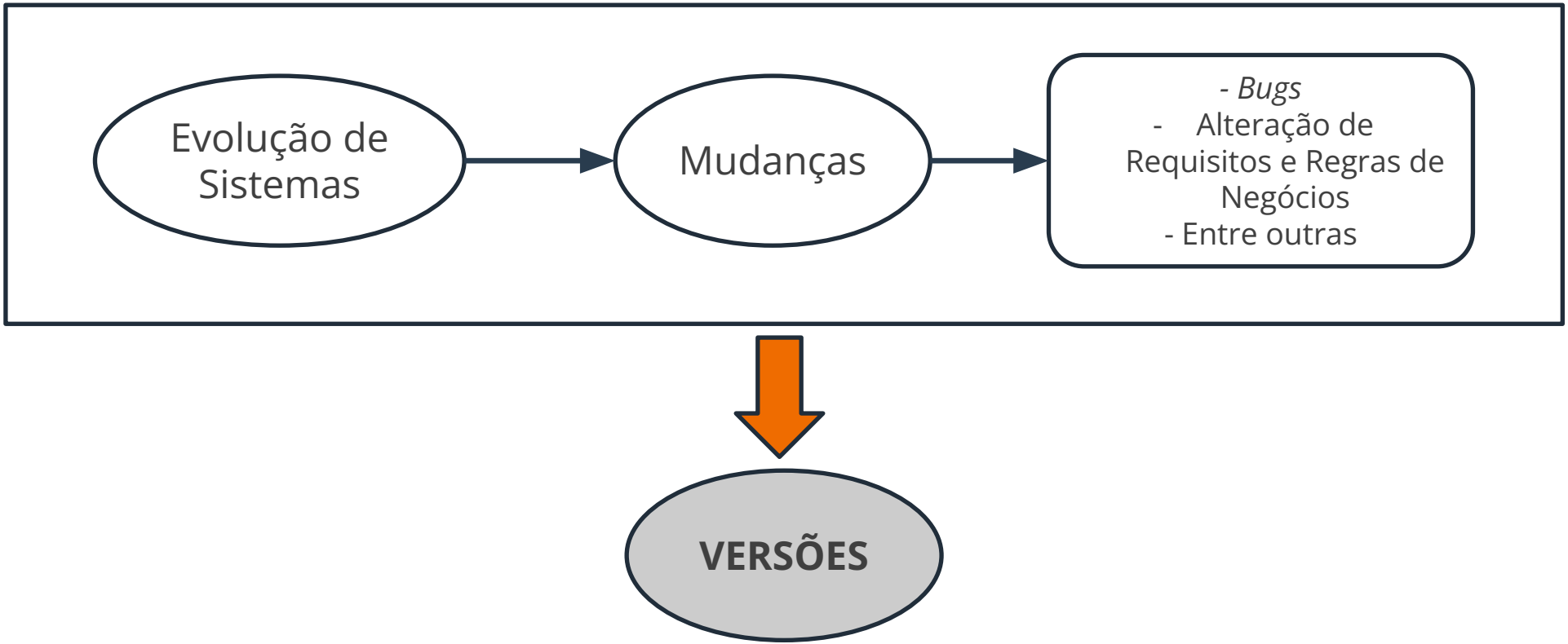
Universidade Estadual de Mato Grosso do Sul

Gerenciamento de Configuração de Software

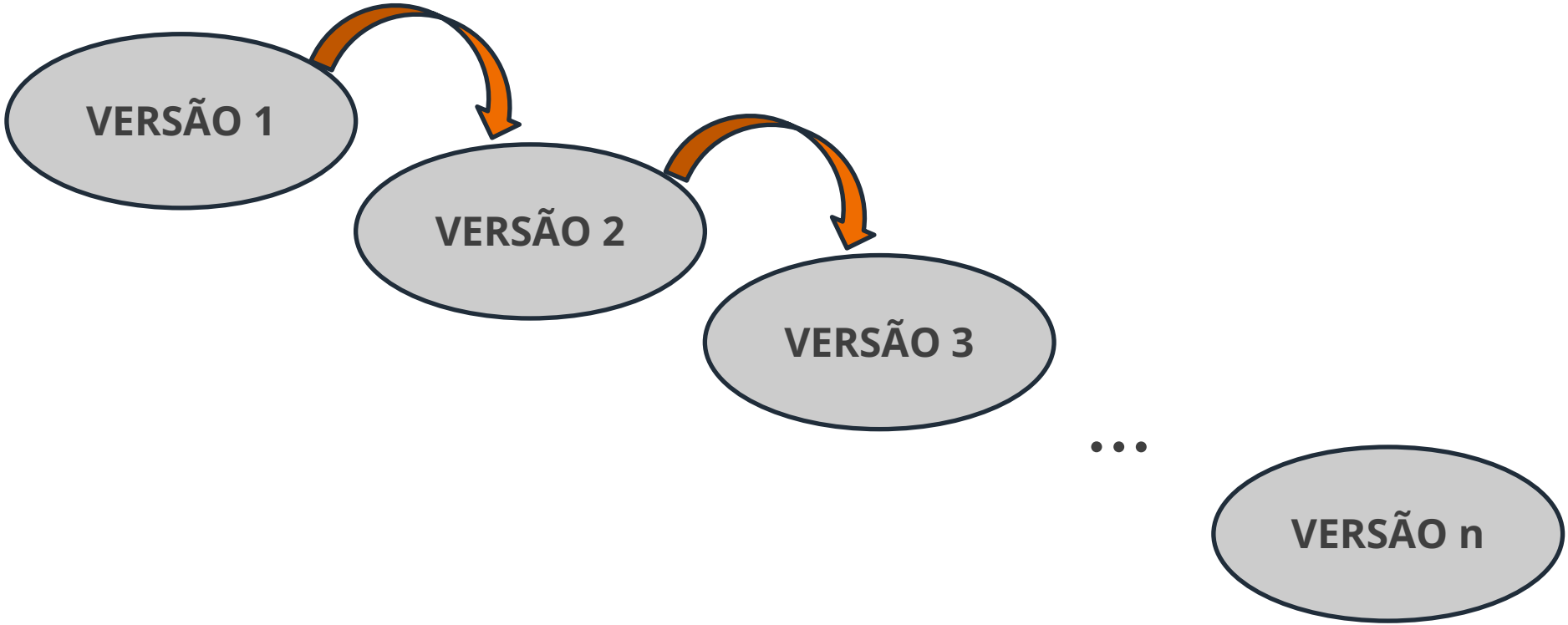
Engenharia de Software II

Profa. Simone de França Tonhão
E-mail: simone.franca@uems.com
Sistemas de Informação

Introdução


Evolução de Sistemas
Mudanças
- *Bugs*
- Alteração de Requisitos e Regras de Negócios
- Entre outras
VERSÕES

Introdução



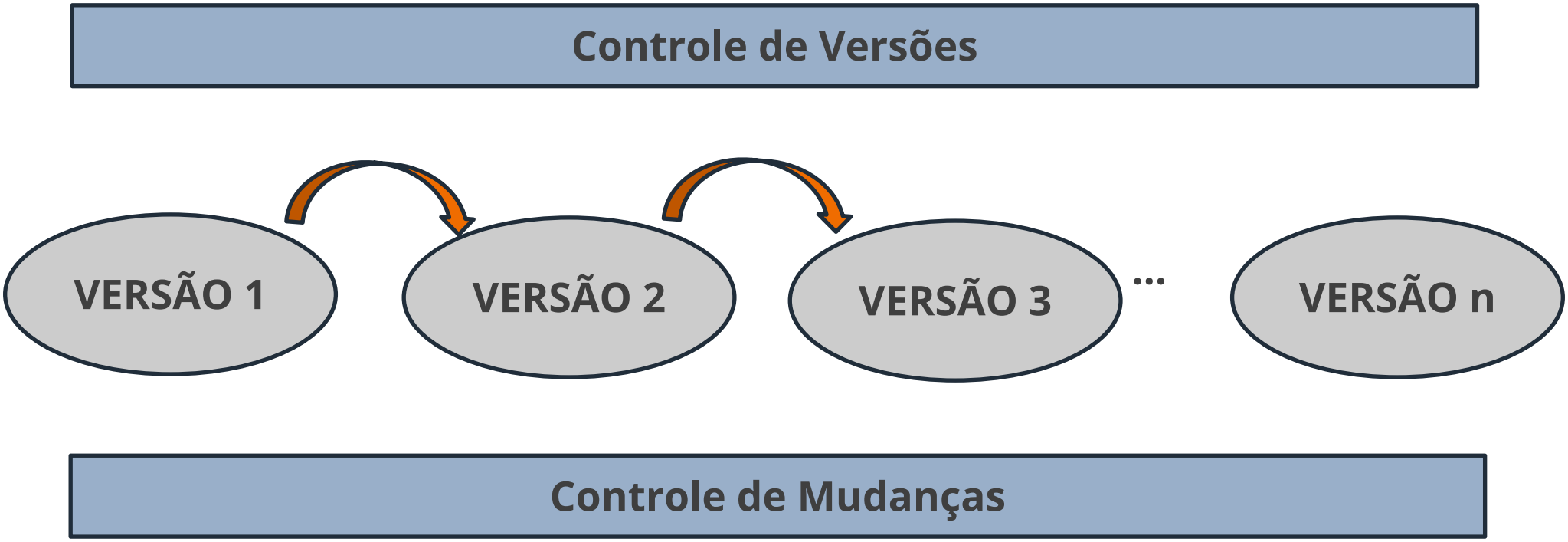
Gerenciamento de Configuração de Software (GCS)

- Necessidade de controlar mudanças;
- Políticas, métodos e ferramentas voltadas para o gerenciamento de mudanças;
- Minimizar erros cometidos durante a evolução do software.

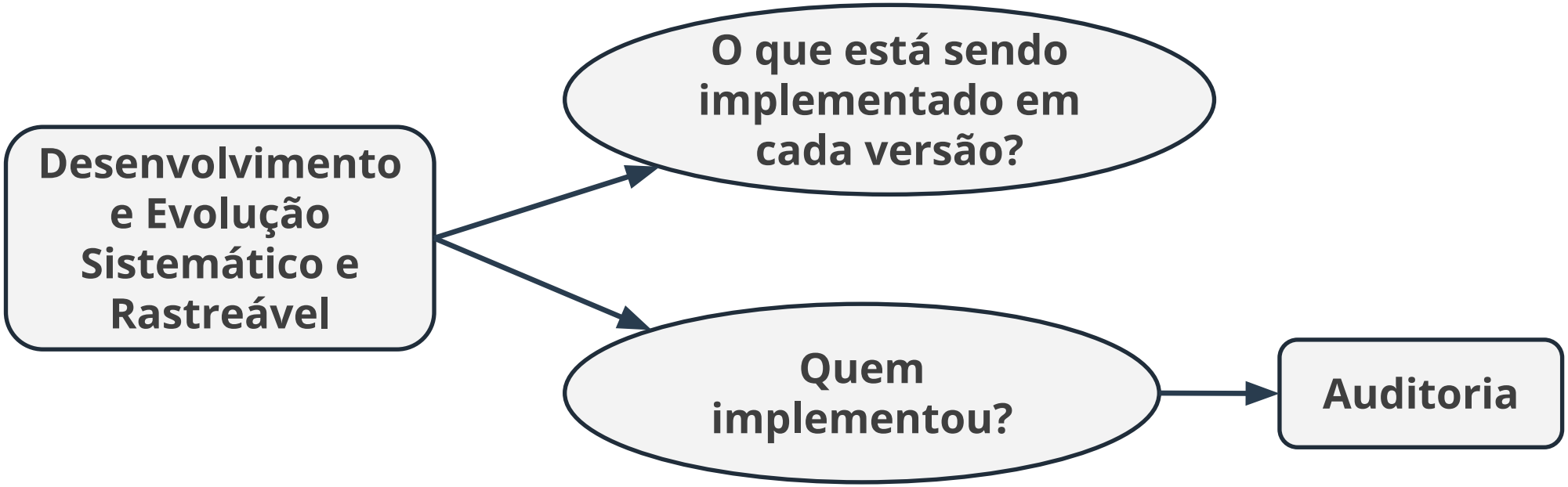
Coordenar o desenvolvimento de software para minimizar a confusão!

Gerenciamento de Configuração de Software (GCS)

- Um importante elemento da **garantia de qualidade de software**;
- É necessário gerenciar sistemas em evolução – **Fácil perder o controle das mudanças**;
- Cada versão implementa propostas de mudanças, correção de defeitos, adaptação de sistemas operacionais e hardware;
- Várias versões em desenvolvimento e uso ao mesmo tempo.


Controle de Versões
VERSÃO 1
VERSÃO 2
VERSÃO 3
...
VERSÃO n
Controle de Mudanças

Gerenciamento de Configuração de Software (GCS)



Gerenciamento de Configuração de Software (GCS)

- É muito utilizado na etapa de manutenção;
- Controlar artefatos produzidos e modificados em outras fases;

Utilidade do GCS

Projetos individuais

É útil!

Projetos em equipe

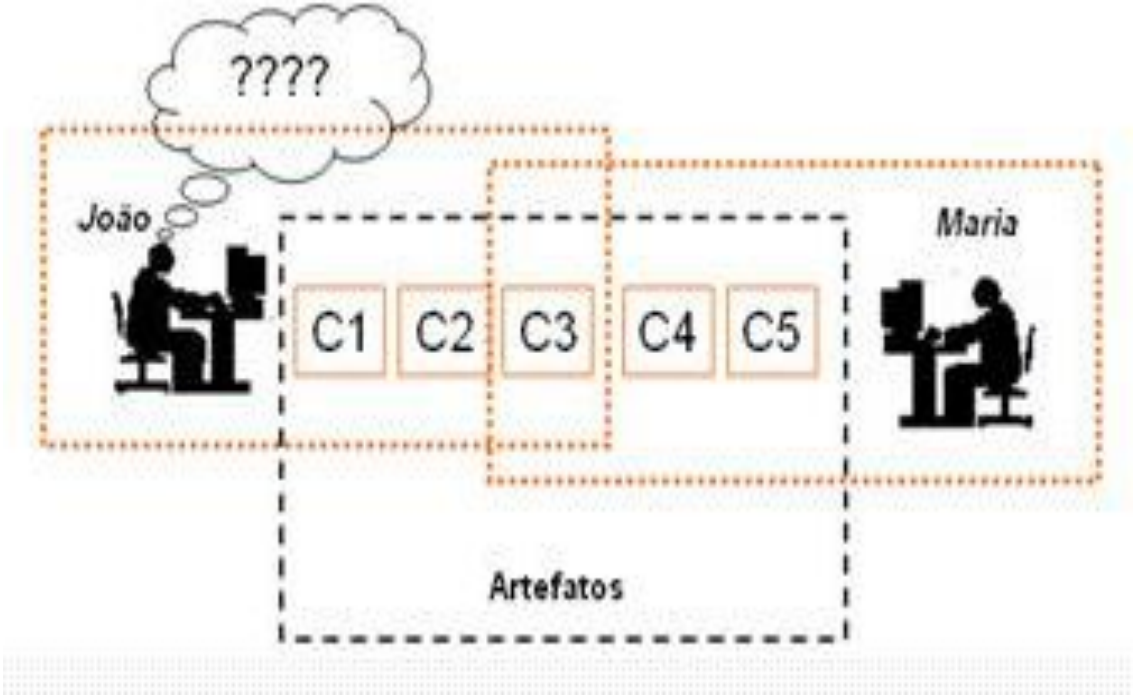
É essencial!

Ter um bom gerenciamento de configuração garante que as equipes tenham acesso às informações sobre um sistema que está sendo desenvolvido, e não interfiram no trabalho de outras pessoas!

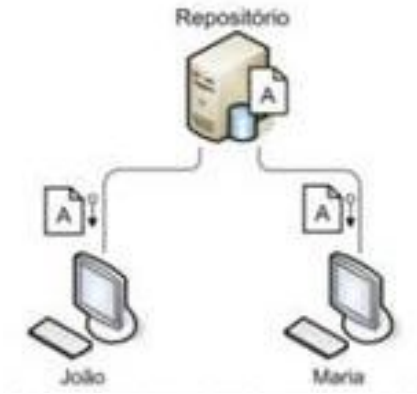
O GCS é importante para que?

- Controlar a mudança que está sendo feita;
- Assegurar que a mudança esteja sendo realizada corretamente;
- Relatar as mudanças aos membros da equipe.

Um exemplo

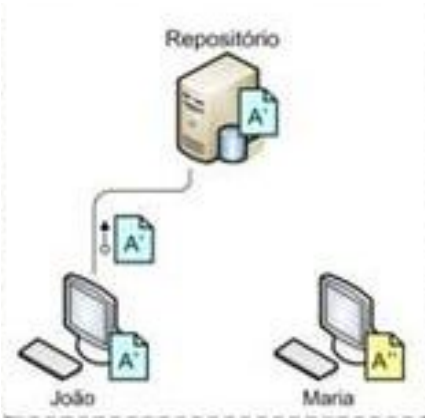


Dois usuários lêem o mesmo arquivo

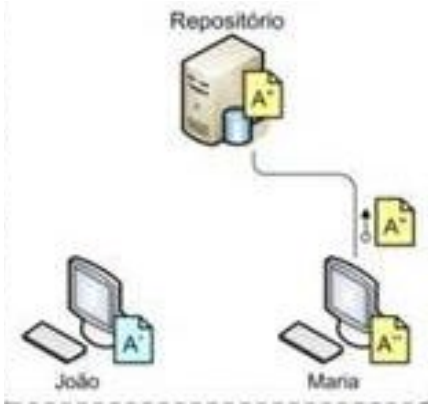


Ambos começam a editar suas cópias

João publica sua versão primeiro

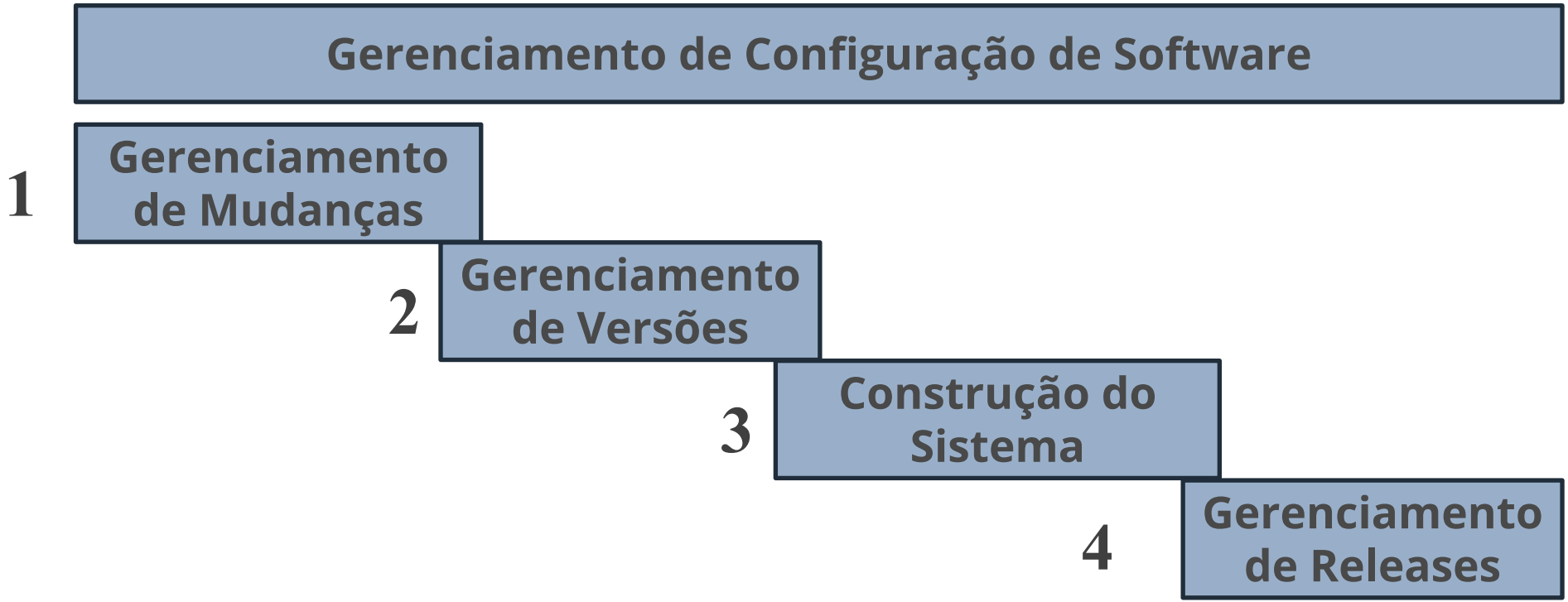


Maria acidentalmente sobrescreve a versão de joão

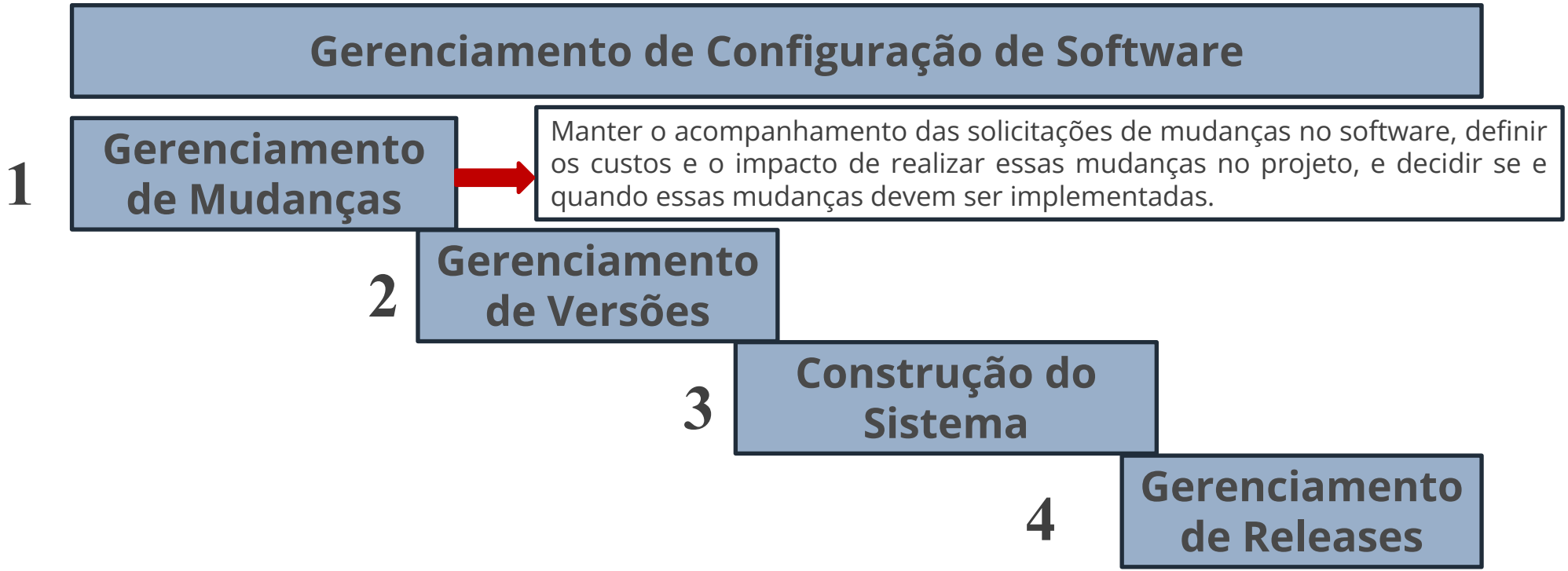


Quando muitas pessoas estão trabalhando no mesmo produto, o GCS coordena o acesso para realizar mudanças no desenvolvimento do software!

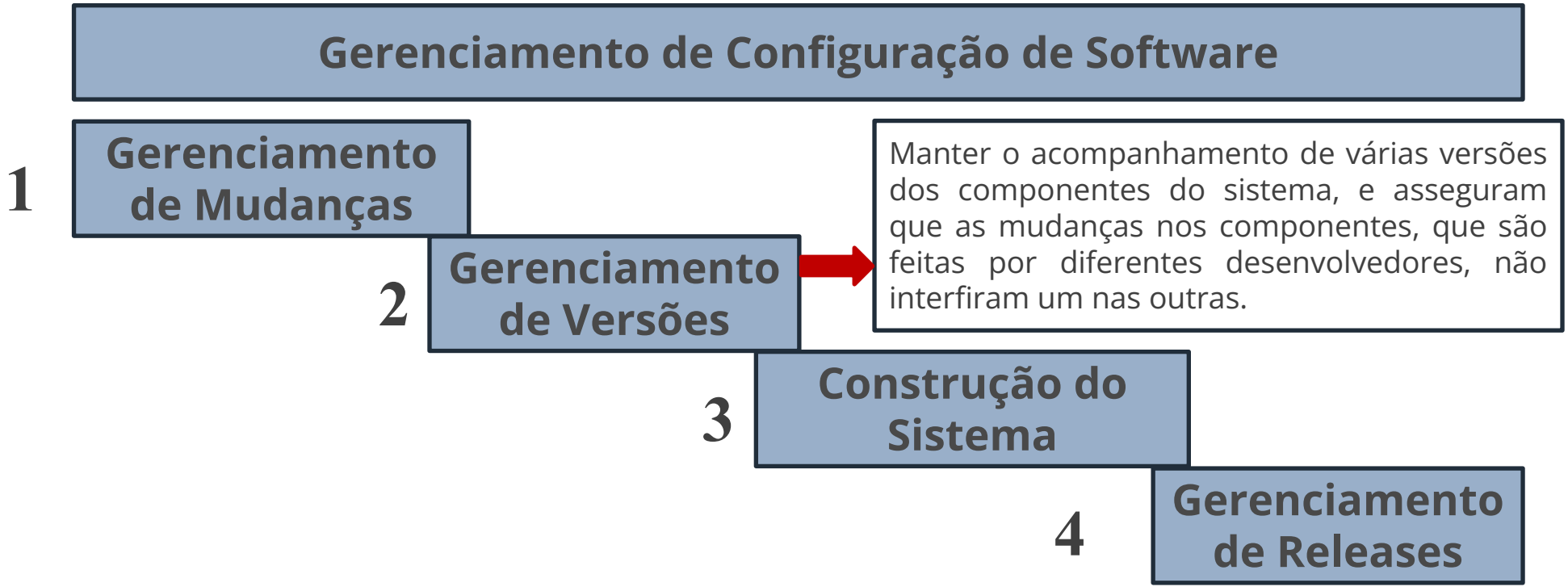
Principais Atividades do GCS



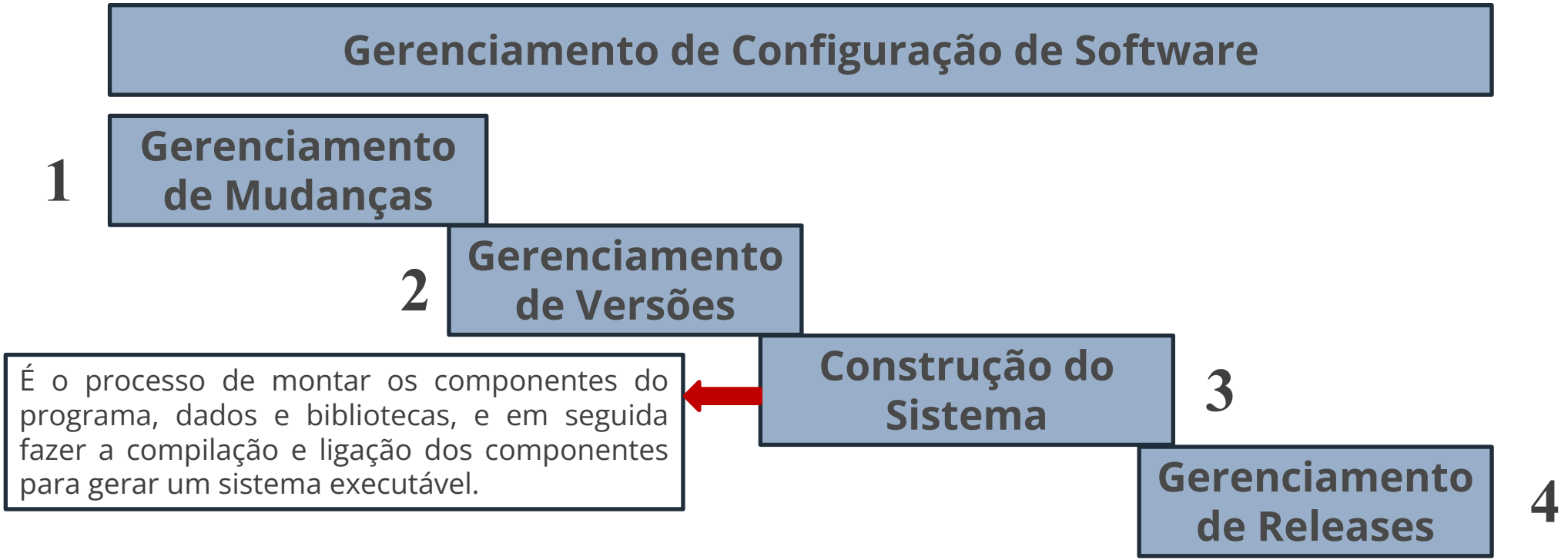
Principais Atividades do GCS



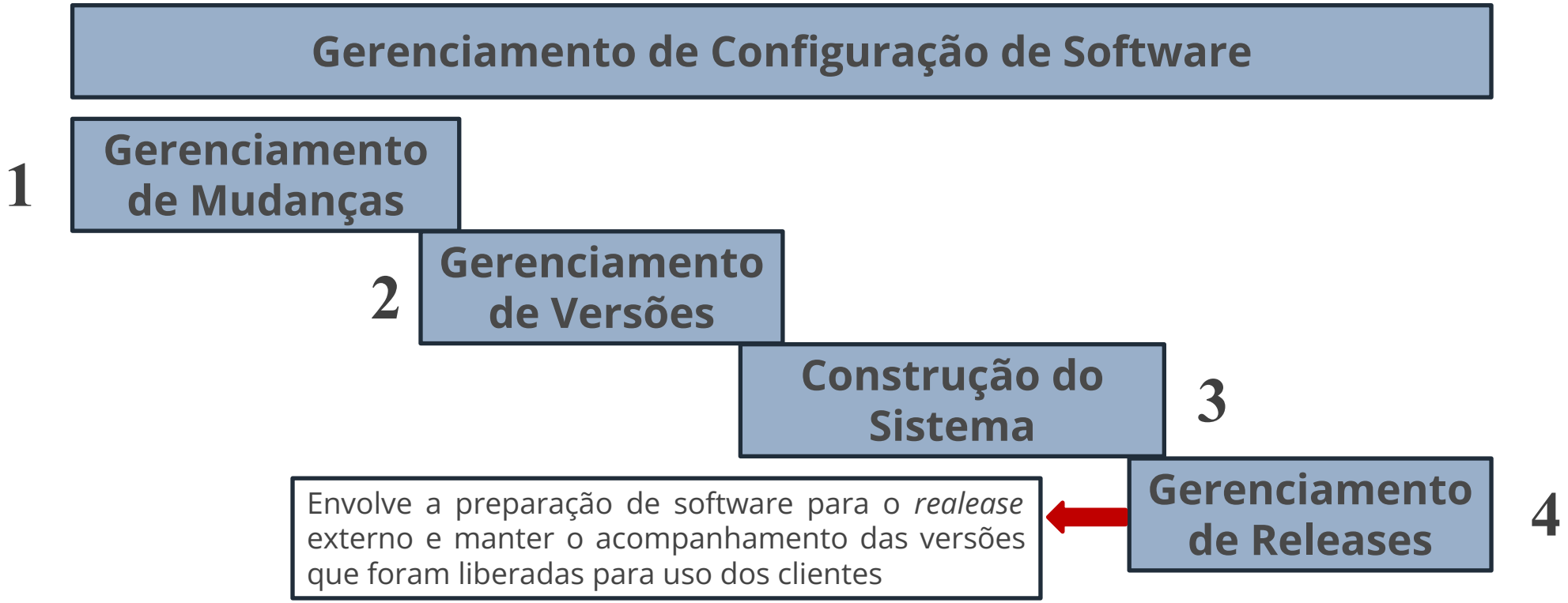
Principais Atividades do GCS



Principais Atividades do GCS



Principais Atividades do GCS



Certificações de Qualidade

- A GCS é a base para as demais áreas da Engenharia de Software;
- Definição e utilização de padrões de GCS são essenciais para certificações de qualidade de padrões ISO 9000, CMM, e CMMI.

Ferramentas de Suporte a GCS

- O GCS envolve lidar com muitas informações;
- Ferramentas das mais simples às mais complexas;
- Quanto mais funcionalidades, maior pode ser o custo.

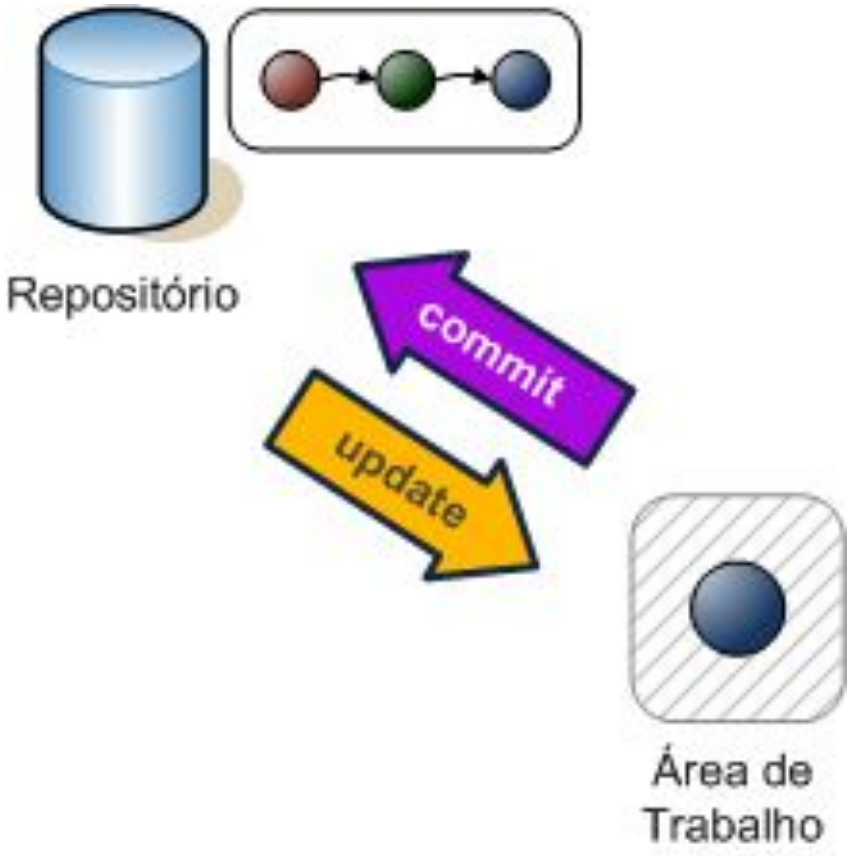
Ferramentas Open Source

Tipo de Ferramenta	Nome
Controle de Versão	CVS Subversion
Controle de Mudança	Mantis Budzilla
Integração (Build)	Scons

Como funciona o Controle de Versão?

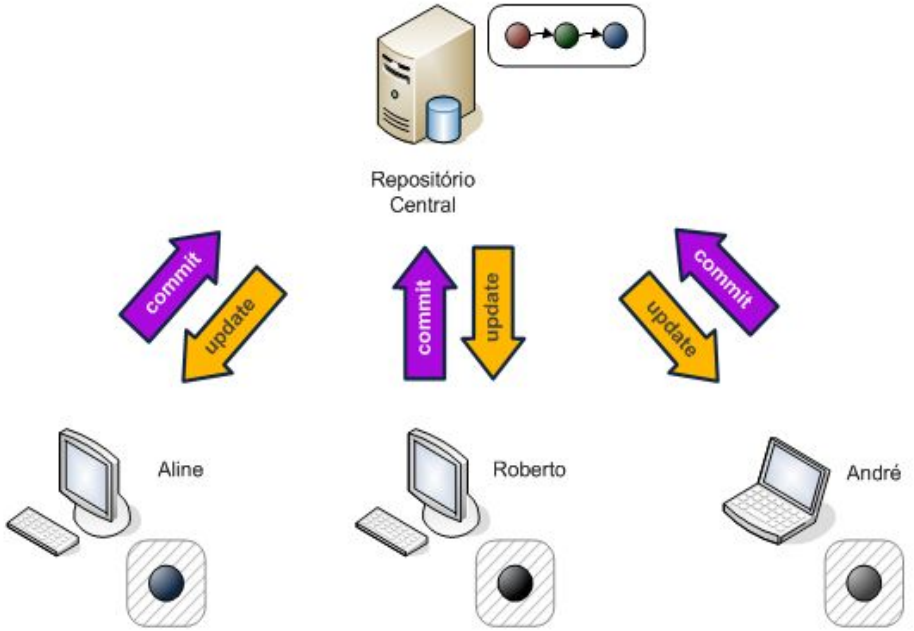
- Composto de duas partes: o **repositório** e a **área de trabalho**.
- O repositório armazena todo o histórico de evolução do projeto, registrando toda e qualquer alteração feita em cada item versionado.
- O desenvolvedor não trabalha diretamente nos arquivos do repositório.
- Usa uma área de trabalho 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.

Como funciona o Controle de Versão?



Como funciona o Controle de Versão?

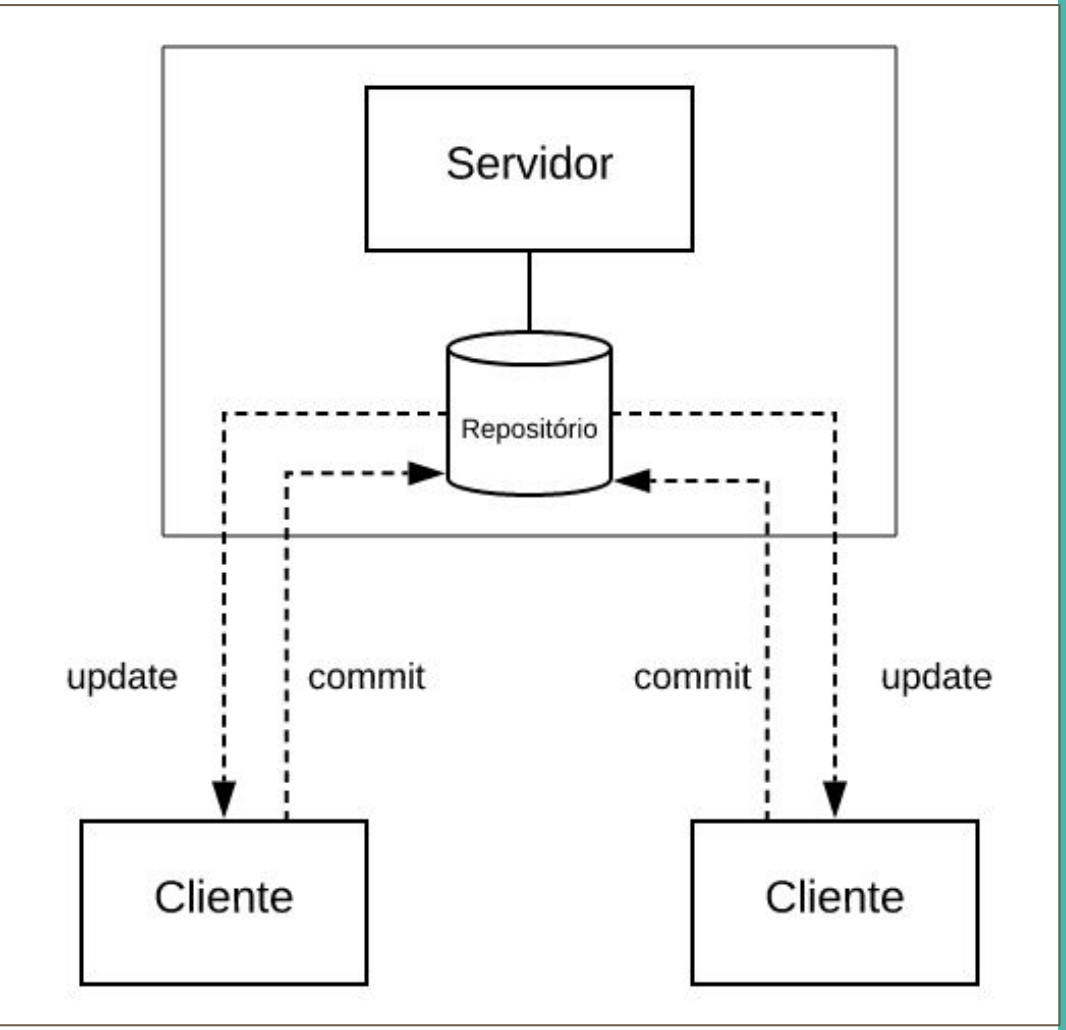
- Há dois tipos de controle de versão: **centralizado** (Subversion) e **distribuído** (Mercurial e o Git).
- Tanto o controle de versão centralizado quanto o distribuído possuem repositórios e áreas de trabalho.
- A diferença está em como cada uma dessas partes está arranjada.



No controle de versão **centralizado** há um único repositório e várias cópias de trabalho que se comunicam apenas através do repositório central.

Centralizado

(exemplo: svn, cvs)

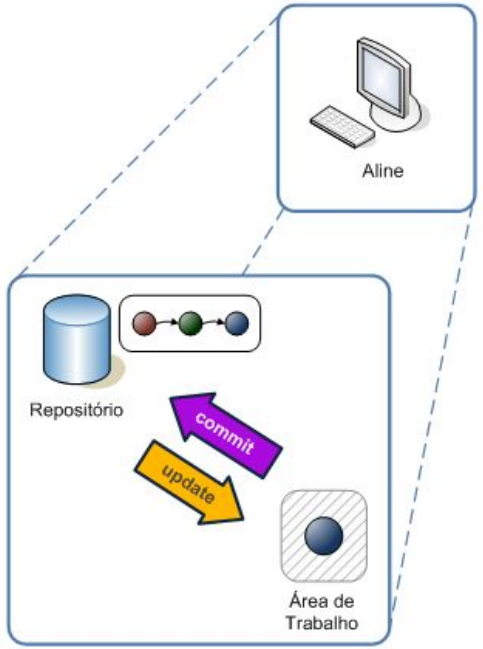


Centralizado

(exemplo: svn, cvs)

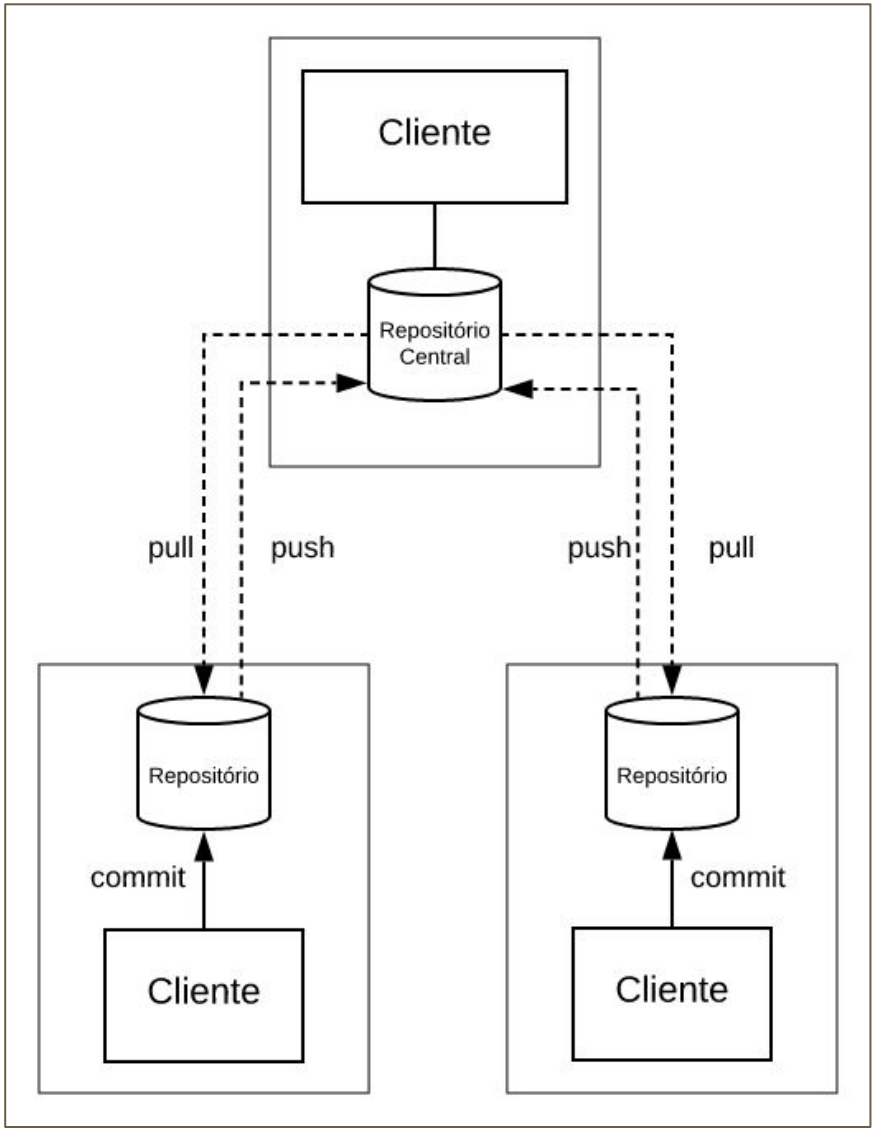
Controle de Versão Centralizado

- 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.



No controle de versão **distribuído** cada desenvolvedor possui um repositório próprio acoplado a uma área de trabalho. A comunicação entre eles continua sendo através de `commit` e `update`.

Distribuído
(exemplo: git, mercurial)



Distribuído

(exemplo: git, mercurial)

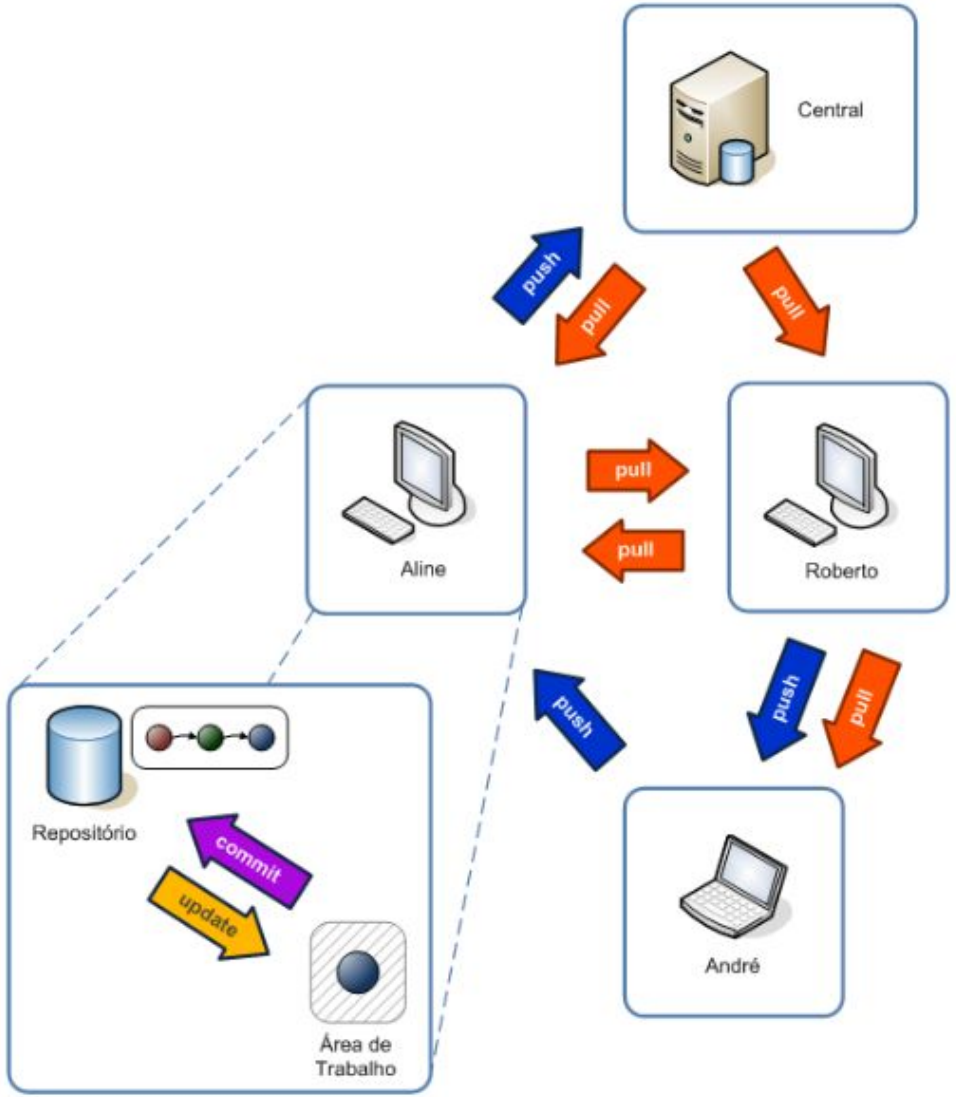
Controle de Versão Distribuído

- 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

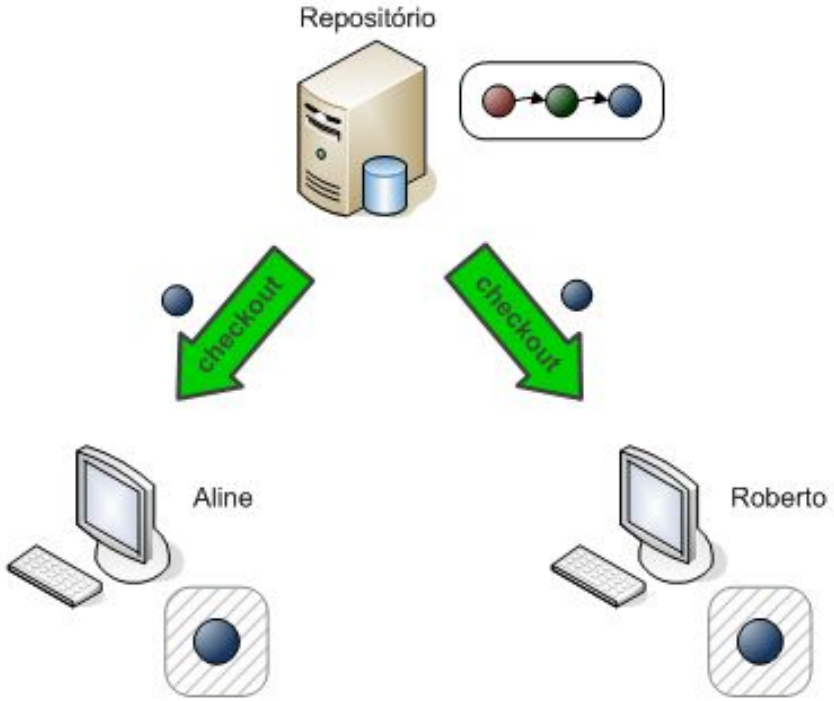
- Um repositório pode se comunicar com qualquer outro através das operações básicas *pull* e *push*:
- ***pull*** (Puxar). Atualiza o repositório local (destino) com todas as alterações feitas em outro repositório (origem).
- ***push*** (Empurrar). Envia as alterações do repositório local (origem) para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

Controle de Versão

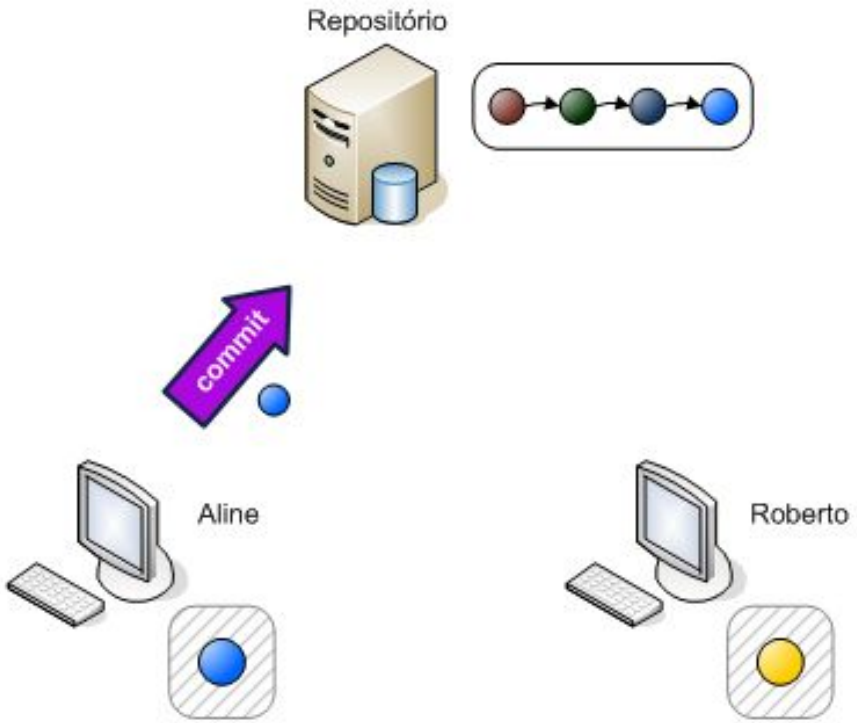


Resumo das Operações Básicas dos Controles de Versão Centralizado e Distribuído

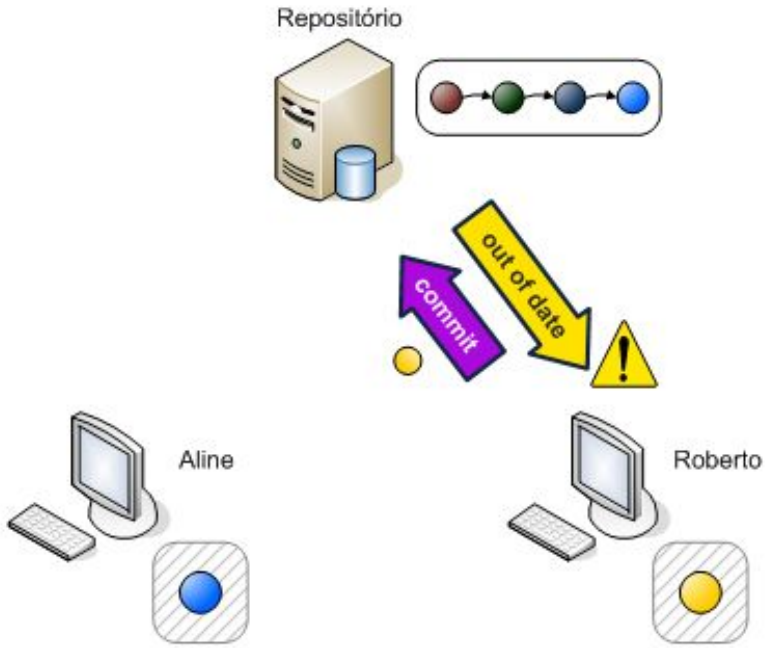
Centralizado	Distribuído	Descrição
`checkout`	`clone`	criação da cópia de trabalho/repositório
`commit`	`commit`	envia alterações para o repositório, criando uma revisão
`update`	`update`	atualiza a cópia/área de trabalho em uma revisão
	`pull`	importa revisões feita em outro repositório
	`push`	envia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



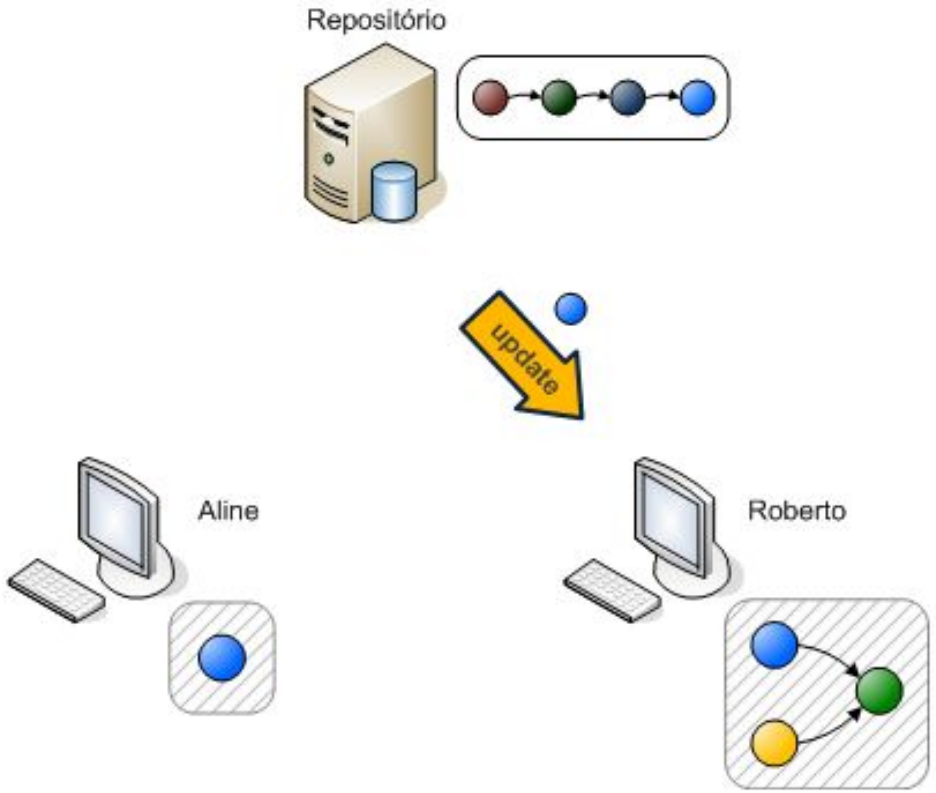
1. Duas cópias de trabalho são criadas a partir do comando `checkout`. As duas iniciam no mesmo estado.



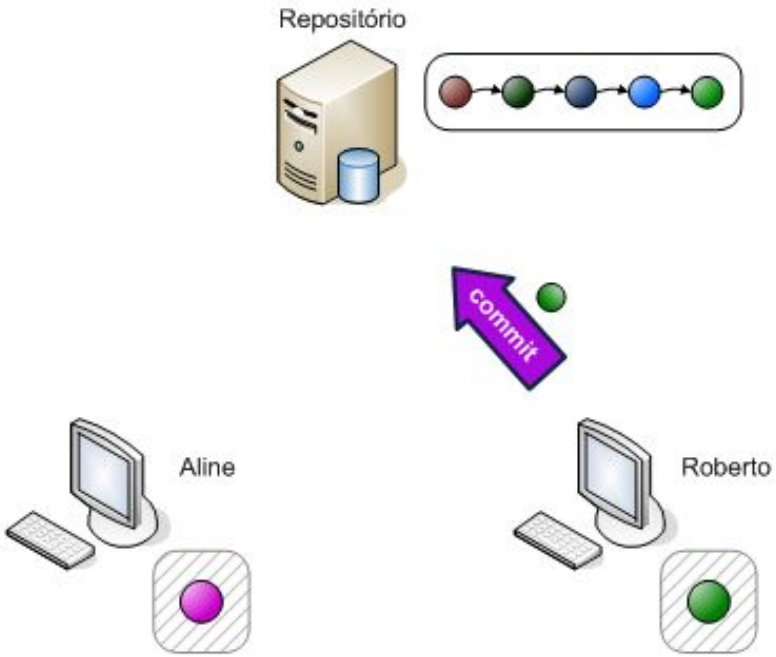
2. Os dois desenvolvedores executam modificações nas suas cópias de trabalho, mas Aline publica antes no repositório.



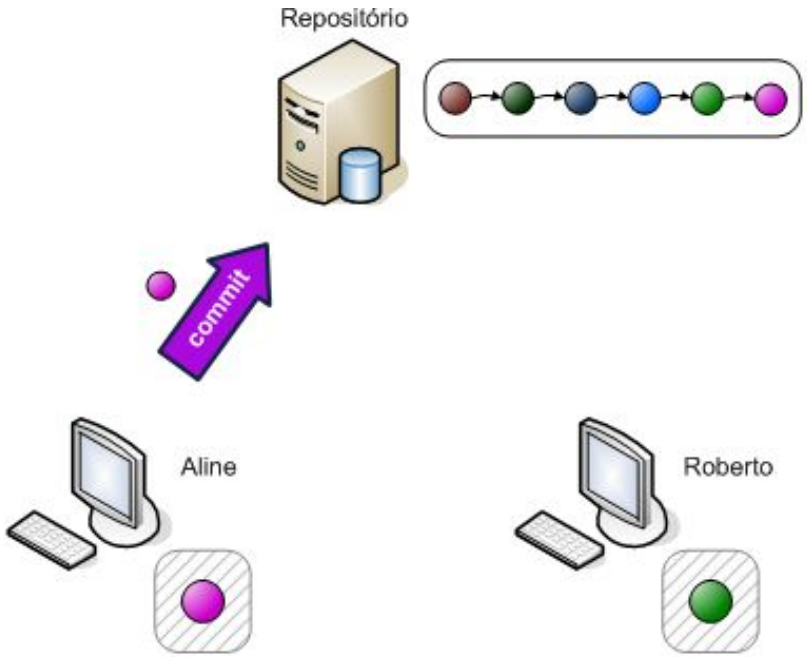
3. Roberto tenta publicar suas alterações, mas o controle de versão recusa justificando que as alterações foram baseadas em arquivos desatualizados. No caso, um ou mais arquivos alterados por Roberto já haviam sido alterados por Aline antes.



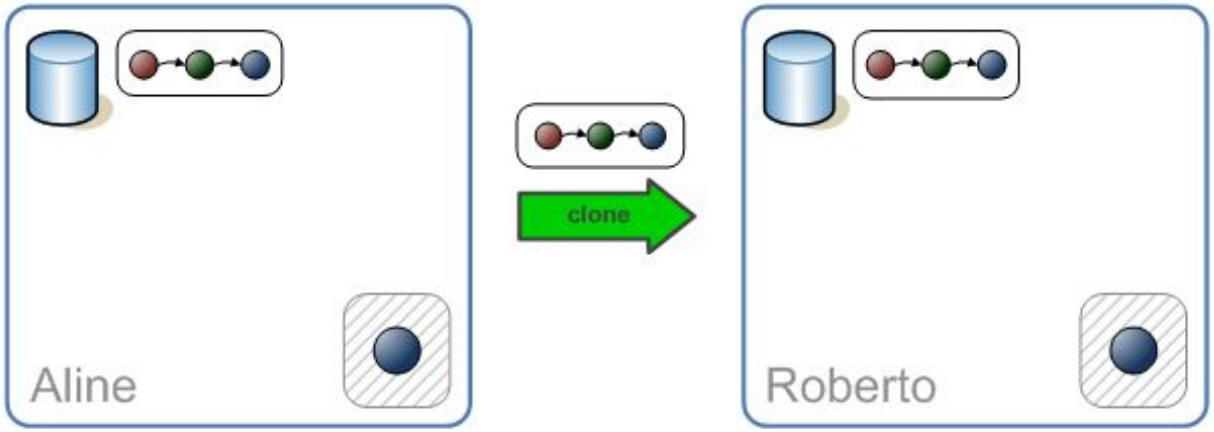
4. Na atualização da cópia de trabalho, o controle de versão já mescla automaticamente as revisões.



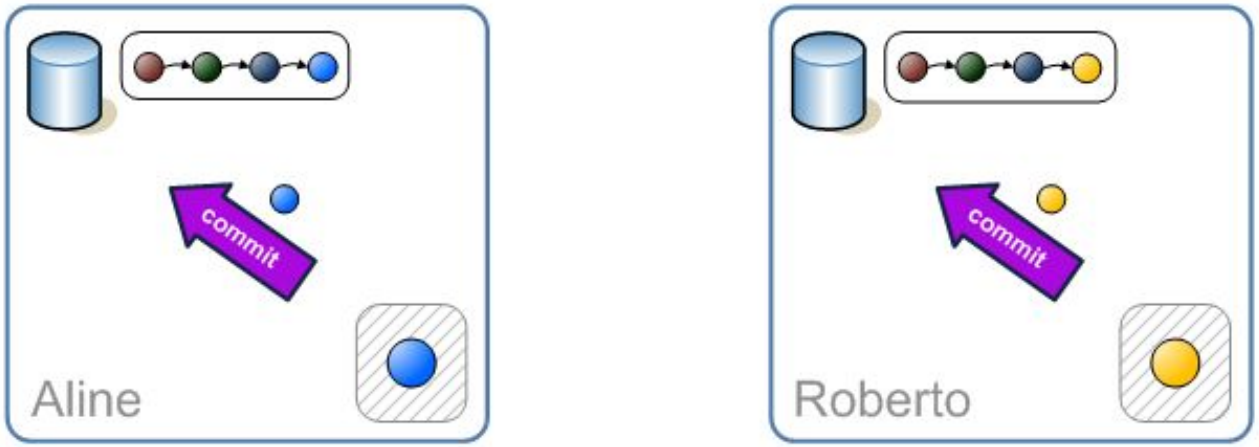
5. Após conferir se a atualização e a mesclagem produziram o resultado desejado, Roberto envia as mudanças ao repositório. Enquanto isso, Aline já trabalha em outra tarefa, executando novas alterações.



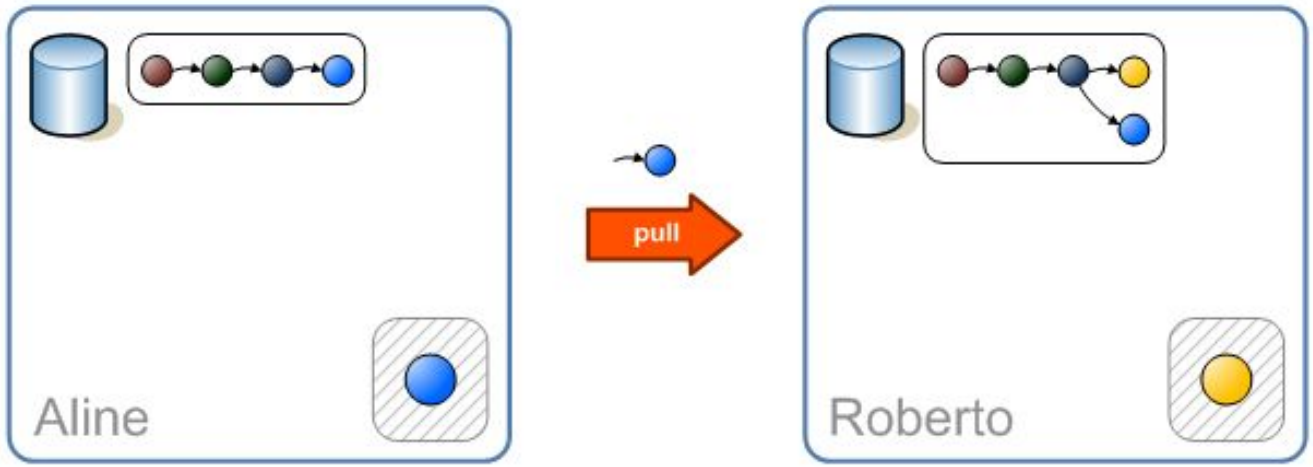
6. O `commit` de Aline pode ser aceito sem problema se nenhuma das revisões que vieram depois da atualização da cópia de trabalho tiver alterado os mesmos arquivos que Aline. É uma situação possível de acontecer, mesmo que não seja comum.



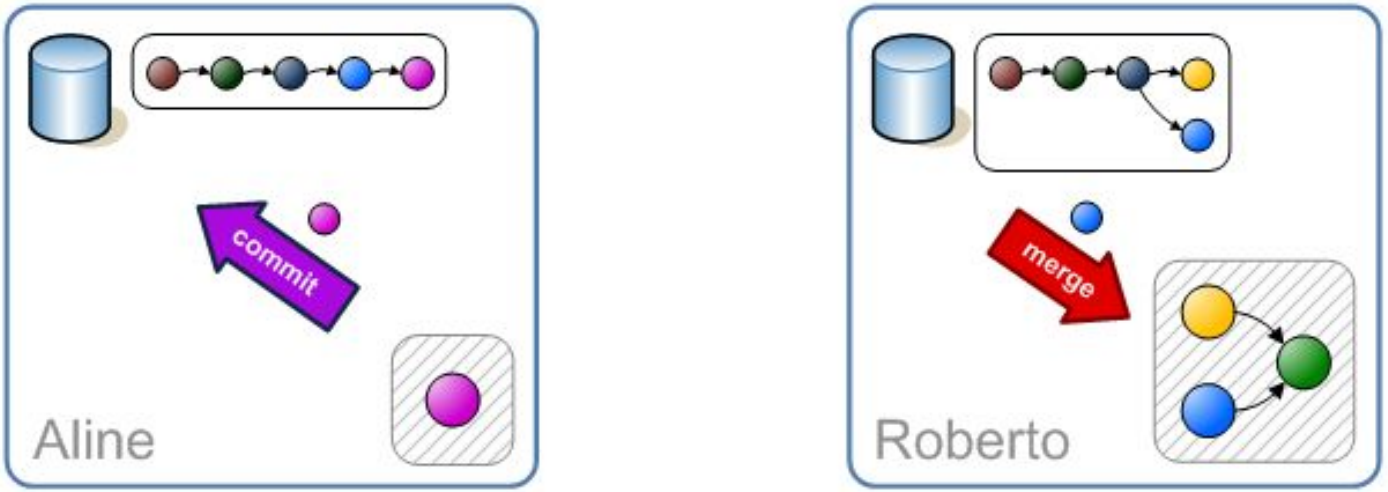
1. Roberto clona o repositório de Aline. Agora, ambos partem do mesmo ponto.



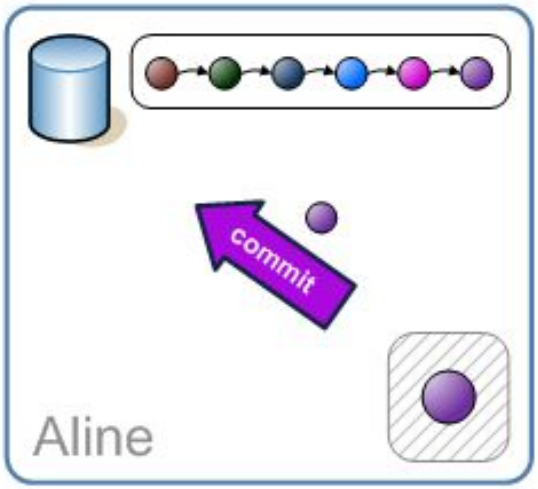
2. Aline e Roberto publicam suas alterações nos seus respectivos repositórios, sem interferir no repositório um do outro.

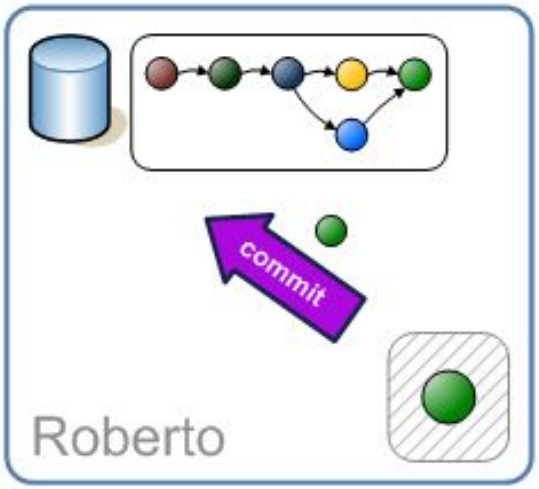


3. Roberto sincroniza seu repositório com as revisões publicadas por Aline. Sua área de trabalho não é afetada pela sincronização.

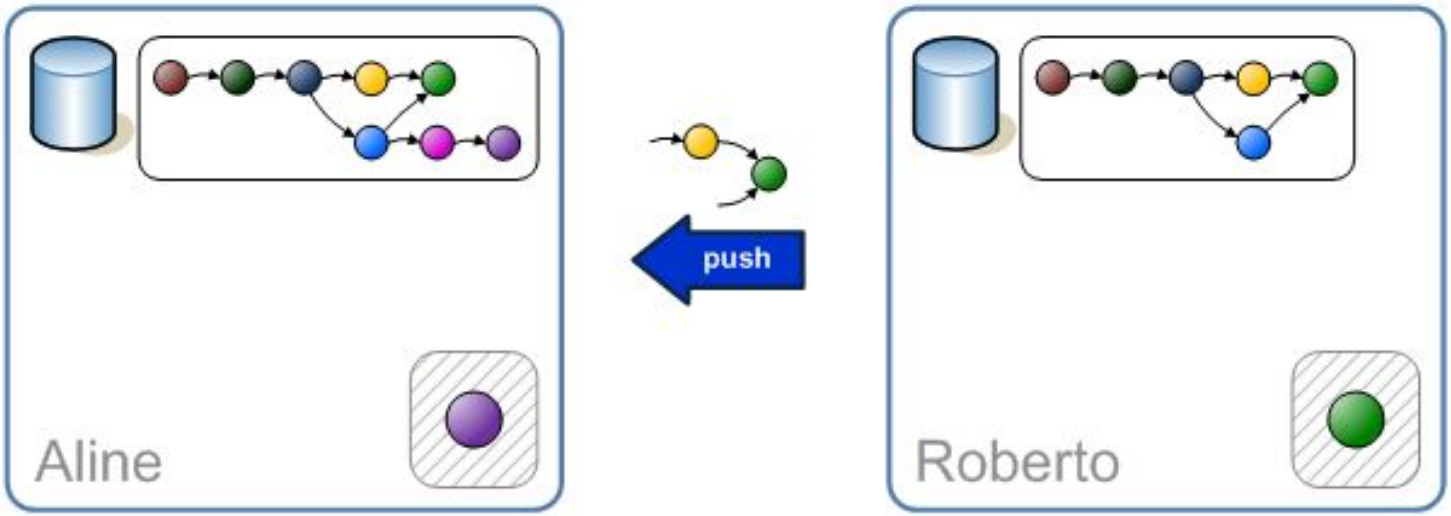


4. A mesclagem entre as revisões de Aline e Roberto é feita explicitamente na área de trabalho de Roberto através de um comando `merge`. Enquanto isso, Aline já gera outra revisão no seu repositório.





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.



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

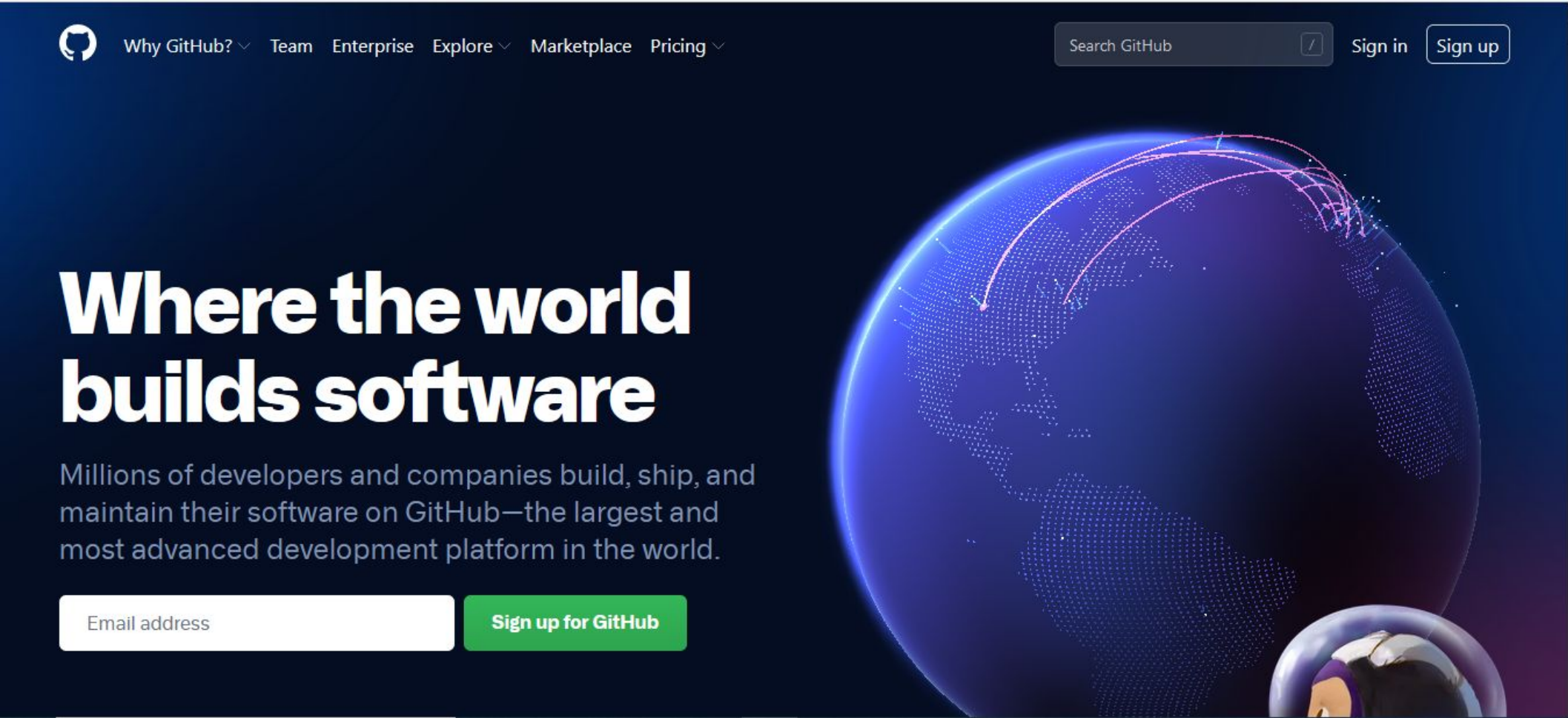
Git - Sistema de Controle de Versão

- Um dos mais modernos e mais utilizados atualmente;
- É um projeto de código aberto – Linus Torvalds;
- Sistema de controle de versão distribuído, diferente do CVS e Subversion;

Comandos Básicos do Git

- ***git init*** – Criar um novo repositório;
- ***git status*** – Verificar o status do repositório;
- ***git add*** – Adicionar arquivos no diretório;
- ***git commit*** – Um pacote de alterações feitas no repositório;
- ***git push*** – Enviar arquivos/diretórios para o repositório remoto;
- ***git pull*** – Atualizar o repositório local de acordo com o repositório remoto;

GitHub


Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
/
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
Sign up for GitHub

Projetos no GitHub

CodeIgniter 4

https://github.com/codeigniter4/CodeIgniter4

Arduino

https://github.com/arduino/Arduino

Vamos criar contas no GitHub.. caso ainda não tenham, e criar nosso primeiro repositório.

Tarefa

Criem suas contas:
https://docs.github.com/pt/get-started/start-your-journey/creating-an-account-on-github

Criem o primeiro repositório e façam o clone para máquina local:
https://docs.github.com/pt/repositories/creating-and-managing-repositories/quickstart-for-repositories

Façam uma alteração no repositório local e realizem o *commit* enviando a alteração para o repositório no GitHub.