

Sistemas de controle de versão - SVN e Git

15º Programa de Verão do LNCC

Wesley da Silva Pereira

Laboratório Nacional de Computação Científica

20 a 23 de Fevereiro de 2017

Apresentação e motivação

Formação acadêmica:

- Graduado em Engenharia Computacional e Ciência da Computação na UFJF.
- Mestre em Matemática Pura na UFJF.
- Doutorando em Modelagem Computacional no LNCC.



Apresentação e motivação

Formação acadêmica:

- Graduado em Engenharia Computacional e Ciência da Computação na UFJF.
- Mestre em Matemática Pura na UFJF.
- Doutorando em Modelagem Computacional no LNCC.

Experiência com controle de versão:

- Curso de Tortoise SVN na graduação.



Apresentação e motivação

Formação acadêmica:

- Graduado em Engenharia Computacional e Ciência da Computação na UFJF.
- Mestre em Matemática Pura na UFJF.
- Doutorando em Modelagem Computacional no LNCC.

Experiência com controle de versão:

- Curso de Tortoise SVN na graduação.
- Problemas com o versionamento no mestrado.



Apresentação e motivação

Formação acadêmica:

- Graduado em Engenharia Computacional e Ciência da Computação na UFJF.
- Mestre em Matemática Pura na UFJF.
- Doutorando em Modelagem Computacional no LNCC.



Experiência com controle de versão:

- Curso de Tortoise SVN na graduação.
- Problemas com o versionamento no mestrado.
- Uso do SVN no projeto de pesquisa.

Apresentação e motivação

Formação acadêmica:

- Graduado em Engenharia Computacional e Ciência da Computação na UFJF.
- Mestre em Matemática Pura na UFJF.
- Doutorando em Modelagem Computacional no LNCC.



Experiência com controle de versão:

- Curso de Tortoise SVN na graduação.
- Problemas com o versionamento no mestrado.
- Uso do SVN no projeto de pesquisa.
- Segunda edição do curso!

Sumário

1 Sistemas de controle de versão

2 Git

- Introdução
- tryGit
- Configurando o Git
- GitHub

3 Subversion (SVN)

- Introdução
- RiouxSVN
- Clientes gráficos
- Exercícios

4 Principais referências

Links importantes para o curso

Sites:

- <https://git-scm.com/>
- <https://git-scm.com/book/en/v2>
- <https://github.com/>
- <https://subversion.apache.org/>
- <http://svnbook.red-bean.com/>
- <https://riouxsvn.com/>
- <http://rabbitvcs.org/>

Slides no GitHub:

<https://github.com/weslleyspereira/cursoSVNeGit>

Controle de versão

Controle de versão é um sistema que guarda o histórico temporal de modificações de um conjunto de arquivos.

- Possibilita o acesso a versões passadas dos arquivos.
- Componente da **Gerência de Configuração de Software** da Engenharia de Software.
- O termo **revisão**, precedido por um número, também é utilizado para identificar a versão da modificação.

Sistemas de controle de versão

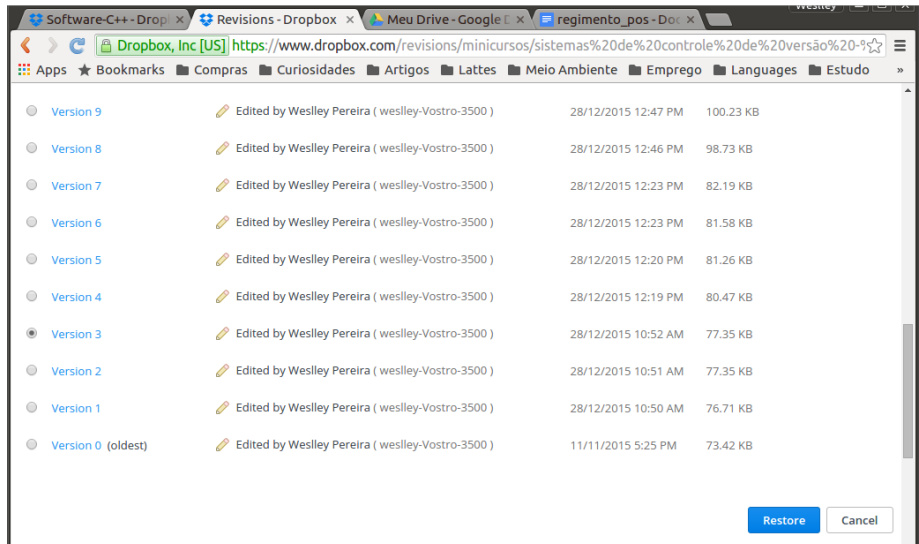
- Sistema de controle de versão é um software que permite gerenciar o processo de versionamento de arquivos.
- Exemplos de funcionalidades: reverter arquivos ou um conjunto de arquivos a um estado anterior; permitir comparação entre versões; guardar informações sobre a modificação (autor, máquina, data, horário).
- Exemplos gratuitos: CVS, Mercurial, Git e SVN.
- Exemplos comerciais: SourceSafe e TFS.

Sistemas de controle de versão

- Sistema de controle de versão é um software que permite gerenciar o processo de versionamento de arquivos.
- Exemplos de funcionalidades: reverter arquivos ou um conjunto de arquivos a um estado anterior; permitir comparação entre versões; guardar informações sobre a modificação (autor, máquina, data, horário).
- Exemplos gratuitos: CVS, Mercurial, Git e SVN.
- Exemplos comerciais: SourceSafe e TFS.

Outros tipos de software também possuem módulos de controle de versão. Os sistemas de armazenamento em nuvem Dropbox e Google Drive são exemplos.

Controle de versão do Dropbox



The screenshot shows a web browser window with the Dropbox Revisions interface. The address bar displays the URL: <https://www.dropbox.com/revisions/minicursos/sistemas%20de%20controle%20de%20vers%C3%A3o%20-%209>. The page title is "Revisions - Dropbox". The interface shows a list of file versions for the document "regimento_pos - Doc". The versions are listed in descending order of age, with "Version 9" being the most recent and "Version 0 (oldest)" being the earliest. Each version entry includes a radio button for selection, the version number, an edit icon, the editor's name and ID, the date and time of the edit, and the file size in KB. The "Restore" button is highlighted in blue at the bottom right of the list.

Version	Edited by	Date and Time	File Size
<input type="radio"/> Version 9	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 12:47 PM	100.23 KB
<input type="radio"/> Version 8	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 12:46 PM	98.73 KB
<input type="radio"/> Version 7	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 12:23 PM	82.19 KB
<input type="radio"/> Version 6	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 12:23 PM	81.58 KB
<input type="radio"/> Version 5	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 12:20 PM	81.26 KB
<input type="radio"/> Version 4	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 12:19 PM	80.47 KB
<input checked="" type="radio"/> Version 3	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 10:52 AM	77.35 KB
<input type="radio"/> Version 2	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 10:51 AM	77.35 KB
<input type="radio"/> Version 1	Edited by Wesley Pereira (wesley-Vostro-3500)	28/12/2015 10:50 AM	76.71 KB
<input type="radio"/> Version 0 (oldest)	Edited by Wesley Pereira (wesley-Vostro-3500)	11/11/2015 5:25 PM	73.42 KB

Restore Cancel

Controle de versão do Dropbox

The screenshot shows the Dropbox web interface in a browser. The address bar displays the URL: <https://www.dropbox.com/home/LNCC-UNICAP-WAVE/Software-C%2B%2B>. The page title is 'Software-C++'. The interface includes a sidebar with navigation options like 'Recents', 'Files', 'Team', 'Paper', 'Photos', 'Sharing', 'Links', 'Events', 'File requests', and 'Deleted files'. The main content area shows a folder named 'Software-C++' containing a list of files. A tooltip is visible over the file 'MHM_LE_C++ (Wesley Pereira's conflicted copy 2015-12-29).zip', indicating it is a conflicted copy.

Name	Modified	Additional sharing
MHM_LE_C++ (Wesley Pereira's conf...5-12-29).zip	17/10/2015 1:02 PM	--
MHM_LE_C++.zip	20/12/2015 6:30 PM	MHM_LE_C++ (Wesley Pereira's conflicted copy 2015-12-29).zip

Controle de versão do Google Drive

Software-C++ - Drop x Revisions - Dropbox x Meu Drive - Google x regimento_pos - Doc x

https://docs.google.com/document/d/1axqRTdnUDt6M8xfyWFJvdS40e1rgzmWQw76l8FYAx4M/edit#

Apps ★ Bookmarks Compras Curiosidades Artigos Lattes Meio Ambiente Emprego Languages Estudo

Histórico de revisões
Hoje, 12:39

100% Total: 2 edições

Histórico de revisões

Hoje, 12:39
Wesley S. Pereira

9 de maio, 08:51
Wesley S. Pereira

☒ Mostrar alterações

Mostrar revisões mais detalhadas

Mestrado e Doutorado - Regimento do Programa

I - Dos Objetivos

Art.1 - O Curso de Pós-Graduação em Modelagem Computacional 'stricto sensu' do LNCC está prioritariamente orientado para a formação de Doutores e Mestre. A partir de uma avaliação global do candidato e, considerando seu potencial **(modificação)**encial em face da proposta do Programa.

Art.2 - A orientação acadêmica fundamental do curso de pós-graduação no LNCC visa formar pesquisadores com capacidade de tratar modelos interdisciplinares.

II - Do Serviço de Análise e Apoio à Formação de Recursos Humanos(SAAFRH)

Art.3 - O SAAFRH é o responsável por todas as atividades de ensino e pós-graduação desenvolvidas pelo LNCC e está estruturado por um Conselho de **P(editado)**esquisa e Formação de Recursos Humanos e três Comitês: O Comitê de Pós-Graduação, de Administração e de Extensão.

Controle de versão do Google Drive

The screenshot shows a web browser window with the Google Drive interface. A modal dialog titled "Gerenciar versões" (Manage versions) is open, displaying the version history for the file "dissertacao.pdf". The dialog indicates that Google Drive maintains previous versions of the file for 30 days. A button "Fazer o upload de nova versão" (Upload new version) is visible. Below, a table lists the current version: "Versão atual" (Current version) for "dissertacao.pdf", dated "11 de abr 14:54" by "Wesley S. Pereira". A dropdown menu is open for this entry, showing a download icon and the option "Fazer o download" (Download). The background shows the Google Drive sidebar with navigation options like "Meu Drive", "Compartilhados comigo", "Google Fotos", "Recentes", "Com estrela", and "Lixeira". The top of the browser shows several tabs, including "Software-C++ - Drop...", "Revisions - Dropbox", "Meu Drive - Google", and "regimento_pos - Doc".

Software-C++ - Drop... Revisions - Dropbox Meu Drive - Google regimento_pos - Doc

https://drive.google.com/drive/my-drive

Apps ★ Bookmarks Compras Curiosidades Artigos Lattes Meio Ambiente Emprego Languages Estudo

Google Drive

NOVO

Meu Drive

Compartilhados comigo

Google Fotos

Recentes

Com estrela

Lixeira

8 GB de 15 GB usados

Fazer upgrade do armazenamento

Gerenciar versões

O Drive mantém as versões anteriores de "dissertacao.pdf" por 30 dias. [Saiba mais](#)

Fazer o upload de nova versão

Versão atual dissertacao.pdf

11 de abr 14:54 Wesley S. Pereira

Fazer o download

Fechar

Por que utilizar?

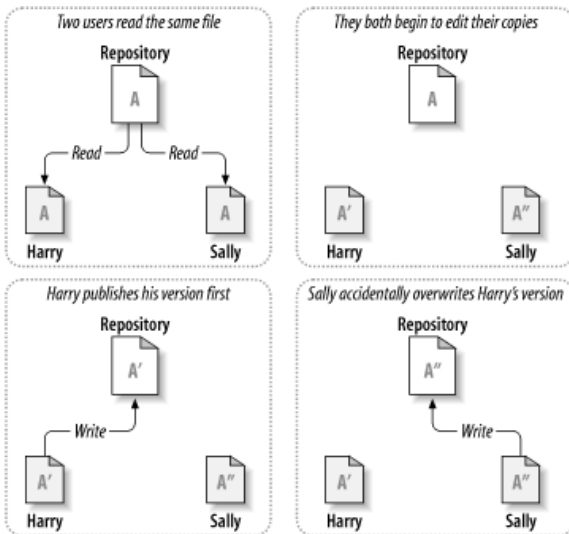
- Facilita trabalho em equipe em um projeto de software.
- Facilita a identificação de bugs.

Por que utilizar?

- Facilita trabalho em equipe em um projeto de software.
- Facilita a identificação de bugs.

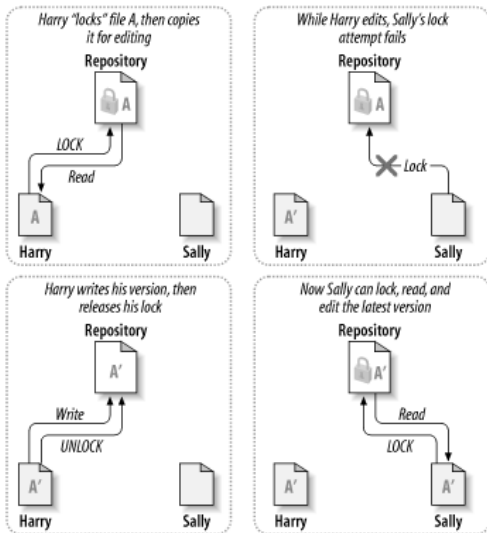
“Suppose we have two coworkers, Harry and Sally. They each decide to edit the same repository file at the same time. If Harry saves his changes to the repository first, it’s possible that (a few moments later) Sally could accidentally overwrite them with her own new version of the file. While Harry’s version of the file won’t be lost forever (because the system remembers every change), any changes Harry made won’t be present in Sally’s newer version of the file, because she never saw Harry’s changes to begin with. Harry’s work is still effectively lost—or at least missing from the latest version of the file—and probably by accident. This is definitely a situation we want to avoid!” [2].

Problema



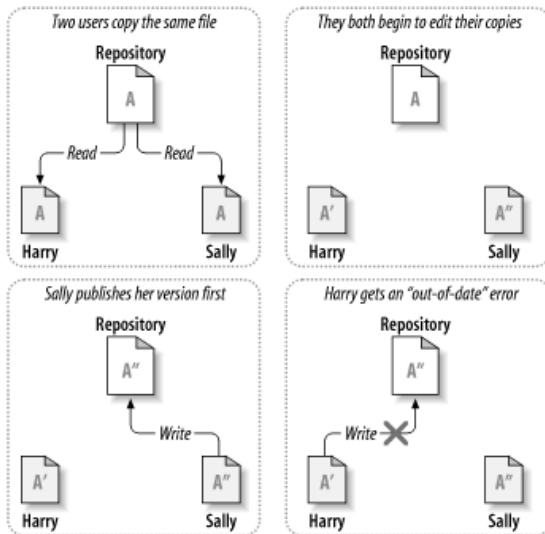
Fonte: [2]

Solução com *lock*



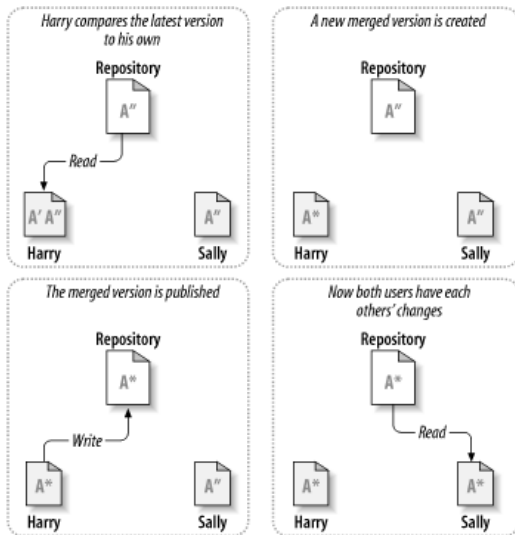
Fonte: [2]

Solução *copy-modify-merge*



Fonte: [2]

Solução *copy-modify-merge*

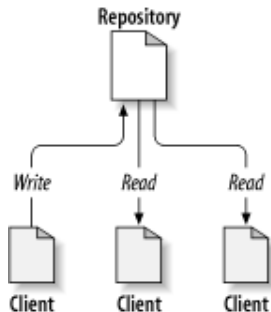


Fonte: [2]

Conceitos básicos

Repositório é um local que guarda todos os dados e meta-dados de um projeto sob controle de versão.

- Normalmente organizado numa hierarquia de arquivos e pastas.
- Clientes se conectam a um repositório para ler ou gravar arquivos.



Fonte: [2]

Conceitos básicos

Banco de dados do projeto é a estrutura que guarda, de maneira organizada, as versões de todos os arquivos.

- Sempre está alocado em um repositório.

Conceitos básicos

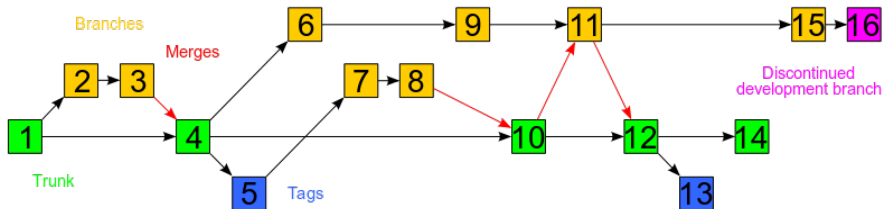
Banco de dados do projeto é a estrutura que guarda, de maneira organizada, as versões de todos os arquivos.

- Sempre está alocado em um repositório.

Para que os clientes trabalhem com um arquivo do repositório, é necessário que possuam uma **cópia local** (*working copy*) do mesmo.

- Cada cliente pode trabalhar com suas cópias locais independentemente do que acontece no repositório.
- Cópias locais podem ser criadas a partir de qualquer versão de um arquivo.

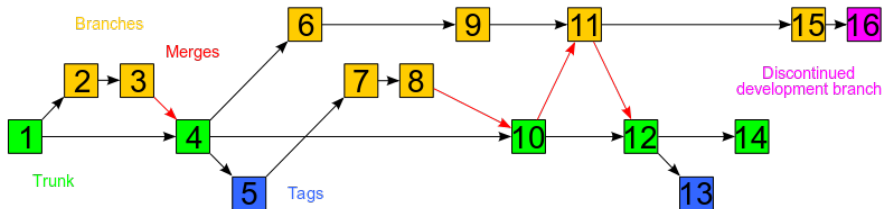
Conceitos básicos



Fonte: https://en.wikipedia.org/wiki/apache_subversion

Um conjunto de arquivos sob controle de versão pode ser separado em **branches**. Cada **branch** corresponde a uma linha de desenvolvimento independente.

Conceitos básicos

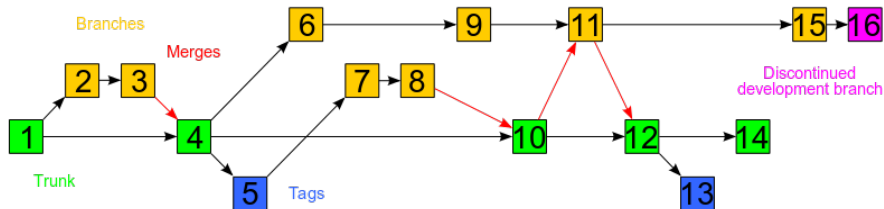


Fonte: https://en.wikipedia.org/wiki/apache_subversion

Um conjunto de arquivos sob controle de versão pode ser separado em **branches**. Cada **branch** corresponde a uma linha de desenvolvimento independente.

O **trunk** é a linha principal de desenvolvimento do projeto. Pode ser entendido como o branch principal.

Conceitos básicos



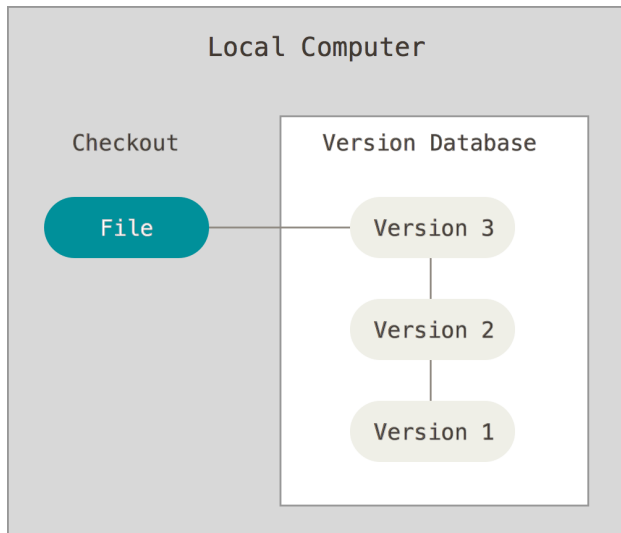
Fonte: https://en.wikipedia.org/wiki/apache_subversion

Um conjunto de arquivos sob controle de versão pode ser separado em **branches**. Cada **branch** corresponde a uma linha de desenvolvimento independente.

O **trunk** é a linha principal de desenvolvimento do projeto. Pode ser entendido como o branch principal.

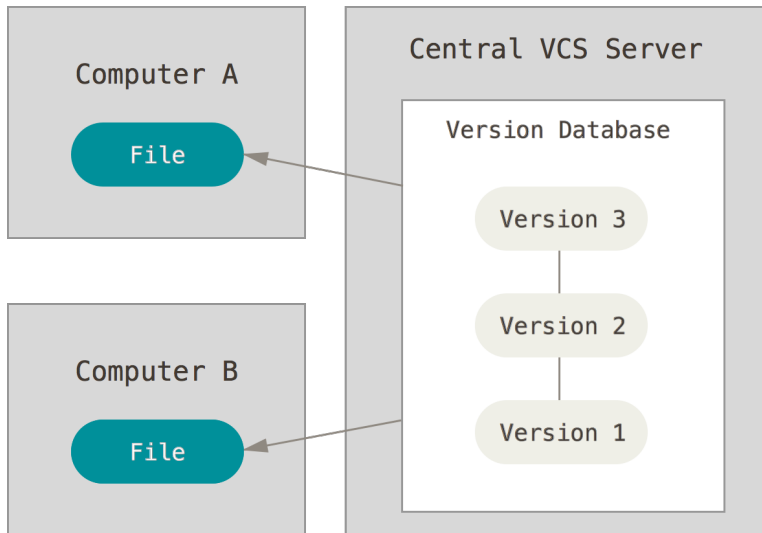
As **tags** são utilizadas para identificar versões específicas do projeto.

Sistemas de controle de versão locais



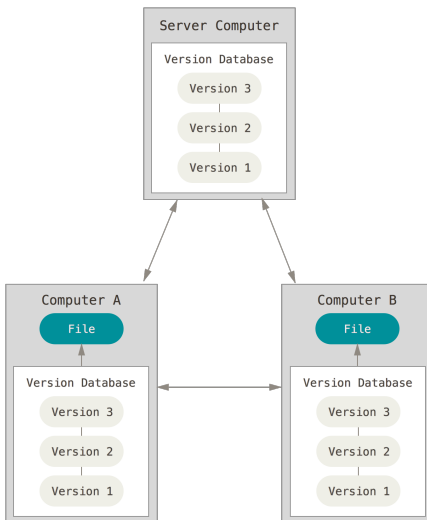
Fonte: [1]

Sistemas de controle de versão centralizados



Fonte: [1]

Sistemas de controle de versão distribuídos



Fonte: [1]

Comandos básicos

- *help*: exibe o menu de ajuda.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.
- *move (mv)*: move ou renomeia itens do diretório local.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.
- *move (mv)*: move ou renomeia itens do diretório local.
- *checkout*: transfere revisões do repositório para o diretório de trabalho.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.
- *move (mv)*: move ou renomeia itens do diretório local.
- *checkout*: transfere revisões do repositório para o diretório de trabalho.
- *commit*: grava alterações locais no repositório.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.
- *move (mv)*: move ou renomeia itens do diretório local.
- *checkout*: transfere revisões do repositório para o diretório de trabalho.
- *commit*: grava alterações locais no repositório.
- *diff*: compara um arquivo em revisões distintas.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.
- *move (mv)*: move ou renomeia itens do diretório local.
- *checkout*: transfere revisões do repositório para o diretório de trabalho.
- *commit*: grava alterações locais no repositório.
- *diff*: compara um arquivo em revisões distintas.
- *merge*: mescla duas alterações feitas em um mesmo arquivo.

Comandos básicos

- *help*: exibe o menu de ajuda.
- *add*: adiciona itens do diretório de trabalho local ao projeto.
- *remove (rm)*: remove itens do diretório local e do controle de versão.
- *move (mv)*: move ou renomeia itens do diretório local.
- *checkout*: transfere revisões do repositório para o diretório de trabalho.
- *commit*: grava alterações locais no repositório.
- *diff*: compara um arquivo em revisões distintas.
- *merge*: mescla duas alterações feitas em um mesmo arquivo.
- *status*: mostra o estado atual de modificações no diretório local de trabalho.

Git

- É um sistema de controle de versão distribuído.
- Criado em 2005 pelos desenvolvedores do kernel Linux (e, em particular, por Linus Torvalds, criador do Linux).



Git

Atributos enfatizados no desenvolvimento inicial do Git:

- Velocidade de execução;
- Design simples;
- Suporte ao desenvolvimento altamente não-linear;
- Totalmente distribuído;
- Capacidade de lidar eficientemente com grandes projetos, como o kernel Linux.

Git

Atributos enfatizados no desenvolvimento inicial do Git:

- **Velocidade de execução;**
- Design simples;
- **Suporte ao desenvolvimento altamente não-linear;**
- Totalmente distribuído;
- **Capacidade de lidar eficientemente com grandes projetos, como o kernel Linux.**

tryGit

Vamos botar a mão na massa?

`http://try.github.com/`

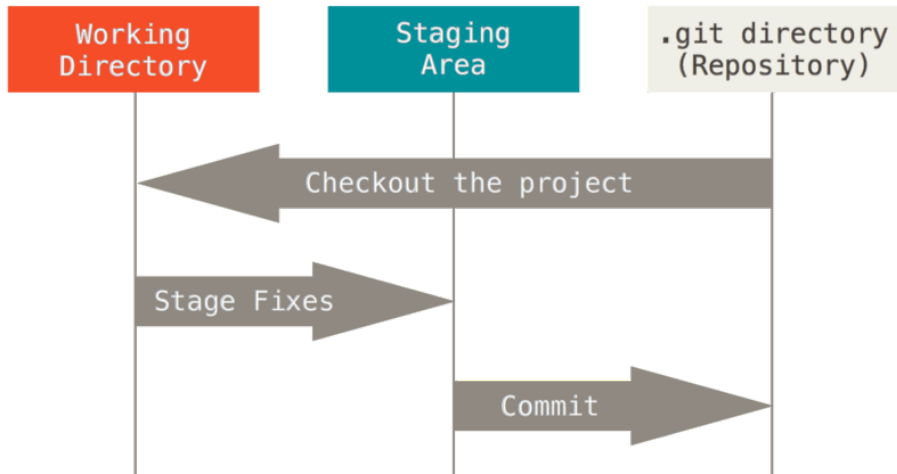


Os três estágios

Git trabalha com três estágios:

- *committed* (enviado): arquivo gravado no repositório.
- *modified* (modificado): arquivo modificado mas não salvo no repositório.
- *staged* (preparado): arquivo modificado que foi marcado para o próximo envio ao repositório.

Os três estágios



Fonte: [1]

Características

- Quase todas as operações são locais. Isto inclui submeter arquivos para o repositório e obter um arquivo em versão passada, por exemplo.

Características

- Quase todas as operações são locais. Isto inclui submeter arquivos para o repositório e obter um arquivo em versão passada, por exemplo.
- Git preserva integridade dos dados através de códigos de *checksum*.

Ex.:

24b9da6552252987aa493b52f8696cd6d3b00373

Utilizado inclusive para identificar arquivos no banco de dados.

Características

- Quase todas as operações são locais. Isto inclui submeter arquivos para o repositório e obter um arquivo em versão passada, por exemplo.
- Git preserva integridade dos dados através de códigos de *checksum*.

Ex.:

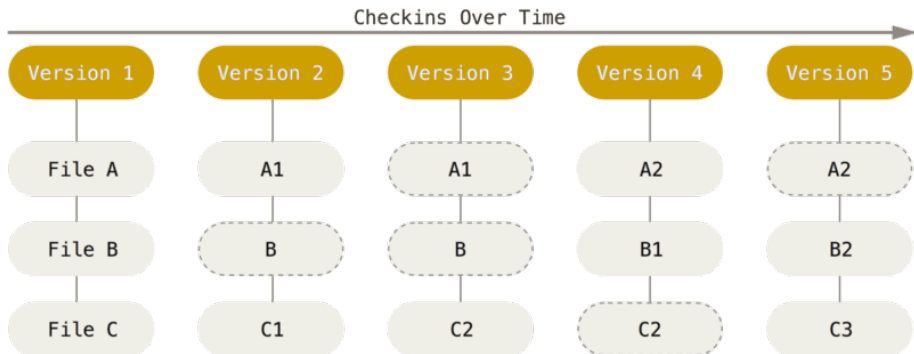
24b9da6552252987aa493b52f8696cd6d3b00373

Utilizado inclusive para identificar arquivos no banco de dados.

- Operações geralmente apenas adicionam dados ao banco de dados do projeto.

Características

- Git armazena versões de arquivos salvando cópias dos mesmos.
- Se arquivos não são modificados entre uma versão e outra, links são criados em vez de cópias de arquivos.



Fonte: [1]

Como usar o Git?

Utilização:

- Linha de comando: possibilita a utilização de todos os comandos do Git.
- Interface gráfica: ferramentas livres disponíveis para as plataformas Windows, Mac e Linux (ver <https://git-scm.com/downloads/guis>). Ex.: RabbitVCS.



Instalação e configuração do Git

Instruções para instalação do Git podem ser encontradas em [1].

Instalação e configuração do Git

Instruções para instalação do Git podem ser encontradas em [1].

O comando **git config** atribui e consulta as variáveis de configuração.

No linux, um repositório Git atende a três arquivos de configuração:

- ① **/etc/gitconfig**: arquivo de configurações do sistema e todos os repositórios. (opção **--system**)

Cada nível sobrescreve as configurações do anterior se nenhuma opção for passada.

Instalação e configuração do Git

Instruções para instalação do Git podem ser encontradas em [1].

O comando **git config** atribui e consulta as variáveis de configuração.

No linux, um repositório Git atende a três arquivos de configuração:

- ① **/etc/gitconfig**: arquivo de configurações do sistema e todos os repositórios. (opção **--system**)
- ② **~/.gitconfig** ou **~/.config/git/config**: arquivo de configurações do usuário. (opção **--global**)

Cada nível sobrescreve as configurações do anterior se nenhuma opção for passada.

Instalação e configuração do Git

Instruções para instalação do Git podem ser encontradas em [1].

O comando **git config** atribui e consulta as variáveis de configuração.

No linux, um repositório Git atende a três arquivos de configuração:

- ① **/etc/gitconfig**: arquivo de configurações do sistema e todos os repositórios. (opção **--system**)
- ② **~/.gitconfig** ou **~/.config/git/config**: arquivo de configurações do usuário. (opção **--global**)
- ③ **.git/config**: arquivo de configurações do repositório.

Cada nível sobrescreve as configurações do anterior se nenhuma opção for passada.

Instalação e configuração do Git

Para que os **commits** possam ser realizados em um projeto, o nome e email do usuário devem ser definidos em um dos arquivos de configuração.

Sugestão

Ao instalar o Git na sua máquina, defina nome e email no arquivo de configurações do usuário. Comandos:

```
git config --global user.name [meu_nome]
```

```
git config --global user.email [meu_email]
```

Instalação e configuração do Git

Para que os **commits** possam ser realizados em um projeto, o nome e email do usuário devem ser definidos em um dos arquivos de configuração.

Sugestão

Ao instalar o Git na sua máquina, defina nome e email no arquivo de configurações do usuário. Comandos:

```
git config --global user.name [meu_nome]
```

```
git config --global user.email [meu_email]
```

O próximo passo é configurar o editor de texto.

```
git config --global core.editor [editor]
```


Instalação e configuração do Git

Para checar as configurações:

git config --list

Instalação e configuração do Git

Para checar as configurações:

git config --list

A ajuda do Git é acionada através de um dos comandos:

git help <comando>

git <comando> --help

man git-<comando>

Instalação e configuração do Git

Para checar as configurações:

git config --list

A ajuda do Git é acionada através de um dos comandos:

git help <comando>

git <comando> --help

man git-<comando>

Vamos testar!

- 1 Abra o terminal com **CTRL+Alt+T**.
- 2 Teste os comandos acima.
- 3 Tente criar um repositório, como feito no TryGit.
- 4 Use o **git config --list** dentro do repositório.

GitHub

É um serviço de hospedagem de repositórios Git, e muito mais!

- Interface web gráfica.
- Fóruns de discussões.
- Ferramentas de gerenciamento de projeto.



GitHub

É um serviço de hospedagem de repositórios Git, e muito mais!

- Contribuição com projetos existentes.
- Colaboração em equipes de desenvolvimento.
- Ferramentas adicionais: GitHub Gist, GitHub Desktop.



Contribuindo com projetos no GitHub

Vamos propor uma contribuição ao repositório Spoon-Knife:

`https://github.com/octocat/Spoon-Knife`

- 1 Clique em **Fork**, para copiar o repositório.
- 2 Encontre sua cópia do repositório Spoon-Knife.
- 3 Copie a URL do repositório.
- 4 Abra o terminal com **CTRL+Alt+T** e use o comando **git clone** para clonar seu repositório Spoon-Knife.

Contribuindo com projetos no GitHub

Alguns comandos úteis no terminal:

- Para saber seu diretório atual, use **pwd**.
- Para visualizar os arquivos do diretório atual, use **ls**.
- Para entrar em um diretório, digite **cd** seguido do nome do diretório.
- Para sair do diretório atual, utilize o comando **cd ..**

Contribuindo com projetos no GitHub

Alguns comandos úteis no terminal:

- Para saber seu diretório atual, use **pwd**.
 - Para visualizar os arquivos do diretório atual, use **ls**.
 - Para entrar em um diretório, digite **cd** seguido do nome do diretório.
 - Para sair do diretório atual, utilize o comando **cd ..**
-
- ① Entre no diretório **Spoon-Knife**.
 - ② Verifique a configuração do diretório remoto com o comando **git remote -v**.
 - ③ Associe o repositório principal **Spoon-Knife** ao nome **upstream**.
Utilize o comando **git remote add upstream https://github.com/octocat/Spoon-Knife.git**.
 - ④ Verifique novamente as configurações de repositórios remotos.

Contribuindo com projetos no GitHub

- 1 Edite o arquivo **index.html**
- 2 Faça com que este arquivo entre na *Staging Area* (preparado para envio ao repositório).
- 3 Envie as modificações ao seu repositório local.
- 4 Envie as modificações ao seu repositório remoto.

Dica

Utilize os comandos do tutorial **tryGit** e fique atento às dicas que o próprio Git fornece no comando **git status**.

Contribuindo com projetos no GitHub

- 1 Edite o arquivo **index.html**
- 2 Faça com que este arquivo entre na *Staging Area* (preparado para envio ao repositório).
- 3 Envie as modificações ao seu repositório local.
- 4 Envie as modificações ao seu repositório remoto.

Dica

Utilize os comandos do tutorial **tryGit** e fique atento às dicas que o próprio Git fornece no comando **git status**.

Vamos criar um **pull request**!

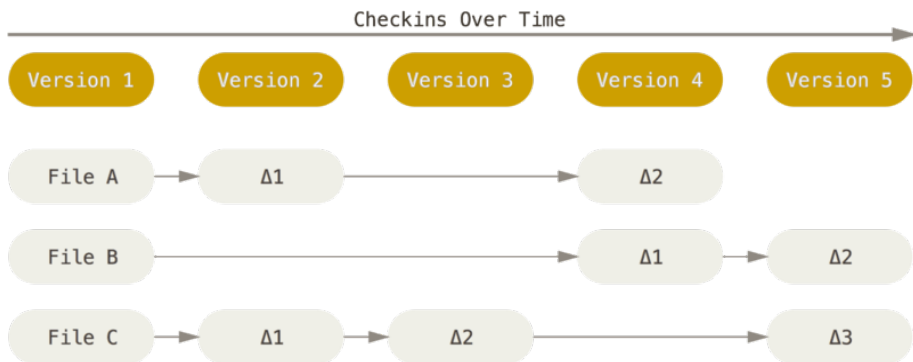
Subversion (SVN)

- Subversion (ou SVN) é um sistema de controle de versão centralizado.
- Começou a ser desenvolvido em 2000, com o objetivo de criar um sistema para substituir o Concurrent Versions System (CVS).
- Em 2010, o sistema Subversion recebeu o nome de “Apache Subversion” pela integração do projeto com a Apache Software Foundation (ASF).



Características

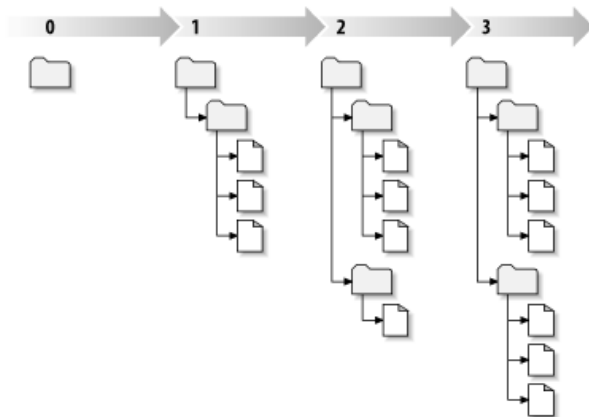
- SVN armazena versões de arquivos salvando as mudanças que ocorreram desde a última versão.



Fonte: [1]

Características

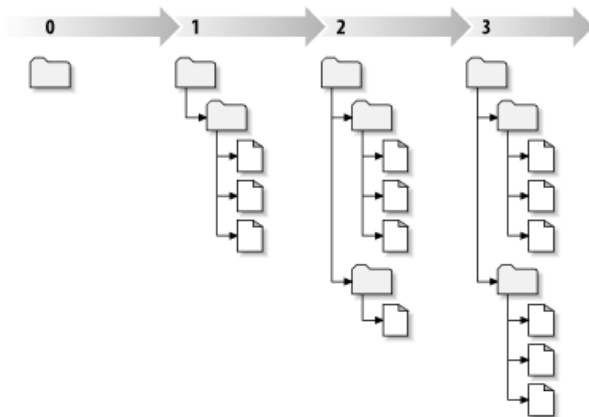
- Transações são atômicas.



Fonte: [2]

Características

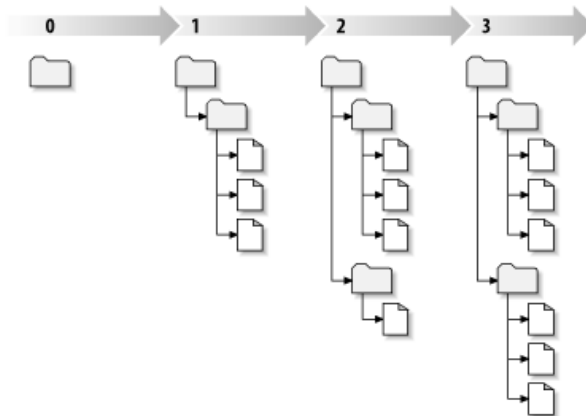
- Transações são atômicas.
- Cada *commit* gera uma revisão que é numerada em ordem crescente.



Fonte: [2]

Características

- Transações são atômicas.
- Cada *commit* gera uma revisão que é numerada em ordem crescente.
- Cada revisão possui sua estrutura de arquivos.



Fonte: [2]

Características

- Cada diretório na raiz do repositório guarda um projeto.

Características

- Cada diretório na raiz do repositório guarda um projeto.
- Diretórios `.svn` (ou `_svn`) guardam os arquivos de configuração de cada projeto.

Características

- Cada diretório na raiz do repositório guarda um projeto.
- Diretórios `.svn` (ou `_svn`) guardam os arquivos de configuração de cada projeto.
- Uma cópia de trabalho local no SVN é uma estrutura completa de diretórios e arquivos.

Características

- Cada diretório na raiz do repositório guarda um projeto.
- Diretórios `.svn` (ou `_svn`) guardam os arquivos de configuração de cada projeto.
- Uma cópia de trabalho local no SVN é uma estrutura completa de diretórios e arquivos.
- Uma cópia de arquivo local é um arquivo da cópia de trabalho local.

Características

Existem 4 estados em que um arquivo de trabalho local pode estar. Estes estados alteram os comandos **svn commit** e **svn update**.

Características

Existem 4 estados em que um arquivo de trabalho local pode estar. Estes estados alteram os comandos **svn commit** e **svn update**.

- Não modificado e atual.

Características

Existem 4 estados em que um arquivo de trabalho local pode estar. Estes estados alteram os comandos **svn commit** e **svn update**.

- Não modificado e atual.
- Modificado e atual.

Características

Existem 4 estados em que um arquivo de trabalho local pode estar. Estes estados alteram os comandos **svn commit** e **svn update**.

- Não modificado e atual.
- Modificado e atual.
- Não modificado e desatualizado.

Características

Existem 4 estados em que um arquivo de trabalho local pode estar. Estes estados alteram os comandos **svn commit** e **svn update**.

- Não modificado e atual.
- Modificado e atual.
- Não modificado e desatualizado.
- Modificado e desatualizado.

Revisões mistas

- *Commits* e *updates* são separados.

Revisões mistas

- *Commits* e *updates* são separados.
- O número da revisão é associado à estrutura inteira de diretórios. Apesar disso, cópias de trabalho mistas são permitidas.

Revisões mistas

- *Commits* e *updates* são separados.
- O número da revisão é associado à estrutura inteira de diretórios. Apesar disso, cópias de trabalho mistas são permitidas.
- Para saber a revisão dos arquivos e da cópia local: **svn info** **<arquivo>**.

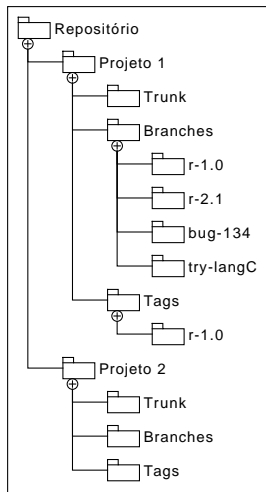
Revisões mistas

- *Commits* e *updates* são separados.
- O número da revisão é associado à estrutura inteira de diretórios. Apesar disso, cópias de trabalho mistas são permitidas.
- Para saber a revisão dos arquivos e da cópia local: **svn info <arquivo>**.
- É proibida a deleção de cópias de arquivos desatualizadas.

Revisões mistas

- *Commits* e *updates* são separados.
- O número da revisão é associado à estrutura inteira de diretórios. Apesar disso, cópias de trabalho mistas são permitidas.
- Para saber a revisão dos arquivos e da cópia local: **svn info <arquivo>**.
- É proibida a deleção de cópias de arquivos desatualizadas.
- A operação de *merge* também é proibida em cópias locais com revisão mista (a partir do Subversion 1.7).

Organização de um repositório SVN



RiouxSVN

É um serviço gratuito de hospedagem de repositórios SVN.

- Repositórios privados por padrão.
- Permite colaboração entre usuários.
- Plano básico é extensível a partir de doações.

Vamos começar a trabalhar em equipe!

<https://riouxsvn.com/>

Dica

Utilize o comando **svn status** com frequência para saber como está sua cópia local. Os comandos **svn info** e **svn log** também ajudam a ter o controle do que está sendo feito.

Clientes gráficos



RabbitVCS

Clientes gráficos



RabbitVCS

Sobre o RabbitVCS:

- Cliente gráfico para SVN e Git integrado ao Nautilus (Linux) e Finder (MAC OS).
- Inspirado no TortoiseSVN (Windows).

Clientes gráficos



RabbitVCS

Sobre o RabbitVCS:

- Cliente gráfico para SVN e Git integrado ao Nautilus (Linux) e Finder (MAC OS).
- Inspirado no TortoiseSVN (Windows).

Vamos aprender mais na prática!

Exercício 1 - Resolvendo conflitos

Criando um repositório no RiouxSVN.

- ① Na aba **Repositories**, clique em **Create new repository...**
- ② O campo **Repository Name** é um identificador, portanto tem limitação de caracteres. Prefira utilizar apenas letras, números e '__'.
- ③ Não marque a caixa **Create trunk, branches and tags directories**

Exercício 1 - Resolvendo conflitos

Criando um repositório no RiouxSVN.

- 1 Na aba **Repositories**, clique em **Create new repository...**
- 2 O campo **Repository Name** é um identificador, portanto tem limitação de caracteres. Prefira utilizar apenas letras, números e '__'.
- 3 Não marque a caixa **Create trunk, branches and tags directories**

Criando cópia local.

- 1 No diretório **home** do linux crie um novo diretório com o seu nome.
- 2 Abra o terminal e vá para o diretório criado.
- 3 Utilize o comando **svn info** para obter informações do repositório.
- 4 Faça o checkout do seu repositório usando o comando **svn checkout**.

Exercício 1

Adicionando um arquivo ao projeto.

- 1 Crie um arquivo de texto **teste.txt** no diretório do projeto com o texto: "Oi Mundo!".
- 2 Verifique o status do arquivo no svn.
- 3 Adicione o arquivo ao controle de versão do svn com o comando **svn add**.
- 4 Utilize o comando **svn commit** para enviar seu arquivo para o repositório.
- 5 Compare a revisão do arquivo com a revisão da cópia local.

Exercício 1

Adicionando um arquivo ao projeto.

- 1 Crie um arquivo de texto **teste.txt** no diretório do projeto com o texto: "Oi Mundo!".
- 2 Verifique o status do arquivo no svn.
- 3 Adicione o arquivo ao controle de versão do svn com o comando **svn add**.
- 4 Utilize o comando **svn commit** para enviar seu arquivo para o repositório.
- 5 Compare a revisão do arquivo com a revisão da cópia local.

Movendo o arquivo.

- 1 Crie um diretório **novo** na cópia local usando o comando **svn mkdir**.
- 2 Mova o arquivo **teste.txt** para o diretório usando o comando **svn mv**.
- 3 Renomeie o arquivo **teste.txt** para **helloWorld.txt** com o mesmo comando do item anterior.
- 4 Verifique as modificações feitas.

Exercício 1

Criando nova cópia local.

- ① Crie uma pasta **novo_usuario** no diretório **home**.
- ② Faça um novo checkout do repositório na pasta **novo_usuario**.
- ③ Compare os números de revisão entre os dois repositórios.
- ④ Troque uma letra no arquivo **helloWorld.txt** e envie a cópia local para o repositório.

Exercício 1

Criando nova cópia local.

- 1 Crie uma pasta **novo_usuario** no diretório **home**.
- 2 Faça um novo checkout do repositório na pasta **novo_usuario**.
- 3 Compare os números de revisão entre os dois repositórios.
- 4 Troque uma letra no arquivo **helloWorld.txt** e envie a cópia local para o repositório.

Gerando conflito.

- 1 Na cópia local que está na pasta com o seu nome, altere o '!' em **helloWorld.txt**.
- 2 Tente fazer a operação **svn commit** (não deveria funcionar).
- 3 Utilize o comando **svn update** para tentar obter a versão atual do **helloWorld.txt**.
- 4 Escolha a opção **p** e dê uma olhada no seu diretório **novo**.

Exercício 1

- Neste momento, seu diretório **novo** possui alguns arquivos a mais.
- Abra os arquivos e verifique as diferentes frases que aparecem. Dois desses arquivos são de revisões salvas no repositório, um deles apresenta sua cópia local antes do comando **svn update** e o último é um arquivo que apresenta o conflito.

Exercício 1

- Neste momento, seu diretório **novo** possui alguns arquivos a mais.
- Abra os arquivos e verifique as diferentes frases que aparecem. Dois desses arquivos são de revisões salvas no repositório, um deles apresenta sua cópia local antes do comando **svn update** e o último é um arquivo que apresenta o conflito.

Resolvendo o conflito.

- 1 Verifique que há um conflito com o **svn status**.
- 2 Utilize o comando **svn resolve** e observe as opções. (A opção **s** mostra uma menu de ajuda, além de outras opções)
- 3 Verifique as diferenças entre as versões digitando **df**.
- 4 Escolha a opção **m**.

Exercício 1

Continuando com a resolução do conflito.

- Existem vários tipos de merge disponíveis.
- Como este é um caso simples, apenas escolha entre opção **1** e **2**.
- Após isso, é necessário marcar o conflito como resolvido com a opção **r**.
- Agora pode realizar o commit de suas modificações.

Exercício 1

Continuando com a resolução do conflito.

- Existem vários tipos de merge disponíveis.
- Como este é um caso simples, apenas escolha entre opção **1** e **2**.
- Após isso, é necessário marcar o conflito como resolvido com a opção **r**.
- Agora pode realizar o commit de suas modificações.

Deletando arquivo.

- ① Utilize o comando **svn update**.
- ② Utilize o comando **svn rm** para remover o arquivo **helloWorld.txt**.
- ③ Envie as mudanças ao repositório.

Exercício 1

Continuando com a resolução do conflito.

- Existem vários tipos de merge disponíveis.
- Como este é um caso simples, apenas escolha entre opção **1** e **2**.
- Após isso, é necessário marcar o conflito como resolvido com a opção **r**.
- Agora pode realizar o commit de suas modificações.

Deletando arquivo.

- ① Utilize o comando **svn update**.
- ② Utilize o comando **svn rm** para remover o arquivo **helloWorld.txt**.
- ③ Envie as mudanças ao repositório.

Mas... e agora? Acho que vou precisar daquele arquivo novamente...

Exercício 1

Encontrando o arquivo deletado. Neste caso, pode ser feito de diferentes maneiras:

- Revertendo o arquivo à revisão anterior com o comando **svn update [caminho do arquivo] -r [número da revisão]**.
- Revertendo a cópia local à revisão anterior com o comando **svn update -r [número da revisão]**.
- Fazendo um checkout na revisão anterior com o comando **svn checkout [url do repositório]@[numero da revisão]**.
- Se o arquivo ainda não tivesse sido enviado ao repositório, o comando **svn revert [caminho do arquivo]** resolveria o problema.

Exercício 1

Encontrando o arquivo deletado. Neste caso, pode ser feito de diferentes maneiras:

- Revertendo o arquivo à revisão anterior com o comando **svn update [caminho do arquivo] -r [número da revisão]**.
- Revertendo a cópia local à revisão anterior com o comando **svn update -r [número da revisão]**.
- Fazendo um checkout na revisão anterior com o comando **svn checkout [url do repositório]@[numero da revisão]**.
- Se o arquivo ainda não tivesse sido enviado ao repositório, o comando **svn revert [caminho do arquivo]** resolveria o problema.

Visualize o log das operações no comando **svn log**.

Resposta local X Resposta do repositório

O SVN retorna respostas locais para alguns comandos, que dizem respeito à revisão que está na cópia de trabalho local. Ex.: **svn diff**, **svn info** e **svn status**.

Para comparar com revisões do repositório:

- **svn diff -r [revisão]** (Use **HEAD** para a última revisão).
- **svn info -r [revisão]**
- **svn status --show-updates**

Exercício 2 - Trabalhando em equipe

- Vamos adicionar todos os nossos usuários ao repositório **eamc_svn_repo**.

Exercício 2 - Trabalhando em equipe

- Vamos adicionar todos os nossos usuários ao repositório **eamc_svn_repo**.
- Vamos fazer o checkout de uma cópia do diretório **project1** repositório. Verifique o que há no projeto: arquivos, estrutura de diretórios, log, número de revisão, etc.

Exercício 2 - Trabalhando em equipe

- Vamos adicionar todos os nossos usuários ao repositório **eamc_svn_repo**.
- Vamos fazer o checkout de uma cópia do diretório **project1** repositório. Verifique o que há no projeto: arquivos, estrutura de diretórios, log, número de revisão, etc.
- Agora faça o checkout da revisão 10 do projeto e cheque novamente a estrutura. Algo novo?

Exercício 2 - Trabalhando em equipe

- Vamos adicionar todos os nossos usuários ao repositório **eamc_svn_repo**.
- Vamos fazer o checkout de uma cópia do diretório **project1** repositório. Verifique o que há no projeto: arquivos, estrutura de diretórios, log, número de revisão, etc.
- Agora faça o checkout da revisão 10 do projeto e cheque novamente a estrutura. Algo novo?
- Volte para a revisão atual do projeto.

Exercício 2

- Se organizem em duplas. Cada dupla precisa de dois computadores.
- Cada dupla deve escolher um arquivo **cancao_do_exilio** para fazer a correção. Observe que o arquivo está com um erro básico.
- Cada membro da dupla irá corrigir uma parte do arquivo.
- Para isso, criaremos um branch para correção de erros.

Exercício 2

- Se organizem em duplas. Cada dupla precisa de dois computadores.
- Cada dupla deve escolher um arquivo **cancao_do_exilio** para fazer a correção. Observe que o arquivo está com um erro básico.
- Cada membro da dupla irá corrigir uma parte do arquivo.
- Para isso, criaremos um branch para correção de erros.

Criando o branch para correção de erros no repositório.

```
svn copy
```

```
https://svn.riouxsvn.com/eamc_svn_repo/project1/trunk
```

```
https://svn.riouxsvn.com/eamc_svn_repo/project1/
```

```
branches/bug-dupla[N] -m "Branch para correção dos acentos"
```

Exercício 2

Fazendo as correções.

- Cada membro da dupla faz o checkout do branch criado na pasta **branches**.
- Façam as correções e façam o commit do branch para o servidor.

Exercício 2

Fazendo as correções.

- Cada membro da dupla faz o checkout do branch criado na pasta **branches**.
- Façam as correções e façam o commit do branch para o servidor.

Observe que cada checkout corresponde a uma nova cópia de trabalho local.

Exercício 2

Voltando ao trunk.

- Ainda no ambiente de trabalho do branch, faça o merge com o trunk e corrija possíveis conflitos.

```
svn merge ^/calc/trunk
```

- Caso sua dupla tenha feito algum commit, é necessário fazer o update do branch antes da etapa anterior. Caso isso aconteça, o **svn** irá notificá-lo.

Exercício 2

Voltando ao trunk.

- Ainda no ambiente de trabalho do branch, faça o merge com o trunk e corrija possíveis conflitos.
`svn merge ^/calc/trunk`
- Caso sua dupla tenha feito algum commit, é necessário fazer o update do branch antes da etapa anterior. Caso isso aconteça, o **svn** irá notificá-lo.

Em apenas um dos computadores:

- Façam o merge final com o trunk e o commit para o repositório, dizendo "Correções prontas!".
- Voltem ao diretório de trabalho do trunk e façam o update.
- Realizem o merge do branch de erro em direção ao trunk.

```
svn merge -{}-reintegrate ^/calc/branches/bug-dupla[N]
```

Exercício 2

Em apenas um dos computadores:

- Faça o commit das mudanças, após verificar que não existem erros.
- Como o branch de correção já não é mais útil, podemos deletá-lo com o comando **svn delete**.

Principais referências



Scott Chacon and Ben Straub.

Pro Git.

Apress, 2 edition, 2014.



Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato.
Version control with subversion: For subversion 1.7 (compiled from r5067).

Obrigado a todos pela atenção!

Agradeço também à mestranda Letícia Fonseca, pela colaboração na elaboração deste minicurso.