CONTROLE DE VERSÃO
Centralizado e Distribuído

Profª. Ana Flávia

Controle de Versão

- Muitos problemas de desenvolvimento de software são causados por falta de controle de versão. Faça uma avaliação rápida da situação de uma equipe de desenvolvimento:
  - Alguém já sobrescreveu o código de outra pessoa por acidente e acabou perdendo as alterações?
  - Tem dificuldades em saber quais as alterações efetuadas em um programa, quando foram feitas e quem fez?
  - Tem dificuldade em recuperar o código de uma versão anterior que está em produção?
  - Tem problemas em manter variações do sistema ao mesmo tempo?

Controle de Versão

- **É fundamental para o desenvolvimento de software**.
- Todos os ambientes de desenvolvimento modernos, como o Visual Studio, Eclipse e o NetBeans, já possuem *plugins* para integração com algum sistema de controle de versão.

Controle de Versão

- **Para que Serve o Controle de Versão?**
- O Controle de versão apoia o desenvolvimento de diversas maneiras:

  - **Registro do Histórico** - registra toda a evolução do projeto, cada alteração sobre cada arquivo. Com essas informações sabe-se quem fez o que, quando e onde.
  - Permite reconstruir uma revisão específica do arquivo sempre que desejado;

Controle de Versão

- **Para que Serve o Controle de Versão?**
  - **Colaboração Concorrente** - O controle de versão possibilita que vários desenvolvedores trabalhem em paralelo sobre os mesmos arquivos sem que um sobrescreva o código de outro, o que traria reaparecimento de defeitos e perda de funcionalidades;
  - **Variações no Projeto** - Mantém linhas diferentes de evolução do mesmo projeto.
    - Por exemplo, mantendo uma versão 1.0 enquanto a equipe prepara uma versão 2.0.

Controle de Versão

- O controle de versão é composto de duas partes: o **repositório** e a **área de trabalho**.

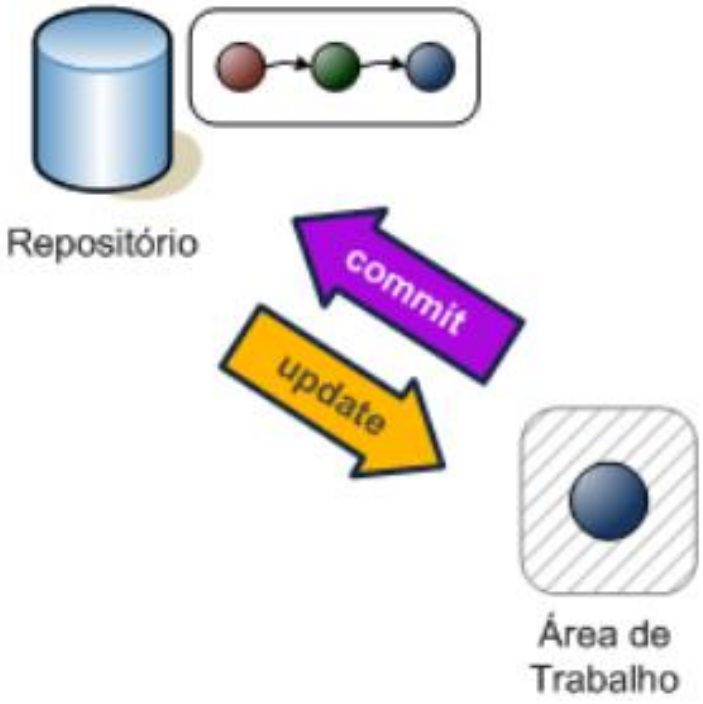
- O **repositório** armazena todo o histórico de evolução do projeto, registrando toda e qualquer alteração feita em cada item versionado.

- O desenvolvedor não trabalha diretamente nos arquivos do repositório. Ao invés disso, usa uma **área de trabalho** que contém a cópia dos arquivos do projeto e que é monitorada para identificar as mudanças realizadas. Essa área é individual e isolada das demais áreas de trabalho.

Controle de Versão

- Estrutura de um controle de versão é composta por **repositório** e **área de trabalho**. A comunicação entre elas se dá através de operações de *commit* e *update*.

Controle de Versão



Repositório x Área de Trabalho

Controle de Versão

- A sincronização entre a área de trabalho e o 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) ao repositório (destino).

  - O ***update*** faz o inverso, isto é, envia as modificações contidas no repositório (origem) para a área de trabalho (destino).

Controle de Versão

- Cada ***commit*** gera uma nova revisão no repositório, contendo as modificações feitas, data e autor. Uma revisão funciona como uma "foto" de todos os arquivos e diretórios em um determinado momento da evolução do projeto. As "fotos" antigas são mantidas e podem ser recuperadas e analisadas sempre que desejado.

- O conjunto dessas revisões é justamente o histórico do projeto.

Controle de Versão

- Tanto o controle de versão **centralizado** quanto o **distribuído** possuem repositórios e áreas de trabalho.

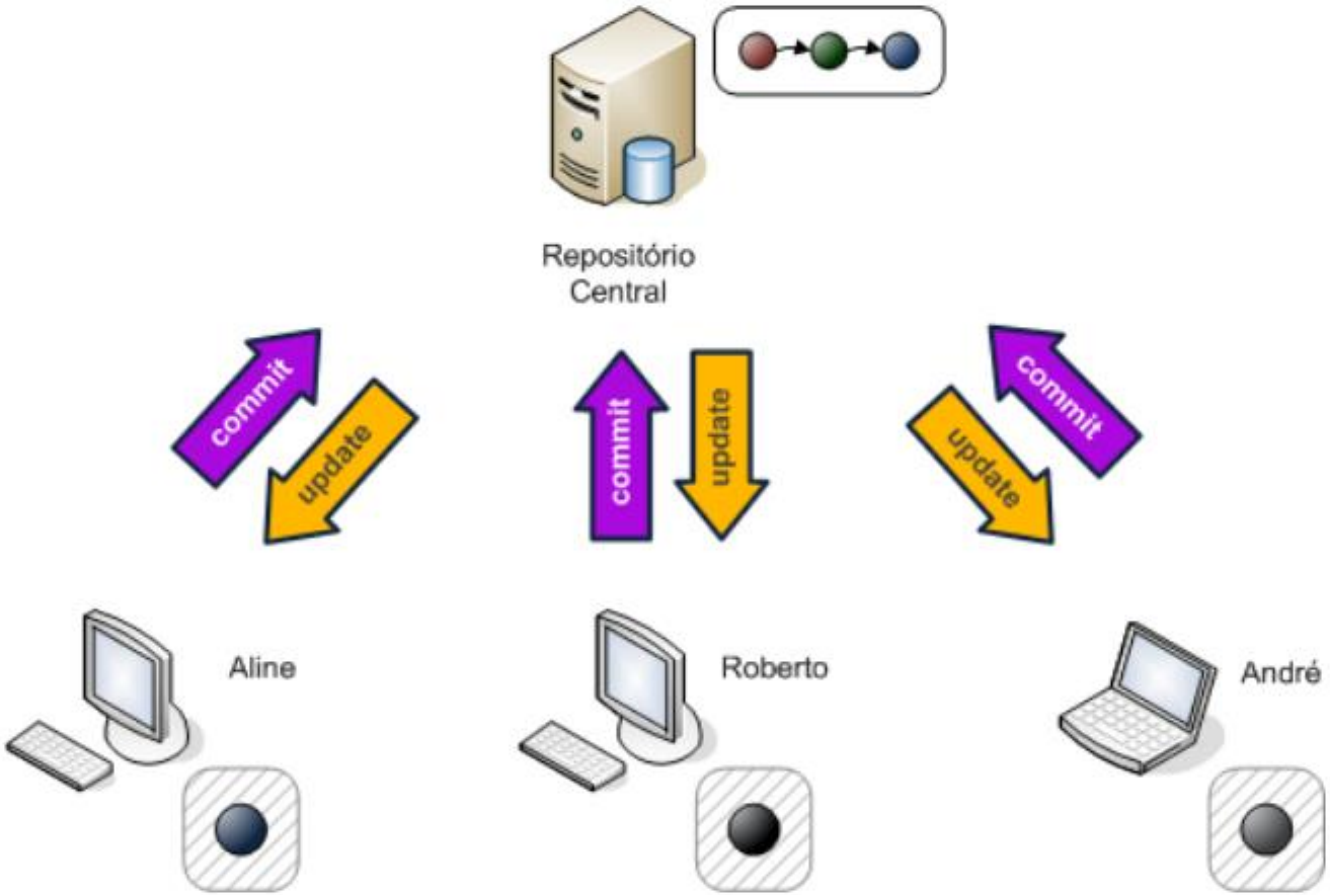
- A diferença está em como cada uma dessas partes está arranjada.

Controle de Versão Centralizado

- O controle de versão centralizado segue a topologia em estrela, havendo apenas um único repositório central mas várias cópias de trabalho, uma para cada desenvolvedor.

- A comunicação entre uma área de trabalho e outra passa obrigatoriamente pelo repositório central.

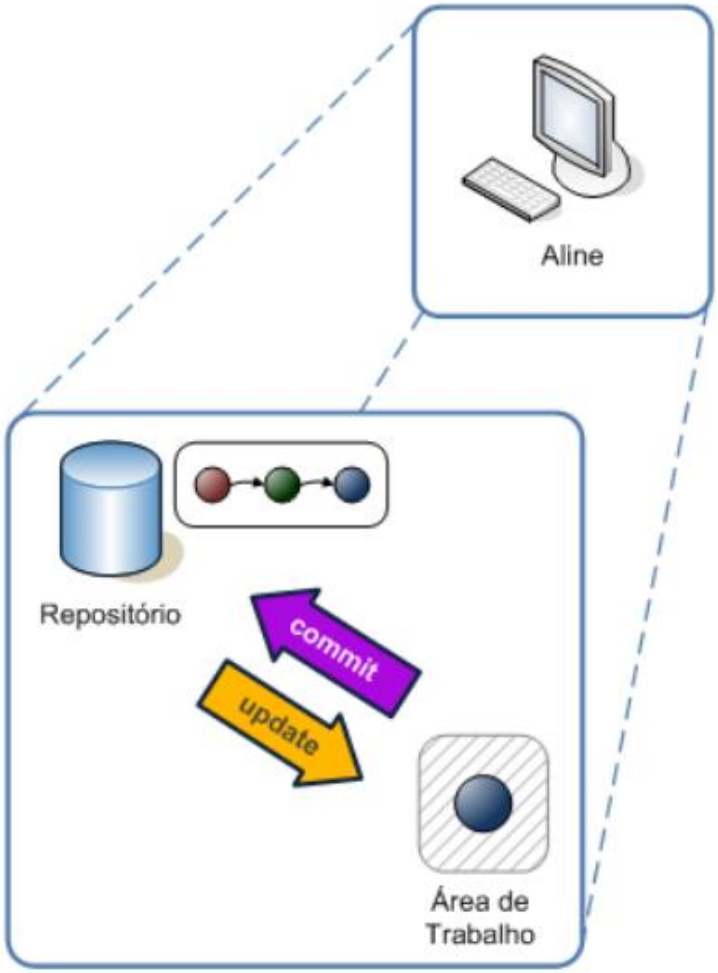
Controle de Versão Centralizado



Controle de Versão Distribuído

- São vários repositórios autônomos e independentes, um para cada desenvolvedor.
- Cada repositório possui uma área de trabalho acoplada e as operações ***commit*** e ***update*** acontecem localmente entre os dois.

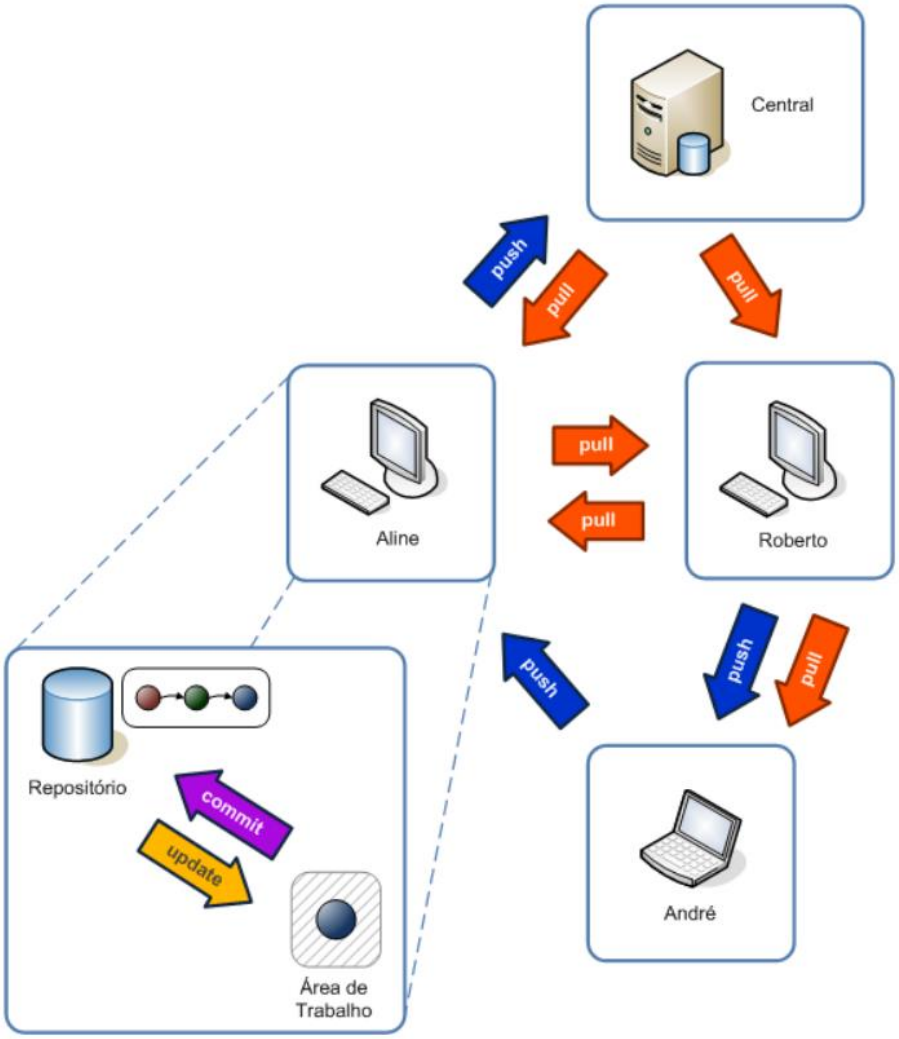
Controle de Versão Distribuído



Controle de Versão Distribuído

- Um repositório pode se comunicar com qualquer outro através das operações básicas ***pull*** e ***push***:
  - ***pull*** (Puxar). Atualiza o repositório local (destino) com todas as alterações feitas em outro repositório (origem).
  - ***push*** (Empurrar). Envia as alterações do repositório local (origem) para um outro repositório (destino).

Controle de Versão Distribuído



Controle de Versão Distribuído

- Um repositório recebe e envia revisões com qualquer outro através de operações ***pull*** e ***push*** sem a necessidade de uma topologia pré-definida.

- A sincronização entre os desenvolvedores acontece de repositório a repositório e não existe, em princípio, um repositório mais importante que o outro, embora o papel de um repositório central possa ser usado para convencionar o fluxo de trabalho.

Resumo das Operações Básicas dos Controles de Versão Centralizado e Distribuído

Centralizado	Distribuído	Descrição
`checkout`	`clone`	criação da cópia de trabalho/repositório
`commit`	`commit`	envia alterações para o repositório, criando uma revisão
`update`	`update`	atualiza a cópia/área de trabalho em uma revisão
	`pull`	importa revisões feita em outro repositório
	`push`	envia revisões locais para outro repositório

Identificação de Revisões

- Uma revisão precisa de uma identificação única.
  - No controle de versão **centralizado**, cada revisão produzida recebe um número inteiro sequencial: 1, 2, 3... Como só existe um repositório, a numeração de revisão é a mesma para todos os desenvolvedores.

Identificação de Revisões

Committed Changes

	Rev	Scores	Commit log message
☆	r5356		Extra import lint from r5314 cl
☆	r5355		Issue 34804: Fix javadoc tag '
☆	r5354		Adds HTMLTable.clear(boolea
☆	r5353		Fixes issue 3647: Long.parse
☆	r5352		Fixed checkstyle errors. Patc
☆	r5351		Fixed checkstyle errors. Patc
☆	r5350		BUILD FIX: restore draftComp
☆	r5349		BUILD FIX: missed a necessa

Identificação de Revisões

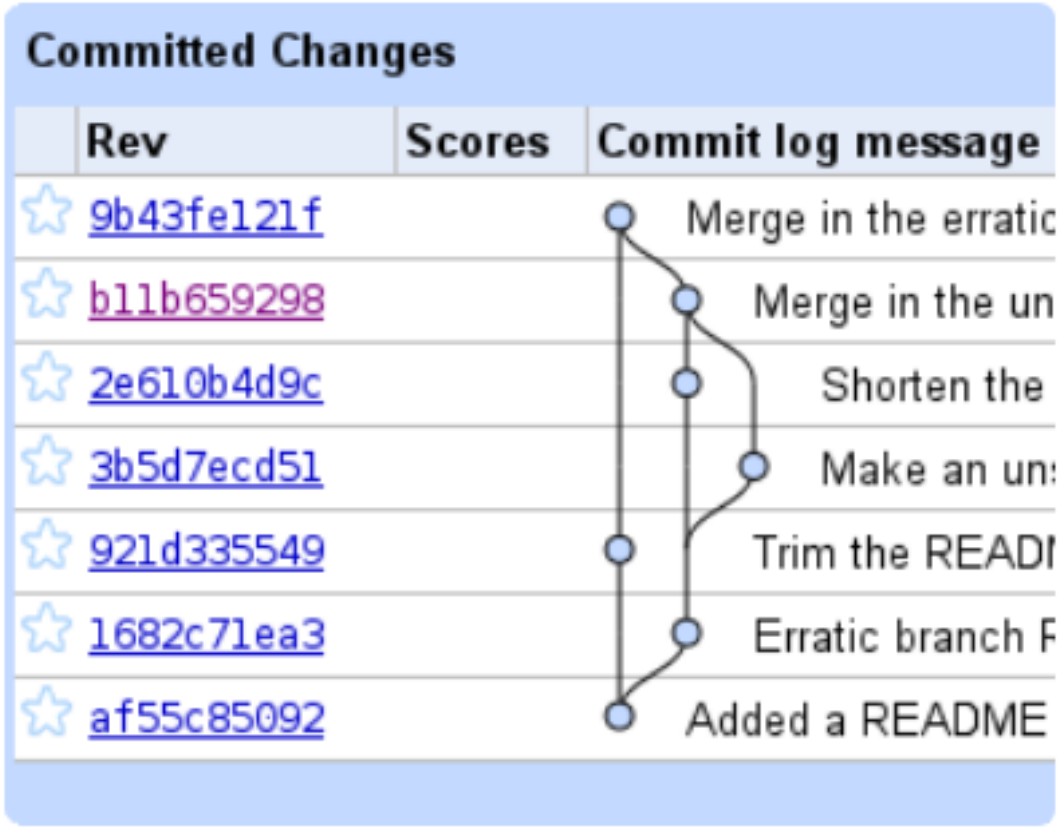
- No sistema **distribuído**, os repositórios são autônomos e portanto não há como definir uma numeração sequencial compartilhada para todos.
  - A solução é identificar cada revisão com uma numeração que nunca se repita em qualquer outro repositório.

Identificação de Revisões

- A forma mais usada é através de um **hash SHA-1**, que produz um número de 160 bits (40 dígitos na forma hexadecimal).

- Esse um número é tão grande e específico que torna extremamente improvável a colisão com um **hash** produzido por outro repositório.

Identificação de Revisões



Sincronização de Mudanças Concorrentes

- Uma das responsabilidades do controle de versão é possibilitar o trabalho paralelo e concorrente de vários desenvolvedores sobre os mesmos arquivos, evitando que um sobrescreva o código de outro, o que resultaria no reaparecimento de defeitos e perda de funcionalidades.

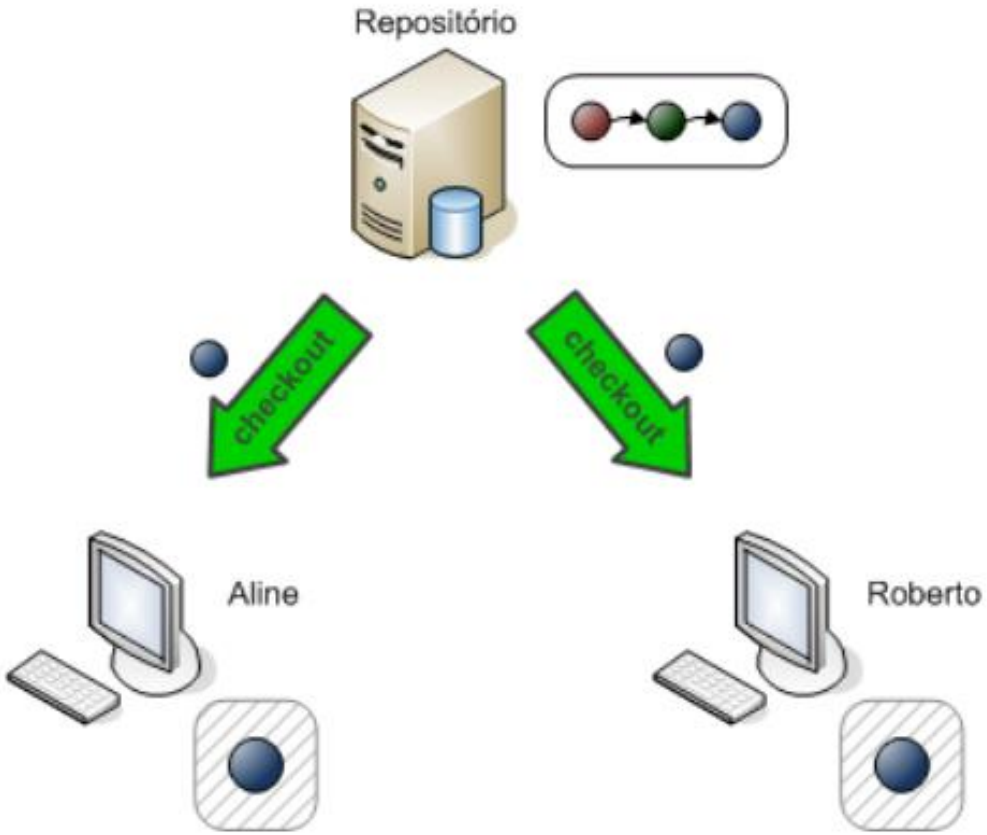
Sincronização de Mudanças Concorrentes

- Em parte, isto é conseguido através da área de trabalho, que fornece a impressão de que o desenvolvedor é o único dono de todo o projeto.
- Mas só a área de trabalho não resolve todo o problema. Ainda é necessário um jeito de sincronizar os esforços dos membros da equipe.

Sincronização de Mudanças Concorrentes

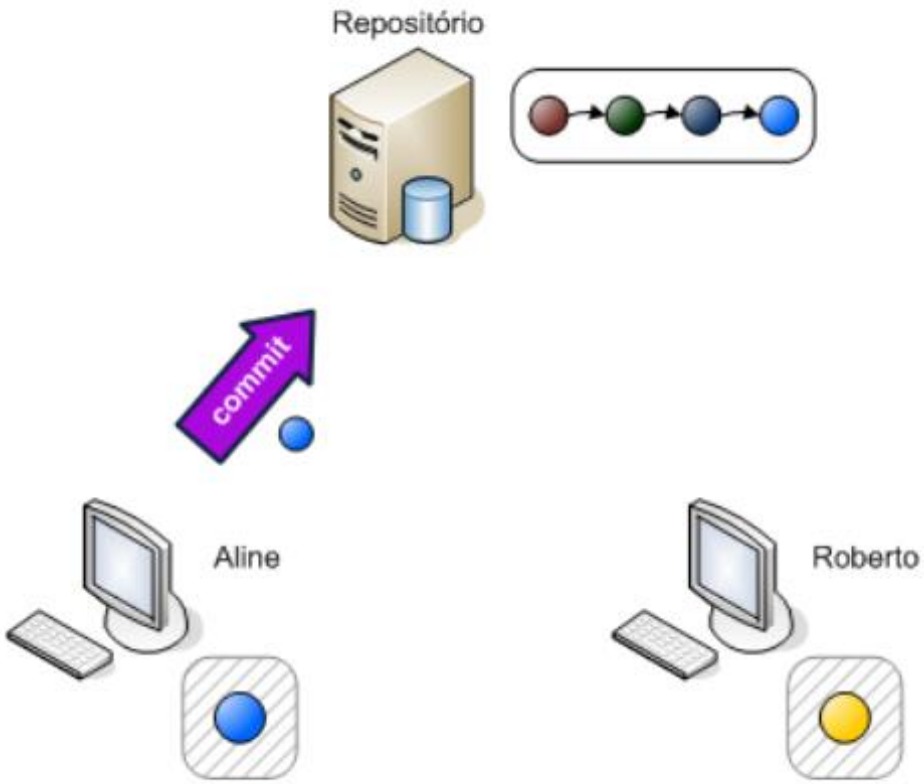
- A sincronização é feita combinando-se revisões concorrentes em uma única resultante.
- Essa operação é conhecida como **merge** (mesclagem) e, apesar de parecer complexa, acontece sem conflitos na maioria das vezes.

Sincronização no Controle de Versão Centralizado



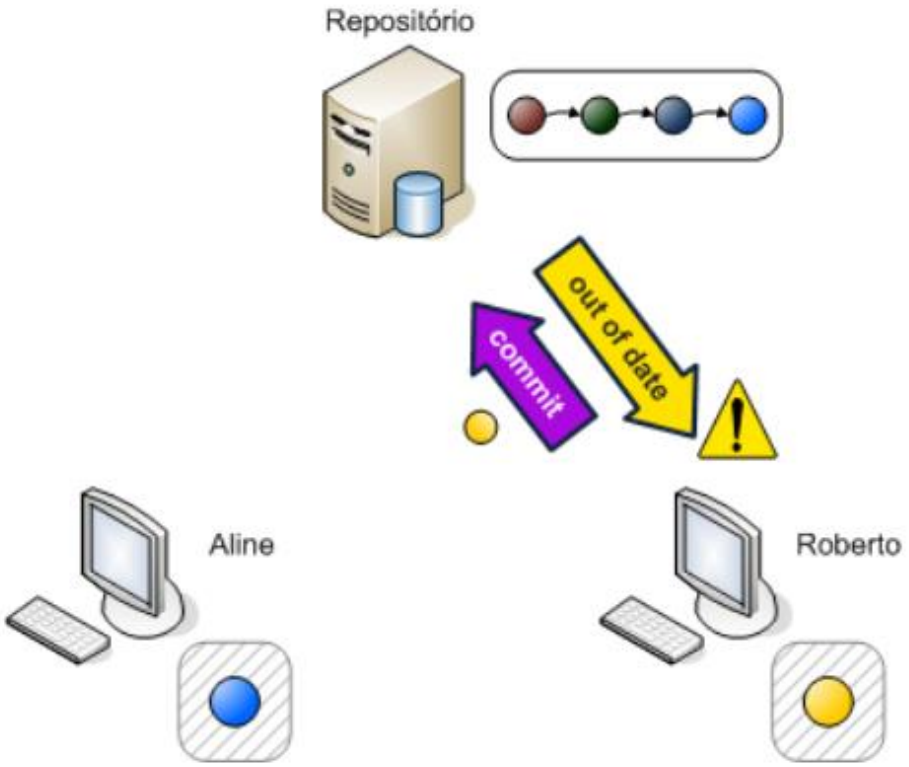
1.Duas cópias de trabalho são criadas a partir do comando `checkout`. As duas iniciam no mesmo estado.

Sincronização no Controle de Versão Centralizado



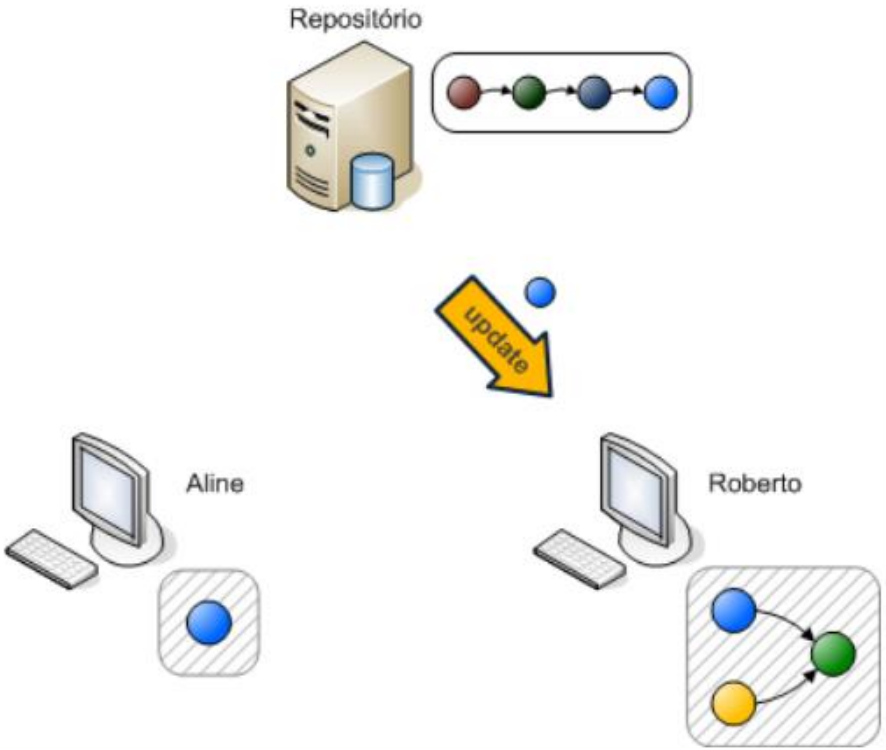
2. Os dois desenvolvedores executam modificações nas suas cópias de trabalho, mas Aline publica antes no repositório.

Sincronização no Controle de Versão Centralizado



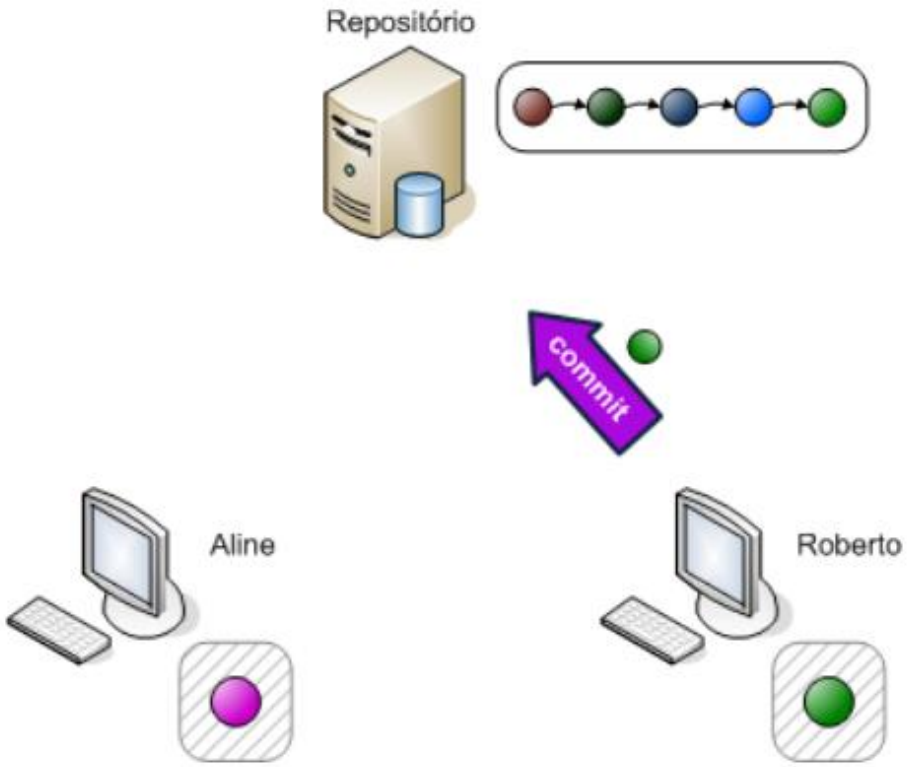
3. Roberto tenta publicar suas alterações, mas o controle de versão recusa justificando que as alterações foram baseadas em arquivos desatualizados. No caso, um ou mais arquivos alterados por Roberto já haviam sido alterados por Aline antes.

Sincronização no Controle de Versão Centralizado



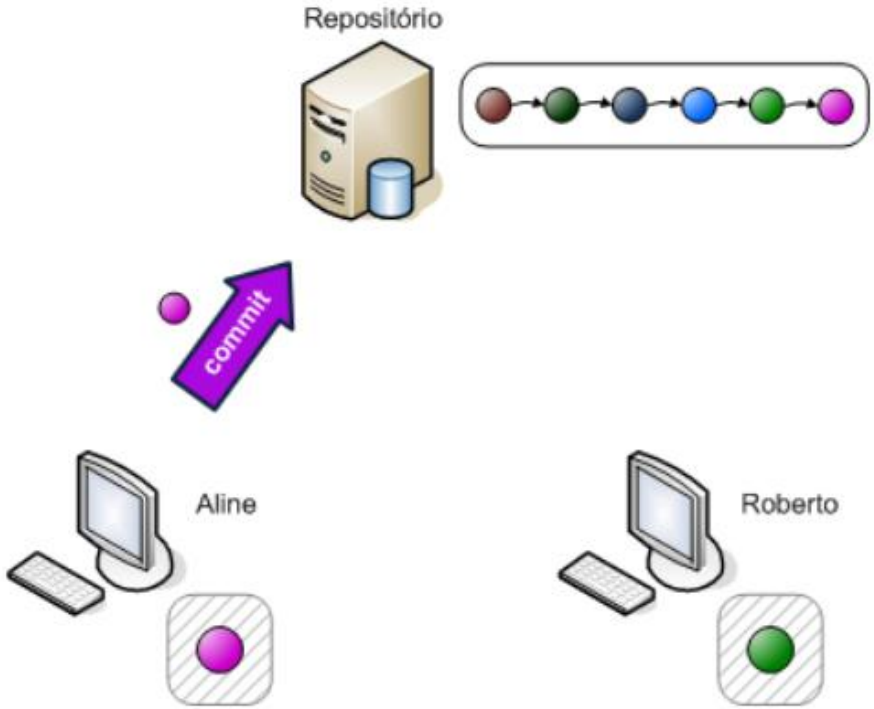
4. Na atualização da cópia de trabalho, o controle de versão já mescla automaticamente as revisões.

Sincronização no Controle de Versão Centralizado



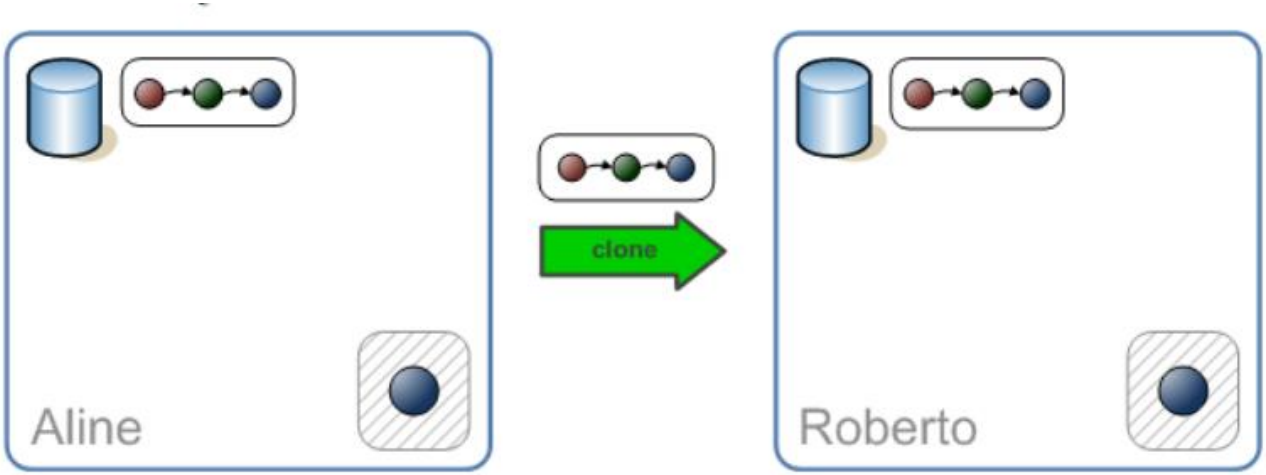
5. Após conferir se a atualização e a mesclagem produziram o resultado desejado, Roberto envia as mudanças ao repositório. Enquanto isso, Aline já trabalha em outra tarefa, executando novas alterações.

Sincronização no Controle de Versão Centralizado



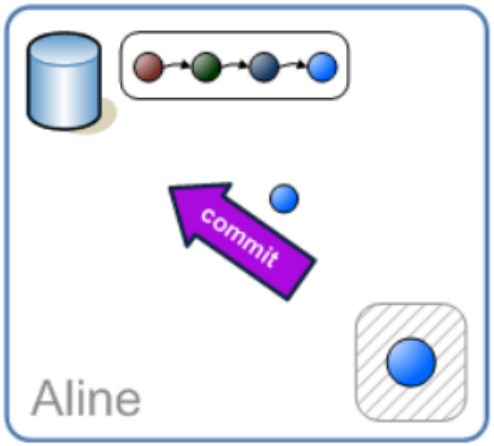
6. O commit de Aline pode ser aceito sem problema se nenhuma das revisões que vieram depois da atualização da cópia de trabalho tiver alterado os mesmos arquivos que Aline. É uma situação possível de acontecer, mesmo que não seja comum.

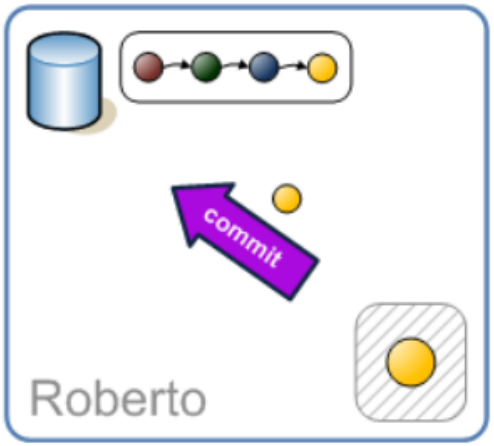
Sincronização no Controle de Versão Distribuído



1. Roberto clona o repositório de Aline. Agora, ambos partem do mesmo ponto.

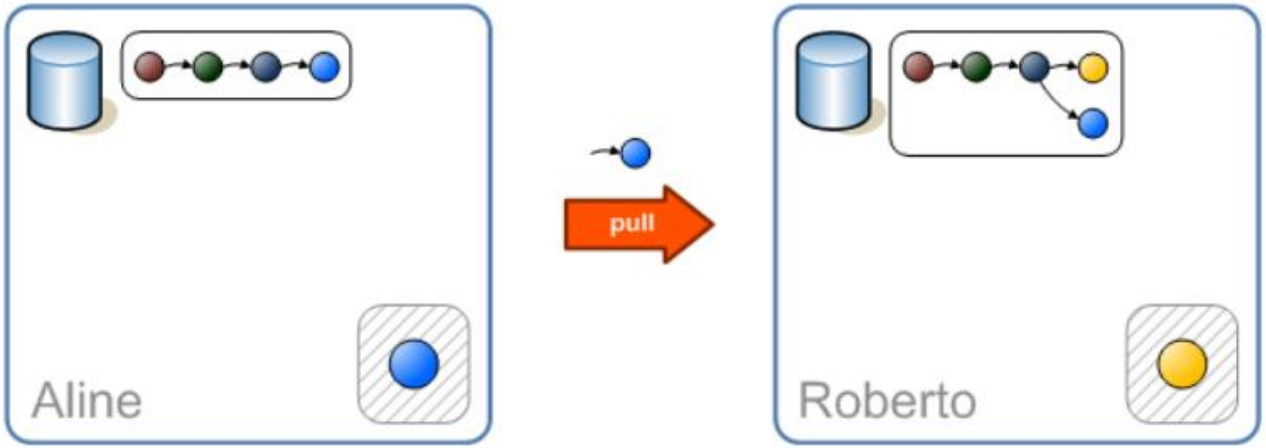
Sincronização no Controle de Versão Distribuído





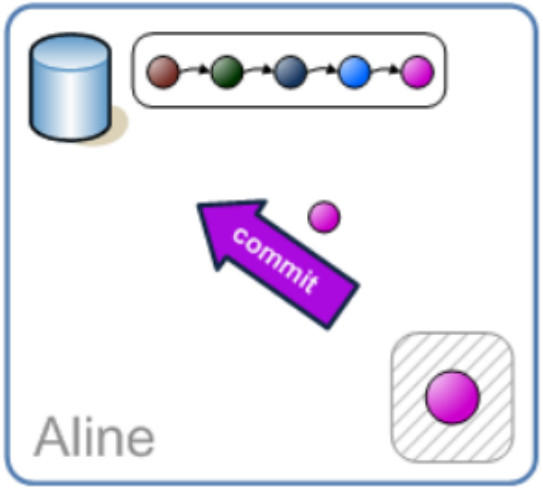
2. Aline e Roberto publicam suas alterações nos seus respectivos repositórios, sem interferir no repositório um do outro.

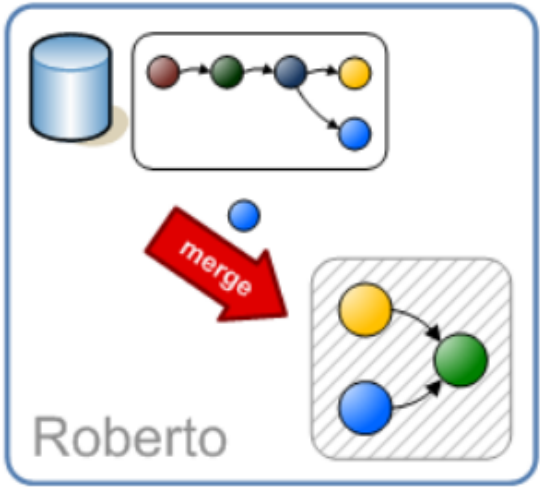
Sincronização no Controle de Versão Distribuído



3. Roberto sincroniza seu repositório com as revisões publicadas por Aline. Sua área de trabalho não é afetada pela sincronização.

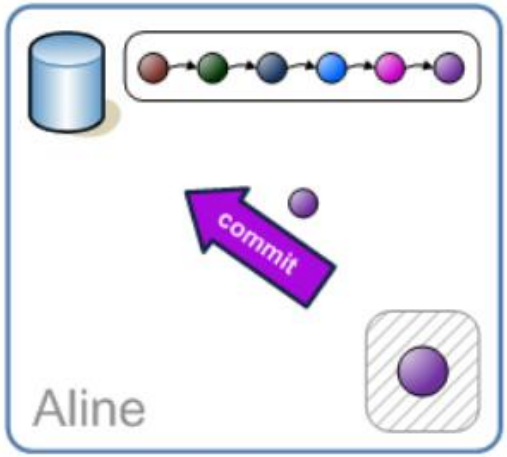
Sincronização no Controle de Versão Distribuído

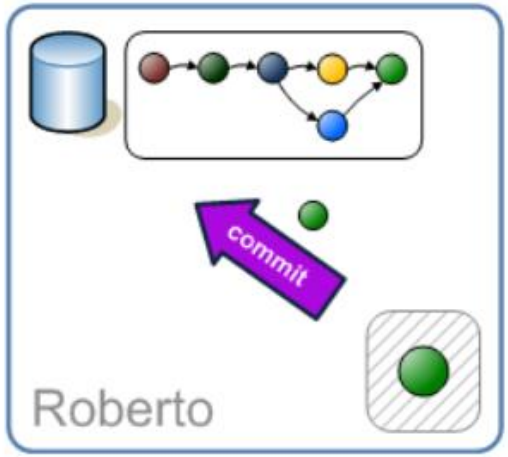




4. A mesclagem entre as revisões de Aline e Roberto é feita explicitamente na área de trabalho de Roberto através de um comando merge. Enquanto isso, Aline já gera outra revisão no seu repositório.

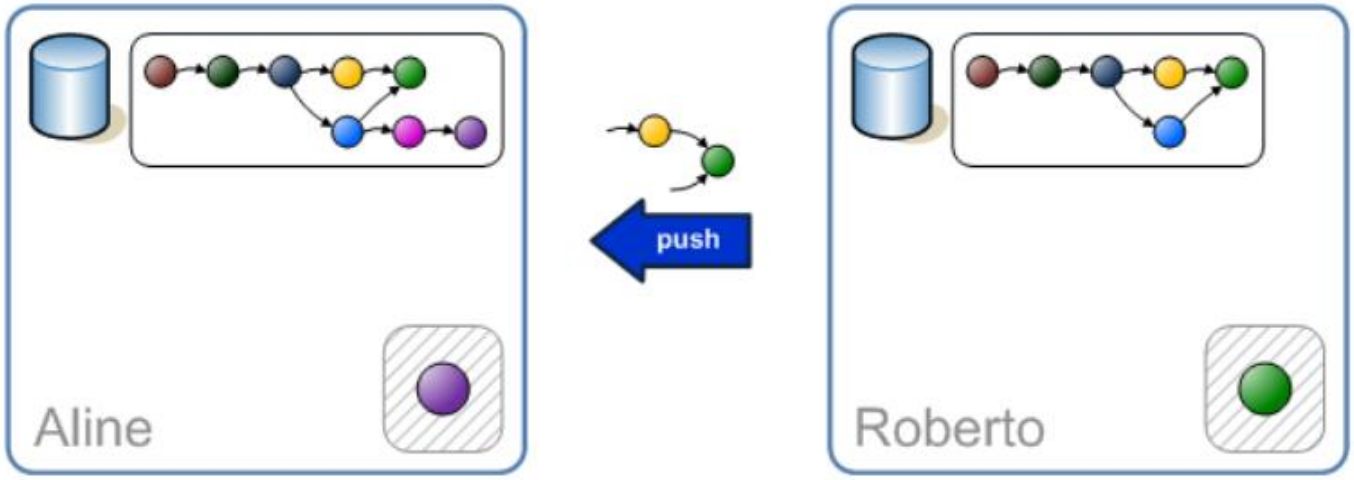
Sincronização no Controle de Versão Distribuído





5. Após conferir se a mesclagem produziram o resultado desejado, Roberto envia as mudanças ao seu repositório. Paralelamente, Aline publica mais uma vez no seu repositório.

Sincronização no Controle de Versão Distribuído



6. Roberto envia suas revisões ao repositório de Aline, que as combina com o histórico de revisões já existente.

Vantagens no Controle de Versão Centralizado

- Usa uma estrutura que atende muito bem a grande parte das equipes de desenvolvimento de software.

- Baseia-se na arquitetura cliente-servidor, com um repositório central (servidor) e cópias de trabalho para os desenvolvedores (clientes).

Vantagens no Controle de Versão Centralizado

- Para equipes de desenvolvimento com acesso ao repositório pela rede local, essa arquitetura funciona bem.

- A velocidade da rede não é problemática, o tempo de resposta do processamento é aceitável e todos os desenvolvedores da equipe têm permissão de leitura e escrita no repositório.

Vantagens no Controle de Versão Centralizado

Não foi para resolver exatamente esse tipo de necessidade que o controle de versão distribuído foi criado. Imagine uma situação diferente:

- **Equipe com centenas de desenvolvedores**. Significa que mais processamento vai ser exigido do servidor central, piorando o tempo de resposta;
- **Equipe espalhada em diferentes filiais da empresa**. Acesso remoto ao repositório com limitações de conexão e de permissão de escrita;

Vantagens no Controle de Versão Centralizado

- A arquitetura **cliente-servidor** não funciona tão bem para essas situações.
- Soluções alternativas como aumentar a capacidade de processamento do servidor ou replicar os repositórios nem sempre são viáveis ou fáceis de serem implementadas.
- Para essas situações, o **controle de versão distribuído** é mais adequado porque se baseia em repositórios autônomos e individuais.

Vantagens no Controle de Versão Distribuído

- As vantagens estão relacionadas à distribuição do processamento, redundância/replicação de repositórios e às novas possibilidades de colaboração entre desenvolvedores (novos fluxos de trabalho).

Vantagens no Controle de Versão Distribuído

- **Do Ponto de Vista do Desenvolvedor**
  - **Rapidez**. As operações são processadas localmente. Não é necessário passar pela rede e contatar o servidor central para fazer um *commit*, *log* ou *diff*.
  - **Autonomia**. A conexão com a rede só é necessária para trocar revisões com outros repositórios. Fora isso, trabalha-se desconectado e em qualquer lugar, como num cliente.

Vantagens no Controle de Versão Distribuído

- **Do Ponto de Vista da Gerência/Coordenação**
  - Parte das decisões gerenciais envolve manter livre o caminho da equipe para que possam trabalhar da melhor maneira possível.
  - Outras decisões importantes são sobre redução de custos.
  - Nestes dois casos específicos, o modelo distribuído oferece as seguintes vantagens:

Vantagens no Controle de Versão Distribuído

- **Confiabilidade.** No sistema centralizado, uma pane no servidor interrompe todo o desenvolvimento. Já no sistema distribuído, além de a equipe poder continuar seu trabalho, os repositórios dos desenvolvedores funcionam como cópias de *backup* de todo o projeto.

- **Redução de custos com servidor.** A carga de processamento fica distribuída nas próprias máquinas dos desenvolvedores. O repositório "central", quando existe, tem o papel do repositório "oficial" e não como processador central das requisições.

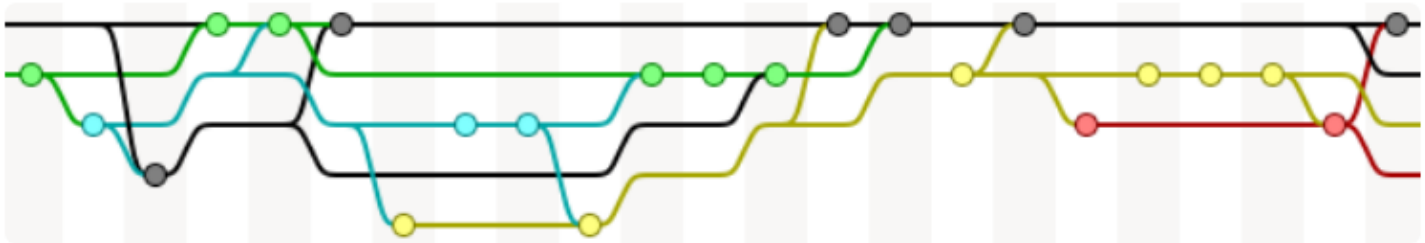
Desvantagens no Controle de Versão Distribuído

- Maior Complexidade

- No centralizado, a forma de trabalho é mais simples de se entender. Mesmo que limitadamente, uma pessoa com pouco conhecimento de controle de versão consegue trabalhar com o resto da equipe.

Desvantagens no Controle de Versão Distribuído

- O DVCS é mais complicado.
- Há diversas combinações de arquiteturas de repositórios e fluxos de trabalho possíveis, o que pode tornar o histórico da evolução do projeto bastante confuso.



Desvantagens no Controle de Versão Distribuído

- Ao contrário do centralizado, não adianta só *commit* e *update* para funcionar "no tranco".

- Todos os desenvolvedores da equipe precisam ter um conhecimento maior do modelo, da ferramenta e, de preferência, também de um processo de desenvolvimento que padronize fluxos de trabalho a serem seguidos.

Desvantagens no Controle de Versão Distribuído

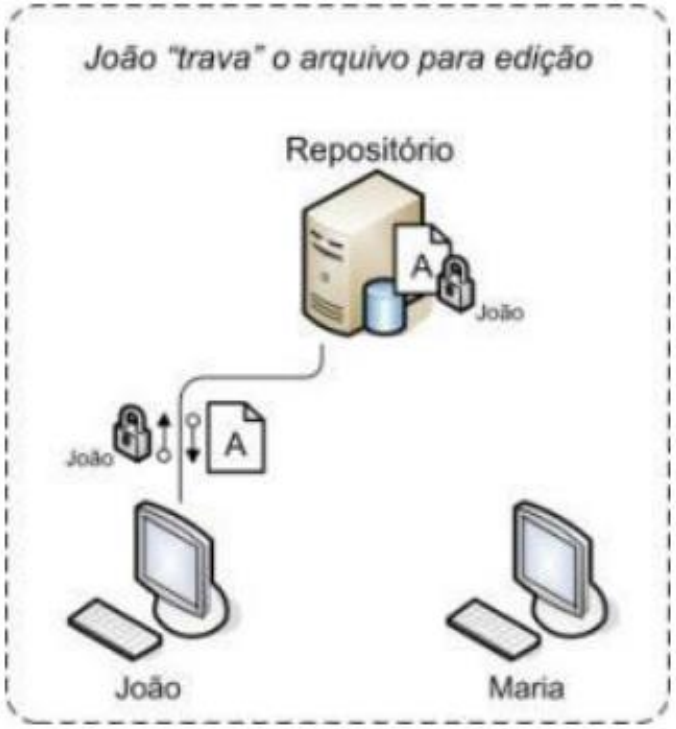
- Travamento de arquivos binários precisa ser centralizado;

- Diferentemente dos arquivos de texto, arquivos binários possuem um formato interno que não é baseado em linhas de texto e, por isso, não podem ser mesclados automaticamente pelo controle de versão ou manualmente pelo desenvolvedor.

Desvantagens no Controle de Versão Distribuído

- A edição concorrente de arquivos binários é problemática.
- Duas pessoas editando ao mesmo tempo uma figura, por exemplo, não conseguirão mesclar as modificações depois e o trabalho de uma delas precisará ser refeito.
- Com arquivos binários, a melhor solução é usar o travamento, isto é, sinalizar que o arquivo está travado para edição e que ninguém mais deve editá-lo enquanto isso.

Desvantagens no Controle de Versão Distribuído



O modelo puramente distribuído não é adequado para lidar com travamento justamente por não possuir um servidor central que possa controlar as travas de todos.

Desvantagens no Controle de Versão Distribuído

- O controle de mudança ainda é centralizado
- Algumas ferramentas de controle de mudança (e.g. Trac, Redmine, Mantis, Bugzilla) ainda não acompanharam as de controle de versão na arquitetura *peer-to-peer*.
- O significa que mesmo usando um DVCS, ainda haverá uma ferramenta centralizada para controle de mudança.

Resumo das Características do Controle de Versão Distribuído

Ponto de Vista	Vantagens	Desvantagens
Desenvolvedor	• Rapidez • Autonomia	• Necessidade de maior conhecimento da ferramenta e do processo
Coordenação/Gerência	• Redução de custos com servidor e infraestrutura externa de rede • Confiabilidade	• Necessidade de maior capacitação de desenvolvedores • Importante ter um fluxo de trabalho bem definido • O controle de mudança ainda precisa ser centralizado

Resumindo

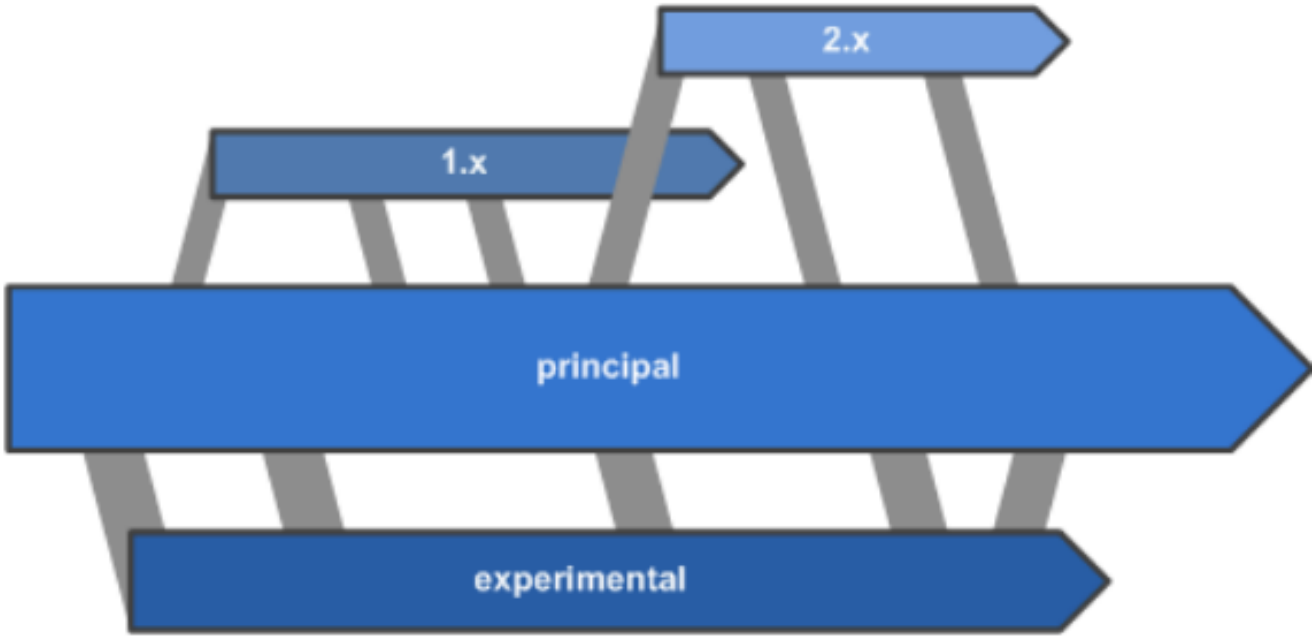
- Controle de versão distribuído oferece algumas vantagens interessantes sobre o centralizado.

- Por outro lado, requer muito mais preparo da equipe e também do processo, o que não chega a ser realmente um impedimento, uma vez que um processo formalizado e equipes capacitadas são fundamentais para produção de software de qualidade.

Diferentes Versões de Projeto

- Muitos projetos precisam de variações específicas.
- Um caso típico é para customizações feitas para atender determinados clientes que precisam de adaptações particulares.
- Outro caso comum é a criação de um ramo para experimentações no projeto, sem comprometer a linha principal de desenvolvimento.

Diferentes Versões de Projeto



O controle de versão oferece funcionalidades que facilitam a coordenação de ramos diferentes de desenvolvimento em um mesmo projeto.

Diferentes Versões de Projeto

- Controle de versão resolve diversos problemas intrínsecos ao desenvolvimento de software.
- É uma prática de engenharia de software comprovadamente eficaz.
- Faz parte das exigências para melhorias do processo de desenvolvimento de certificações tais como CMMi, MPS-Br e SPICE.