Google olabi

<SISTEMA.DE CONTROLE.DE VERSÃO>

{ Para que serve um Sistema de Controle de Versão?

O Sistema de Controle de Versão (ou **VCS**, sigla de *Version Control System* em inglês) é um método ou categoria de ferramentas de *software* fundamental para o desenvolvimento dele, uma vez que ajudam no acompanhamento de alterações no código. Se algo der errado, é possível comparar diferentes versões de código e reverter facilmente para versões anteriores. O **VCS** fornece maior flexibilidade e controle em relação a outras soluções.

Se duas pessoas trabalharem em um arquivo ao mesmo tempo e decidirem mesclá-lo, em caso de conflitos entre as duas versões, o **VCS** permite identificar esses conflitos e tomar uma decisão sobre a melhor forma de mesclar essas diferentes versões em um terceiro documento. Nessa abordagem, você também mantém um histórico da versão anterior, caso deseje voltar para uma delas.

{ Vantagens

Esse sistema permite ainda que pessoas desenvolvedoras, designers e analistas de qualidade (dentre outros perfis) trabalhem em um mesmo projeto. É essencial para garantir que todos tenham acesso ao código mais recente, uma vez que, à medida que o desenvolvimento fica mais complexo, há uma necessidade maior de gerenciamento de várias versões de produtos inteiros.

Em linhas gerais, o **VCS** permite ao usuário:

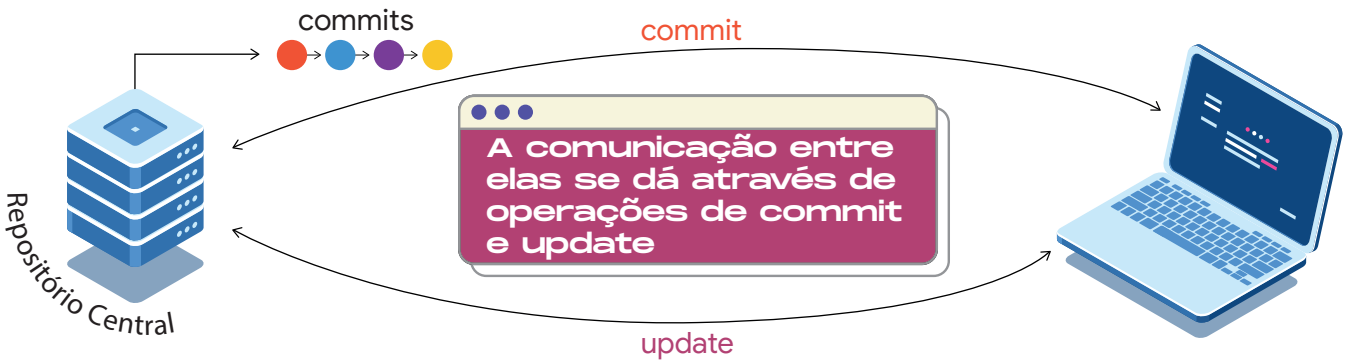
- Acompanhar desenvolvimentos e mudanças em seus arquivos
- Registrar alterações feitas em seu arquivo de uma forma que você consiga entender posteriormente
- Explorar diferentes versões de um arquivo, mantendo a versão original
- Mesclar duas versões de um arquivo e gerenciar conflitos entre elas
- Reverter alterações através do histórico de versões anteriores do arquivo

{ Como funciona o Controle de Versão?

O Controle de Versão é composto por duas partes: Repositório e área de trabalho. O repositório armazena todo o histórico de evolução do projeto registrando toda e qualquer alteração feita em cada item versionado.

O desenvolvedor não trabalha diretamente nos arquivos do Repositório. Ao invés disso, ele usa uma área de trabalho contendo a cópia dos arquivos do projeto monitorada para identificar mudanças realizadas. Essa área é individual e isolada das demais áreas de trabalho.

Controle de Versão



A comunicação e sincronização entre a área de trabalho e repositório é feita através dos comandos *commit* e *update*. O *commit* envia um pacote contendo uma ou mais modificações feitas na área de trabalho (origem) para o repositório (destino). O *update* faz o inverso: envia as modificações contidas no repositório (origem) para a área de trabalho (destino).

Cada *commit* gera uma nova revisão no repositório contendo modificações feitas, data e autor. Uma revisão funciona como uma "foto" de todos os arquivos e diretórios durante determinado momento de evolução do projeto. As "fotos" antigas são mantidas para recuperação e análise sempre que desejado. O conjunto de revisões é o histórico do projeto.

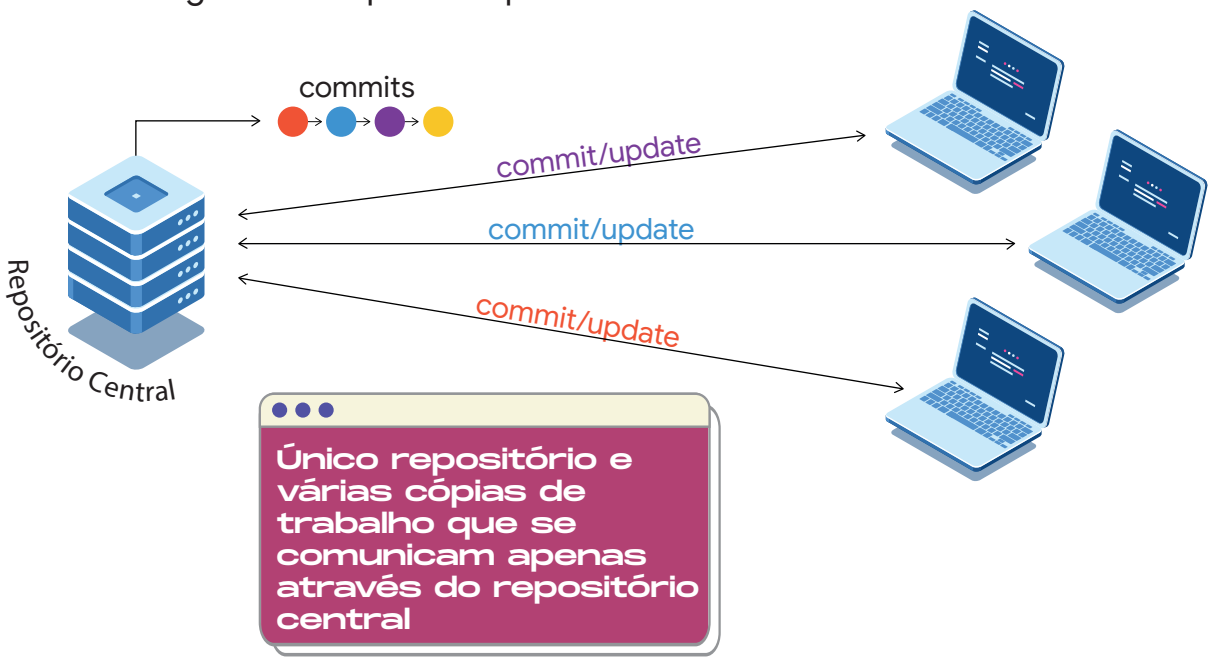
Tanto o controle de versão centralizado quanto distribuído possuem repositórios e áreas de trabalho próprias. A diferença reside em como cada uma dessas partes está interligada.

{ Tipos de Controle de Versão

<Controle de Versão Centralizado (CVCS)>

No Controle de Versão Centralizado (**CVCS**), há um único repositório e várias cópias de trabalho, que se comunicam apenas através do repositório central. O sistema funciona no modelo cliente-servidor e contém todas as informações transferidas para o cliente.

É uma espécie de repositório compartilhado, que fornece o código mais recente para os desenvolvedores. Dessa forma, eles trabalharão sempre na cópia principal. Basta puxar a última cópia do código, trabalhar nela, submeter as alterações e enviar o código de volta para o repositório.



A principal desvantagem do **CVCS** é ser um ponto único de falha. Se o servidor central cair, você não poderá usá-lo; se estiver fazendo um *commit* remoto, também levará algum tempo e precisará de conectividade com a Internet para confirmar quaisquer alterações.

{ Exemplos

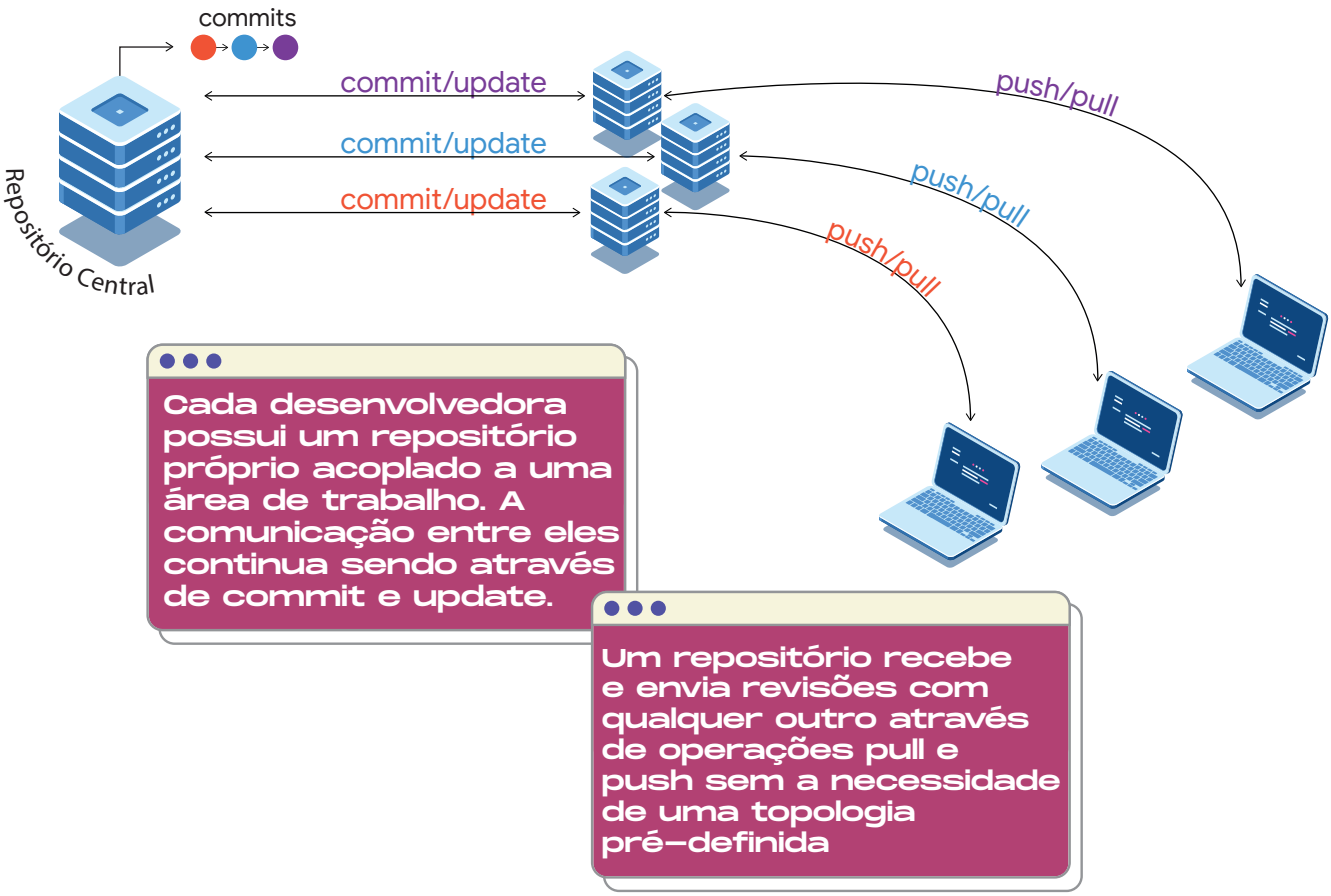
Código aberto: Concurrent Versions System (**CVS**), Subversion (**SVN**)

Código fechado: TeamCity, Vault, IBM Configuration Management Version Control (**CMVC**)

<Controle de Versão Distribuído (DVCS)>

No Controle de Versão Distribuído (**DVCS**), cada desenvolvedor possui um repositório próprio acoplado a uma área de trabalho. A comunicação entre estes continua sendo através de *commit* e *update*.

O sistema de **DVCS** possui repositório centralizado, da mesma forma que todos os desenvolvedores têm uma cópia local do repositório.



Pessoas desenvolvedoras podem trabalhar em cópias locais simultaneamente, uma vez que estas não exigem conectividade com a Internet para trabalhar no código. É possível fazer tudo no código exceto *push* e *pull*. Se o servidor central cair, também não haverá impacto, devido ao repositório local.

{ Exemplos

Código aberto: Git, Mercurial, Bazaar

Código fechado: Visual Studio Team Services, Plastic SCM

{ Diferenças entre arquiteturas SVN (VCS centralizado) e GIT (VCS distribuído)

Na arquitetura de Subversion (VCS centralizado), temos apenas um repositório central, enquanto no GIT (VCS distribuído), temos um repositório central e um local.

Benefícios de sistemas distribuídos:

• **Segurança:** *Softwares* de controle de versão de mecanismos para evitar possíveis corrupções em arquivos. Apenas pessoas autorizadas e identificadas podem mexer no código fonte controlado.

• **Versionamento:** Caso deseje voltar para versão de um determinado arquivo (devido a algum erro cometido ou simplesmente à mudança de escopo), é possível fazê-lo de forma simples e estruturada, de forma a minimizar eventuais erros e efeitos colaterais.

• **Rastreabilidade:** Quando se trata de algo importante, é sempre interessante saber o "quem", "quando", "como", "onde" e o "por quê". Todos esses metadados estão disponíveis nas ferramentas mais populares de controle de versão.

• **Organização:** Os sistemas com interface visual possuem visualização completa do ciclo de vida de cada arquivo controlado, desde a criação até o momento atual.

• **Colaboração:** O trabalho em equipe, principalmente em times distribuídos, é muito facilitado. Pessoas que talvez nem se conheçam conseguem colaborar em um determinado projeto cujo repositório central é disponibilizado a todos os envolvidos.

• **Confiança:** O uso de repositórios remotos ajuda muito na recuperação de eventos imponderáveis. Riscos como "perdemos o projeto inteiro na máquina de fulano" são minimizados. Também é possível testar novas ideias sem comprometer a linha-base do desenvolvimento.

{ Gerenciamento de controle com Mercurial

O Mercurial é uma ferramenta gratuita e distribuída de gerenciamento de controle de origem (repositório central). Ele oferece ferramentas para lidar de maneira eficiente com projetos de qualquer tamanho, por meio de interface intuitiva (Fonte: Site oficial Mercurial).

Operações iniciais:

- Inicialização de repositório
- Adição, exclusão, cópia e renomeação de arquivos
- Status das modificações
- Consolidação
- Visualização do histórico de mudanças
- Possibilidade de ignorar arquivos
- Possibilidade de reverter alterações

Comandos a serem aplicados:

https://www.mercurial-scm.org/wiki/Tutorial

Google olabi

<SISTEMA.DE CONTROLE.DE VERSÃO>

/exercícios/

{ Exercício coletivo I

Criaremos um repositório para resolver os desafios abaixo.

Primeiro, **crie um novo repositório** chamado **projeto-config** com os arquivos abaixo e insira uma mensagem em cada um deles, para que o sistema possa identificar as novas modificações:

- README.md
- config.json
- get_config.sh

{ Exercício coletivo II:

Acesse o repositório **projeto-config** e **verifique os estados dos arquivos.** Em seguida, adicione-os à área de staging e verifique novamente o status do versionamento.

{ Exercício coletivo III:

Com os arquivos adicionados, **reverta o estado do arquivo** get_config.sh para a área *working directory*.

Para consolidar os arquivos na área de *staging*, utilize comando para r**egistrar as modificações realizadas** e escreva uma mensagem que descreve de forma objetiva as modificações.

{ Exercício de reforço:

Para reforçar nossos conhecimentos, vamos praticar os exercícios no link https://mercurial.aragost.com/kick-start/pt_BR/basic/#exercicios . Serão sete exercícios práticos, com comandos que utilizamos no dia a dia.