Gestão de Configuração e Mudanças de Software

Allan Lima

Arquitetura, Design e Implementação de Sistemas para Internet
Pós Graduação
Faculdade 7 de Setembro

arglbr@gmail.com

nov/2014

Subversion

Parte 1

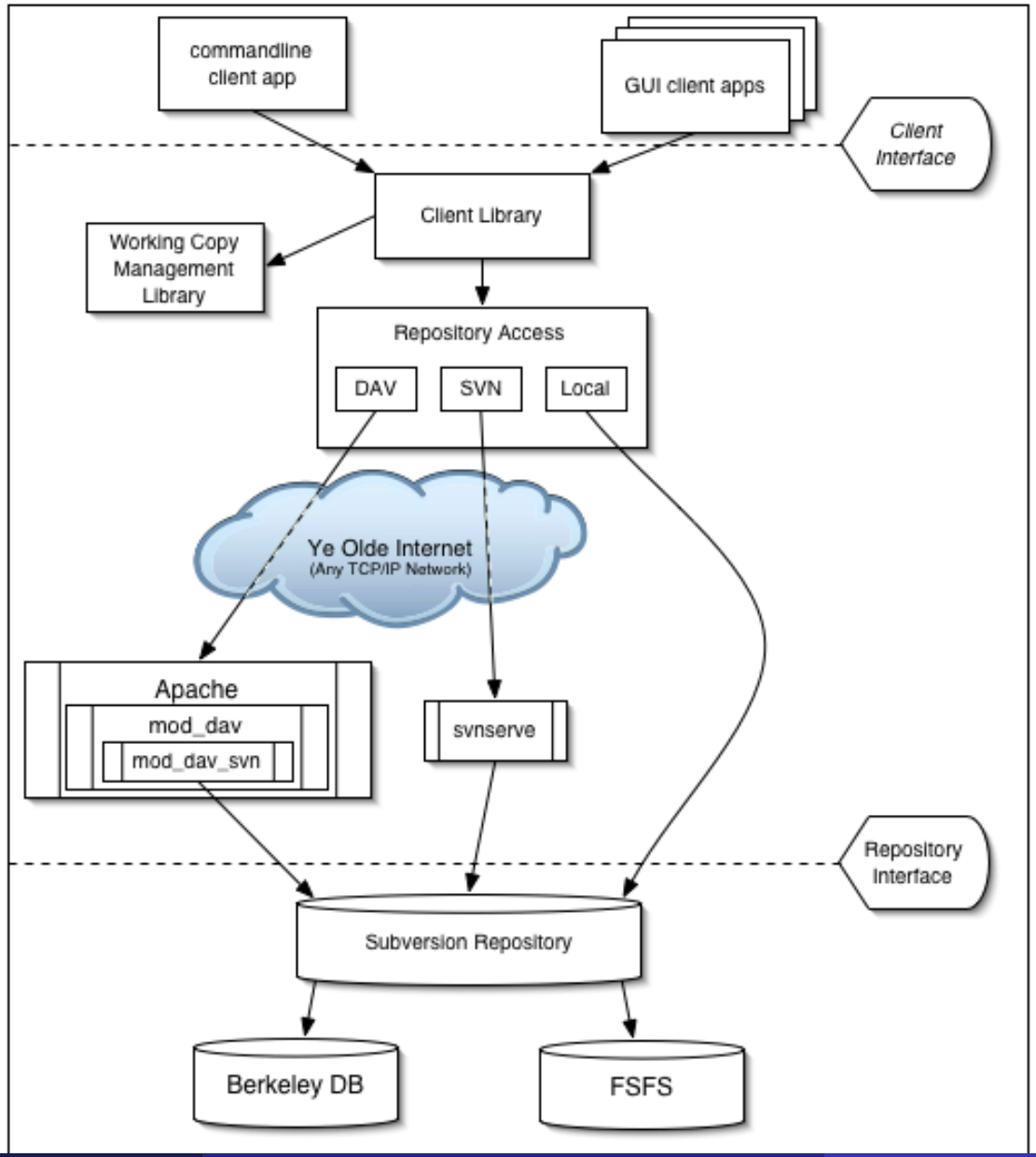
Histórico

- No início de 2000, a CollabNet Inc. (http://www.collab.net) começou a procurar desenvolvedores para escrever um substituto para o CVS.
- Em 2009, a CollabNet iniciou o processo de integração com a Apache Software Foundation (ASF).
- No início de 2010, o SVN foi totalmente adotado pela ASF com sua nova página na web (http://subversion.apache.org), e foi rebatizado de "Apache Subversion".

Características

- Controle de versão centralizado mais popular.
- Substituiu o CVS.
- Disponível para várias plataformas.
- Plugins para várias ferramentas.

Arquitetura



URLs do Repositório

Esquema	Método de Acesso
`file:///`	acesso direto ao repositório (em um disco local).
`http://`	acesso via protocolo WebDAV em um servidor Apache.
`https://`	mesmo que `http://`, mas com encriptação SSL.
`svn://`	acesso via protocolo próprio em um servidor svnserve.
`svn+ssh://`	mesmo que `svn://`, mas através de um túnel SSH.

Table: URLs de Acesso ao Repositório

Componentes

- **svn:** Cliente de linha de comando
- **svnversion:** Informa o estado (em termos de revisões) de uma cópia de trabalho
- **svnlook:** Ferramenta para inspecionar um repositório
- **svnadmin:** Ferramenta para criação, ajuste ou manutenção de um repositório
- **mod_dav_svn:** Módulo plug-in para o Apache HTTP Server, usado para disponibilizar seu repositório a outros através da rede
- **svnserve:** Programa servidor independente, executável como um processo; outra forma de disponibilizar o repositório a através da rede
- **svndumpfilter:** Programa para filtragem de fluxos de um repositório Subversion
- **svnsync:** Programa para espelhamento incremental de um repositório para outro através da rede
- **svndump:** Ferramenta de backup

Instalação

- Inicie o VirtualBox da sua estação
- Inicie a VM Ubuntu disponível
- O usuário é **gcs**
- A senha é **gcs123**
- Essa VM funcionará como o Servidor para as práticas
- Tente sempre usar a mesma estação de trabalho

Instalação

- Instale a última versão estável do SVN conforme indicado em: `http://subversion.apache.org/packages.html`
- Para a versão Ubuntu, use o comando abaixo:
  - `apt-get install subversion libapache2-svn`
- Para a versão Windows, baixe o arquivo:
  - svn-win32-X.Y.Z.zip

Instalação

- Para Windows, instale o TortoiseSVN:
  - `http://tortoisesvn.net/downloads.html`
- Para Mac OS, SCPlugin:
  - `http://scplugin.tigris.org/files/documents/1368/47351/SCPlugin-0.8.2-SVN.1.6.5.dmg`

Instalação - Linux

- Através da linha de comando a seguir estaremos iniciando o servidor e definindo o diretório dos repositórios para a pasta:
  - `mkdir -p ~/curso-gcs/repositorio-svn/`
  - `svnserve -d -r ~/curso-gcs/repositorio-svn/`

Instalação - Linux

- Com o servidor rodando, devemos criar um repositório através do comando `svnadmin`, criaremos então o repositório **teste** em:
  - `~/curso-gcs/repositorio-svn/teste`
  - `svnadmin create ~/curso-gcs/repositorio-svn/teste`
- Observe que os repositórios podem ser criados em qualquer pasta que esteja vazia, mas apenas poderão ser acessados os repositórios criados a partir do repositório principal definido através do comando `svnserve`.

Instalação - Configuração Servidor

- O comando `svnadmin` cria a estrutura básica do repositório que armazenará todos os dados.
- Após a criação da estrutura inicial do repositório devemos editar o arquivo `svnserver.conf` que está na pasta:
  - `~/curso-gcs/repositorio-svn/teste/conf`
  - `svnadmin create ~/curso-gcs/repositorio-svn/teste`
- Modifique o conteúdo do arquivo para:
  - ```
    [general]
    anon-access = read
    auth-access = write
    password-db = passwd
    ```
    ```

Instalação - Configuração Servidor

- O termo **anon-acess** define como será o acesso anônimo ao repositório, podemos definir como **none**, **read** ou **write** para impedir o acesso anônimo, permitir somente leitura ou permitir para leitura e gravação respectivamente.
- O termo **auth-acess** define como poderá ser o acesso para usuários autorizados e pode ser definido com os mesmos valores none, read ou write.
- O termo **password-db** define o nome do arquivo contendo a lista de `usuário = senha` autorizados no repositório, no exemplo deveremos ter um arquivo chamado passwd a seguir um exemplo de um arquivo de senhas:
  - ```
    [users]
    #user = pass
    joao = 123
    maria = abc
    ```
    ```

Instalação - Configuração Servidor

- No exemplo, temos um repositório que permite leitura dos documentos através de um acesso anônimo e leitura e gravação para os usuários autenticados.
- O acesso autorizado pode ser feito pelos usuários `joao` e `maria` através da senha 123 e `abc` respectivamente.
- *Importante: tanto usuário como senha são case sensitive.*

Subversion

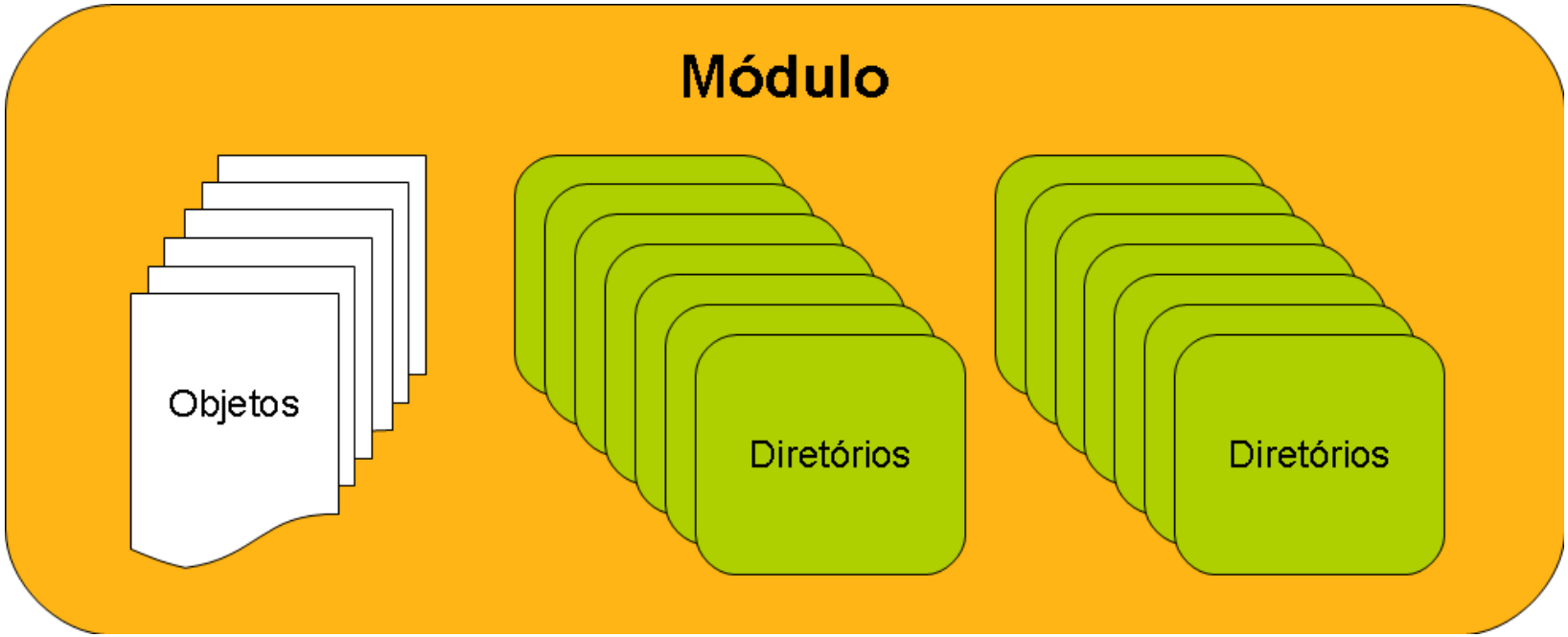
Parte 2

O Repositório

- É um conjunto de módulos, configurações e credenciais de usuários.

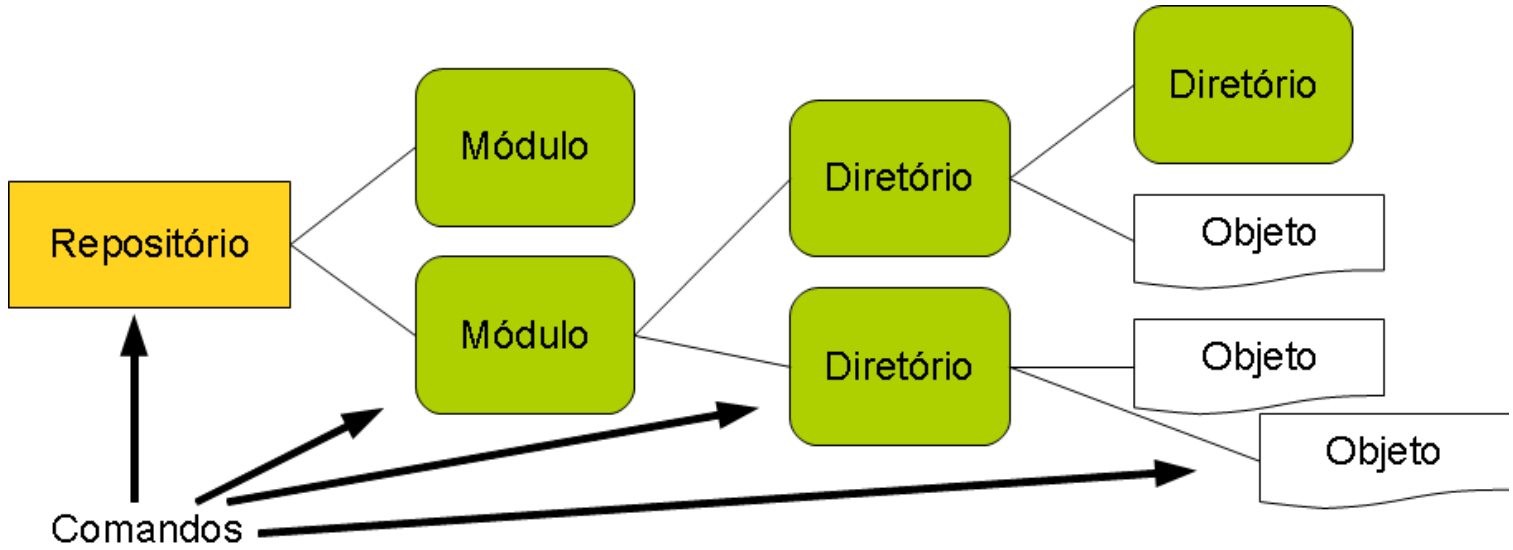
O Módulo

- É um conjunto de objetos e sub-diretórios.



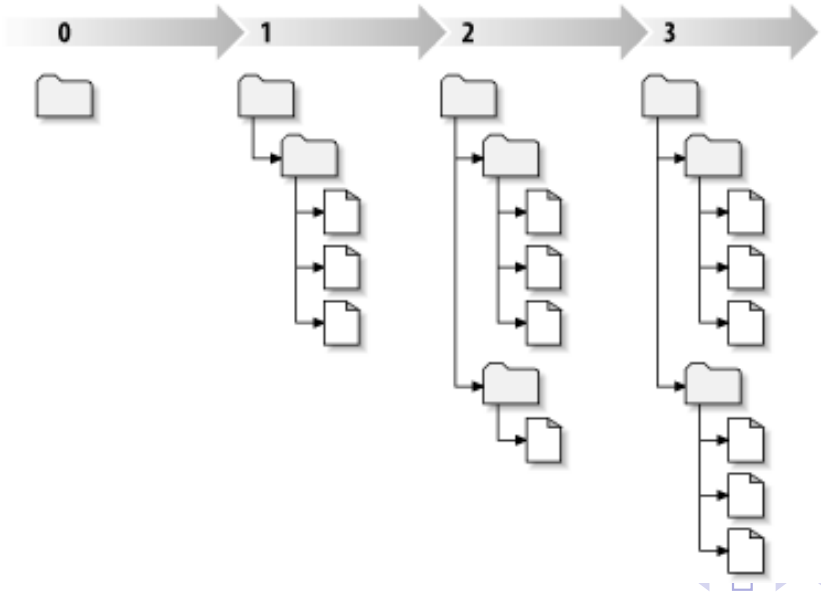
O Módulo

- É a maior unidade na hierarquia que deve receber comandos.
- Os comandos recebidos por um Módulo são propagados recursivamente a todos os objetos e diretórios dentro dele.



Revisões

- Uma operação **svn commit** publica as alterações feitas em qualquer número de arquivos e diretórios como uma única transação atômica.
- Cada vez que o repositório recebe um **commit**, é criado um novo estado da árvore de arquivos, chamado revisão.
- A cada revisão é atribuído um número único, maior do que o número atribuído é revisão anterior.
- A revisão inicial de um repositório recém criado é 0 e consiste em nada mais que um diretério raiz vazio.



Organização do repositório

- Embora não exista obrigatoriedade, recomenda-se dividir os Módulos em diretórios que determinam estágios no desenvolvimento.
  - Branches
  - Tags
  - Trunk

Organização do repositório

- **Branches:** Contém todo o código que foi ramificado a partir do tronco principal.
- **Tags:** Sempre que uma nova versão é feita, uma nova marca será criada nesta pasta.
- **Trunk:** é o tronco principal e que geralmente contém a versão mais recente do projeto.

Organização do repositório

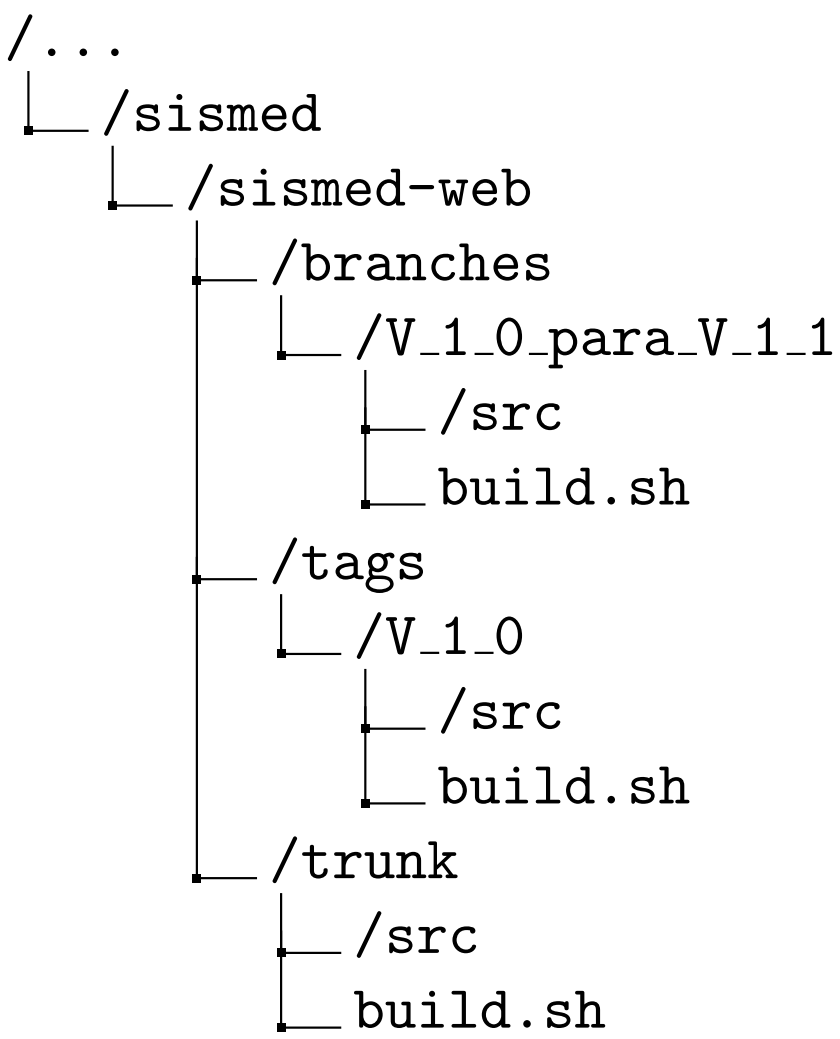
Exemplo Lógico:

```
/...
└── /repositorio
    ├── /modulo1
    │   ├── /branches
    │   ├── /tags
    │   └── /trunk
    └── /modulo2
        ├── /branches
        ├── /tags
        └── /trunk
```

Exemplo Real:

```
/...
└── /sismed
    ├── /sismed-web
    │   ├── /branches
    │   ├── /tags
    │   └── /trunk
    └── /sismed-ejb
        ├── /branches
        ├── /tags
        └── /trunk
```

Organização do repositório



Comandos

Os comandos SVN tem uma nomenclatura própria, assim como conceitos que devem ser compreendidos para darmos início a um estudo mais profundo da ferramenta.

Colocando dados em seu Repositório

Há dois modos de colocar novos arquivos em seu repositório Subversion:

- `svn import`
  - é um modo rápido para copiar uma árvore de arquivos não versionada em um repositório, criando diretórios intermediários quando necessário.
  - não requer uma cópia de trabalho `checkout`, e seus arquivos são imediatamente submetidos ao repositório.
  - Isto é tipicamente usado quando se tem uma árvore de arquivos existente que se quer adicioná-la ao controle de versão no repositório Subversion.
- `svn add`
  - Agenda o arquivo, diretório, ou link simbólico para ser adicionado ao repositório. No próximo `commit`, o artefato passará a fazer parte do diretório pai onde estiver.

Checkout Inicial

- Na maioria das vezes, inicia-se o uso de um repositório Subversion fazendo um `checkout` do projeto.
- Fazer um `checkout` de um repositório cria uma "cópia de trabalho" na máquina local.
- Esta cópia contém o HEAD (revisão mais recente) do repositório Subversion que foi especificado no `checkout`.

```
$ svn checkout http://svn.collab.net/repos/svn/trunk
A    trunk/Makefile.in
A    trunk/ac-helpers
A    trunk/ac-helpers/install.sh
A    trunk/ac-helpers/install-sh
A    trunk/build.conf
...
Gerado cópia de trabalho para revisão 8810.
```

Ciclo Básico de Trabalho

- Atualizar sua cópia de trabalho
  - `svn update`
- Fazer alterações
  - `svn add`
  - `svn delete`
  - `svn copy`
  - `svn move`
- Verificar suas alterações
  - `svn status`
  - `svn diff`
- Possivelmente desfazer algumas alterações
  - `svn revert`
- Resolver conflitos (combinar alterações de outros)
  - `svn update`
  - `svn resolved`
- Submeter suas alterações
  - `svn commit`

Examinando o Histórico

- O repositório Subversion mantém um registro de cada modificação submetida, e permite a explorar este histórico examinando versões anteriores de seus arquivos e diretórios bem como os metadados a eles relacionados.
- É possível realizar um `checkout` do repositório para um estado exatamente como ele era em uma certa data ou em um determinado número de revisão no passado.
  - `svn log`
    - Exibe informações tais como: mensagens de log com informações de data e autor anexadas és revisões, e quais caminhos mudaram em cada revisão.
  - `svn diff`
    - Exibe detalhes a nível das linhas de uma alteração em particular.
  - `svn cat`
    - Recupera um arquivo como ele era em uma dada revisão e exibe seu conteúdo na tela.
  - `svn list`
    - Exibe os arquivos em um diretório para uma dada revisão.

Às Vezes Você Só Precisa Limpar

- Se uma operação do Subversion é interrompida (se o processo for morto, ou se a máquina travar, por exemplo), o arquivo de log permanece no disco.
- Executando novamente os arquivos de log, o Subversion pode completar a operação previamente iniciadas, e a cópia de trabalho pode voltar para um estado consistente.
  - `svn cleanup`
    - Varre a cópia de trabalho e executa quaisquer arquivos de log que esteja incompleto, removendo quaisquer impedimentos.

Propriedades

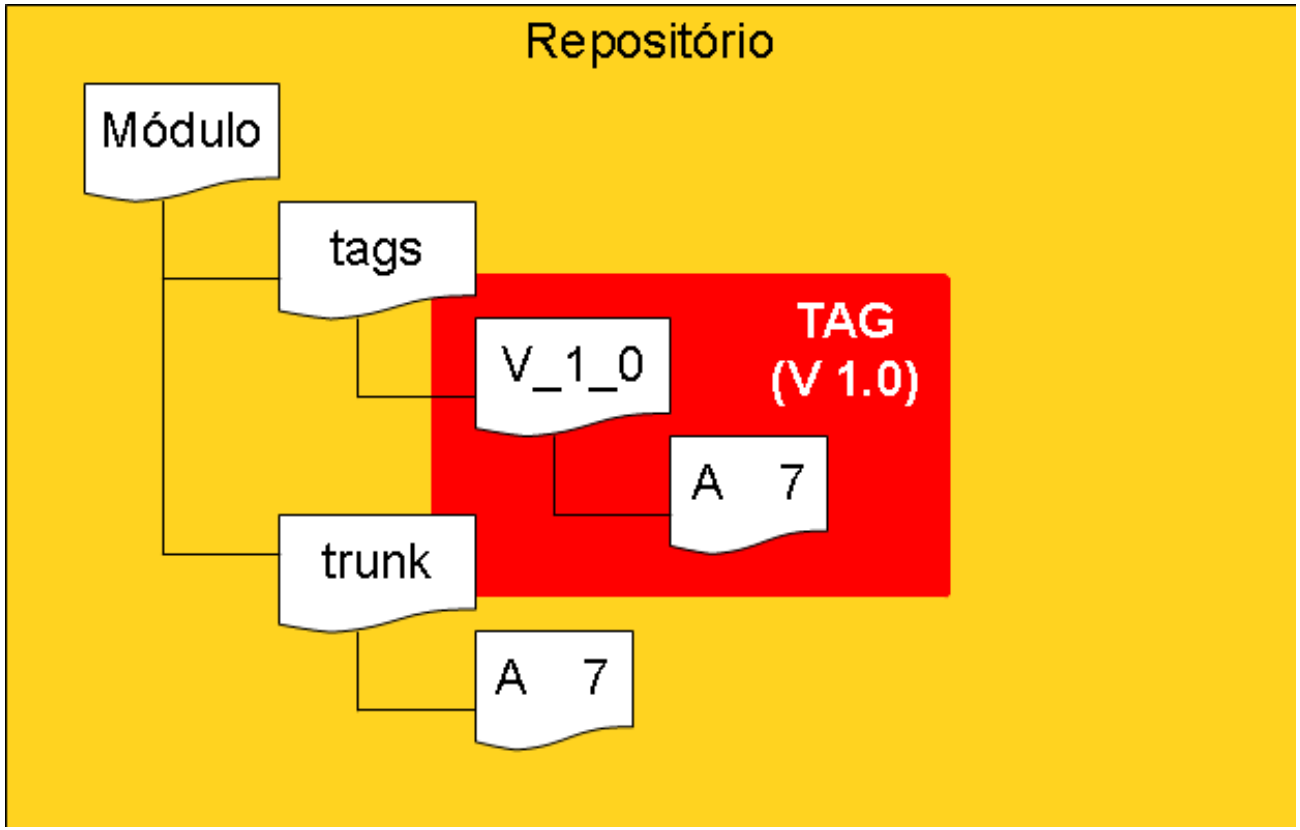
- Adicionalmente ao versionamento de seus arquivos e diretórios, o Subversion permite interfaces para adição, modificação e remoção de metadados versionados em cada um de seus arquivos e diretórios sob controle de versão.
- Chamamos estes **metadados** de propriedades, e eles podem ser entendidos como tabelas de duas colunas que mapeiam nomes de propriedades a valores arbitrários anexados a cada item em sua cópia de trabalho.
- Os nomes e valores das propriedades podem ser texto livre, com a restrição de que os nomes devem ser texto legível por humanos.
- Podem ser versionadas, tal como o conteúdo textual de seus arquivos.

Propriedades

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/butt
property 'copyright' set on 'calc/button.c'
$
$ svn propset license -F /path/to/LICENSE calc/button.c
property 'license' set on 'calc/button.c'
$
$ svn proplist calc/button.c
Properties on 'calc/button.c':
  copyright
  license
$ svn propget copyright calc/button.c
(c) 2006 Red-Bean Software
```

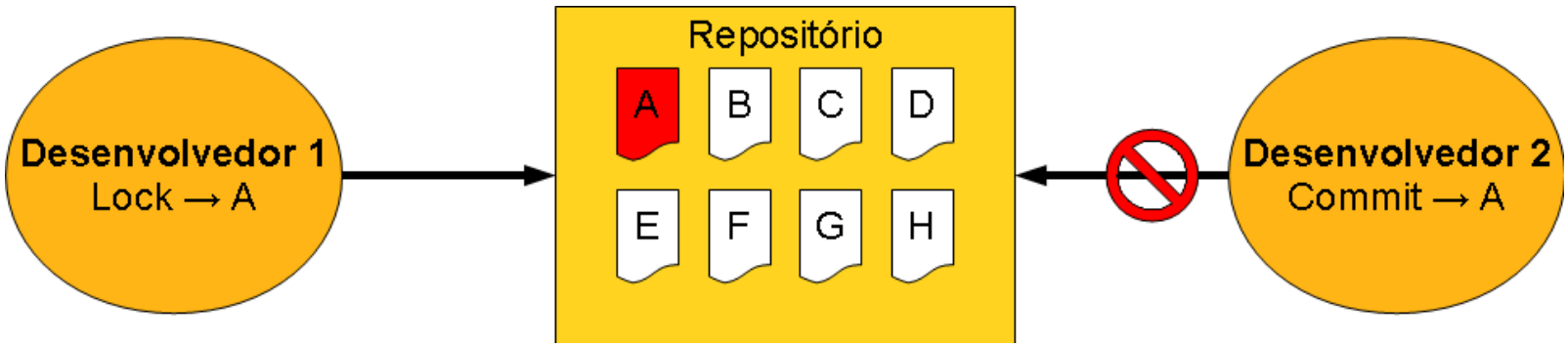
Tag (Rótulo)

- É uma fotografia do projeto em um dado tempo.
- Uma TAG é gerada a partir de uma revisão.
- Não deve haver alteração.



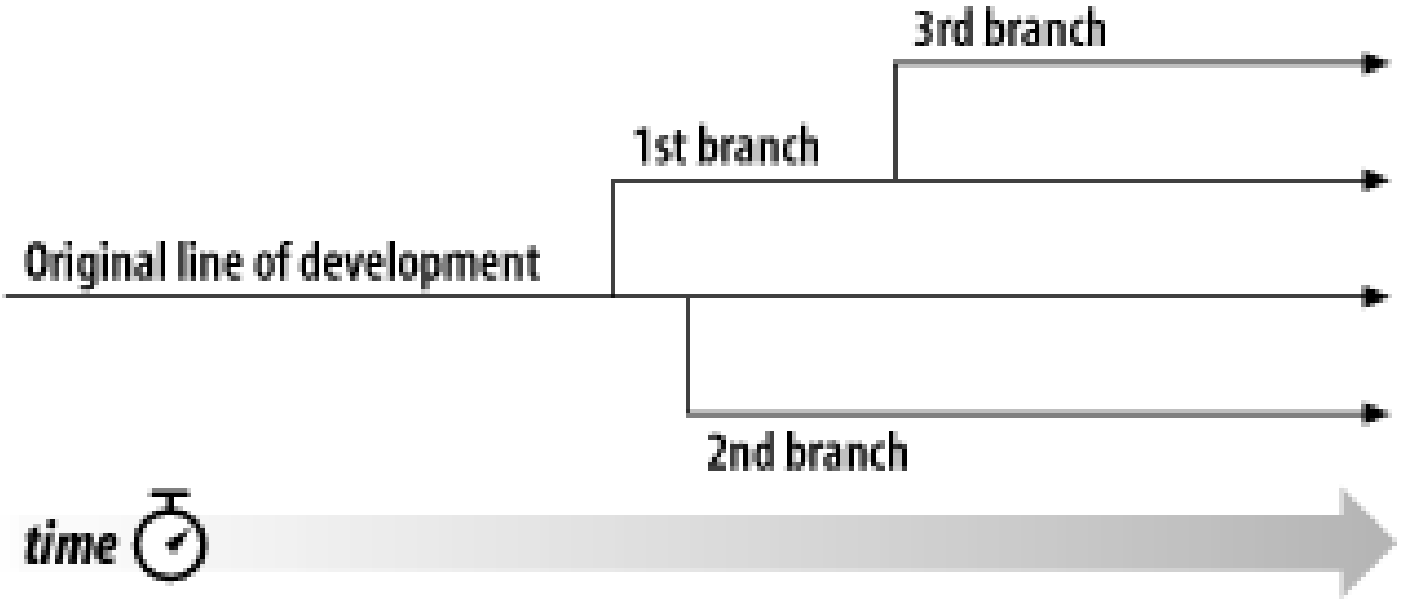
Lock / Release lock

- Bloqueia/libera a operação de commit para apenas um usuário.
- O usuário que bloquear um arquivo impede que os demais façam novos commits, impedindo o desenvolvimento, até que o arquivo seja liberado.



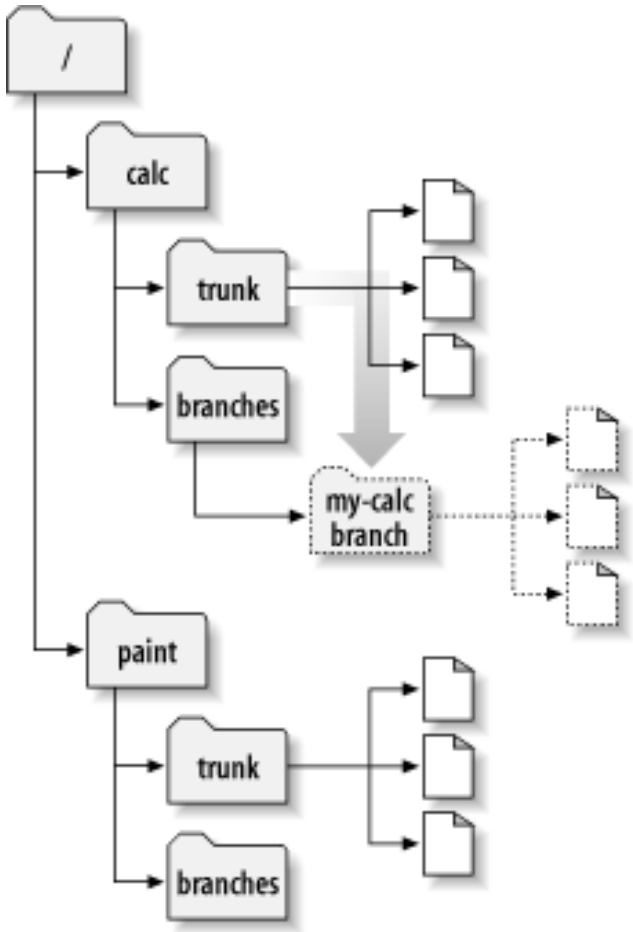
Branches (Ramos)

- Um **branch** é uma linha de desenvolvimento que existe independente de outra linha, e ainda, compartilham um histórico em comum.



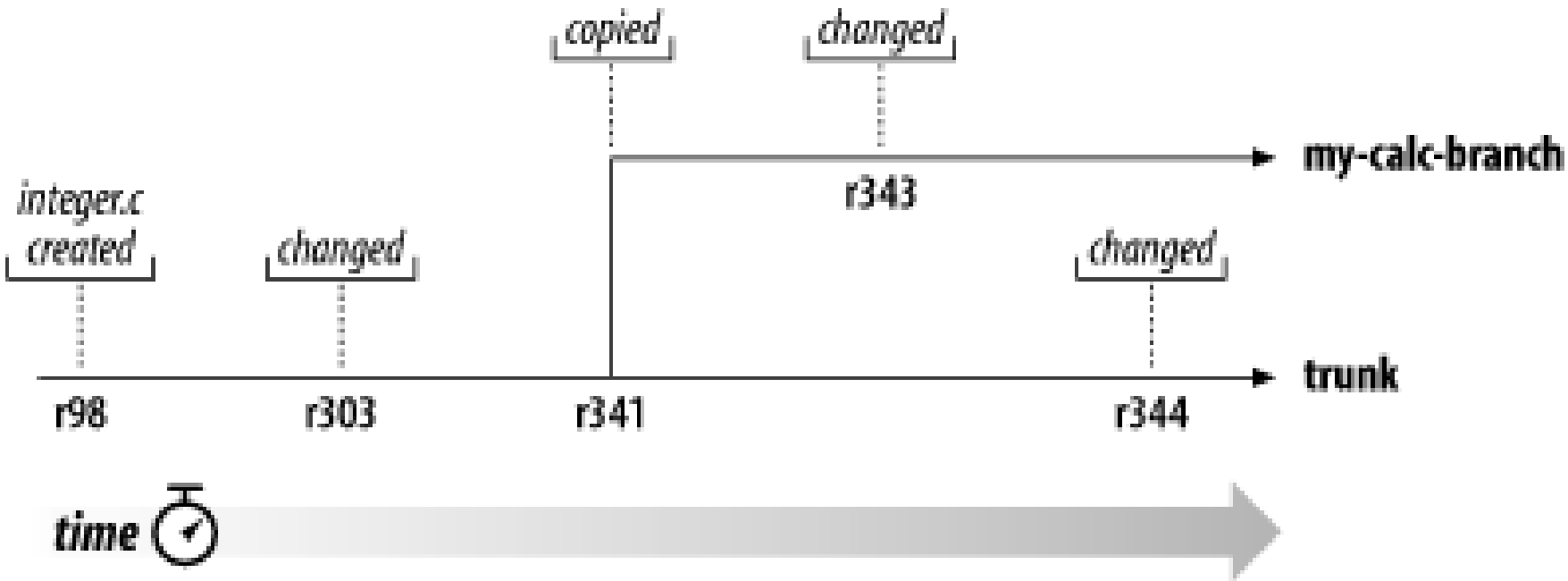
Branches (Ramos)

- Um **branch** nada mais é que uma cópia do diretório `trunk`
- Se faz uma cópia do projeto no repositório usando o comando `svn copy`.



Branches (Ramos)

- Um **branch** sempre se inicia como cópia de outra coisa, e segue rumo próprio a partir desse ponto, gerando seu próprio histórico.

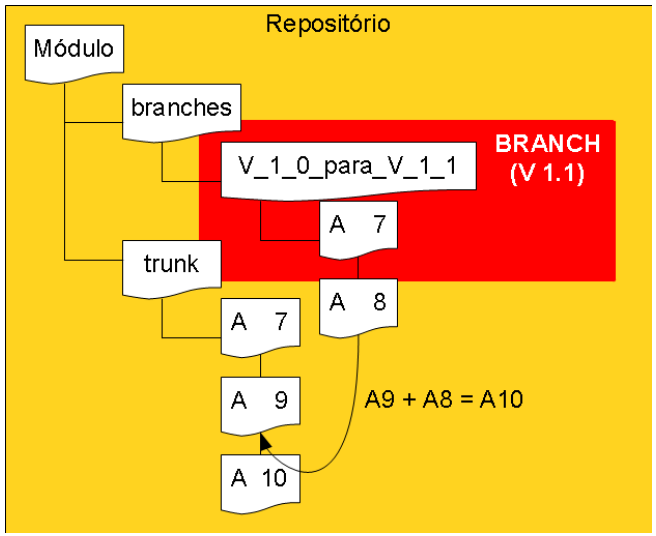


Copiando Modificações Entre Branches (Ramos)

- Para projetos que tenham um grande numero de colaboradores, é comum que cada um tenha sua cópia de trabalho do `trunk`.
- Sempre que alguem precise fazer uma longa modificação que possa corromper o `trunk`, o procedimento padrão é criar um ramo privado e fazer os commits neste ramo até que todo o trabalho esteja concluido.
- Deve-se continuamente compartilhar as modificações ao longo do trabalho, para não "se isolar" e ter problemas de sincronização de código no futuro

Merge (diff-and-apply)

- Faz a fusão (`merge`) de uma ramificação (`branch`) com o tronco principal (`trunk`) ou outros `branches`.
- O comando `svn merge` aplica as diferenças diretamente na cópia de trabalho classificando-as como modificações locais.
- O comando `merge` usa os números das revisões parar realizar a fusão de código.
- Usa propriedades para `svn:mergeinfo` para rastrear as fusões já realizadas.



Merge (diff-and-apply): conceito chave

Duas árvores de repositório são comparadas, e a diferença é aplicada a uma cópia de trabalho.

- Uma árvore de repositório inicial (geralmente chamada de lado esquerdo da comparação),
- Uma árvore de repositório final (geralmente chamada de lado direito da comparação),
- Uma cópia de trabalho para receber as diferenças como modificação local (geralmente chamada de destino da fusão).

Merge (diff-and-apply): conceito chave

- Uma vez especificados estes três argumentos, as duas árvores são comparadas, e o resultado das diferenças são aplicadas sobre a cópia de trabalho de destino, como modificações locais.
- Se o resultado for adequado se faz o `commit`.
- Caso contrário, pode-se simplesmente reverter as mudanças com o comando `svn revert`.

Ferramentas

- TortoiseSVN: Integração com Windows Explorer
- Subclise: Plugin Eclipse
- SmartSVN: Desenvolvido em Java
- KDESVN: Integração com o KDE
- RabbitVCS: Integração com o Nautilus do Gnome

RabbitVCS



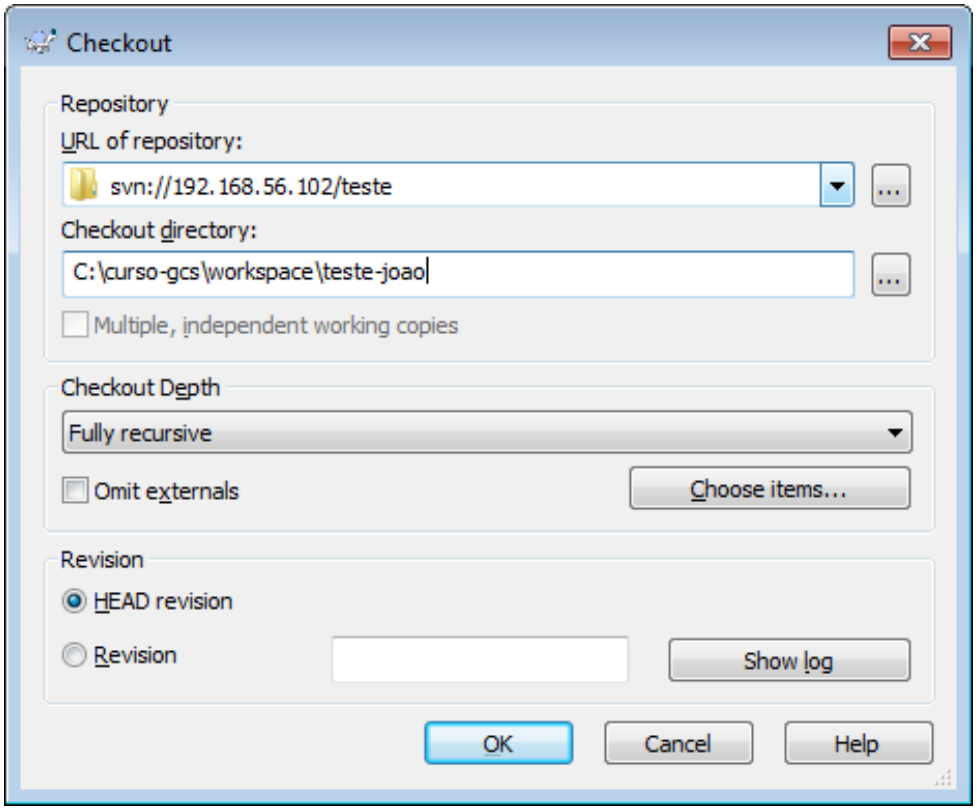
TortoiseSVN

Subversion

Parte 3

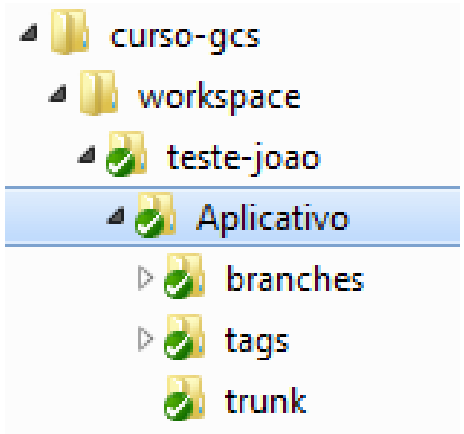
Prática

- Verifique se o servidor SVN, no Linux, está no ar. Caso contrário execute o comando abaixo:
  - `svnserve -d -r ~/curso-gcs/repositorio-svn/`
- Faça o `checkout` do repositório para uma cópia local:
  - `C:\curso-gcs\workspace\teste-joao`



Prática

- Crie o módulo **Aplicativo** com a estrutura padrão: branches, tags e trunk.
- Use o comando `add` para incluir no repositório.
- Use o comando `commit` para persistir no repositório
- Dentro do `/trunk` crie dois arquivos:
  - Classe.java
  - build.bat



Prática

- Classe.java
  - ```
    /**
     * Classe exemplo.
     * @author desenvolvedor@email.com
     */
    public class Classe {
        /**
         * Metodo inicial.
         * @param args Argumento de linha de comando
         */
        public static void main(String[] args) {
            // Comandos
            System.out.println("Ola, mundo!");
        }
    }
    ```
- build.bat
  - `javac *.java`
    ```

Prática

- Use o comando `add` para incluir no repositório.
- Use o comando `commit` para persistir no repositório.
- Utilize o comando `tag` para criar uma marca da situação do `trunk` atual:
  - `/Teste/Aplicativo/trunk` → `/Teste/Aplicativo/tags/V_1_0`
- Utilize o comando `branch` para criar um ramo da tag `V_1_0`:
  - `/Teste/Aplicativo/tags/V_1_0` → `/Teste/Aplicativo/branches/V_1_0_para_V_1_1`
- Utilize a funcionalidade *Revision Graph* para visualizar o ramo criado.
- Faça alterações no `branch` e `trunk` para ver o que mudou no gráfico.

Prática

Para testar o comando de `lock` é necessário fazer `checkout` com outro usuário:

- `C:\curso-gcs\workspace\teste-maria`
- Faça o bloqueio de um arquivo com o usuário **joao** e tente fazer `commit` com o usuário **maria**.
- Faça uma alteração no arquivo com o usuário **joao**, faça `commit`, perceba que o `commit` libera o bloqueio.
- O usuário **maria** poderá fazer seu `commit`

Prática

- Instala a ferramenta KDiff para auxiliar nos `merges`: `http://sourceforge.net/projects/kdiff3/files/kdiff3/0.9.97/`
- Caso as alterações feitas pelos usuários sejam em linhas distintas, o merge será automático, mas necessita de verificação manual.
- Caso as alterações feitas sejam na mesma linha, o merge não será automático. Será necessário a resolução de conflito.
- Simule as duas situações e veja como se comporta:
- Utilize as funções *Edit conflicts* e *Resolved...* do TortoiseSVN quando necessário
- Faça o `merge` de uma versão específica com um `branch` ou `trunk`

Prática

- Para testar o comando de `merge`, devemos alterar os arquivos no branch:
- Após isso, no `trunk`, devemos executar o comando `merge`
- Devemos resolver os conflitos e gerar nova versão

Prática

- Exclua um arquivo através o comando `delete` com o usuário `joao`:
- Atualize a cópia de trabalho do usuário **maria** para ver o resultado
- Visualize o `log` dos arquivos para ver mais informações
- Altere as propriedades de um arquivo
- Recupere uma cópia de um arquivo excluído

Referências

- Version Control with Subversion: `http://svnbook.red-bean.com`
- Pragmatic Version Control: Using Subversion (The Pragmatic Starter Kit Series)(2nd Edition)