Engenharia de Software II / Qualidade e Teste de Software

Aula 05: Controle de Versões

Breno Lisi Romano

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)
Tecnologia em Sistemas para Internet – TSI (QTSI6)

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Revisão: Teste de Software

- Representam uma oportunidade de detectar **defeitos** antes do software ser entregue aos usuários
- A atividade de testes pode ser feita de forma **manual** e/ou **automática** e tem por objetivos:
  - **Detectar Erros** para **Eliminar** os **Defeitos** e **Evitar** as **Falhas**
  - **Produzir casos de teste** que tenham elevadas probabilidades de **revelar um defeito ainda não descoberto**, com uma quantidade mínima de tempo e esforço
  - **Comparar o resultado dos testes com os resultados esperados** → produzir uma indicação da qualidade e da confiabilidade do software. Quando há diferenças, inicia-se um processo de depuração para descobrir a causa

Revisão: Estágios de Teste

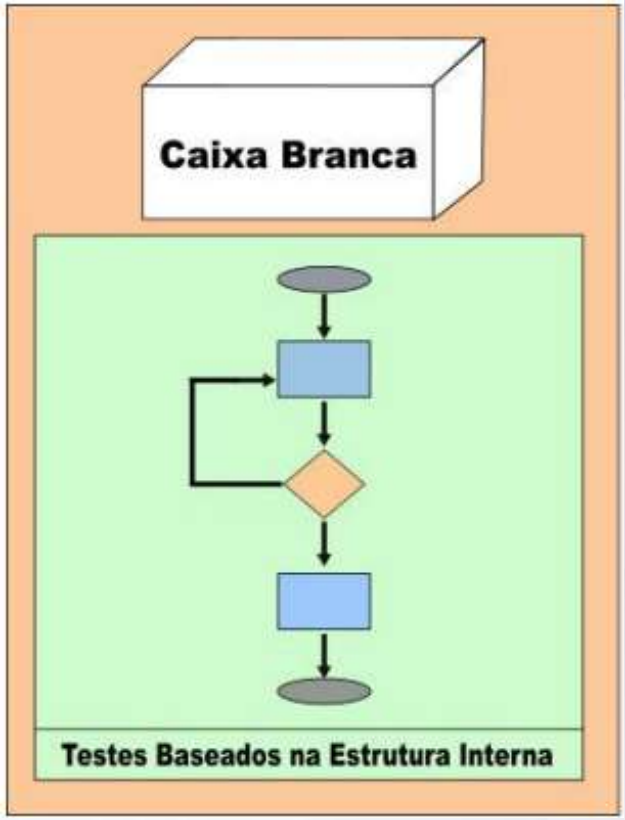
- Existem diferentes estágios de teste associados ao desenvolvimento de um produto de software:
  - Teste de Unidade
  - Teste de Integração
  - Teste de Sistemas
  - Teste de Aceitação (Homologação)

Revisão: Abordagens de Teste

Abordagem Funcional (Caixa Preta):

-

Abordagem Estrutural (Caixa Branca):

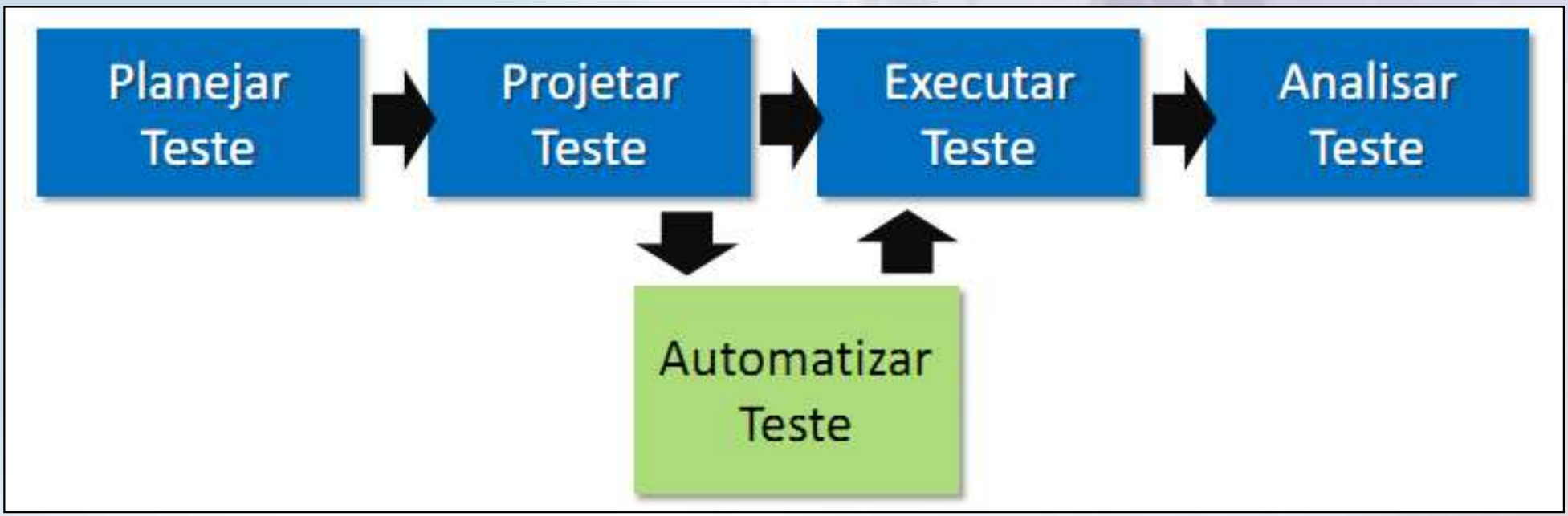


-

Revisão: Tipos de Teste de Software

- Existem vários tipos de teste que podem ser executados nos diversos estágios de teste e utilizando as diferentes abordagens existentes:
  - **Teste de Funcionalidade**
  - **Teste de Recuperação de Falhas**
  - **Teste de Segurança de Acesso**
  - **Teste de Carga**
  - **Teste de Desempenho**
  - **Teste de Portabilidade**
  - **Teste de Regressão**

Revisão: Processo Básico de Teste de Software



Revisão: Projetar Casos de Teste

Caso de Teste: CT#01: Débito em Conta com Sucesso	
Descrição do Caso de Teste:	**Resultados Esperados:**
Este caso de teste verifica o débito em uma conta válida e com saldo positivo, permitindo a retirada da quantia solicitada.	O débito de R$200,00 na conta será permitido! Novo Saldo da Conta: R$450,00 Mensagem Apresentada ao Usuário ao Realizar o Débito: O Débito foi realizado com sucesso.
Entradas:	
Número da Conta: 22321-3	
Saldo da Conta: R$650,00	
Valor do Débito: R$200,00	

Controle de Versões

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Mudanças (1)

- "*Não há nada permanente, exceto a mudança*" (Heráclito, 500 a.C)

- A maioria das modificações no software é justificável
  - Não vale a pena se queixar delas
  - Certifique-se de que dispõe de mecanismos para cuidar delas

Mudanças (2)

- Origens Comuns das Modificações:
  - Novas condições de negócio ou mercado
  - Novas necessidades do cliente
  - Reorganização ou crescimento / diminuição dos negócios
  - Restrições de orçamentos ou cronogramas
  - **Defeitos encontrados com a realização da Atividade de Testes de Software**

Motivação: Uso de Sistemas para Controle de Versões

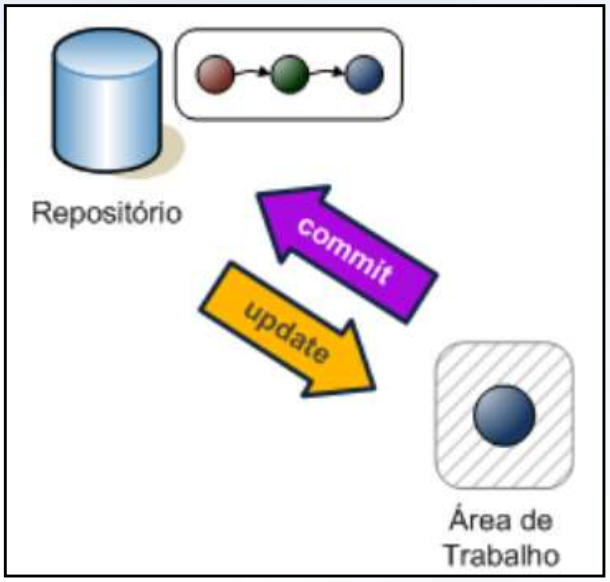
- Muitos problemas de **desenvolvimento de software** são causados por **falta de controle de versão e mudanças**
- Avalie o local que vocês trabalham:
  - Alguém já sobrescreveu o código de outra pessoa por acidente e acabou perdendo as alterações?
  - Tem dificuldades em saber quais as alterações efetuadas em um software, 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
- Melhoramos o ambiente de trabalho com o uso de Sistemas de Controle de Versões
  - Centralizado (Subversion) e Distribuído (Mercurial e o Git)
- Disciplina de **Gerenciamento de Configuração e Mudança** no RUP lida com isto → Base para Realização bem sucedida de Teste de Software

Para que Serve o Controle de Versão?

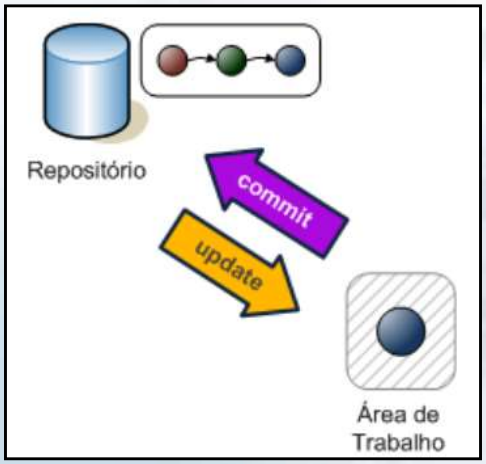
- **Registro do Histórico:** Registra toda a evolução do projeto, cada alteração sobre cada arquivo
  - Sabe-se quem fez o que, quando e onde
  - Permite reconstruir uma revisão específica do arquivo sempre que desejado
- **Colaboração Concorrente**: Possibilita que vários desenvolvedores trabalhem em 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

Como funciona um Sistema de Controle de Versões? (1)

- **Composto** de duas partes: o **Repositório** e a **Área de Trabalho**
  - **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 arquivo do repositório → Usa uma Área de trabalho
  - **Área de Trabalho (*Workspace*):** contém a cópia dos arquivos do projeto e que é monitorada para identificar as mudanças realizadas
    - É individual e isolada das demais áreas de trabalho



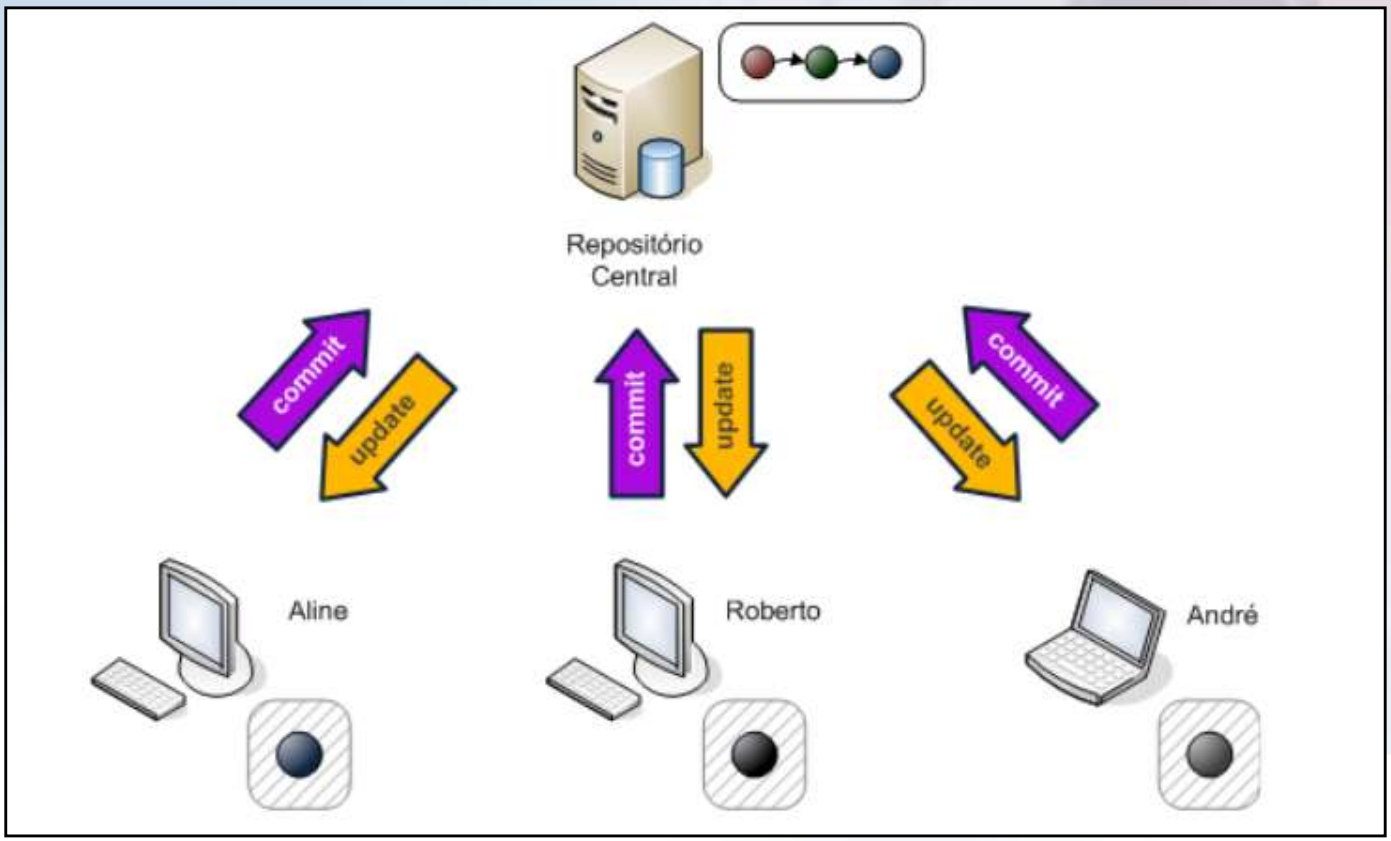
Como funciona um Sistema de Controle de Versões? (2)



- Sincronização entre a área de trabalho e o repositório é feita através dos comandos **commit** e **update**
  - **Commit:** envia um pacote contendo uma ou mais modificações feitas na área de trabalho (origem) ao repositório (destino)
    - Gera uma nova revisão no repositório, contendo as modificações feitas, data e autor
    - Revisão é 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 → Histórico do Projeto
  - **Update**: faz o inverso, isto é, envia as modificações contidas no repositório (origem) para a área de trabalho (destino)
- Tanto o controle de versão centralizado quanto o distribuído possuem Repositórios e Áreas de trabalho → Diferença está em como cada uma dessas partes está arranjada

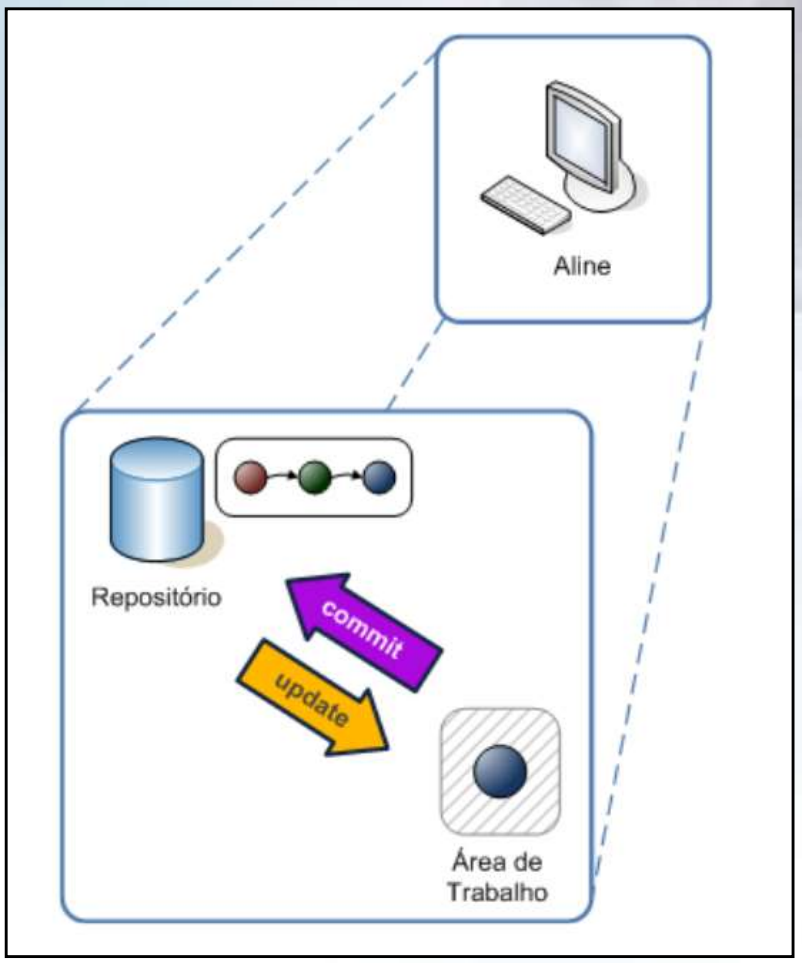
Controle de Versões Centralizado

- **Único repositório centralizado com várias cópias de Áreas de Trabalho**
  - Comunicação entre uma área de trabalho e outra passa obrigatoriamente pelo repositório central



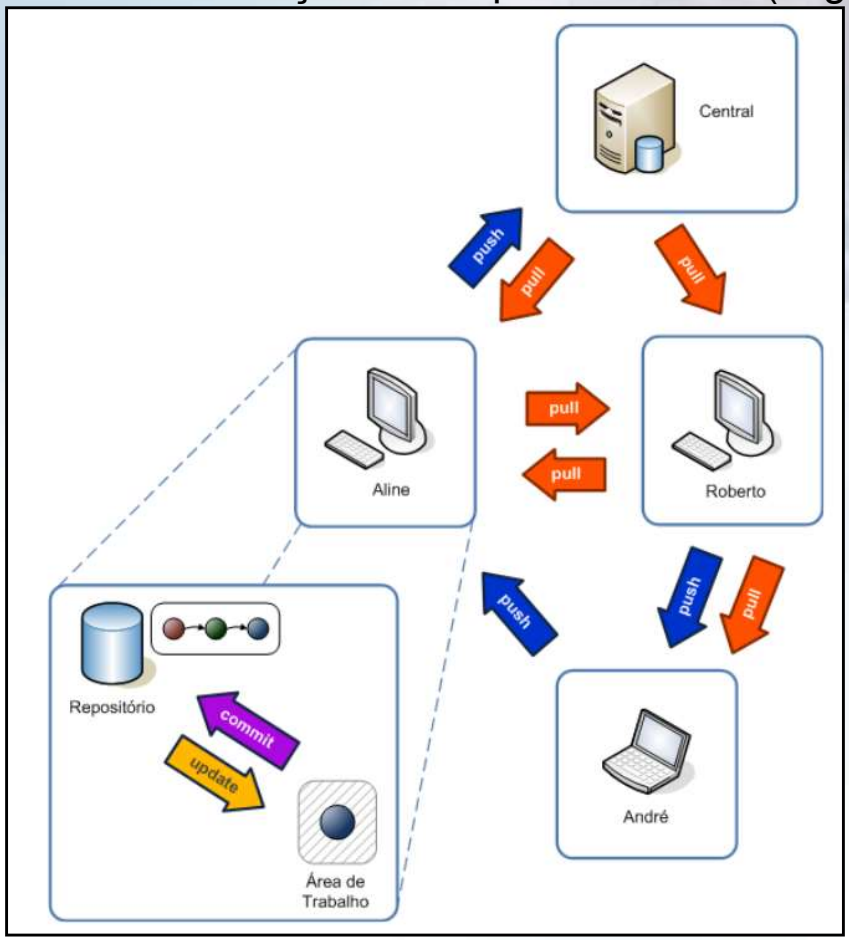
Controle de Versões Distribuído (1)

- **Vários repositórios autônomos e independentes, um para cada desenvolvedor**
  - Cada **Repositório** possui uma **Área de Trabalho** acoplada e as Operações **commit** e **update (ou pull)** acontecem **localmente** entre os dois



Controle de Versões Distribuído (2)

- Um Repositório pode se comunicar com qualquer outro através das operações básicas: **Pull** e **Push**
  - **Pull (Puxar) ou Fetch:** 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)

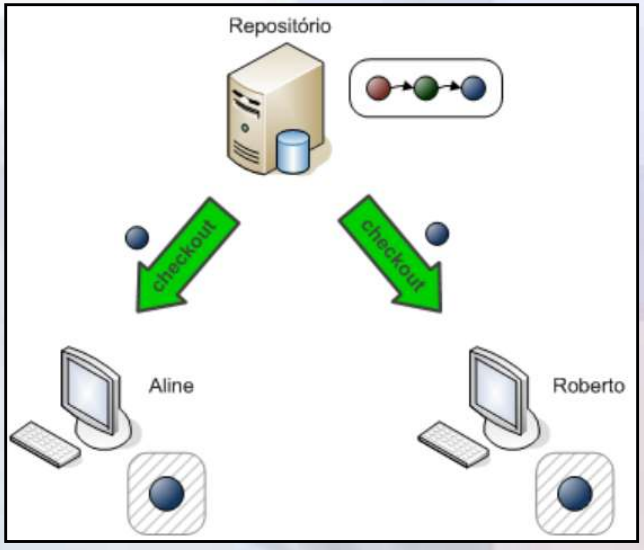


Resumo das Operações Básicas dos Controles de Versão Centralizado e Distribuído

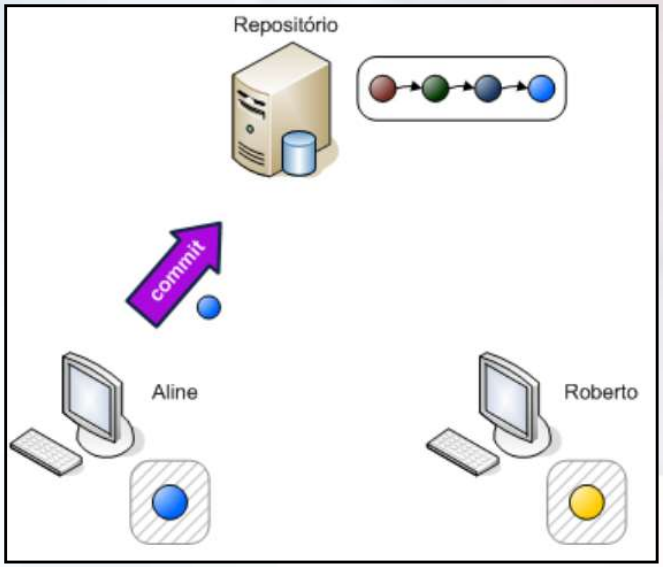
Centralizado	Distribuído	Descrição
`checkout`	`clone`	criação da cópia de trabalho/repositório
`commit`	`commit`	envia alterações para o repositório, criando uma revisão
`update`	`update`	atualiza a cópia/área de trabalho em uma revisão
	`pull`	importa revisões feita em outro repositório
	`push`	envia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

Sincronização de Mudanças: Sistema de Controle de Versões Centralizado (1)

- Cópias de trabalho são criadas a partir do comando **Checkout**

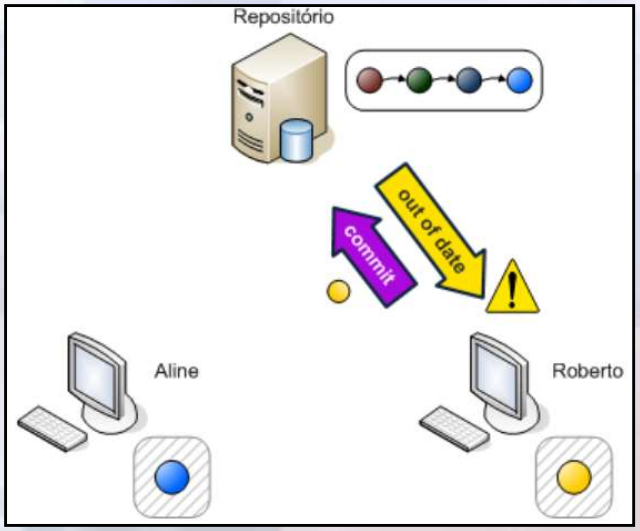


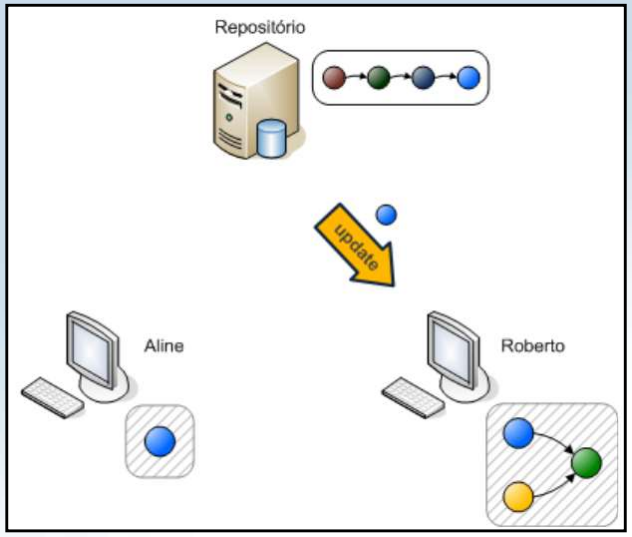
- Dois desenvolvedores executam modificações nas suas cópias de trabalho, mas Aline realiza o **commit** antes no Repositório Centralizado



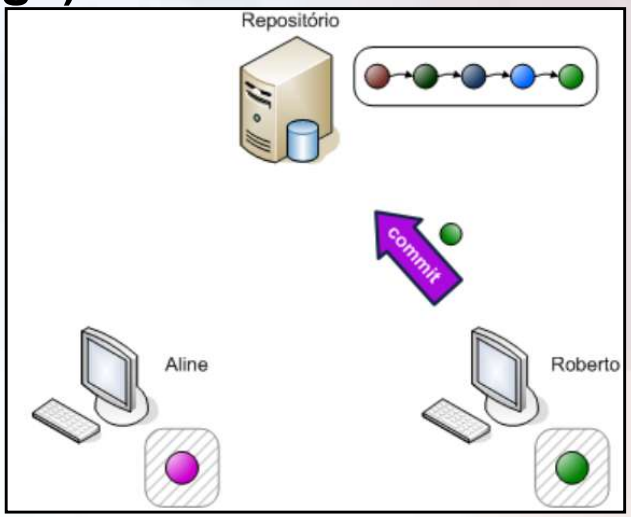
Sincronização de Mudanças: Sistema de Controle de Versões Centralizado (2)

- 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





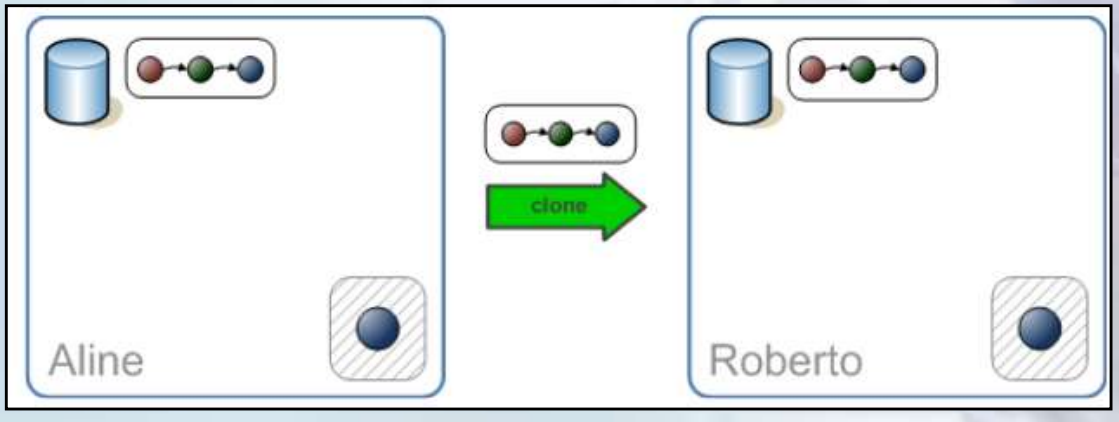
- Na **atualização da cópia de trabalho**, o controle de versão já **mescla automaticamente as revisões (Operação Merge)**



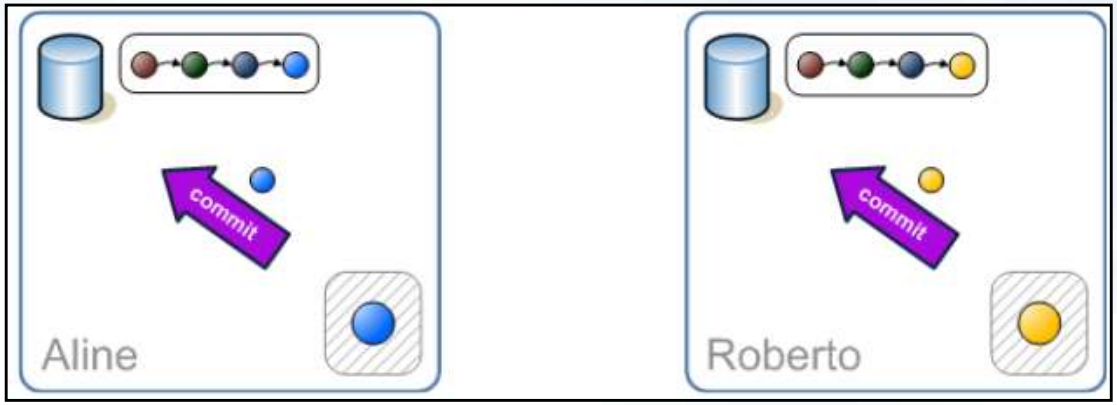
- Após **conferir** se a **atualização** e a **mesclagem** produziram o resultado desejado, Roberto realiza o **commit** as **mudanças** ao repositório

Sincronização de Mudanças: Sistema de Controle de Versões Distribuído (1)

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

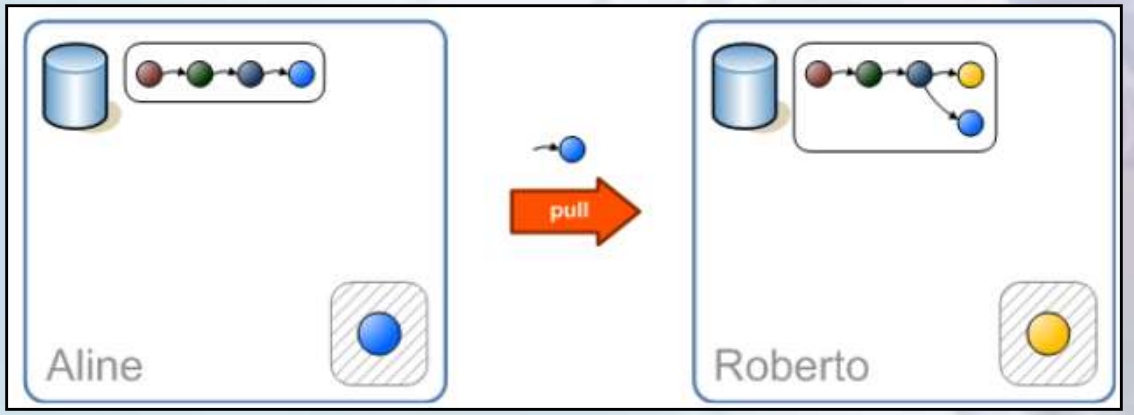


- Aline e Roberto realizam o **commit de suas alterações** nos seus **respectivos repositórios**, **sem interferir** no **repositório** um do outro

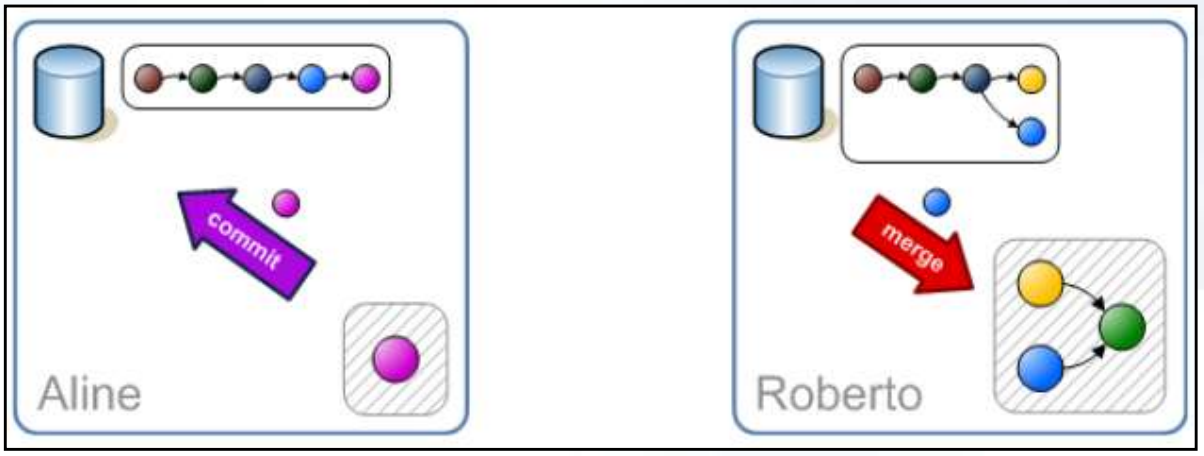


Sincronização de Mudanças: Sistema de Controle de Versões Distribuído (2)

- Roberto **sincroniza** seu repositório com as revisões publicadas por Aline **(Operação Pull)** . Sua área de trabalho não é afetada pela sincronização

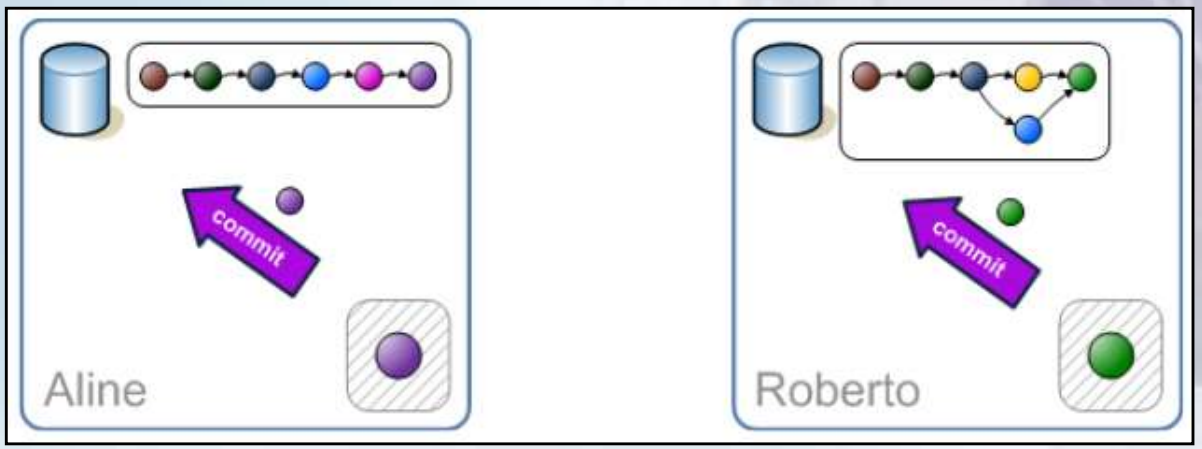


- A **mesclagem entre as revisões** de Aline e Roberto é feita **explicitamente na Área de Trabalho** de Roberto através de um comando **Merge. Enquanto isso,** Aline já **gera outra revisão** no seu **Repositório**

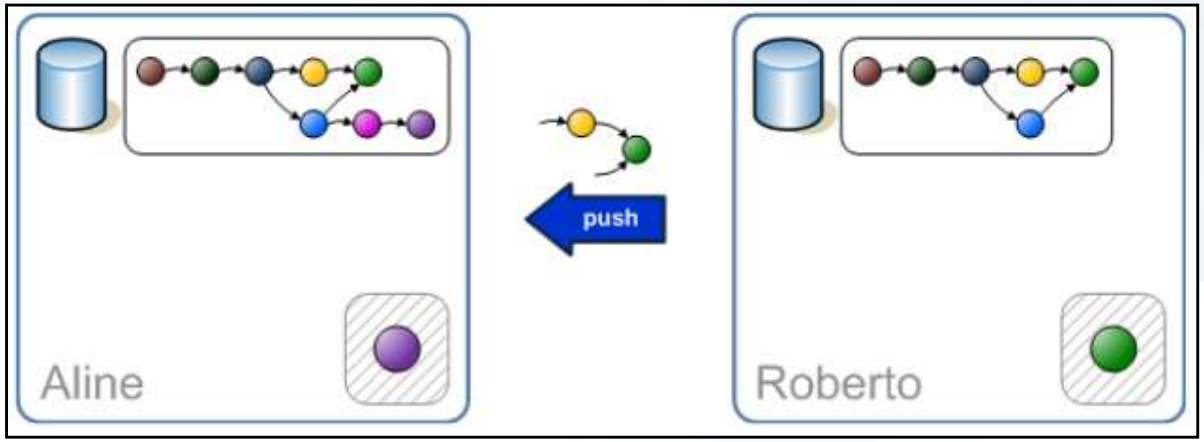


Sincronização de Mudanças: Sistema de Controle de Versões Distribuído (3)

- 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**



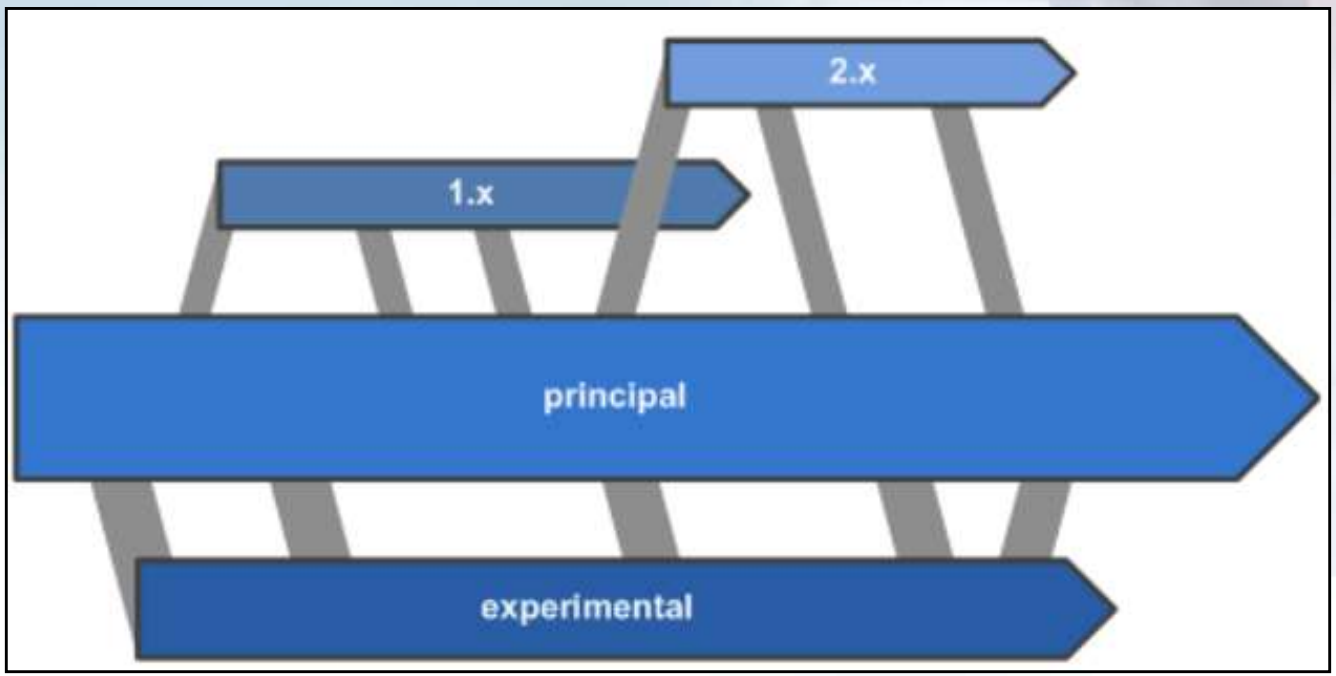
- Roberto **envia suas revisões** ao **repositório** de Aline (**Operação Push**), que as **combina com o histórico de revisões** já existente



Diferentes Versões do Projeto (Conceito de Branch)

- 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

- O **controle de versão** oferece funcionalidades que facilitam a **coordenação de Ramos (Branch) diferentes** de desenvolvimento em um mesmo projeto



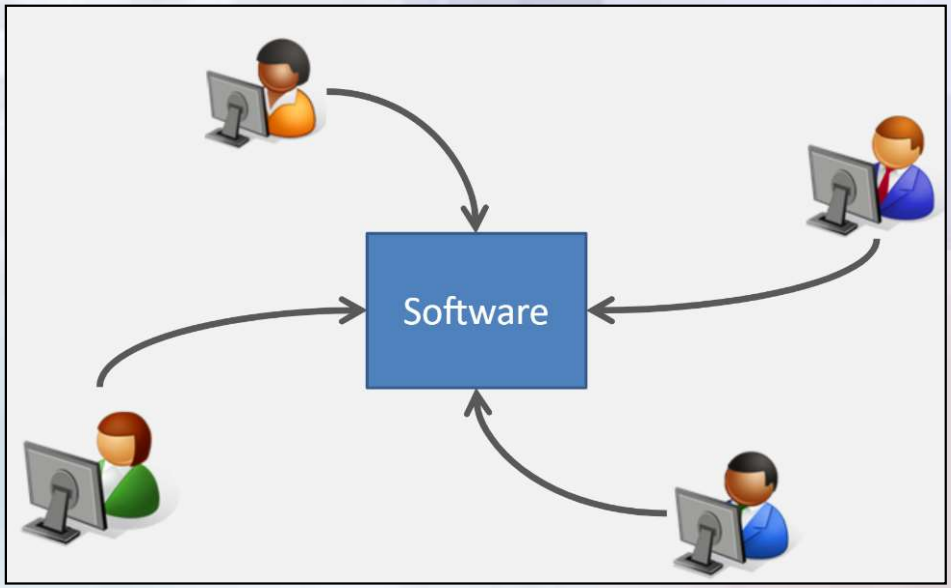
Controle de Versões: Subversion (SVN)

Repositório: https://svn.sbv.ifsp.edu.br/svn/esw-tsw2024/

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Controle de Versões: Subversion (SVN)

- Manutenção do histórico de modificações de artefatos na nuvem
- Possibilidade de criação de linhas paralelas de desenvolvimento
- Controle de acesso concorrente



https://tortoisesvn.net/downloads.html

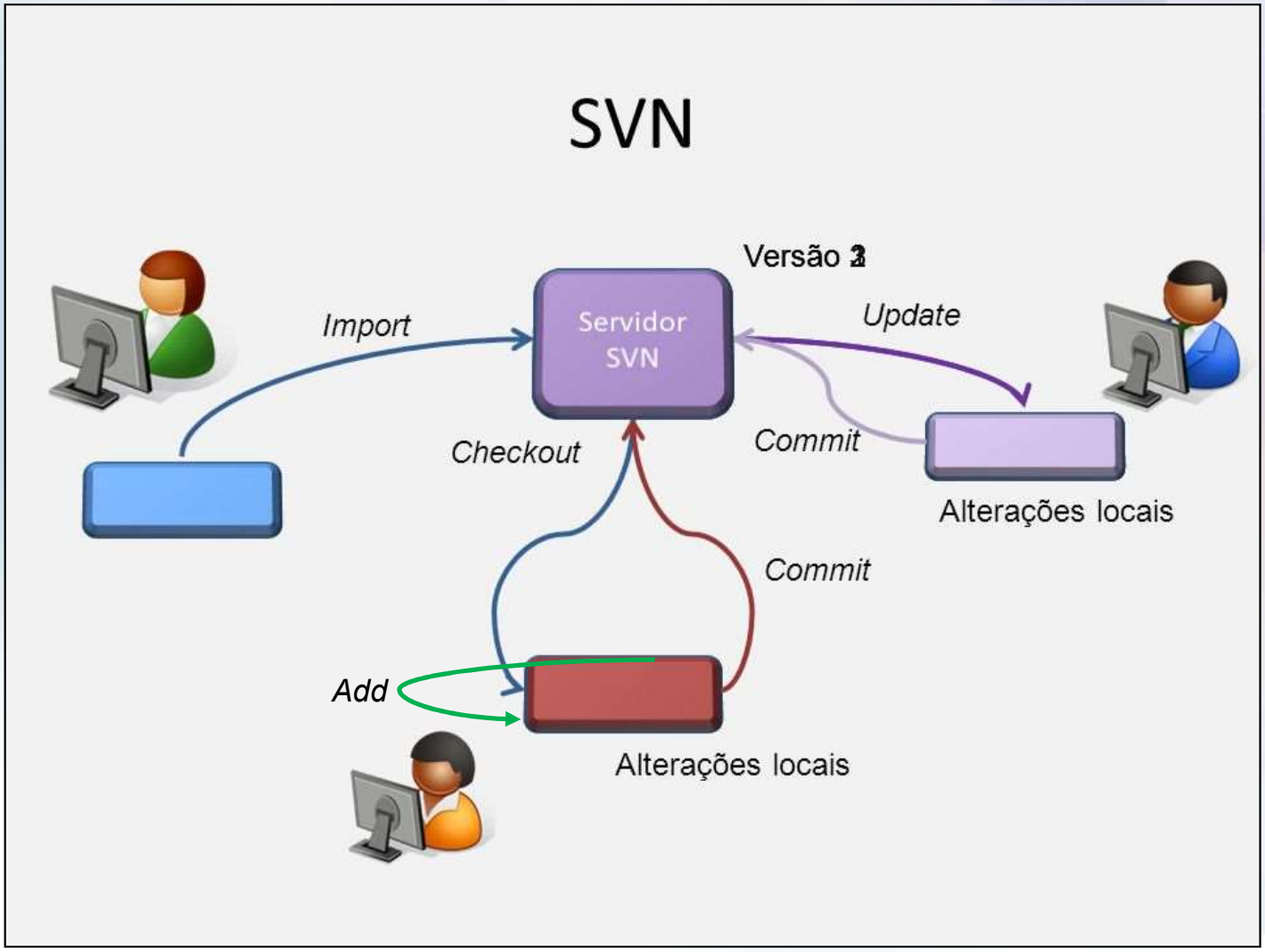
SUBVERSION

Servidor

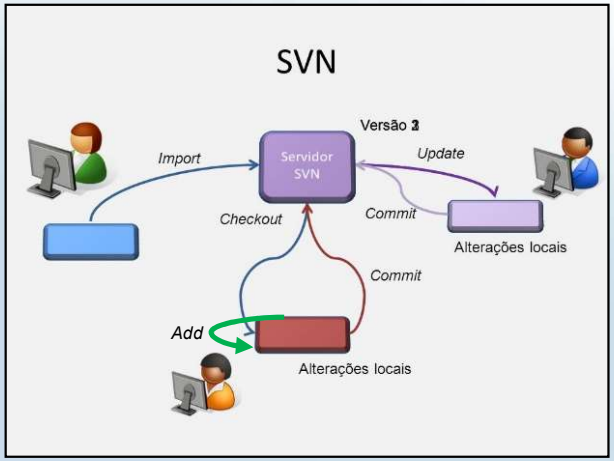
TortoiseSVN

Cliente SVN

Subversion (SVN): Operações Principais (1)



Subversion (SVN): Operações Principais (2)

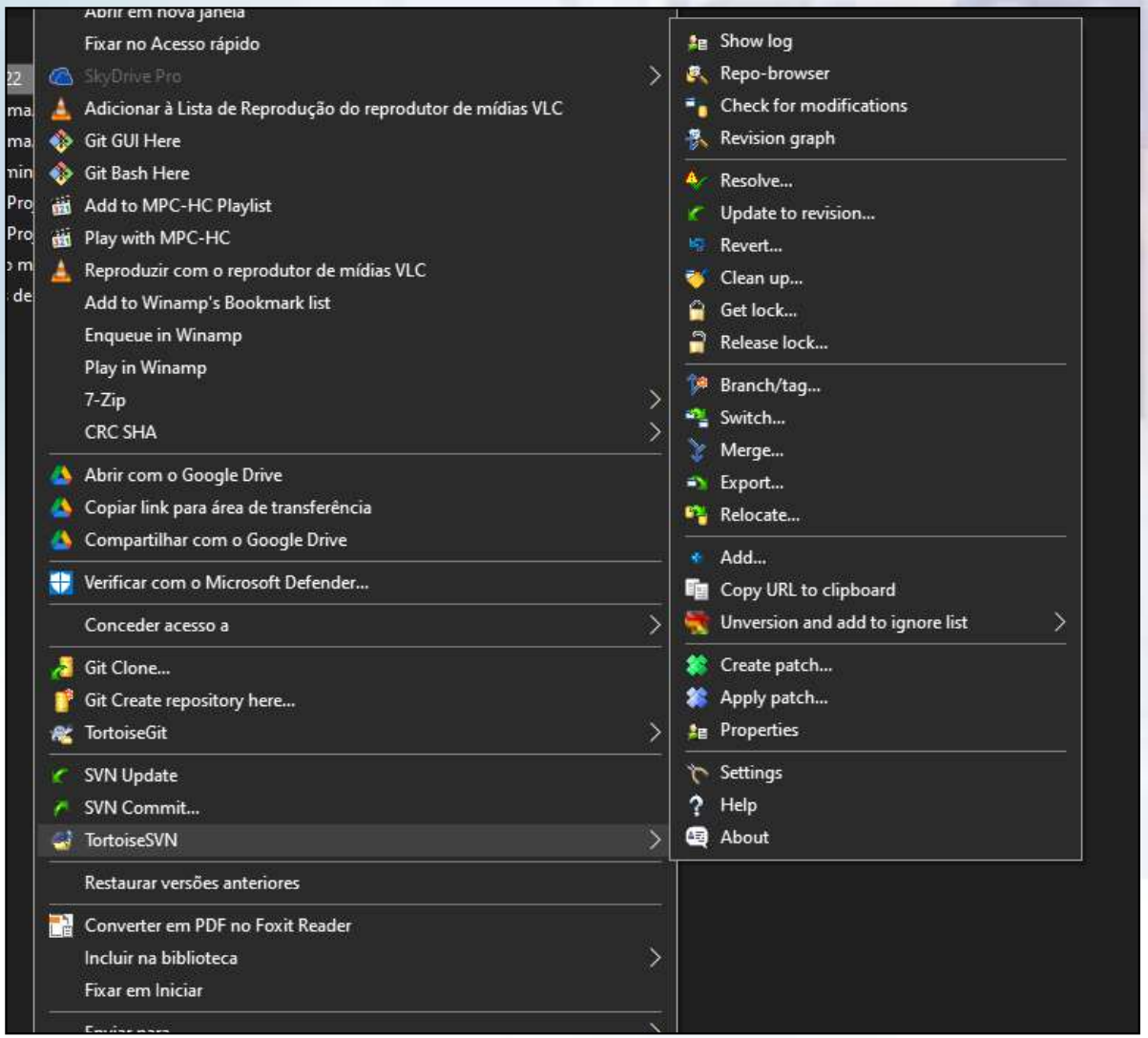


Resumo das principais operações:

- ***Import***: Iniciar o versionamento de um conjunto de arquivos (Upload para o SVN) – Já fizemos
- ***Checkout***: Importar o estado atual do repositório para uma cópia local, sob controle de versão
- ***Update***: Atualiza o estado da cópia local para concordar com mudanças no repositório (Ex.: Adicionando, Removendo, Modificando Arquivos...)
  - Primeira coisa a ser feita antes de iniciar os trabalhos
- ***Commit***: Postar suas modificações no repositório oficial
- ***Add***: Adicionar arquivos/pastas ao versionamento. A modificação é valida a partir do próximo commit realizado
- ***Delete***: Remover arquivos do repositório

Subversion (SVN): Operações Principais (3)

- Para acessar as operações no Computador, utilizar o cliente Tortoise SVN, com o botão direito do mouse em uma pasta:



Breve Explicação sobre as Ferramentas Utilizadas no Projeto - SVN

- **Subversion (SVN) – IFSP-SBV**
  - Controle de Versões dos arquivos desenvolvidos nos Projetos Práticos de Engenharia de Software
  - Repositório: https://svn.sbv.ifsp.edu.br/svn/esw-tsw2024/ **(USAR ESTE ENDEREÇO PARA CONFIGURAR O SVN NA SUA MÁQUINA)**

- Deixar um tempo para que todos verifiquem o acesso no Projeto do SVN

Simulação de Todas as Operações no Repositório do Projeto - SVN

- **Subversion (SVN)**
  - Explicar a arquitetura de um Controle de Versões
  - Mostrar como faz *checkout* do repositório (Cria-se o repositório pela primeira vez na máquina do desenvolvedor) – utilizar o TortoiseSVN
  - Mostrar como adiciona arquivos no repositório
  - Mostrar como faz *commit* do repositório (enviar os arquivos para o servidor)
  - Mostrar como atualiza o repositório (baixar a última versão dos arquivos do servidor para a máquina do usuário) – **PRIMEIRA COISA QUE FAZ ANTES DE TRABALHAR**
  - Mostrar o Relatório dos Arquivos Enviados ao Repositório (Show Logs)
  - Mostrar como limpa os dados de autenticação do SVN nas máquinas do Laboratório
- PRATICAR: Utilizar a pasta de Pratica para criar arquivos, realizar commits, updates, etc..

Controle de Versões: Git / Github

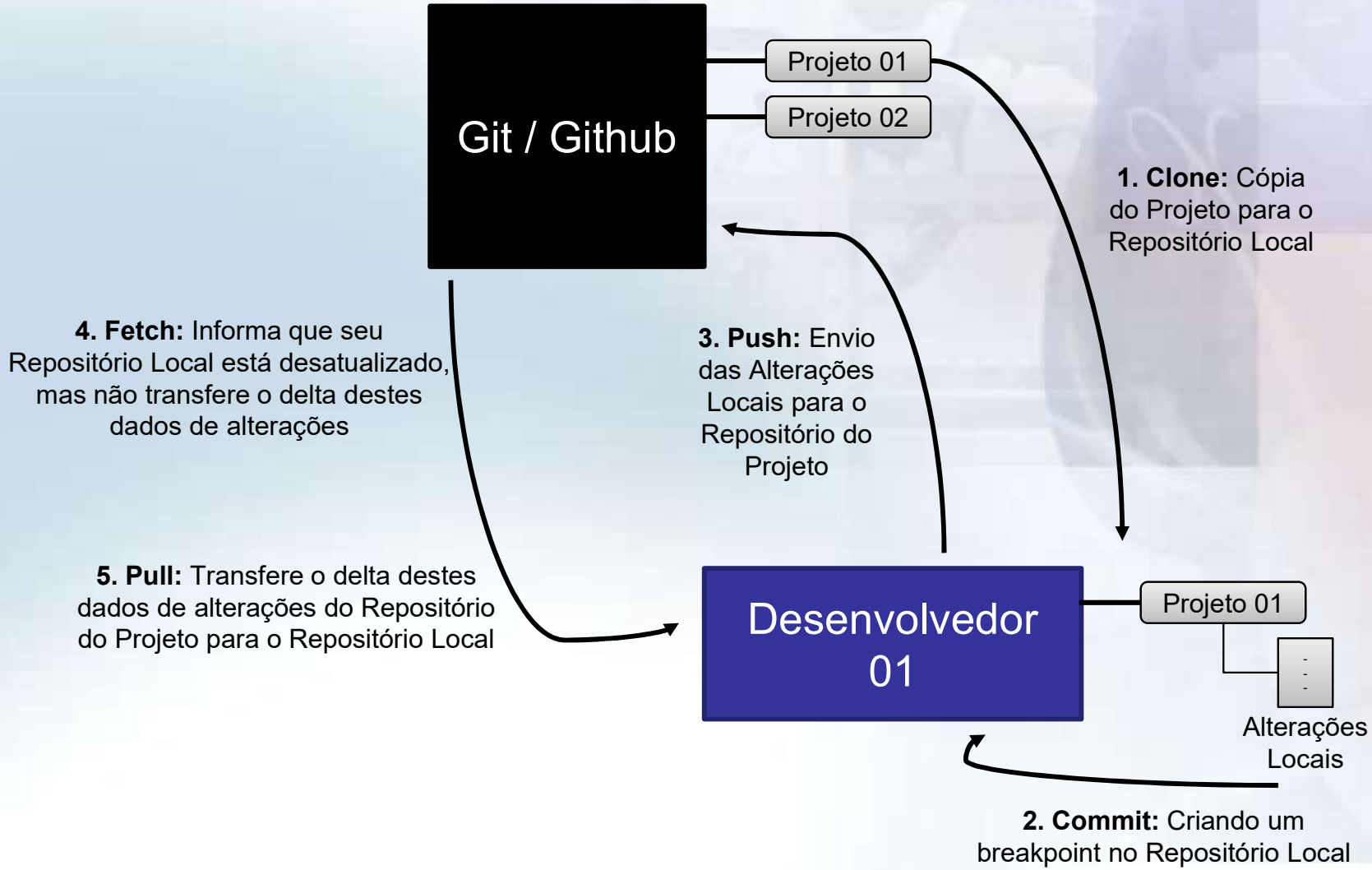
Repositório: https://github.com/blromano/esw-tsw2024.git

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

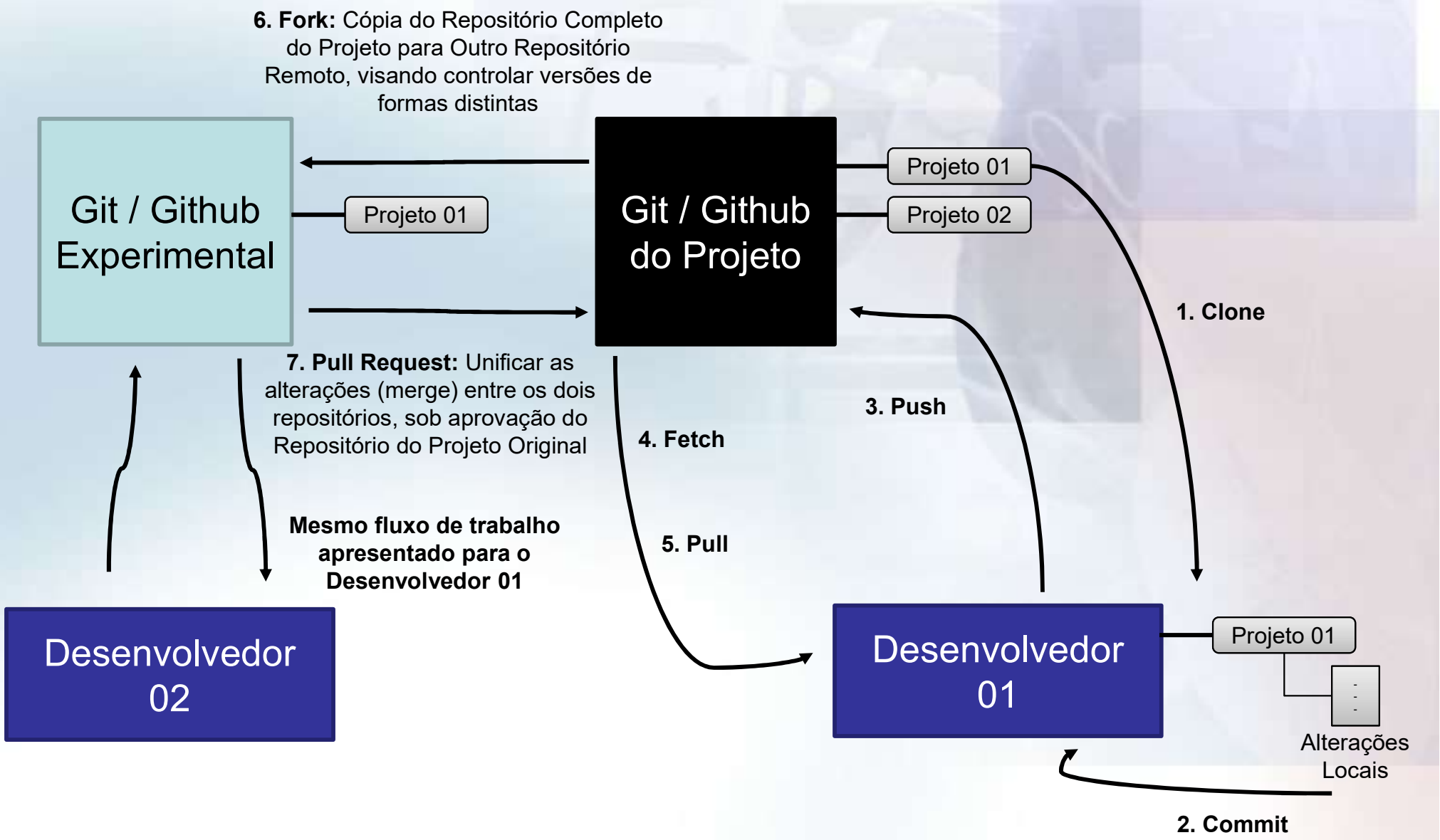
Orientações Gerais

- Criar uma conta no GitHub:
  - https://github.com/
- Instalação do Git nos Computadores:
  - https://git-scm.com/
  - Verificar se a instalação funcionou: Abrir o Git Bash e digitar git --version
- Instalar o GitHub Desktop:
  - https://desktop.github.com/
- Playlist do Curso de Git no Youtube:
  - https://www.youtube.com/playlist?list=PLqdwHeoSjEN9agzBYxXytf0guto3jIOxS
- Endereço do Repositório (Clone):
  - https://github.com/blromano/esw-tsw2024.git

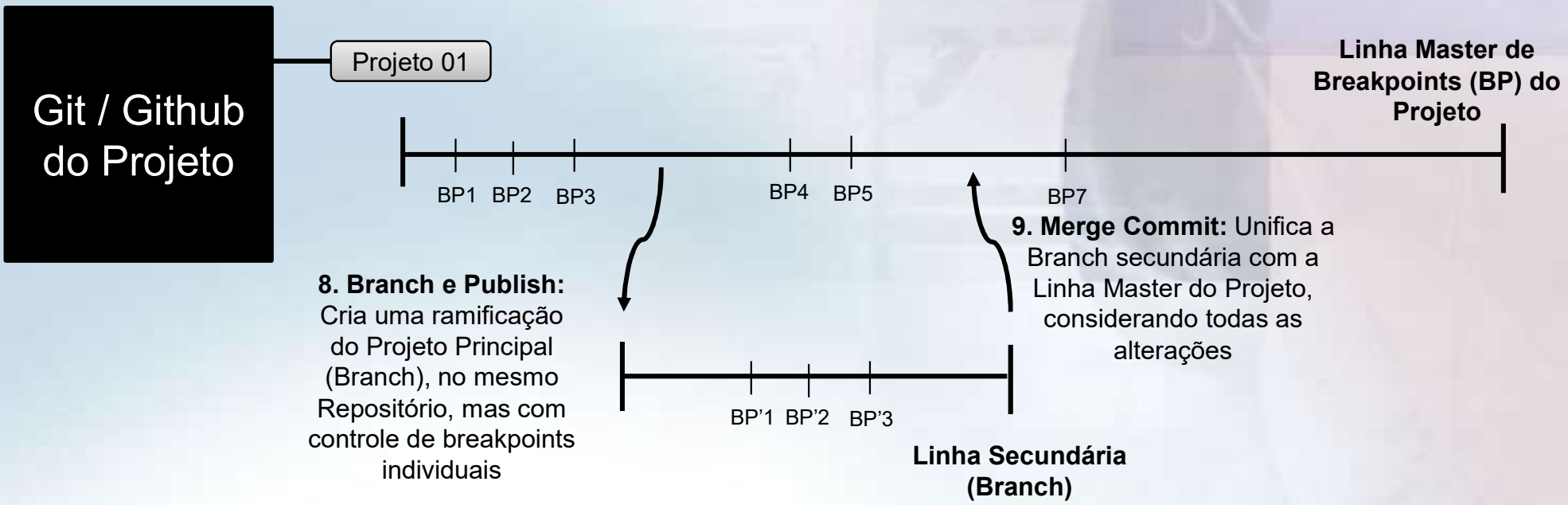
Git / Github: Operações Principais (1)



Git / Github: Operações Principais (2)

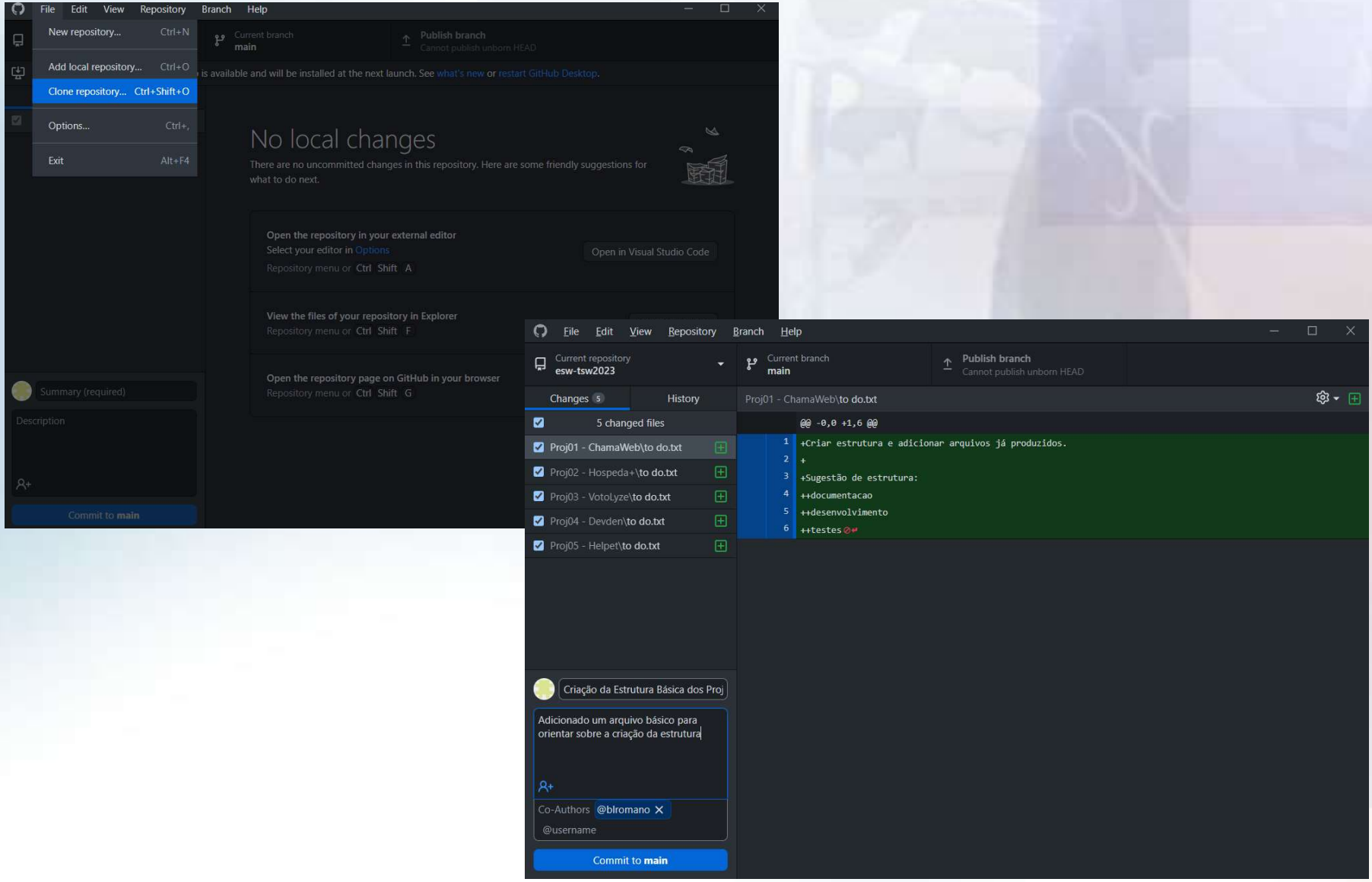


Git / Github: Operações Principais (3)



Breakpoints (BP): São Commits e Pushs que realizamos no Projeto

Github Desktop



Git / Github: Operações Principais (4)

Resumo das principais operações:

- **Clone:** Cópia do Projeto para o Repositório Local
- **Commit:** Criando um breakpoint no Repositório Local
- **Push:** Envio das Alterações Locais para o Repositório do Projeto
- **Fetch:** Informa que seu Repositório Local está desatualizado, mas não transfere o delta destes dados de alterações
- **Pull:** Transfere o delta destes dados de alterações do Repositório do Projeto para o Repositório Local
- **Fork:** Cópia do Repositório Completo do Projeto para Outro Repositório Remoto, visando controlar versões de formas distintas
- **Pull Request:** Unificar as alterações (merge) entre os dois repositórios, sob aprovação do Repositório do Projeto Original
- **Branch e Publish:** Cria uma ramificação do Projeto Principal (Branch), no mesmo Repositório, mas com controle de breakpoints individuais
- **Merge Commit:** Unifica a Branch secundária com a Linha Master do Projeto, considerando todas as alterações

Engenharia de Software II / Qualidade e Teste de Software

Aula 05: Controle de Versões

Breno Lisi Romano

Dúvidas?

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)
Tecnologia em Sistemas para Internet – TSI (QTSI6)

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista