão, mas também um bug tracker e wiki integrado.
* BitKeeper: Um dos primeiros VCS distribuídos e usado por grandes projetos, como o kernel do Linux, antes do surgimento do Git.
* Git: O mais popular entre os VCS distribuídos. Criado por Linus Torvalds para gerenciar o desenvolvimento do kernel Linux, Git oferece um fluxo de trabalho descentralizado, permitindo que desenvolvedores trabalhem de forma independente e colaborem com facilidade.



Principais Diferenças:

Centralizado: Um único repositório central. Fácil de gerenciar em equipes pequenas, mas pode ser limitado em termos de escalabilidade e flexibilidade.

Distribuído: Cada desenvolvedor tem uma cópia completa do projeto. Permite um fluxo de trabalho mais descentralizado, ideal para equipes distribuídas ou grandes projetos.

Git é um dos sistemas distribuídos mais usados atualmente, devido à sua flexibilidade, velocidade e suporte a fluxos de trabalho variados. Subversion (SVN), por outro lado, ainda é bastante usado em ambientes que preferem um sistema mais centralizado.

**Principais vantagens de utilizar um sistema de controle de versão:**1. Controle de histórico: Analisar o código em versões passadas e como ele foi evoluindo e até mesmo voltar a versões anteriores.  
2. Trabalho em equipe e Ramificação do projeto: Um tipo de separar as áreas que tem que ser trabalhada no programa. Como uma ramificação para o front end, outra para o back end, uma para o banco de dados, outra para o designer. Para que no final seja possível juntar essas ramificações e ter o projeto final, isso tudo com cada integrante do projeto trabalhando de maneira separada.

3. Segurança: As ramificações proporcionam segurança entre as áreas de trabalho, uma pessoa trabalhando no back não pode alterar o código do front por exemplo.  
4. Organização: Por trabalhar com controle de histórico e ramificação, o git proporciona um trabalho organizado que gera um controle muito grande, uma economia de espaço e uma economia de trabalho.