TortoiseSVN

Um aplicativo do Subversion para Windows

Versão 1.6.16

Stefan Küng
Lübbe Onken
Simon Large

TortoiseSVN: Um aplicativo do Subversion para Windows: Versão 1.6.16

por Stefan Küng, Lübbe Onken, e Simon Large
tradutor-tradução: tradudor-créditos

Publicado 2011/01/21 21:21:17 (r20750)

Índice

Lista de Figuras

Lista de Tabelas

Prefácio

- Você trabalha com uma equipe?
- Você tem trabalhado em um arquivo, e outra pessoa também tem trabalhado no mesmo arquivo ao mesmo tempo? Você perdeu as modificações que fez por causa disso?
- Você modificou o arquivo, e depois quis desfazer as modificações que fez? Você já desejou ver como era o arquivo tempos atrás?
- Você encontrou um erro em seu projeto e quer saber quando o erro foi inserido em seus arquivos?

Se você respondeu “sim” para alguma dessas questões, então TortoiseSVN é para você! Simplesmente leia e descubra sobre como TortoiseSVN pode lhe ajudar no trabalho. Isto não é difícil.

1. Público alvo

This book is written for computer literate folk who want to use Subversion to manage their data, but are uncomfortable using the command line client to do so. Since TortoiseSVN is a windows shell extension it's assumed that the user is familiar with the windows explorer and knows how to use it.

2. Guia de leitura

This Prefácio explains a little about the TortoiseSVN project, the community of people who work on it, and the licensing conditions for using it and distributing it.

O Capítulo 1, *Introdução* explica o que é o TortoiseSVN, o que faz, de onde vem e o básico para instalar no computador.

Em Capítulo 2, *Conceitos básicos do Controle de Versão* damos uma introdução sobre o controle de revisão *Subversion* que é a base do TortoiseSVN. Isto é uma cópia da documentação do projeto Subversion e explica as diferentes formas de controlar versão, e como Subversion funciona.

O capítulo Capítulo 3, *O Repositório* explica como carregar um repositório local, que é vantajoso para testar Subversion e TortoiseSVN em um único computador. Também explica um pouco sobre o gerenciamento de um repositório que é relevante para repositórios localizados em um servidor. Existe também um capítulo sobre como configurar um servidor caso você precise de um.

O Capítulo 4, *Guia do Uso Diário* é a parte mais importante pois explica todas as principais características do TortoiseSVN e como usá-las. Com o formato de tutorial, inicia explicando a função de obter uma cópia de trabalho, depois como modificar a cópia de trabalho, então submeter as modificações, etc. E então continua com os tópicos avançados.

Capítulo 5, *The SubWCRev Program* é um outro aplicativo que acompanha TortoiseSVN que permite extrair informações da cópia de trabalho e escrever isto em arquivos.. Isto é útil para incluir informações sobre a construção dos seus projetos.

A seção Apêndice B, *Como eu faço...* responde as dúvidas comuns sobre a execução das tarefas que não estão explicitamente descritas em outras seções.

A parte Apêndice D, *Automatizando o TortoiseSVN* mostra como a as janelas do TortoiseSVN podem ser chamadas através da linha de comando. Isso pode ser útil em scripts que precisam da interação do usuário.

O Apêndice E, *Command Line Interface Cross Reference* mostra a relação entre os comandos do TortoiseSVN e seus equivalentes na linha de comando do aplicativo Subversion `svn.exe`.

3. TortoiseSVN é grátis!

TortoiseSVN é grátis. Você não precisa pagar nada para usar, e você pode usar da maneira que quiser. É um projeto desenvolvido sobre a Licença Pública Genérica - GNU General Public Licence (GPL).

TortoiseSVN is an Open Source project. That means you have full read access to the source code of this program. You can browse it on this link *http://code.google.com/p/tortoisesvn/source/browse/*. You will be prompted to enter username and password. The username is `guest`, and the password must be left blank. The most recent version (where we're currently working) is located under `/trunk/`, and the released versions are located under `/tags/`.

4. Comunidade

Both TortoiseSVN and Subversion are developed by a community of people who are working on those projects. They come from different countries all over the world and work together to create wonderful programs.

5. Agradecimentos

Tim Kemp
: por iniciar o projeto TortoiseSVN

Stefan Küng
: for the hard work to get TortoiseSVN to what it is now

Lübbe Onken
: pelos belos ícones, logomarca, encontrar erros, traduções e gerenciar as traduções

Simon Large
: for helping with the documentation and bug hunting

O livro Subversion
: por ensinar sobre o Subversion e pelo capítulo 2 de onde copiamos

O estilo de projeto Tigris
: pelos estilos da documentação de onde copiamos

Nossos Colaboradores
: for the patches, bug reports and new ideas, and for helping others by answering questions on our mailing list.

Nossos Contribuidores
: por muitas horas de diversão com a música que nos enviaram

6. Termos usados neste documento

Para facilitar a leitura do documento, o nome de todas as telas e Menus do TortoiseSVN estão em destaque. A Janela de Log por exemplo.

As opções de menu estão indicadas com uma seta. TortoiseSVN → Mostrar Log significa: selecionar *Mostrar Log* no menu *TortoiseSVN*.

Quando referenciar um menu específico dentro de uma das janelas do TortoiseSVN, será usado algo como: Menu Específico → Salvar como ...

Os Botões da Interface do Usuário são indicados como: Clique em OK para continuar.

As Ações do Usuário são indicados com a fonte em negrito. **Alt+A**: mantenha a tecla **Alt** pressionada no teclado até pressionar a tecla **A**. Right-drag: clique com o botão direito do mouse e mantenha pressionado *arrastando* o item até o novo local.

Saídas de vídeo e entradas de dados são indicadas com uma fonte `different`.

Importante

Detalhes importantes são indicados com um ícone.

Dica

Dicas tornam sua vida mais fácil.

Cuidado

Lugares onde você deve tomar cuidado com o que faz.

Atenção

Where extreme care has to be taken, data corruption or other nasty things may occur if these warnings are ignored.

Capítulo 1. Introdução

Controle de versão é a arte de administrar as mudanças das informações. Isto é uma ferramenta crítica para programadores, que normalmente gastam horas fazendo pequenas modificações em seus aplicativos e então desfazem ou verificam algumas dessas modificações no dia seguinte. Imagine uma equipe de várias desenvolvedores trabalhando juntos - e talvez simultaneamente em mesmos arquivos! - e você precisa ver porque um bom controle é necessário para *controlar uma possível desordem*.

1.1. O que é o TortoiseSVN?

TortoiseSVN is a free open-source client for the *Subversion* version control system. That is, TortoiseSVN manages files and directories over time. Files are stored in a central *repository*. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to recover older versions of your files and examine the history of how and when your data changed, and who changed it. This is why many people think of Subversion and version control systems in general as a sort of “time machine”.

Alguns sistemas de controle de versão também são um aplicativo de gerenciamento de configuração (SCM). Esses sistemas são especificamente adaptados para controlar estruturas de código fonte, e tem muitas características de um aplicativo específico de desenvolvimento - como um aplicativo para uma linguagem de programação específica, or fornecendo ferramentas de construção de software. Subversion, entretanto, não é um desses sistemas; é um sistema genérico que pode ser usado para administrar *qualquer* conjunto de arquivos, incluindo código fonte.

1.2. História do TortoiseSVN

In 2002, Tim Kemp found that Subversion was a very good version control system, but it lacked a good GUI client. The idea for a Subversion client as a Windows shell integration was inspired by the similar client for CVS named TortoiseCVS.

Tim studied the source code of TortoiseCVS and used it as a base for TortoiseSVN. He then started the project, registered the domain `tortoisesvn.org` and put the source code online. During that time, Stefan Küng was looking for a good and free version control system and found Subversion and the source for TortoiseSVN. Since TortoiseSVN was still not ready for use then he joined the project and started programming. Soon he rewrote most of the existing code and started adding commands and features, up to a point where nothing of the original code remained.

As Subversion became more stable it attracted more and more users who also started using TortoiseSVN as their Subversion client. The user base grew quickly (and is still growing every day). That's when Lübbe Onken offered to help out with some nice icons and a logo for TortoiseSVN. And he takes care of the website and manages the translation.

1.3. Caracterísiticas do TortoiseSVN

O que faz do TortoiseSVN um bom aplicativo cliente para Subversion? Aqui está uma pequena lista de recursos.

Interface integrada

TortoiseSVN integrates seamlessly into the Windows shell (i.e. the explorer). This means you can keep working with the tools you're already familiar with. And you do not have to change into a different application each time you need functions of the version control!

And you are not even forced to use the Windows Explorer. TortoiseSVN's context menus work in many other file managers, and in the File/Open dialog which is common to most standard Windows

applications. You should, however, bear in mind that TortoiseSVN is intentionally developed as extension for the Windows Explorer. Thus it is possible that in other applications the integration is not as complete and e.g. the icon overlays may not be shown.

Sobreposição dos ícones
: A situação de cada arquivo e diretório controlado é indicado por uma pequena sobreposição de ícones. O que permite a você ver rapidamente qual é a situação da sua cópia de trabalho.

Fácil acesso aos comandos do Subversion
: Todos os comandos do Subversion estão disponíveis nos menus do explorer. TortoiseSVN adiciona seu próprio submenu.

Uma vez que TortoiseSVN é um aplicativo cliente do Subversion, também gostaríamos de mostrar a você algumas das funcionalidades do Subversion:

Controle de diretório
: CVS somente mantém o histórico de alterações de arquivos individuais, mas Subversion usa um controle “virtual” de sistema de arquivos que mantém o histórico de toda a estrutura de diretório ao longo do tempo. Arquivos *e* diretórios são controlados. E como resultado, temos verdadeiros comandos para **mover** e **copiar** arquivos e diretórios.

Submissão atômica
: Cada submissão é enviada completamente para o repositório, ou não é enviado nada. Isto permite aos desenvolvedores construir e submeter as alterações em partes coesas.

Metadados controlados
: Cada arquivo e diretório possue um conjunto de “propriedades” invisíveis. Você pode inventar e gravar qualquer conjunto de chave/valor que desejar. Propriedades são controladas ao longo do tempo, exatamente como o conteúdo dos arquivos.

Escolha das camadas da rede
: Subversion tem uma noção abstrata de acesso ao repositório, tornando fácil para as pessoas desenvolverem novos mecanismos de rede. O servidor de rede avançado do Subversion é um módulo para o servidor web Apache, do qual expõe uma variante do HTTP chamada WebDAV/DeltaV. Isto dá ao Subversion uma grande vantagem em estabilidade e interoperabilidade, e provê várias funcionalidades chave de graça: autenticação, autorização, compressão, e navegação no repositório, por exemplo. Uma característica menor, um processo servidor autônomo do Subversion também está disponível. Este servidor exterioriza um protocolo específico que pode ser facilmente encapsulado sobre o protocolo ssh.

Manipulação consiste de dados
: Subversion apresenta as diferenças de arquivos usando um algoritmo de comparação binária, que funciona igualmente para arquivos texto (compreensíveis) and binários (ilegíveis). Ambos os tipos de arquivos são gravados compactados da mesma forma no repositório, e as diferenças são trasmitidas em ambas as direções através da rede.

Ramificação e Rotulação eficiente
: Os recursos necessários para ramificar e rotular não é propocional ao tamanho do projeto. Subversion cria ramos e rótulos simplesmente copiando o projeto, usando um mecanismo parecido ao hard-link. Deste modo estas operações são realizadas rapidamente sem variação de tempo, e consomem muito pouco espaço no repositório.

Hackability
: Subversion não tem uma bagagem histórica; ele foi desenvolvido como uma coleção de bibliotecas compartilhas em C com APIs bem definidas. Isto torna o Subversion de fácil manutenção e utilizável por outras aplicações e linguagens.

1.4. Instalando TortoiseSVN

1.4.1. Requerimentos do sistema

TortoiseSVN runs on Windows 2000 SP2, Windows XP or higher. Windows 98, Windows ME and Windows NT4 are no longer supported since TortoiseSVN 1.2.0, but you can still download the older versions if you really need them.

If you encounter any problems during or after installing TortoiseSVN please refer to Apêndice A, *Frequently Asked Questions (FAQ)* first.

1.4.2. Instalação

TortoiseSVN comes with an easy to use installer. Double click on the installer file and follow the instructions. The installer will take care of the rest.

Importante

Você precisa instalar o TortoiseSVN como Administrador do sistema.

1.4.3. Pacotes de Idioma

The TortoiseSVN user interface has been translated into many different languages, so you may be able to download a language pack to suit your needs. You can find the language packs on our *translation status page* [http://tortoisesvn.net/translation_status]. And if there is no language pack available yet, why not join the team and submit your own translation ;-)

Each language pack is packaged as a `.exe` installer. Just run the install program and follow the instructions. Next time you restart, the translation will be available.

1.4.4. Corretor ortográfico

TortoiseSVN inclui um corretor ortográfico que lhe permite verificar as mensagens de log das submissões. Isto é especialmente útil se o idioma do projeto não é seu idioma nativo. O corretor ortográfico usa o mesmo arquivo de dicionário que *OpenOffice* [http://openoffice.org] e *Mozilla* [http://mozilla.org].

The installer automatically adds the US and UK English dictionaries. If you want other languages, the easiest option is simply to install one of TortoiseSVN's language packs. This will install the appropriate dictionary files as well as the TortoiseSVN local user interface. Next time you restart, the dictionary will be available too.

Ou você pode instalar o dicionário você mesmo. Se você tem o OpenOffice ou o Mozilla instalado, você pode copiar esses dicionários, que estão localizados nos diretórios da instalação dessas aplicações. Entretanto, você precisa baixar os arquivos necessários do dicionário de *http://wiki.services.openoffice.org/wiki/Dictionaries*

Uma vez que você tenha os arquivos do dicionário, você provavelmente precisa renomeá-los para que os nomes dos arquivos somente tenham caracteres do idioma. Por exemplo:

- `en_US.aff`
- `en_US.dic`

Então apenas copie eles para o subdiretório `bin` do diretório de instalação do TortoiseSVN. Normalmente o caminho será `C:\Arquivos de Programs\TortoiseSVN\bin`. Se você não quer bagunçar o subdiretório `bin`, você pode ao invés disso mover os arquivos do seu idioma para `C:`

`\Arquivos de Programas\TortoiseSVN\Languages`. Se este diretório não existir, você precisar primeiro criá-lo. Na próxima vez que iniciar o TortoiseSVN, o seu idioma estará disponível.

Se você instalar vários dicionários, TortoiseSVN usa essas regras para selecionar uma para usar.

1. Verifique a configuração `tsvn:projectlanguage`.Veja Seção 4.17, “Configurações do Projeto” para informações sobre a configuração.

2. Se o idioma do projeto não estiver configurado, ou o idioma não estiver instalado, tente o idioma correspondente da configuração do Windows.

3. Se o exato idioma do Windows não funcionar, tente o idioma “Padrão”, ex: `de_CH` (Alemão-Suiço) troca por `de_DE` (Alemão).

4. Se nada disso funcionar, então o idioma padrão é o Inglês, que está incluido na instalação normal.

Capítulo 2. Conceitos básicos do Controle de Versão

Este capítulo é uma versão levemente modificada do mesmo capítulo do livro do Subversion. Há uma versão disponível do livro do Subversion aqui: *http://svnbook.red-bean.com/*.

Este capítulo é uma pequena, simplória introdução ao Subversion. Se você é novato em controle de versões, este capítulo com certeza é para você. Nós começamos com uma abordagem geral dos conceitos do controle de versão, seguindo nossas idéias sobre o Subversion, e mostrando alguns exemplos simples de uso do Subversion.

Embora os exemplos desse capítulo mostre pessoas compartilhando uma porção de código de fonte, tenha em mente que Subversion pode controlar qualquer conjunto de arquivos - não está limitado a ajudar programadores.

2.1. O Repositório

Subversion é um sistema centralizador de compartilhamento de informação. A essência é um *repositório*, que é um arquivo central de dados. O repositório grava informação no formato de *diretório de arquivo de sistemas* - uma típica hierarquia de arquivos e diretórios. Quantos *clientes* quiser se conectam no repositório, e então leem e escrevem nesses arquivos. Escrevendo dados, um usuário torna a informação disponível para outros; lendo os dados, um usuário recebe a informação de outros.

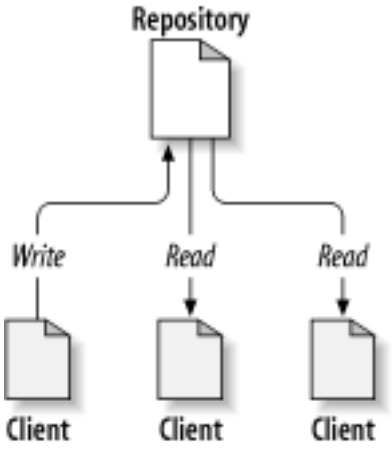


Figura 2.1. Um típico sistema Cliente/Servidor

Então porque isto é interessante? Até agora, isso se parece com a definição de um típico servidor de arquivos. E de verdade, o repositório *é* um tipo de servidor de arquivos, mas não é sua função normal. O que o repositório do Subversion faz de especial é que *ele guarda cada modificação* feita: cada modificação para cada arquivo e mesmo as modificações na prória estrutura de diretórios, como adição, exclusão e reorganização de arquivos e diretórios.

When um cliente lê uma informação do repositório, normalmente vê apenas a última versão da estrutura de arquivos. Mas o usuário também pode ver estruturas mais *antigas*. Por exemplo, um usuário pode fazer perguntas como, “o que este diretório continha na últma quarta-feira?”, ou “quem foi a última pessoa a modificar este arquivo, e que modificações foram feitas?” Estas são os tipos de perguntas principais em qualquer *sistema de controle de versão*: sistemas que são projetados para gravar e armazenar modificações dos dados ao longo do tempo.

2.2. Modelo de controles

Todo sistema de controle de versão precisa resolver algumas problemas fundamentais: como o sistema vai permitir aos usuários compartilhar a informação, mas prevenindo que um não atrapalhe o outro? É muito fácil para os usuários acidentalmente sobreescrever as modificações de outros no repositório.

2.2.1. O problema do compartilhamento de arquivo

Considere o cenário: suponha que nós temos dois usuários, Harry e Sally. Cada um deles decide editar o mesmo arquivo no mesmo repositório ao mesmo tempo. Se Harry salvar suas alterações no repositório primeiro, é possível que (alguns tempo depois) Sally poderia acidentalmente sobreescrever as alterações com sua versão do arquivo. Enquanto a versão do arquivo do Harry se perderia para sempre (porque o sistema guarda cada modificação), qualquer alteração que Harry fez *não estariam* presentes na nova versão do arquivo de Sally, porque ela nunca viu as modificações do Harry. O trabalho do Harry foi perdido - ou no mínimo estaria na versão mais recente - e provavelmente por acidente. Esta é definitivamente um situação que nós queremos evitar!

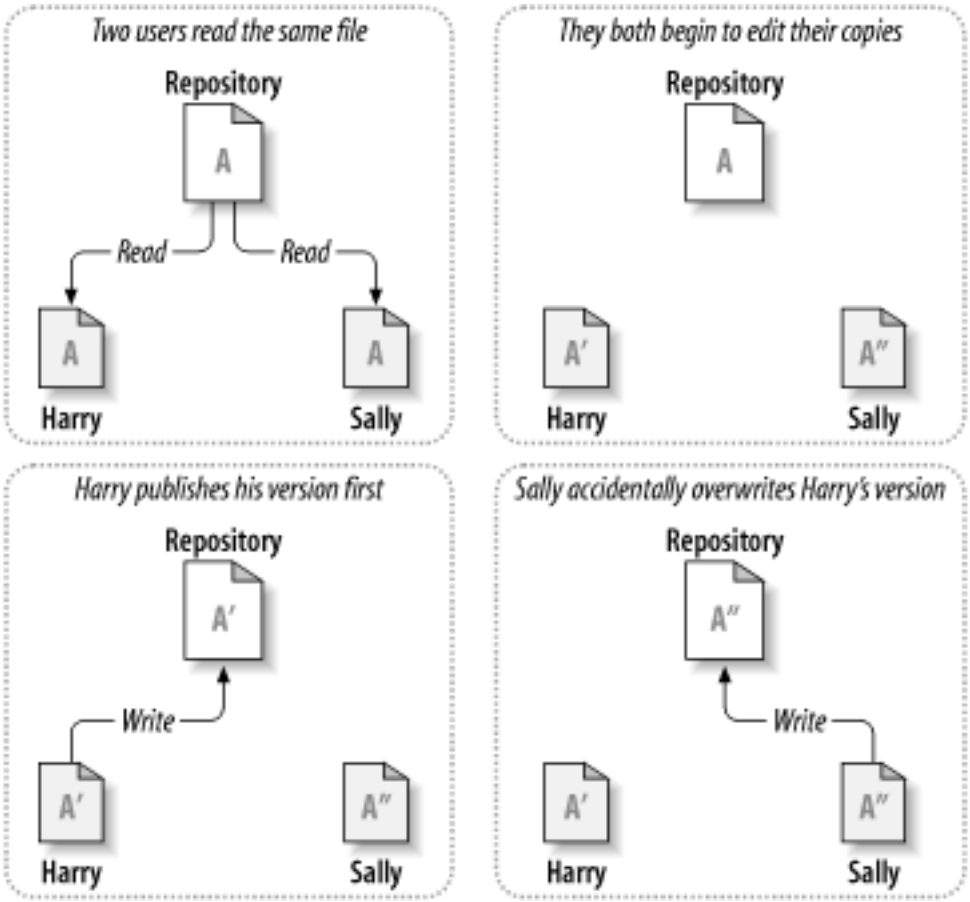


Figura 2.2. O problema a ser evitado

2.2.2. A solução Alocar-Modificar-Desalocar

Muitos sistemas de controle de versão usam o modelo *alocar-modificar-desalocar* para resolver este problema, o qual é uma solução simples. Em cada sistema, o repositório permite somente uma pessoa por vez modificar o arquivo. Primeiro Harry deve *alocar* o arquivo antes que possa fazer as alterações. Alocar um arquivo é como um controle de biblioteca/ se Harry alocou o arquivo, então Sally não pode fazer qualquer alteração nele. Se ela tentar alocar o arquivo, o repositório vai negar essa solicitação. Tudo que ela pode fazer é ler o arquivo, e esperar até que Harry acabe as suas alterações e libere o arquivo. Depois que Harry desalocar o arquivo, sua vez acaba, e então é a vez de Sally alocar o arquivo e fazer suas alterações.

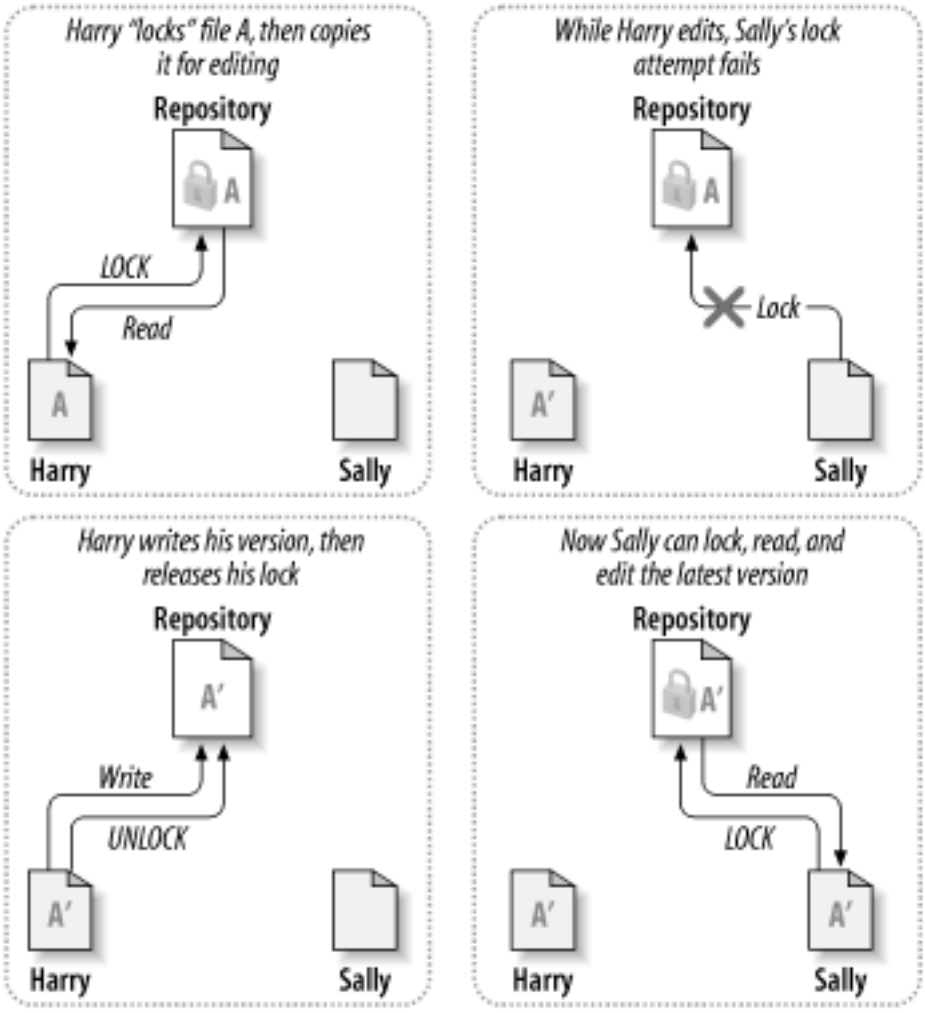


Figura 2.3. A solução Alocar-Modificar-Desalocar

O problema com o modelo alocar-modificar-desalocar é que é muito restritivo, e muitas vezes é um empecilho para os usuários:

- *Alocação pode causar problemas administrativos.* Algumas vezes Harry vai alocar o arquivo e então esquecer dele. Entretanto, porque Sally continua esperando para para editar o arquivo, suas mãos estão atadas. E então Harry sai de férias. Agora Sally precisa pedir a um administrador para liberar o arquivo alocado por Harry. A situação acaba causando atrasos e uma porção de tempo perdido desnecessário.

- *Alocação pode causa uma serialização desnecessária.* O que fazer se Harry estava editando o início de um arquivo texto, e Sally quer editar apenas o fim do arquivo? Estas alterações não se sobrepoem. Eles poderiam facilmente editar o arquivo ao mesmo tempo, e nenhum grande problema ocorreria, assumindo que as modificações seriam corretamente unificadas. Não é necessário travar o arquivo neste caso.

- *Alocar pode criar uma falta noção de segurança.* Suponhamos que Harry aloque e altere o arquivo A, enquanto ao mesmo tempo Sally aloca e edita o arquivo B. Supondo que A e B dependem um do outro, e que as modificações feitas em cada um são semanticamente incompatíveis. Imprevisivelmente A e B não funcionam mais juntos. O sistema de alocação não tem como prever este problema - ainda que esse sistema passe uma falta sensação de segurança. É fácil para Harry e Sally imaginar que alocando arquivos, cada um está seguro, numa tarefa isola, e deste modo evitando discussões precosss sobre as modificações.

2.2.3. A solução Copiar-Modificar-Unificar

Subversion, CVS e outros sistemas de controle de versão usam o modelo *copiar-modificar-unificar* como alternativa para a alocação. Neste modelo, cada usuário lê o repositório e cria uma *cópia de trabalho* pessoal dos arquivos do projeto. Eles então trabalham de forma paralela, modificando suas próprias cópias. No final, as cópias locais são unificadas com uma nova versão, uma versão final. O sistema de controle de versão oferece ajuda com a unificação, mas no final uma intervenção humana é que decide como a unificação será feita.

Aqui vai um exemplo. Digamos que Harry e Sally criam cada um uma cópia de trabalho de um mesmo projeto, copiado do repositório. Eles trabalham ao mesmo tempo, e fazem modificações em um mesmo arquivo A em suas cópias. Sally salva suas alterações no repositório primeiro. Quando Harry tenta salvar suas modificações após Sally, o repositório informa a ele que o arquivo A está *desatualizado*. Em outras palavras, o arquivo A do repositório tem alguma alteração desde a última vez que ele foi copiado. Então Harry é questionado sobre a *unificação* das modificações no repositório serem inseridas em sua cópia do arquivo A. Oportunamente as modicações de Sally não foram sobreescritas pelas dele; uma vez que ele unificou ambas as alterações, ele salva a sua cópia no repositório.

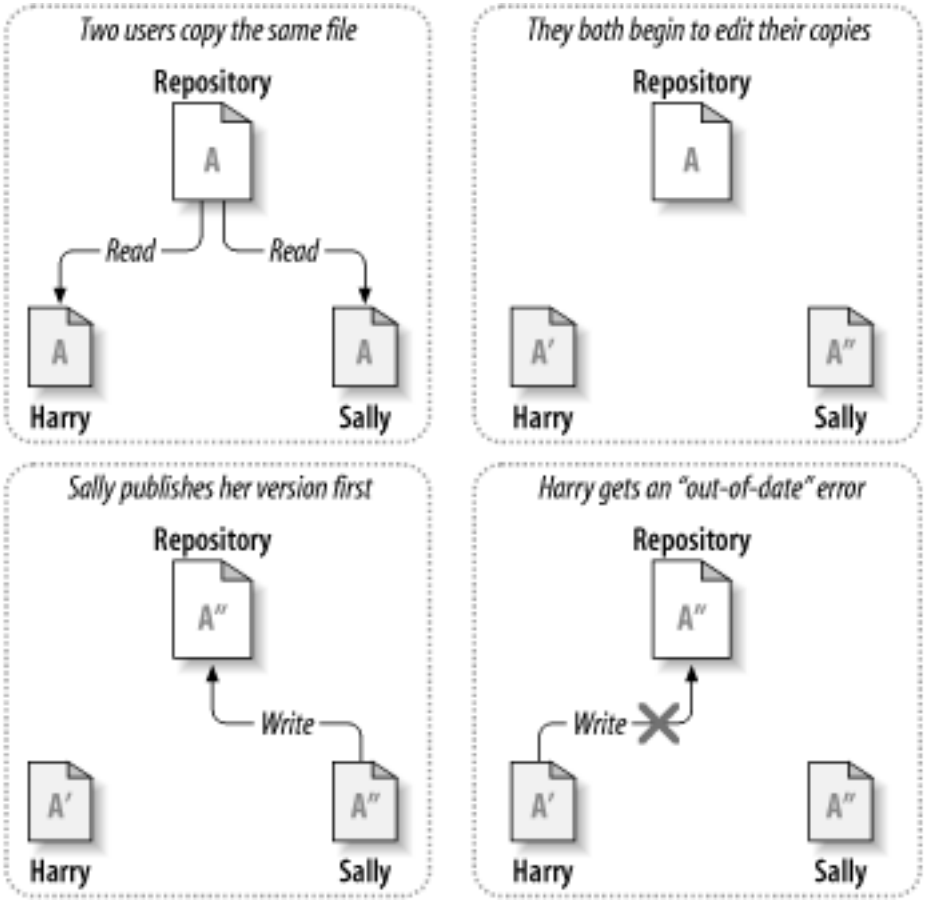


Figura 2.4. A solução Copiar-Modificar-Unificar

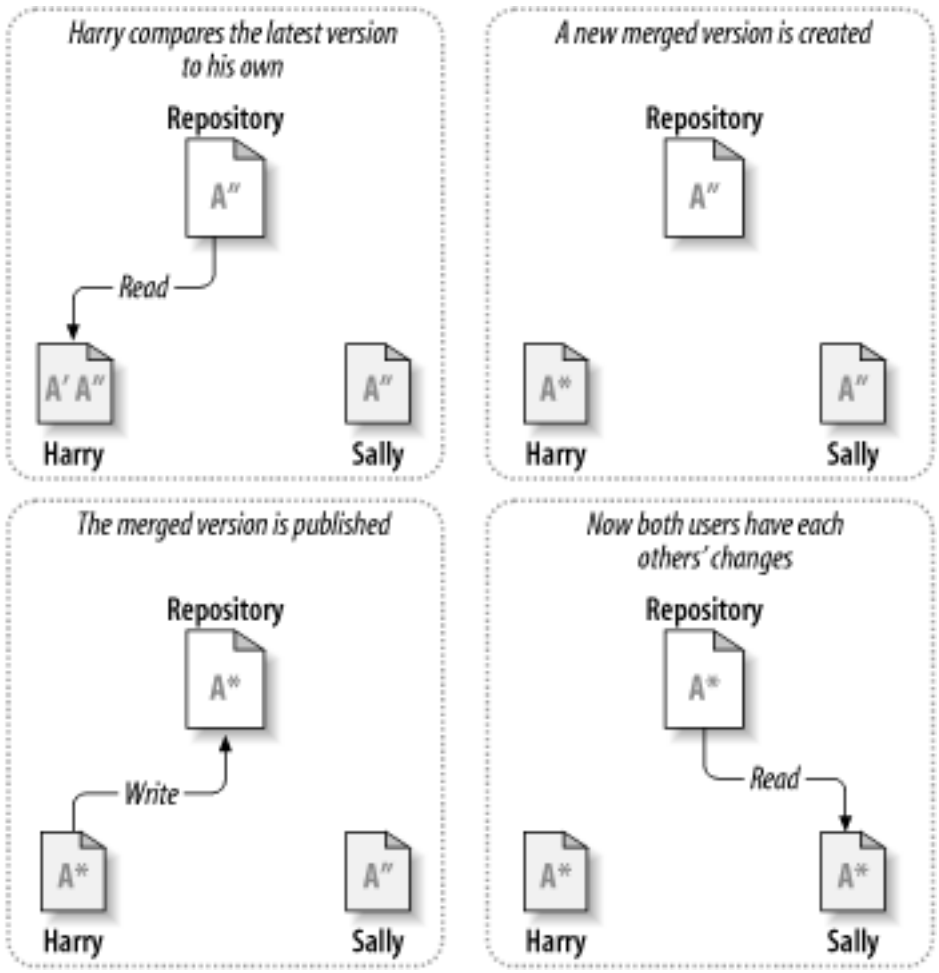


Figura 2.5. ...Continuando com Copiar-Modificar-Unificar

Mas o que acontece se as alterações de Sally se *sobrepoem* sobre as alterações de Harry? O que então deve ser feito? Esta situação é chamada de *conflito*, e em geral isto não é um problema. Quando Harry for questionado sobre a unificação das últimas alterações no repositório em sua cópia de trabalho, sua cópia do arquivo A de qualquer maneira é marcada como conflitante: ele verá todas as modificações em conflito, e manualmente resolverá. Entenda que o software não pode resolver automaticamente os conflitos; somente humanos são capazes de entender quais são as escolhas lógicas a serem tomadas. Uma vez que Harry manualmente resolveu o que estava se sobreponde (talvez precise discutir o conflito com Sally!), ele pode seguramente salvar o arquivo unificado de volta no repositório.

O modelo copiar-modificar-unificar pode parecer bagunçado, mas na prática, isto funciona grandemente. Usuários podem trabalhar paralelamente, sem esperar por outros. Quando eles trabalham num mesmo arquivo, a maioria das alterações não se sobrepoem; conflitos são pouco frequentes. E na maioria das vezes o tempo que leva para resolver os conflitos é muito menor que o tempo perdido em um sistema de alocação.

No final das contas, tudo isso acaba em um ponto crítico: comunicação entre os usuários. Quando os usuários não se comunicam direito, aumenta-se os conflitos sintáticos e semênticos. Nenhum sistema pode forçar uma perfeita comunicação, nenhum sistema pode detectar conflitos de semântica. Então não há porque se entusiasmar com falsas promessas de que um sistema de alocação evitará conflitos; na prática, alocações parecem restringir a produtividade mais que qualquer outra cosa.

Existe uma situação em comum onde o modelo alocar-modificar-desalocar se torna melhor, e é com arquivos que não são unificáveis. Por exemplo, se seu repositório contém alguns arquivos de imagens, e

duas pessoas mudam a imagem ao mesmo tempo, não há nenhuma maneira de combinar as alterações. Ou Harry ou Sally perderá sua alteração.

2.2.4. O que o Subversion faz?

Subversion usa a solução copiar-modificar-unificar como padrão, e na maioria dos casos é o suficiente. Contudo, a partir da versão 1.2, Subversion também permite alocar um arquivo, e se existem arquivos que não são unificáveis, você pode adotar uma política de alocação, que o Subversion está preparado para o que você precisa.

2.3. Subversion em Ação

2.3.1. Cópias de Trabalho

Você está pronto para ler sobre cópias de trabalho; agora vamos demonstrar como o aplicativo do Subversion cria e usa as cópias.

Uma cópia de trabalho do Subversion é uma estrutura de diretórios como outra qualquer em seu sistema local, contendo um conjunto de arquivos. Você pode editar esses arquivos como desejar, e se os arquivos são códigos fonte, você pode compilar seu programa a partir desta cópia como sempre fez. Sua cópia de trabalho é sua própria área pessoal. Subversion nunca vai incorporar alterações de outras pessoas, nem disponibilizar suas alterações para outros, até que você mesmo o faça.

Depois que você fez algumas modificações nos seus arquivos em sua cópia de trabalho e verificou que estão funcionando corretamente, Subversion provê a você comandos para *publicar* suas alterações para outras pessoas que trabalham com você em seu projeto (através da escrita no repositório). Se outras pessoas publicarem suas próprias modificações, Subversion provê comandos para unificar estas modificações na sua cópia de trabalho (através da leitura do repositório).

Uma cópia de trabalho também contém alguns arquivos extras, criados e mantidos pelo Subversion, para ajudar a executar os comandos. Em questão, cada diretório da sua cópia de trabalho contém um subdiretório chamado `.svn`, também conhecido como cópia de trabalho do *diretório administrativo*. Os arquivos em cada diretório administrativo ajuda o Subversion a reconhecer quais arquivos possuem alterações não publicadas, e quais arquivos estão desatualizados em relação ao repositório.

Um repositório típico do Subversion geralmente guarda os arquivos (ou código fonte) para diversos projetos; normalmente, cada projeto é um subdiretório na estrutura de arquivos do repositório. Desta forma, uma cópia de trabalho de um usuário normalmente corresponde a uma específica subestrutura do repositório.

Por exemplo, suponha que você tenha um repositório que contem dois projetos de software.

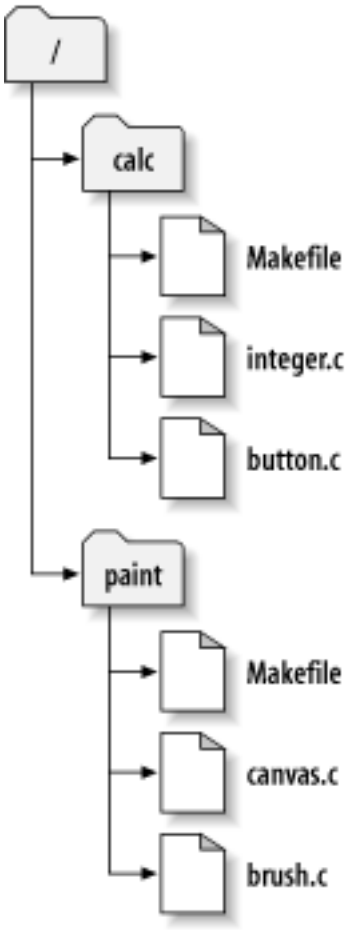


Figura 2.6. O Sistema de Arquivos do Repositório

Em outras palavras, o diretório principal do repositório tem dois subdiretórios: `paint` e `calc`.

Para pegar uma cópia de trabalho, você deve *obter* alguma subestrutura do repositório. (O termo *obter* pode soar como uma alocação ou reserva de recursos, mas não é; isto simplesmente cria uma cópia local do projeto para você).

Suponha que você fez mudanças para `button.c`. Já que o diretório `.svn` sabe a data de modificação do arquivo e seu conteúdo original, Subversion pode avisar que você fez modificações no arquivo. Contudo, Subversion não publica nenhuma alteração até que você realmente o faça. O ato de publicar suas alterações é mais conhecido como *submeter* (ou *enviar*) modificações para o repositório.

Para publicar suas modificações para outras pessoas, você pode usar o commando **submeter** do Subversion.

Agora suas modificações para `button.c` foram enviadas para o repositório; se outro usuário obter uma cópia de `/calc`, eles verão suas modificações na última versão do arquivo.

Suponha que você tenha uma colaboradora, Sally, que obteve uma cópia de `/calc no mesmo momento que você. Quando você submeter suas modificações de button.c, a cópia de trabalho de Sally não será modificada; Subversion somente modificará as cópias dos usuários que solicitarem a atualização.`

Para atualizar seu projeto, Sally deve pedir ao Subversion para *atualizar* sua cópia de trabalho, através do comando **atualizar** do Subversion. Isto incorporará sua modificação na cópia de trabalho dela, assim como qualquer outra modificação que tenha sido submetido desde que ela obteve sua cópia de trabalho.

Note que Sally não precisou especificar quais arquivos atualizar; Subversion usa a informação do diretório `.svn`, e além disso informações do repositório, para decidir quais arquivos precisam ser atualizados.

2.3.2. URLs do Repositório

Repositórios do Subversion podem ser acessados através de vários métodos direfentes - como disco local, ou através de vários protocolos de rede. Um local de repositório, contudo, é sempre uma URL. O esquema de URL indica o método de acesso:

Esquema	Método de Acesso
`file://`	Acesso direto ao repositório em um local ou dispositivo de rede.
`http://`	Acesso através do protocolo WebDAV para o módulo Subversion do servidor Apache.
`https://`	O mesmo que `http://`, mas com criptografia SSL
`svn://`	Acesso TCP/IP não autenticado através do protocolo customizado para um servidor `svnserve`.
`svn:ssh://`	acesso TCP/IP autenticado e criptografado através de um protocolo customizado para um servidor `svnserve`

Tabela 2.1. URLs de Acesso ao Repositório

Para a maioria, URLs do Subversion usam a sintaxe padrão, permitido nomes de servidores e números de porta em uma parte específica da URL. O método de acesso `file://` é normalmente usado para acesso local, apesar de que isto pode ser usado com caminhos UNC em uma máquina da rede. A URL portanto usa o formato `file://hostname/path/to/repos`. Para a máquina local, a parte do `hostname` da URL deve ser suprimida ou deve ser `localhost`. Por esta razão, caminhos locais normalmente aparecem com três barras, `file:///path/to/repos`.

Também, usuários do esquema `file://` em plataformas Windows vão precisar usar uma sintaxe “padrão” não oficial para acessar repositórios que estão na mesmo máquina, mas em um dispositivo diferente que o atual dispositivo de trabalho da máquina. Qualquer uma das duas seguintes sintaxes funcionarão onde `X` é o dispositivo no qual o repositório está:

```
file:///X:/path/to/repos
...
file:///X|/path/to/repos
...
```

Note que a URL usa barras normais embora a forma nativa (não URL) de um caminho no Windows seja barra invertida.

Você pode garantir o acesso a um repositório FSFS através de um compartilhamento de rede, mas você *não pode* acessar um repositório BDB desta forma.

Atenção

Não crie or acesse um repositório Berkeley DB em uma rede compartilhada. Isto *não pode* existir em um sistema de arquivos remoto. Nem mesmo se você tem dispositivos de rede mapeados em uma unidade de disco. Se você tentar usar Berkeley DB em uma rede compartilhada, os resultados são imprevisíveis - você poderá ver erros misteriosos imediatamente, ou meses antes descobrir que seu repositório está sutilmente corrompido.

2.3.3. Revisões

Um comando **svn submeter** pode publicar modificações de qualquer quantidade de arquivos e diretórios como uma única transação atômica. Em sua cópia de trabalho, você pode mudar o conteudo de um

arquivo, criar, excluir, renomear e copiar arquivos e diretórios, e depois enviar todas as modificações como uma única e completa alteração.

Em um repositório, cada submissão é tratada como uma transação atômica: ou todas as mudanças são enviadas, ou nenhuma delas. Subversion mantém essa atomicidade por causa de falhas de programas, falhas de sistema, falhas de rede, e ações de outros usuários.

Cada vez que o repositório aceita uma submissão, ele cria um novo estado da estrutura de arquivos, chamada de *revisão*. Cada revisão é associada a um número natural, maior que o número da revisão anterior. A revisão inicial de um repositório recém criado é zero, e não contém nada mais que um diretório principal vazio.

Uma maneira interessante de enxergar o repositório é como uma série de estruturas. Imagine uma matriz de números de revisões, começando em 0, e se estendendo da esquerda para a direita. Cada número de revisão com uma estrutura de arquivos abaixo, e cada divisão já em seguida da maneira como o repositório realizou cada submissão.

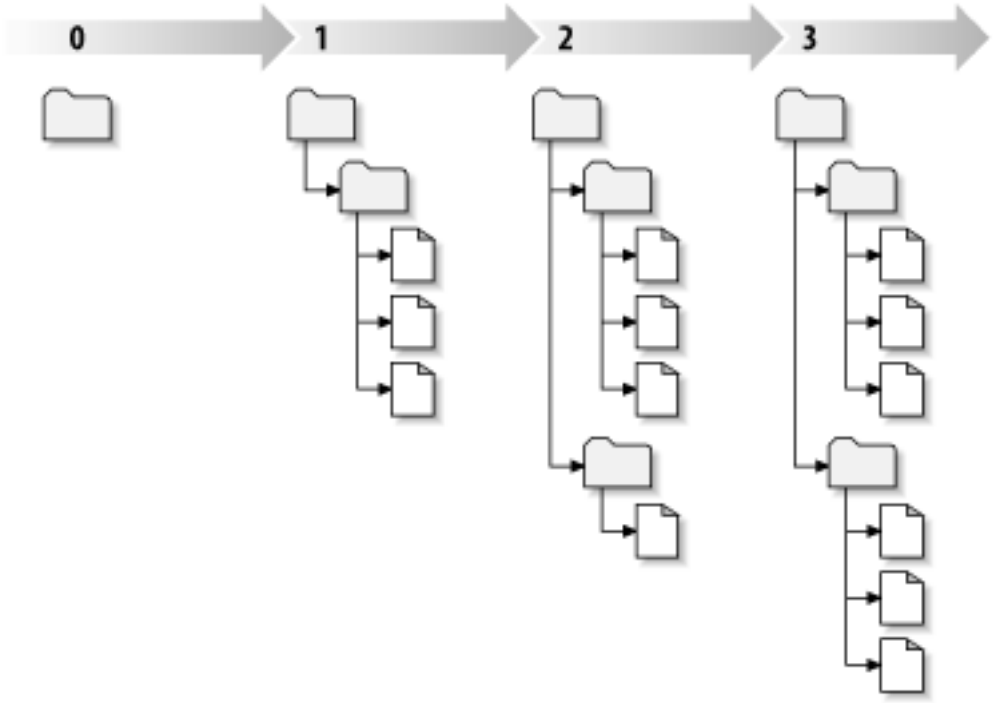


Figura 2.7. O Repositório

> **Números Globais de Revisão**
>
> Ao contrário de muitos outros sistemas de controle de versão, o número da revisão do Subversion é aplicado para a *estrutura inteira*, e não para os arquivos individuais. Cada número de revisão seleciona em uma estrutura inteira, uma estado em particular do repositório depois de algumas alteração submetida. Outra forma de enxergar isto é que a revisão N representa o estado da estrutura do repositório depois da enésima submissão. Quando o usuário do Subversion fala sobre a ``revisão 5 do `foo.c`", eles realmente querem dizer ``foo.c` como está na revisão 5." Repare que em geral, revisões N e M de um arquivo *não* necessariamente são diferentes.

É importante notar que cópias de trabalho nem sempre correspondem a uma única revisão qualquer no repositório; eles podem conter arquivos de diferentes revisões. Por exemplo, suponha que você obteve uma cópia de trabalho de um repositório do qual a revisão mais recente é a 4:

```
calc/Makefile:4
```

```
integer.c:4
button.c:4
```

Neste momento, esta cópia de trabalho corresponde exatamente à revisão 4 no repositório. Entretanto, suponha que você fez modificações em `button.c`, e submeteu essas alterações. Assumindo que não houve nenhuma outra submissão, o envio vai criar a revisão 5 no repositório, e sua cópia de trabalho vai se parecer com isto:

```
calc/Makefile:4
     integer.c:4
     button.c:5
```

Suponha que, neste momento, Sally envia uma alteração do `integer.c`, criando a revisão 6. Se você usar o **svn atualizar** para atualizar sua cópia de trabalho, então você terá algo como isto:

```
calc/Makefile:6
     integer.c:6
     button.c:6
```

As modificações de Sally para `integer.c` vão aparecer em sua cópia de trabalho, e sua alteração vai continuar presente em `button.c`. Neste exemplo, o texto de `Makefile` é identifico nas revisões 4, 5 e 6, mas Subversion vai marcar sua cópia de trabalho do `Makefile` com a revisão 6 indicando que ainda é a versão atual. Então, depois que você completou todas as atualizações na sua cópia de trabalho, isto geralmente corresponderá a uma exata revisão do repositório.

2.3.4. Como Cópia de Trabalho Acompanham o Repositório

Para cada arquivo do diretório de trabalho, Subversion grava duas partes essenciais da informação na área administrativa `.svn/`:

- qual revisão o arquivo da sua cópia de trabalho esta baseado (isto é chamado de *revisão de trabalho* do arquivo), e
- a informação de data/hora de quando foi a última atualização da cópia do repositório.

Com estas informação, através de troca de informações com o repositório, Subversion pode identificar qual dos quatro estados seguintes é o estado de um arquivo:

Não modificado, e atualizado
: O arquivo não foi modificado na cópia de trabalho, e nenhuma nova modificação foi submetida no repostório considerando a revisão de trabalho. Um **submetero> do arquivo natualizar do arquivo tamb**

Localmente alterado, e atualizado
: O arquivo possui modificações no diretório de trabalho, e nenhuma modificação para o arquivo foi enviada para o repositório considerando a revisão atual. Existem alterações que não foram submetidas para o repositório, deste modo um **submeter** do arquivo vai obter sucesso ao publicar suas alterações, e um **atualizar** do arquivo não fará nada.

Não modificado, e desatualizado
: O arquivo não possui alterações no diretório de trabalho, mas há modificações no repositório. O arquivo deverá eventualmente ser atualizado, para sincronizar com a revisão publica. Um **submeter** do arquivo não fará nada, e um **atualizar** do arquivo trará as últimas alterações para a cópia local.

Localmente modificado, e desatualizado
: O arquivo possui modificações tanto no diretório de trabalho, como no repositório. Um **submeter** do arquivo vai falhar com o erro *desatualizado*. O arquivo deverá ser atualizado primeiro; o comando **atualizar** vai tentar unificar as alterações do repositório com as alterações locais. Se o Subversion

não conseguir completar a unificação de forma plausível automaticamente, ele deixará para o usuário resolver o conflito.

2.4. Resumo

Nós apresentamos vários conceitos fundamentais do Subversion neste capítulo:

- Nós introduzimos as noções de um repositório central, a cópia de trabalho do usuário, e uma porção da estrutura de revisões do repositório.
- Nós mostramos alguns exemplos simples de como dois colaboradores podem usar o Subversion para publicar e receber alterações um do outro, usando o modelo 'copiar-modificar-unificar'.
- Nós falamos um pouco sobre a forma como o Subversion acompanha e controla as informações em uma cópia de trabalho.

Capítulo 3. O Repositório

Não importa qual protocolo você use para acessar seus repositórios, você sempre precisará criar pelo menos um repositório. Isto pode ser feito igualmente com a linha de comando do Subversion ou com o TortoiseSVN.

Se você não criou ainda um repositório do Subversion, a hora é agora.

3.1. Criação do Repositório

Você pode criar um repositório com o formato FSFS ou com o velho formato Berkeley Database (BDB). O formato FSFS é geralmente mais rápido e mais fácil de administrar, e funciona em redes compartilhadas e Windows 98 sem problemas. O formato BDB é foi considerado mais estável simplesmente porque tem sido usado por mais tempo, mas desde que FSFS tem sido usado agora por vários anos, o argumento é agora bastante fraco. Leia *Chosing a Data Store* [http://svnbook.red-bean.com/en/1.5/svn.reposadmin.planning.html@svn.reposadmin.basics.backends] no livro do Subversion para mais detalhes.

3.1.1. Criando um Repositório com a linha de comando do cliente

1. Cria um diretório vazio com o nome SVN (ex: `D:\SVN\`), o qual é usado como diretório principal para todos os seus repositórios.
2. Cria outro diretório `MyNewRepository` dentro de `D:\SVN\`
3. Abra a janela de comandos (ou janela DOS), vá para `D:\SVN\` e digite

```
svnadmin create --fs-type bdb MyNewRepository
```

ou

```
svnadmin create --fs-type fsfs MyNewRepository
```

Agora você tem uma novo repositório localizado em `D:\SVN\MyNewRepository`.

3.1.2. Criando um Repositório com o TortoiseSVN

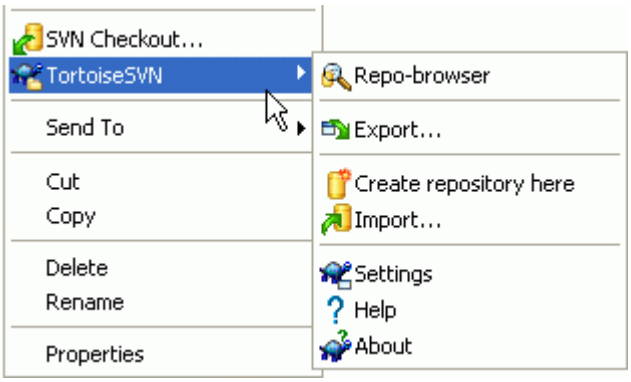


Figura 3.1. O menu do TortoiseSVN para diretórios não controlados

1. Abra o windows explorer
2. Crie um novo diretório e chame de, por exemplo `SVNRepository`
3. Clique-botão-direito no diretório recentemente criado e selecione TortoiseSVNCriar Repositório aqui....

   Um repositório é então criado dentro do novo diretório. *Não edite esses arquivos você mesmo!!!*. Se apresentar algum erro tenha certeza que o diretório está vazio e sem proteção de escrita.

Dica

TortoiseSVN não oferece a opção de criar um repositório BDB, apesar de que você pode continuar usando a linha de comando para criar um. Repositórios FSFS são geralmente mais fáceis para manter, e também é mais fácil para nós mantermos o TortoiseSVN devido aos problemas de compatibilidade entre as diferentes versões do BDB.

Future versions of TortoiseSVN will not support `file://` access to BDB repositories due to these compatibility issues, although it will of course always support this repository format when accessed via a server through the `svn://`, `http://` or `https://` protocols. For this reason, we strongly recommend that any new repository which must be accessed using `file://` protocol is created as FSFS.

Claro, nós também recomendamos que você não use `file://` por qualquer razão que não sejam apenas testes locais. Usando um servidor é mais seguro e mais confiável e também para um único desenvolvedor.

3.1.3. Acesso Local para o Repositório

Para acessar seu repositório local você precisa do caminho para a pasta. Apenas lembre-se que Subversion esperar todos os caminhos dos repositórios no formato `file:///C:/SVNRepository/`. Note o uso de barras em todo o caminho.

Para acessar o repositório localizado em uma rede compartilhada você pode também usar uma unidade mapeada, ou você pode usar um caminho UNC. Para caminhos UNC, o formato é `file://NomeServidor/caminho/para/repos/`. No que há apenas 2 barras aqui.

Antes do SVN 1.2, caminhos UNC tinham que ser informados num formato mais complicado `file:///\NomeServidor/caminho/para/repos`. Este formato ainda é suportado, mas não é recomendado.

Atenção

Não crie ou acesse um repositório Berkeley DB em uma rede compartilhada. Ele *não pode* ser usado em uma sistema de arquivos remoto. Nem mesmo se você tem uma unidade mapeada da rede em uma unidade de disco. Se você tentar usar Berkeley BD em uma rede compartilhada, o resultado é imprevisível - você poderá ver erros misteriosos de imediato, ou poderá levar meses para descobrir que os dados do repositório estão sutilmente corrompidos.

3.1.4. Acessando um Repositório em uma Rede Compartilhada

Apesar de que em teoria é possível guardar um repositório FSFS em uma rede compartilhada e acessar concorrentemente o repositório usando o protocolo `file://`, é definitivamente *não* é recomendado. De fato nós *fortemente* desencorajamos isto, e não suportamos tal uso.

Primeiramente você está dando para todos os usuários dreito de escrita no repositório, então qualquer usuário pode acidentalmente apagar o repositório inteiro ou corrompe-lo de alguma maneira.

Em segundo lugar nem todo protocolo de compartilhamento de arquivo suporta as travas necessárias para o Subversion, então você pode encontrar seu repositório corrompido. Isto pode não acontecer logo, mas um dia dois usuários vão tentar acessar o repositório ao mesmo tempo.

Em terceiro lugar as permissões do arquivo precisam ser bem definidas. Você pode achar fácil num compartilhamento Windows, mas SAMBA é relativamente difícil.

O acesso através de `file://` é destinado para local, um único usuário, e particularmente para testes e depurações. Quando você quer compartilhar o repositório você *realmente* deve configurar um servidor, e isto não é tão difícil como você deve estar pensando. Leia Seção 3.5, “Acessando o Repositório” para orientações sobre escolhas e configurações de um servidor.

3.1.5. Leiaute do Repositório

Antes de você importar seus dados para o repositório você deve primeiro pensar sobre como você quer organizar os dados. Se você usar um dos formatos recomendados depois você o terá muito mais organizado.

Existem alguns padrões e recomendadas maneiras de organizar um repositório. Muitas pessoas criam um diretório `trunk` para guardar a “linha principal” de desenvolvimento, um diretório `branches` para guardar as ramificações, e um diretório `tags` para guardar as versões concluidas. Se o repositório guarda somente um único projeto, então frequentemente as pessoas criam estes diretórios no nível mais alto:

```
/trunk
/branches
/tags
```

Se o repositório contém vários projetos, as pessoas normalmente organizam sua estrutura por ramificação:

```
/trunk/paint
/trunk/calc
/branches/paint
/branches/calc
/tags/paint
/tags/calc
```

... ou por projeto:

```
/paint/trunk
/paint/branches
/paint/tags
/calc/trunk
/calc/branches
/calc/tags
```

Organizar por projeto faz sentido se os projetos não são fortemente relacionados e cada um deles é controlado individualmente. Para projetos relacionados onde você pode querer controlar todos os projeto de uma vez, ou onde os projetos são todos amarrados uns aos outros em um único pacote de distribuição, é então mais comum organizar por ramificação. Desta maneira você tem somente um trunk para controlar, e os relacionamentos entre os subprojetos são vislumbrados mais facilmente.

Se você adotar como nível mais alto algo como`/trunk /tags /branches`, não há nada para dizer que você tenha que copiar o trunk inteiro para cada ramificação e liberação, e de alguma forma esta estrutura oferece mais flexibilidade.

Para projetos não relacionados você deve dar preferência para repositórios separados. Quando você submeter alterações, o novo número de revisão será do repositório inteiro, não o apenas a revisão do projeto. Tendo 2 projetos não relacionados compartilhando um mesmo repositório pode significar grandes lacunas no número da revisão. Os projetos Subversion e o TortoiseSVN aparecem no mesmo endereço principal, mas são repositórios completamente separados permitindo o desenvolvimento independente, e sem confusões no número da liberação.

Claro, você é livre para ignorar estes leiautes padrões. Você pode criar qualquer tipo de variação, seja o que funcionar melhor para você e sua equipe. Lembre-se que qualquer que seja a sua escolha, isto não é uma decisão permanente. Você pode reorganizar seu repositório a qualquer momento. Pelos diretórios branches e tags serem diretórios comuns, TortoiseSVN pode mover ou renomeá-los do jeito que você quiser.

Mudar de um leiaute para outro é apenas questão de mover uns diretórios no servidor; se você não gosta da maneira que as coisas estão organizadas no repositório, basta manipular os diretórios no repositório.

Então se você ainda não tem criada uma estrutura base de diretórios dentro do seu repositório você deverá fazer agora. Existem duas maneiras de fazer isso. Se você simplesmente quer criar uma estrutura `/trunk /tags /branches`, você pode usar o navegador de repositórios para criar as três pastas (em três separadas submissões). Se você quer criar um hierarquia mais estruturada então crie a estrutura primeiro no disco local e importe-a em uma única submissão, como isto:

1. criar uma nova pasta vazia em seu disco rígido

2. criar seu diretório da estrutura principal dentro da pasta - não coloque nenhum arquivo dentro ainda!

3. import this structure into the repository via a right click on the folder and selecting TortoiseSVN → Import... This will import your temp folder into the repository root to create the basic repository layout.

Note que o nome do diretório que você está importando não aparece no repositório, somente o seu conteúdo. Por exemplo, crie a seguinte estrutura de diretórios:

```
C:\Temp\Novo\trunk
C:\Temp\Novo\branches
C:\Temp\Novo\tags
```

Importar `C:\Temp\Novo` dentro do diretório principal do repositório, o qual então vai parecer com isto:

```
/trunk
/branches
/tags
```

3.2. Cópia de Segurança do Repositório

Não importante o tipo de repositório que você usa, é vitalmente importante que você mantenha cópias de segurança regulares, e que você verifique a cópia de segurança. Se o servidor falhar, você terá condições de acessar uma versão recente de seus arquivos, mas sem o repositório todo o histórico estará perdido para sempre.

Uma simplória (mas não recomendada) maneira de fazer é copiar o diretório do repositório para a midia da cópia de segurança. Contudo, você precisa ter absoluta certeza que nenhum processo está acessando os dados. Neste contexto, acessando significa *qualquer* acesso de qualquer maneira. Um repositório BDB é escrito até mesmo quando a operação parece requerer apenas leitura, como verificar a situação. Se seu repositório estiver sendo acessado durante a cópia, (navegador web aberto, WebSVN, etc.) a cópia de segurança será inútil.

A maneira recomendada é executar

```
svnadmin hotcopy caminho/para/repositório caminho/para/cópiaseguranca --clean-log
```

para criar uma cópia de seu repositório em modo seguro. E então guarde esta cópia. A opção `--clean-logs` não é obrigatório, mas remove qualquer registro de arquivo duplicado quando você copia um repositório BDB, o que pode economizar algum espaço.

A ferramenta `svnadmin` é instalada automaticamente quando você instala o cliente de linha de comando do Subversion. Se você está instalando as ferramentas de linha de commando em uma computador Windows, a melhor maneira de fazê-lo é usar a versão Windows do instalador. O instalar é mais compacto que a versão `.zip`, então o arquivo fica menor, e ele se encarrega de configurar os caminhos para você. Você pode baixar a versão mais atual do cliente de linha de comando do Subversion de *http://subversion.apache.org/getting.html*.

3.3. Rotinas de eventos no servidor

Uma rotina para evento é um programa iniciado por algum evento do repositório, como a criação de uma nova revisão ou a modificaç~!ao de um propriedade não controlada. Cada evento é entregue com informação sufiente para dizer qual é o evento, qual é(são) o(s) destino(s) desta operação, e qual é o nome de usuário da pessoa que disparou o evento. Dependendo da saída da rotina ou da situação de retorno, a rotina para evento pode continuar a ação, parar, ou então suspender de alguma forma. Por favor veja o capítulo *Rotina para Eventos* [http://svnbook.red-bean.com/en/1.5/svn.reposadmin.create.html#svn.reposadmin.create.hooks] no livro do Subversion para detalhamento completo sobre os eventos que estão disponíveis.

Estas rotinas para eventos são executadas pelo servidor que hospeda o repositório. TortoiseSVN também permite que você configura no cliente rotinas para eventos que são executando localmente em consequência de certos acontecimentos. Veja Seção 4.30.8, “Client Side Hook Scripts” para mais informações.

Exemplos de rotinas para eventos podem ser encontradas no diretório `hooks` do repositório. Estas rotinas de exemplos são adaptadas para servidores Unix/Linux e precisam ser modificadas se seu servidor utiliza Windows. A rotina pode ser um arquivo em lote ou um executável. O exemplo abaixo mostra um arquivo em lote que pode ser usado para implementar um evento pre-revpro-change.

```
rem Somente permitir mensagens de alteração.
if "%4" == "svn:log" exit 0
echo Property '%4' não pode mudar >&2
exit 1
```

Note que qualquer saída em vídeo (stdout) será ignorada. Se você quer que uma mensagem seja mostrar na tela de Rejeição da Submissão você deve enviar através da saída de erro (stderr). No arquivo em lote isso pode ser feito usando `>&2`

3.4. Vínculos externos

Se você quer disponibilizar seu repositório Subversion para outras pessoas você pode incluir um vínculo no seu website. Uma forma de tornar isso mais fácil é incluir um *vínculo externo* para outros usuários do TortoiseSVN.

Quando você instala o TortoiseSVN, ele registra um novo protocolo `tsvn:`. Quando um usuário do TortoiseSVN clica num vínculo como este, a janela de obtenção será aberta automaticamente com a URL do repositório já preenchida.

Para incluir tal vínculo em sua própria página HTML, você precisa adicionar um código parecido com este:

```
<a href="tsvn:http://project.domain.org/svn/trunk">
</a>
```

Claro, isto parecerá melhor se você incluir a imagem correta. Você pode usar o *logo do TortoiseSVN* [http://tortoisesvn.tigris.org/images/TortoiseCheckout.png] ou você pode providenciar sua própria imagem.

```
<a href="tsvn:http://project.domain.org/svn/trunk">
<img src=TortoiseCheckout.png></a>
```

Você pode também fazer o vínculo apontar para uma revisão específica, por exemplo

```
<a href="tsvn:http://project.domain.org/svn/trunk?100">
</a>
```

3.5. Acessando o Repositório

Para usar o TortoiseSVN (ou qualquer outro cliente do Subversion), você precisa de um lugar para guardar seus repositórios. Você pode também guardar seus repositórios e acessá-los usando o protocolo `file://` ou você pode colocá-los em um servidor e acessá-los usando os protocolos `http://` ou `svn://`. Os dois protocolos podem também ser criptografados. Você pode usar `https://` ou `svn+ssh://`, ou você pode usar `svn://` com SASL.

Se você está usando um servidor público como *Google Code* [http://code.google.com/hosting/] ou um servidor disponibilizado por outra pessoa então não há nada que você possa fazer. Veja Capítulo 4, *Guia do Uso Diário*.

Se você não tem um servidor e vai trabalhar sozinho, ou se você está apenas avaliando o Subversion e o TortoiseSVN isoladamente, então repositórios locais são provavelmente a melhor opção. Apenas crie um repositório em seu próprio computador como descrito anteriormente em Capítulo 3, *O Repositório*. Você pode pular o resto deste capítulo e ir diretamente para Capítulo 4, *Guia do Uso Diário* para saber como começar a usá-los.

Se você está pensando em configurar um repositório para vários usuários em uma pasta compartilhada, pense novamente. Leia Seção 3.1.4, “Acessando um Repositório em uma Rede Compartilhada” para descobrir porque isto é uma má idéia. Configurando um servidor não é tão difícil quanto parece, e isto vai ter mais segurança e provavelmente mais velocidade.

As próximas seções são um guia passo-a-passo de como você pode configurar um servidor em uma máquina Windows. Claro que você pode também configurar um servidor em uma máquina Linux, mas isto está fora do escopo deste guia. Informações mais detalhadas das opções de um servidor Subversion, e como escolher a melhor arquitetura para cada situação, podem ser encontradas no manual do Subversion na seção *Configurações do Servidor* [http://svnbook.red-bean.com/en/1.5/svn.serverconfig.html].

3.6. Svnserve Baseado em Servidor

3.6.1. Introdução

Subversion inclui o Svnserve - um servidor autônomo leve que utiliza um procotolo customizado usado uma conexão TPC/IP comum. Isto é ideal para pequenas instalações, ou quando uma instalação completa do servidor Apache não puder ser usado.

Na maioria dos casos svnserve é fácil de configurar e executa rapidamente em um servidor baseado no Apache, apesar de não ter algumas características avançadas. E agora que o suporte a SASL está incluído é também fácil garantir a segurança.

3.6.2. Instalando o svnserve

1. Pegue a última versão do Subversion de *http://subversion.apache.org/getting.html*. Uma alternativa é usar um pacote de instalação do CollabNet em *http://www.collab.net/downloads/subversion*. Este instalador vai configurar o svnserve como um serviço do Windows, e também vai incluir algumas ferramentas necessárias caso esteja usando SASL para segurança.

2. Se você já tem uma versão instalada do Subversion, e svnserve está sendo executado, você precisa primeiro parar o serviço antes de continuar.

3. Execute o instalador do Subversion. Se você executar o instalador em um servidor (recomendado) você pode pular o passo 4.

4. Abra o windows explorer, vá no diretório de instalação do Subversion (normalmente `C:\Arquivos de Programas\Subversion`) e no diretório `bin`, encontre os arquivos `svnserve.exe`, `intl3_svn.dll`, `libapr.dll`, `libapriconv.dll`, `libapriutil.dll`, `libdb*.dll`, `libeay32.dll` e `ssleay32.dll` - copie esses arquivos, ou simplesmente copie todo o diretório `bin`, para dentro do direitório do seu servidor ex. `c:\svnserve`

3.6.3. Executando o svnserve

Agora que o svnserve está instalado, você precisa executá-lo no seu servidor. A forma mais simples é executar através da linha de comando do DOS ou criar um atalho no windows:

```
svnserve.exe --daemon
```

svnserve vai então iniciar esperando por requisições na porta 3690. A opção --daemon diz ao svnserve para executar como um serviço, logo ele vai sempre estar disponível até que seja manualmente terminado.

Se você ainda não criou um repositório, siga as instruções dadas para configurar o servidor Apache em Seção 3.7.4, “Configurações”.

Para verificar se o svnserve está funcionando, use TortoiseSVN → Navegador de Repositório para visualizar um repositório.

Assumindo que seu repositório está localizado em `c:\repos\TestRepo`, e seu servidor é chamado de `localhost`, use:

```
svn://localhost/repos/TestRepo
```

quando pedido no navegador de repositório.

Você pode também aumentar a segurança e poupar tempo digitando URLSs com o svnserve usando a opção --root para definir a pata principal e restringir o acesso para um diretório específico no servidor:

```
svnserve.exe --daemon --root drive:\path\to\repository\root
```

Usando o teste anterior como guia, svnserve poderá agora ser executado assim:

```
svnserve.exe --daemon --root c:\repos
```

E no navegador de repositório do TortoiseSVN a URL é reduzida para:

```
svn://localhost/TestRepo
```

Note que a opção --root é também necessária se seu repositório está localizado em uma partição ou unidade diferente de onde o svnserve está no seu servidor.

Svnserve vai atender qualquer número de repositórios. Apenas deixe eles dentro da pasta principal, para acessá-los a partir desta pasta.

Atenção

Não crie or acesse um repositório Berkeley DB em uma rede compartilhada. Isto *não pode* existir em um sistema de arquivos remoto. Nem mesmo se você tem dispositivos de rede mapeados em uma unidade de disco. Se você tentar usar Berkeley DB em uma rede compartilhada, os resultados são imprevisíveis - você poderá ver erros misteriosos imediatamente, ou meses antes descobrir que seu repositório está sutilmente corrompido.

3.6.3.1. Executar svnserve como um Serviço

Executando o svnserve como um usuário normalmente não é a melhor forma. Isto significa que sempre será necessário logar com um usuário no servidor, e lembrar de executá-lo depois de reiniciar o servidor. Uma opção melhor para executar o svnserve é como um serviço no windows. A partir da versão 1. do Subversion, svnserve pode ser instalado como um serviço nativo no windows.

To install svnserve as a native windows service, execute the following command all on one line to create a service which is automatically started when windows starts.

```
sc create svnserve binpath= "c:\svnserve\svnserve.exe --service
    --root c:\repos" displayname= "Subversion" depend= tcpip
    start= auto
```

If any of the paths include spaces, you have to use (escaped) quotes around the path, like this:

```
sc create svnserve binpath= "
    \"C:\Program Files\Subversion\bin\svnserve.exe\"
    --service --root c:\repos" displayname= "Subversion"
    depend= tcpip start= auto
```

You can also add a description after creating the service. This will show up in the Windows Services Manager.

```
sc description svnserve "Subversion server (svnserve)"
```

Observe um pouco o formato do não usual comando usado pelo `sc`. Nos pares `chave= valor` não deve haver espaço entre a chave e o = mas deve ter um espaço antes da chave.

Dica

Microsoft recomenda agora que serviços sejam executados usando a conta de Serviço Local ou do Serviço de Rede. Veja *Os Serviços e as Contas de Serviços do Guia de Planejamento de Segurança* [http://www.microsoft.com/technet/security/topics/serversecurity/serviceaccount/default.mspx]. Para criar um serviço com uma conta de Serviço Local, concatene ao comando anterior o exemplo que segue.

```
obj= "NT AUTHORITY\LocalService"
```

Note que você você terá de configurar os direitos de acesso apropriados para a conta de Serviço Local para o Subversion e seus repositórios, assim como qualquer aplicação que seja usada pelas rotinas para eventos. O grupo de direitos para isto é chamado de "LOCAL SERVICE".

Uma vez que você instalou o serviço, você precisa ir até o gerenciador de serviços para iniciá-lo (apenas desta vez; ele será iniciado automaticamente quando o servidor reiniciar).

Para informações mais detalhadas, acesse *Suporte para o Serviço do Svnserve no Windows* [http://svn.collab.net/repos/svn/trunk/notes/windows-service.txt].

Se você instalou uma versão mais atual do svnserve usando a interface `SVNService`, e agora você quer usar ao suporte nativo, você vai precisar desregistrar a interface como um serviço (lembre-se de parar o serviço primeiro!). Simplesmente use o comando

```
svnservice -remove
```

para remover a referência do serviço.

3.6.4. Autenticação Básica com svnserve

O padrão de configuração do svnserve provê acesso de leitura para usuário anônimos. Isto significa que você pode usar uma URL `svn://` para obter e atualizar, ou você pode usar o Navegador de Repositórios do TortoiseSVN para visualizar o repositório, mas você não poderá submeter nenhuma alteração.

Para habilitar o acesso de escrito no repositório, você precisa editar o arquivo `conf/svnserve.conf` no diretório do seu repositório. Este arquivo controla a configuração do serviço do svnserve, e também contém uma documentação muito útil.

VocÊ pode habilitar acesso de escrita para usuário anônimos apenas configurando:

```
[general]
anon-access = write
```

Entretanto você não vai saber quem fez as alterações no repositório, já que a propriedade `svn:author` estará vazia. Você também não poderá controlar quem pode fazer alterações no repositório. Isto é um pouco de risco de configuração.

One way to overcome this is to create a password database:

```
[general]
anon-access = none
auth-access = write
password-db = userfile
```

Where `userfile` is a file which exists in the same directory as `svnserve.conf`. This file can live elsewhere in your file system (useful for when you have multiple repositories which require the same access rights) and may be referenced using an absolute path, or a path relative to the `conf` directory. If you include a path, it must be written `/the/unix/way`. Using \ or drive letters will not work. The `userfile` should have a structure of:

```
[users]
username = password
...
```

This example would deny all access for unauthenticated (anonymous) users, and give read-write access to users listed in `userfile`.

Dica

If you maintain multiple repositories using the same password database, the use of an authentication realm will make life easier for users, as TortoiseSVN can cache your credentials so that you only have to enter them once. More information can be found in the Subversion book, specifically in the sections *Create a 'users' file and realm* [http://svnbook.red-bean.com/en/1.5/svn.serverconfig.svnserve.html#svn.serverconfig.svnserve.auth.users] and *Client Credentials Caching* [http://svnbook.red-bean.com/en/1.5/svn.serverconfig.netmodel.html#svn.serverconfig.netmodel.credcache]

3.6.5. Melhor Segurança com SASL

3.6.5.1. O que é SASL?

The Cyrus Simple Authentication and Security Layer is open source software written by Carnegie Mellon University. It adds generic authentication and encryption capabilities to any network protocol, and as of Subversion 1.5 and later, both the svnserve server and TortoiseSVN client know how to make use of this library.

For a more complete discussion of the options available, you should look at the Subversion book in the section *Using svnserve with SASL* [http://svnbook.red-bean.com/en/1.5/svn.serverconfig.svnserve.html#svn.serverconfig.svnserve.sasl]. If you are just looking for a simple way to set up secure authentication and encryption on a Windows server, so that your repository can be accessed safely over the big bad Internet, read on.

3.6.5.2. Autenticação SASL

To activate specific SASL mechanisms on the server, you'll need to do three things. First, create a `[sasl]` section in your repository's `svnserve.conf` file, with this key-value pair:

```
use-sasl = true
```

Second, create a file called `svn.conf` in a convenient location - typically in the directory where subversion is installed.

Thirdly, create two new registry entries to tell SASL where to find things. Create a registry key named `[HKEY_LOCAL_MACHINE\SOFTWARE\Carnegie Mellon\Project Cyrus\SASL Library]` and place two new string values inside it: `SearchPath` set to the directory path containing the `sasl*.dll` plug-ins (normally in the Subversion install directory), and `ConfFile` set to the directory containing the `svn.conf` file. If you used the CollabNet installer, these registry keys will already have been created for you.

Edit the `svn.conf` file to contain the following:

```
pwcheck_method: auxprop
auxprop_plugin: sasldb
mech_list: DIGEST-MD5
```

```
sasldb_path: C:\TortoiseSVN\sasldb
```

The last line shows the location of the authentication database, which is a file called `sasldb`. This could go anywhere, but a convenient choice is the repository parent path. Make sure that the svnserve service has read access to this file.

If svnserve was already running, you will need to restart it to ensure it reads the updated configuration.

Now that everything is set up, all you need to do is create some users and passwords. To do this you need the `saslpasswd2` program. If you used the CollabNet installer, that program will be in the install directory. Use a command something like this:

```
saslpasswd2 -c -f C:\TortoiseSVN\sasldb -u realm username
```

The `-f` switch gives the database location, `realm` must be the same as the value you defined in your repository's `svnserve.conf` file, and username is exactly what you expect it to be. Note that the realm is not allowed to contain space characters.

You can list the usernames stored in the database using the `sasldblistusers2` program.

3.6.5.3. Criptografia SASL

To enable or disable different levels of encryption, you can set two values in your repository's `svnserve.conf` file:

```
[sasl]
use-sasl = true
min-encryption = 128
max-encryption = 256
```

The `min-encryption` and `max-encryption` variables control the level of encryption demanded by the server. To disable encryption completely, set both values to 0. To enable simple checksumming of data (i.e., prevent tampering and guarantee data integrity without encryption), set both values to 1. If you wish to allow (but not require) encryption, set the minimum value to 0, and the maximum value to some bit-length. To require encryption unconditionally, set both values to numbers greater than 1. In our previous example, we require clients to do at least 128-bit encryption, but no more than 256-bit encryption.

3.6.6. Autenticação com svn+ssh

Another way to authenticate users with a svnserve based server is to use a secure shell (SSH) to tunnel requests through. It is not as simple to set up as SASL, but it may be useful is some cases.

With this approach, svnserve is not run as a daemon process, rather, the secure shell starts svnserve for you, running it as the SSH authenticated user. To enable this, you need a secure shell daemon on your server.

A basic method for setting up your server is given in Apêndice G, *Securing Svnserve using SSH*. You can find other SSH topics within the FAQ by searching for “SSH”.

Further information about svnserve can be found in the *Version Control with Subversion* [http://svnbook.red-bean.com].

3.6.7. Path-based Authorization with svnserve

Starting with Subversion 1.3, svnserve supports the same `mod_authz_svn` path-based authorization scheme that is available with the Apache server. You need to edit the `conf/svnserve.conf` file in your repository directory and add a line referring to your authorization file.

```
[general]
authz-db = authz
```

Here, `authz` is a file you create to define the access permissions. You can use a separate file for each repository, or you can use the same file for several repositories. Read Seção 3.7.6, “Autorização Baseado nos Caminhos” for a description of the file format.

3.7. Servidor Baseado no Apache

3.7.1. Introdução

The most flexible of all possible server setups for Subversion is the Apache based one. Although a bit more complicated to set up, it offers benefits that other servers cannot:

WebDAV
: The Apache based Subversion server uses the WebDAV protocol which is supported by many other programs as well. You could e.g. mount such a repository as a “Web folder” in the Windows explorer and then access it like any other folder in the file system.

Navegando No Repositório
: You can point your browser to the URL of your repository and browse the contents of it without having a Subversion client installed. This gives access to your data to a much wider circle of users.

Autenticação
: You can use any authentication mechanism Apache supports, including SSPI and LDAP.

Segurança
: Since Apache is very stable and secure, you automatically get the same security for your repository. This includes SSL encryption.

3.7.2. Instalando o Apache

The first thing you need before installing Apache is a computer with Windows 2000, Windows XP+SP1, Windows 2003, Vista or Server 2008.

Atenção

Please note that Windows XP without the service pack 1 will lead to bogus network data and could therefore corrupt your repository!

1. Download the latest version of the Apache web server from *http://httpd.apache.org/download.cgi*. Make sure that you download the version 2.2.x - the version 1.3.xx won't work!

   The msi installer for Apache can be found by clicking on `other files`, then browse to `binaries/win32`. You may want to choose the msi file `apache-2.2.x-win32-x86-openssl-0.9.x.msi` (the one that includes OpenSSL).

2. Once you have the Apache2 installer you can double click on it and it will guide you through the installation process. Make sure that you enter the server-URL correctly (if you don't have a DNS name for your server just enter the IP-address). I recommend to install Apache *for All Users, on Port 80, as a Service*. Note: if you already have IIS or any other program running which listens on port 80 the installation might fail. If that happens, go to the programs directory, `\Apache Group\Apache2\conf` and locate the file `httpd.conf`. Edit that file so that `Listen 80` is changed to a free port, e.g. `Listen 81`. Then restart the installation - this time it should finish without problems.

3. Now test if the Apache web server is running correctly by pointing your web browser to `http://localhost/` - a preconfigured Website should show up.

Cuidado

If you decide to install Apache as a service, be warned that by default it will run as the local system account. It would be a more secure practice for you to create a separate account for Apache to run as.

Make sure that the account on the server that Apache is running as has an explicit entry in the repository directory's access control list (right-click directory | properties | security), with full control. Otherwise, users will not be able to commit their changes.

Even if Apache runs as local system, you still need such an entry (which will be the SYSTEM account in this case).

If Apache does not have this permission set up, your users will get “Access denied” error messages, which show up in the Apache error log as error 500.

3.7.3. Instalando o Subversion

1. Download the latest version of the Subversion Win32 binaries for Apache. Be sure to get the right version to integrate with your version of Apache, otherwise you will get an obscure error message when you try to restart. If you have Apache 2.2.x go to *http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=8100*.

2. Run the Subversion installer and follow the instructions. If the Subversion installer recognized that you've installed Apache, then you're almost done. If it couldn't find an Apache server then you have to do some additional steps.

3. Using the windows explorer, go to the installation directory of Subversion (usually `c:\program files\Subversion`) and find the files `/httpd/mod_dav_svn.so` and `mod_authz_svn.so`. Copy these files to the Apache modules directory (usually `c:\program files\apache group\apache2\modules`).

4. Copy the file `/bin/libdb*.dll` and `/bin/intl3_svn.dll` from the Subversion installation directory to the Apache bin directory.

5. Edit Apache's configuration file (usually `C:\Program Files\Apache Group\Apache2\conf\httpd.conf`) with a text editor such as Notepad and make the following changes:

   Uncomment (remove the '#' mark) the following lines:

   ```
   #LoadModule dav_fs_module modules/mod_dav_fs.so
   #LoadModule dav_module modules/mod_dav.so
   ```

   Add the following two lines to the end of the `LoadModule` section.

   ```
   LoadModule dav_svn_module modules/mod_dav_svn.so
   LoadModule authz_svn_module modules/mod_authz_svn.so
   ```

3.7.4. Configurações

Now you have set up Apache and Subversion, but Apache doesn't know how to handle Subversion clients like TortoiseSVN yet. To get Apache to know which URL will be used for Subversion repositories you

have to edit the Apache configuration file (usually located in `c:\program files\apache group\apache2\conf\httpd.conf`) with any text editor you like (e.g. Notepad):

1. At the end of the config file add the following lines:

```
<Location /svn>
  DAV svn
  SVNListParentPath on
  SVNParentPath D:\SVN
  #SVNIndexXSLT "/svnindex.xsl"
  AuthType Basic
  AuthName "Subversion repositories"
  AuthUserFile passwd
  #AuthzSVNAccessFile svnaccessfile
  Require valid-user
</Location>
```

   This configures Apache so that all your Subversion repositories are physically located below `D:\SVN`. The repositories are served to the outside world from the URL: `http://MyServer/svn/` . Access is restricted to known users/passwords listed in the `passwd` file.

2. To create the `passwd` file, open the command prompt (DOS-Box) again, change to the `apache2` folder (usually `c:\program files\apache group\apache2`) and create the file by entering

```
bin\htpasswd -c passwd <username>
```

   This will create a file with the name `passwd` which is used for authentication. Additional users can be added with

```
bin\htpasswd passwd <username>
```

3. Reinicie o serviço do Apache novamente

4. Point your browser to `http://MyServer/svn/MyNewRepository` (where `MyNewRepository` is the name of the Subversion repository you created before). If all went well you should be prompted for a username and password, then you can see the contents of your repository.

A short explanation of what you just entered:

Definindo	Explicação
<Location /svn>	means that the Subversion repositories are available from the URL `http://MyServer/svn/`
DAV svn	tells Apache which module will be responsible to serve that URL - in this case the Subversion module.
SVNListParentPath on	For Subversion version 1.3 and higher, this directive enables listing all the available repositories under `SVNCaminhoAcima`.
SVNParentPath D:\SVN	tells Subversion to look for repositories below `D:\SVN`
SVNIndexXSLT "/svnindex.xsl"	Used to make the browsing with a web browser prettier.
AuthType Basic	is to activate basic authentication, i.e. Username/password
AuthName "Subversion repositories"	is used as an information whenever an authentication dialog pops up to tell the user what the authentication is for

Definindo	Explicação
AuthUserFile passwd	specifies which password file to use for authentication
AuthzSVNAccessFile	Location of the Access file for paths inside a Subversion repository
Require valid-user	specifies that only users who entered a correct username/password are allowed to access the URL

Tabela 3.1. Configuração do Apache para `httpd.conf`

But that's just an example. There are many, many more possibilities of what you can do with the Apache web server.

- If you want your repository to have read access for everyone but write access only for specific users you can change the line

  ```
  Require valid-user
  ```

  to

  ```
  <LimitExcept GET PROPFIND OPTIONS REPORT>
  Require valid-user
  </LimitExcept>
  ```

- Using a `passwd` file limits and grants access to all of your repositories as a unit. If you want more control over which users have access to each folder inside a repository you can uncomment the line

  ```
  #AuthzSVNAccessFile svnaccessfile
  ```

  and create a Subversion access file. Apache will make sure that only valid users are able to access your `/svn` location, and will then pass the username to Subversion's `AuthzSVNAccessFile` module so that it can enforce more granular access based upon rules listed in the Subversion access file. Note that paths are specified either as `repos:path` or simply `path`. If you don't specify a particular repository, that access rule will apply to all repositories under `SVNParentPath`. The format of the authorization-policy file used by `mod_authz_svn` is described in Seção 3.7.6, “Autorização Baseado nos Caminhos”

- To make browsing the repository with a web browser 'prettier', uncomment the line

  ```
  #SVNIndexXSLT "/svnindex.xsl"
  ```

  and put the files `svnindex.xsl`, `svnindex.css` and `menucheckout.ico` in your document root directory (usually `C:/Program Files/Apache Group/Apache2/htdocs`). The directory is set with the `DocumentRoot` directive in your Apache config file.

  You can get those three files directly from our source repository at *http://tortoisesvn.googlecode.com/svn/trunk/contrib/svnindex*. (Seção 3, “TortoiseSVN é grátis!” explains how to access the TortoiseSVN source repository).

  The XSL file from the TortoiseSVN repository has a nice gimmick: if you browse the repository with your web browser, then every folder in your repository has an icon on the right shown. If you click on that icon, the TortoiseSVN checkout dialog is started for this URL.

3.7.5. Vários Repositórios

If you used the SVNParentPath directive then you don't have to change the Apache config file every time you add a new Subversion repository. Simply create the new repository under the same location as the first repository and you're done! In my company I have direct access to that specific folder on the server via SMB (normal windows file access). So I just create a new folder there, run the TortoiseSVN command TortoiseSVN → Create repository here... and a new project has a home...

If you are using Subversion 1.3 or later, you can use the SVNListParentPath on directive to allow Apache to produce a listing of all available projects if you point your browser at the parent path rather than at a specific repository.

3.7.6. Autorização Baseado nos Caminhos

The mod_authz_svn module permits fine-grained control of access permissions based on user names and repository paths. This is available with the Apache server, and as of Subversion 1.3 it is available with svnserve as well.

An example file would look like this:

```
[groups]
admin = john, kate
devteam1 = john, rachel, sally
devteam2 = kate, peter, mark
docs = bob, jane, mike
training = zak
# Default access rule for ALL repositories
# Everyone can read, admins can write, Dan German is excluded.
[/]
* = r
@admin = rw
dangerman =
# Allow developers complete access to their project repos
[proj1:/]
@devteam1 = rw
[proj2:/]
@devteam2 = rw
[bigproj:/]
@devteam1 = rw
@devteam2 = rw
trevor = rw
# Give the doc people write access to all the docs folders
[/trunk/doc]
@docs = rw
# Give trainees write access in the training repository only
[TrainingRepos:/]
@training = rw
```

Note that checking every path can be an expensive operation, particularly in the case of the revision log. The server checks every changed path in each revision and checks it for readability, which can be time-consuming on revisions which affect large numbers of files.

Authentication and authorization are separate processes. If a user wants to gain access to a repository path, she has to meet *both*, the usual authentication requirements and the authorization requirements of the access file.

3.7.7. Authentication With a Windows Domain

As you might have noticed you need to make a username/password entry in the `passwd` file for each user separately. And if (for security reasons) you want your users to periodically change their passwords you have to make the change manually.

But there's a solution for that problem - at least if you're accessing the repository from inside a LAN with a windows domain controller: `mod_auth_sspi`!

The original SSPI module was offered by Syneapps including source code. But the development for it has been stopped. But don't despair, the community has picked it up and improved it. It has a new home on *SourceForge* [http://sourceforge.net/projects/mod-auth-sspi/].

- Download the module which matches your apache version, then copy the file `mod_auth_sspi.so` into the Apache modules folder.
- Edit the Apache config file: add the line

  ```
  LoadModule sspi_auth_module modules/mod_auth_sspi.so
  ```

  to the `LoadModule` section. Make sure you insert this line *before* the line

  ```
  LoadModule auth_module modules/mod_auth.so
  ```

- To make the Subversion location use this type of authentication you have to change the line

  ```
  AuthType Basic
  ```

  to

  ```
  AuthType SSPI
  ```

  also you need to add

  ```
  SSPIAuth On
  SSPIAuthoritative On
  SSPIDomain <domaincontroller>
  SSPIOmitDomain on
  SSPIUsernameCase lower
  SSPIPerRequestAuth on
  SSPIOfferBasic On
  ```

  within the `<Location /svn>` block. If you don't have a domain controller, leave the name of the domain control as `<domaincontroller>`.

Note that if you are authenticating using SSPI, then you don't need the `AuthUserFile` line to define a password file any more. Apache authenticates your username and password against your windows domain instead. You will need to update the users list in your `svnaccessfile` to reference `DOMAIN\username` as well.

Importante

The SSPI authentication is only enabled for SSL secured connections (https). If you're only using normal http connections to your server, it won't work.

To enable SSL on your server, see the chapter: Seção 3.7.9, “Autenticação no servidor com SSL”

Dica

Subversion `AuthzSVNAccessFile` files are case sensitive in regard to user names (`JUser` is different from `juser`).

In Microsoft's world, Windows domains and user names are not case sensitive. Even so, some network administrators like to create user accounts in CamelCase (e.g. `JUser`).

This difference can bite you when using SSPI authentication as the windows domain and user names are passed to Subversion in the same case as the user types them in at the prompt. Internet Explorer often passes the username to Apache automatically using whatever case the account was created with.

The end result is that you may need at least two entries in your `AuthzSVNAccessFile` for each user -- a lowercase entry and an entry in the same case that Internet Explorer passes to Apache. You will also need to train your users to also type in their credentials using lower case when accessing repositories via TortoiseSVN.

Apache's Error and Access logs are your best friend in deciphering problems such as these as they will help you determine the username string passed onto Subversion's `AuthzSVNAccessFile` module. You may need to experiment with the exact format of the user string in the `svnaccessfile` (e.g. `DOMAIN\user` vs. `DOMAIN//user`) in order to get everything working.

3.7.8. Origem Múltiplica de Atenticação

It is also possible to have more than one authentication source for your Subversion repository. To do this, you need to make each authentication type non-authoritative, so that Apache will check multiple sources for a matching username/password.

A common scenario is to use both Windows domain authentication and a `passwd` file, so that you can provide SVN access to users who don't have a Windows domain login.

- To enable both Windows domain and `passwd` file authentication, add the following entries within the `<Location>` block of your Apache config file:

```
AuthBasicAuthoritative Off
SSPIAuthoritative Off
```

Here is an example of the full Apache configuration for combined Windows domain and `passwd` file authentication:

```
<Location /svn>
  DAV svn
  SVNListParentPath on
  SVNParentPath D:\SVN

  AuthName "Subversion repositories"
  AuthzSVNAccessFile svnaccessfile.txt

# NT Domain Logins.
  AuthType SSPI
  SSPIAuth On
  SSPIAuthoritative Off
  SSPIDomain <domaincontroller>
  SSPIOfferBasic On
```

```
# Htpasswd Logins.
  AuthType Basic
  AuthBasicAuthoritative Off
  AuthUserFile passwd

  Require valid-user
</Location>
```

3.7.9. Autenticação no servidor com SSL

Even though Apache 2.2.x has OpenSSL support, it is not activated by default. You need to activate this manually.

1. In the apache config file, uncomment the lines:

   ```
   #LoadModule ssl_module modules/mod_ssl.so
   ```

   and at the bottom

   ```
   #Include conf/extra/httpd-ssl.conf
   ```

   then change the line (on one line)

   ```
   SSLMutex "file:C:/Program Files/Apache Software Foundation/\
   Apache2.2/logs/ssl_mutex"
   ```

   to

   ```
   SSLMutex default
   ```

2. Next you need to create an SSL certificate. To do that open a command prompt (DOS-Box) and change to the Apache folder (e.g. `C:\program files\apache group\apache2`) and type the following command:

   ```
   bin\openssl req -config conf\openssl.cnf -new -out my-server.csr
   ```

   You will be asked for a passphrase. Please don't use simple words but whole sentences, e.g. a part of a poem. The longer the phrase the better. Also you have to enter the URL of your server. All other questions are optional but we recommend you fill those in too.

   Normally the `privkey.pem` file is created automatically, but if it isn't you need to type this command to generate it:

   ```
   bin\openssl genrsa -out conf\privkey.pem 2048
   ```

   Next type the commands

   ```
   bin\openssl rsa -in conf\privkey.pem -out conf\server.key
   ```

   and (on one line)

```
bin\openssl req -new -key conf\server.key -out conf\server.csr \
-config conf\openssl.cnf
```

and then (on one line)

```
bin\openssl x509 -in conf\server.csr -out conf\server.crt
                 -req -signkey conf\server.key -days 4000
```

This will create a certificate which will expire in 4000 days. And finally enter (on one line):

```
bin\openssl x509 -in conf\server.cert -out conf\server.der.crt
                 -outform DER
```

These commands created some files in the Apache `conf` folder (`server.der.crt`, `server.csr`, `server.key`, `.rnd`, `privkey.pem`, `server.cert`).

3. Reinicie o serviço Apache.
4. Point your browser to `https://servername/svn/project` ...

SSL e Internet Explorer

If you're securing your server with SSL and use authentication against a windows domain you will encounter that browsing the repository with the Internet Explorer doesn't work anymore. Don't worry - this is only the Internet Explorer not able to authenticate. Other browsers don't have that problem and TortoiseSVN and any other Subversion client are still able to authenticate.

If you still want to use IE to browse the repository you can either:

- define a separate `<Location /path>` directive in the Apache config file, and add the `SSPIBasicPreferred On`. This will allow IE to authenticate again, but other browsers and Subversion won't be able to authenticate against that location.
- Offer browsing with unencrypted authentication (without SSL) too. Strangely IE doesn't have any problems with authenticating if the connection is not secured with SSL.
- In the SSL "standard" setup there's often the following statement in Apache's virtual SSL host:

  ```
  SetEnvIf User-Agent ".*MSIE.*" \
              nokeepalive ssl-unclean-shutdown \
              downgrade-1.0 force-response-1.0
  ```

  There are (were?) good reasons for this configuration, see *http://www.modssl.org/docs/2.8/ssl_faq.html#ToC49* But if you want NTLM authentication you have to use `keepalive`. If You uncomment the whole `SetEnvIf` you should be able to authenticate IE with windows authentication over SSL against the Apache on Win32 with included `mod_auth_sspi`.

Forçando o acesso SSL

When you've set up SSL to make your repository more secure, you might want to disable the normal access via non-SSL (http) and only allow https access. To do this, you have to add another directive to the Subversion `<Location>` block: `SSLRequireSSL`.

An example `<Location>` block would look like this:

```
<Location /svn>
  DAV svn
  SVNParentPath D:\SVN
  SSLRequireSSL
  AuthType Basic
  AuthName "Subversion repositories"
  AuthUserFile passwd
  #AuthzSVNAccessFile svnaccessfile
  Require valid-user
</Location>
```

3.7.10. Usango um certificados de cliente com servidor SSL virtual

Enviado para a lista de discussão do TortoiseSVN por Nigel Green. Obrigado!

In some server configurations you may need to setup a single server containing 2 virtual SSL hosts: The first one for public web access, with no requirement for a client certificate. The second one to be secure with a required client certificate, running a Subversion server.

Adding an `SSLVerifyClient Optional` directive to the *per-server* section of the Apache configuration (i.e. outside of any `VirtualHost` and `Directory` blocks) forces Apache to request a client Certificate in the initial SSL handshake. Due to a bug in `mod_ssl` it is essential that the certificate is requested at this point as it does not work if the SSL connection is re-negotiated.

The solution is to add the following directive to the virtual host directory that you want to lock down for Subversion:

```
SSLRequire %{SSL_CLIENT_VERIFY} eq "SUCCESS"
```

This directive grants access to the directory only if a client certificate was received and verified successfully.

To summarise, the relevant lines of the Apache configuration are:

```
SSLVerifyClient Optional

### Virtual host configuration for the PUBLIC host
### (not requiring a certificate)

<VirtualHost 127.0.0.1:443>
  <Directory "pathtopublicfileroot">
  </Directory>
</VirtualHost>

### Virtual host configuration for SUBVERSION
### (requiring a client certificate)
<VirtualHost 127.0.0.1:443>
  <Directory "subversion host root path">
    SSLRequire %{SSL_CLIENT_VERIFY} eq "SUCCESS"
  </Directory>

  <Location /svn>
    DAV svn
    SVNParentPath /pathtorepository
```

```
  </Location>
</VirtualHost>
```

Capítulo 4. Guia do Uso Diário

Este documento descreve o uso diário do cliente TortoiseSVN. Este capítulo *não* é uma introdução aos sistemas de controle de versão, e *não* é uma introdução ao Subversion (SVN). Este capítulo é como um documento auxiliar para olhar quando você sabe mais ou menos o que quer fazer, mas não está absolutamente certo de como fazer.

Se você precisa aprender sobre o controle de versão usando Subversion, então nós recomendamos que você leia o fantástico livro: *Controle de Versão com Subversion* [http://svnbook.red-bean.com/].

Este documento é também um trabalho em progresso, assim como TortoiseSVN e Subversion também são. Se você encontrar qualquer erro, por favor reporte para a lista de discussão para que possamos atualizar a documentação. Algumas imagens do Guia do Uso Diário (GUD) pode não refletir a última versão do aplicativo. Por favor, perdoe-nos. Nós trabalhamos no TortoiseSVN no nosso tempo livre.

A fim de obter o máximo do Guia de Uso Diário:

- Você já deverá ter instalado o TortoiseSVN.
- Você deverá estar familiarizado com um sistema de controle de versões.
- Você deverá ter conhecimentos básicos do Subversion.
- Você deve ter configurado um servidor e/ou ter acesso a um repositório Subversion.

4.1. Começando

4.1.1. Sobreposição dos Ícones

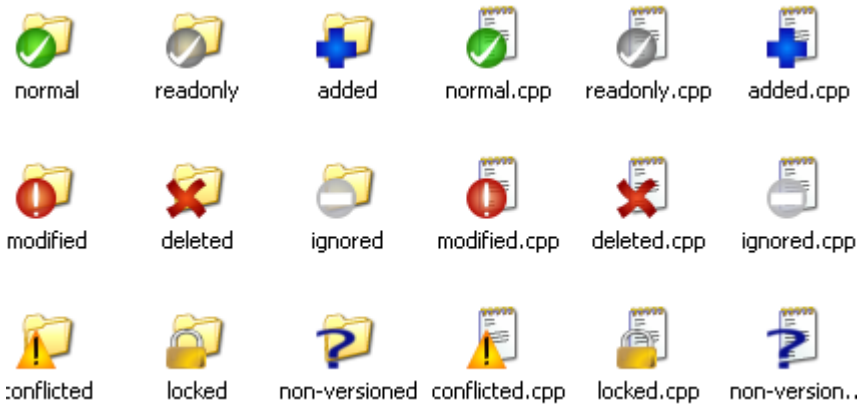


Figura 4.1. Explorer mostra os ícones sobrepostos

Um dos recursos mais visíveis do TortoiseSVN é o de sobreposição de ícones que aparece em arquivos em sua cópia de trabalho. Eles te mostram em um instante quais de seus arquivos foram modificados. Consulte Seção 4.7.1, “Sobreposição dos Ícones” para descobrir o que as diferentes sobreposições representam.

4.1.2. Menus do Contexto

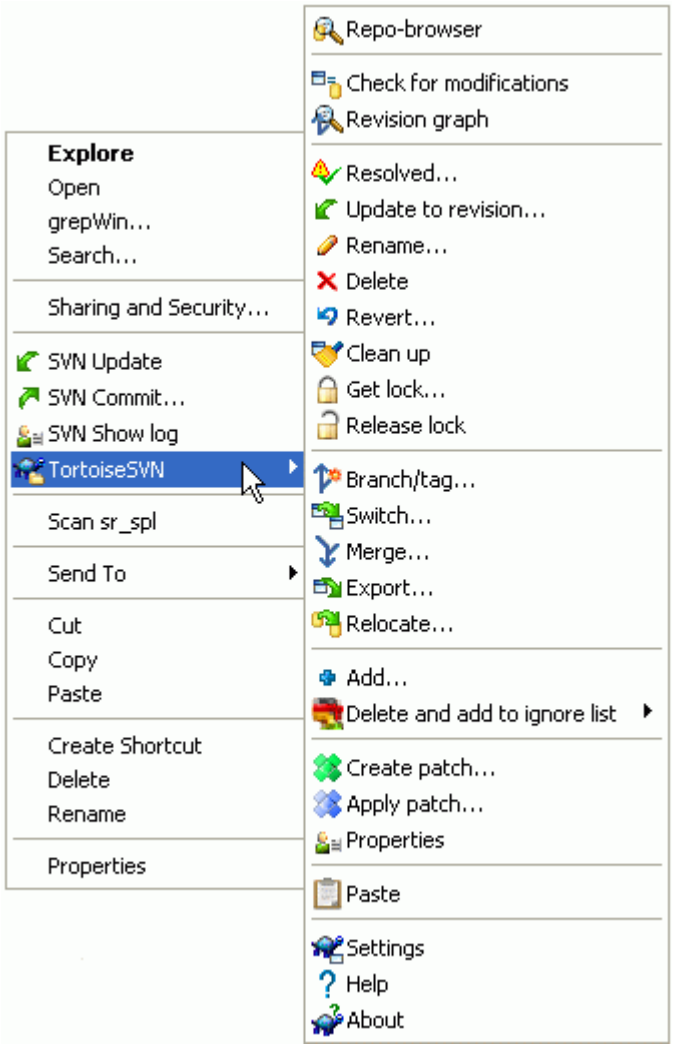


Figura 4.2. Menu do contexto para diretórios controlados

Todos os comandos do TortoiseSVN são acessados através do menu de contexto do Windows Explorer. A maioria são diretamente visíveis quando se clica com o botão direito em um arquivo ou diretório. Os comandos estarão disponíveis se o arquivo ou o diretório que o contém está no controle de versão. Você também pode acessar o menu do TortoiseSVN como parte do menu Arquivo, no Windows Explorer.

Dica

Alguns comandos que são usados muito raramente, só estão disponíveis no menu de contexto extendido. Para trazê-los ao menu de contexto, segure pressionado o botão **Shift** enquanto clica com o botão direito do mouse.

Em alguns casos você poderá ver várias entradas do TortoiseSVN. Isto não é um bug!

Figura 4.3. Atalho no menu arquivo do Explorer em um diretório controlado

Este exemplo é de um atalho não-versionado dentro de um diretório versionado, e no menu Arquivo do Windows Explorer há *três* entradas para o TortoiseSVN. Uma para o diretório, uma para o atalho, e uma terceira para o objeto para o qual o atalho aponta. Para ajudar a distinguir entre elas, os ícones um indicador no canto inferior direito para mostrar se a entrada se refere a um arquivo, um diretório, um atalho ou para múltiplos ítens selecionados.

Se você utiliza Windows 2000 descobrirá que os menus de contexto são mostrados com texto liso, sem os ícones de menu por cima. Estamos conscientes que isto foi trabalhado em versões anteriores, mas a Microsoft mudou a forma como os manipuladores de ícones funcionam para o vista, exigindo que nós usássemos um método diferente de mostrá-los, que infelizmente não funciona em Windows 2000.

4.1.3. Arrastar e Soltar

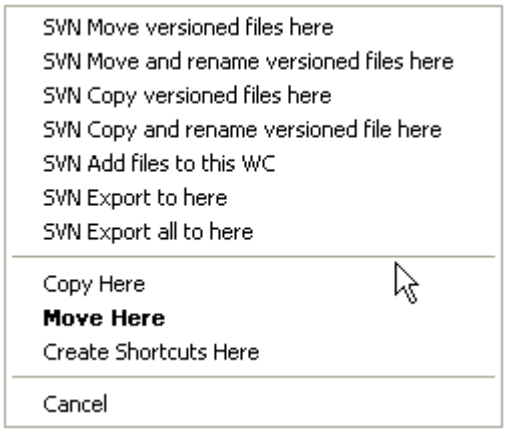


Figura 4.4. Menu de quando se clica com o botão direito e se arrasta um diretório que está sob o controle de versão.

Outros comandos estão disponíveis ao se clicar e arrastar, quando você clica com o botão direito e arrasta arquivos ou diretórios para um novo endereço dentro da cópia de trabalho ou quando você clica com o botão direito e arrasta um arquivo ou diretório não versionado para um diretório sob o controle de versão.

4.1.4. Atalhos Comuns

Algumas operações comuns do Windows tem atalhos bem conhecidos, mas não aparecem em botões e menus. Se você não sabe como fazer algo óbvio como atualizar uma visualização, cheque aqui.

F1
: Ajuda, claro.

F5
: Atualiza a visualização atual. Este talvez seja o mais útil comando de uma única tecla. Por exemplo... No Explorer ele irá atualizar os ícones sobrepostos na sua cópia de trabalho. Na janela de Commit irá reexplorar a Cópia de Trabalho para ver o que precisa ser enviado. Na janela Revision Log irá verificar o repositório novamente a procura de novas modificações recentes.

Ctrl-A
: Seleciona tudo. Pode ser usado se você tiver uma mensagem de erro e precisar copiar e colar em um email. Use Ctrl-A para selecionar a mensagem e então...

Ctrl-C
: ... Copia o texto selecionado.

4.1.5. Autenticação

Se o repositório que você está tentando acessar é protegido por senha, uma janela de autenticação aparecerá.

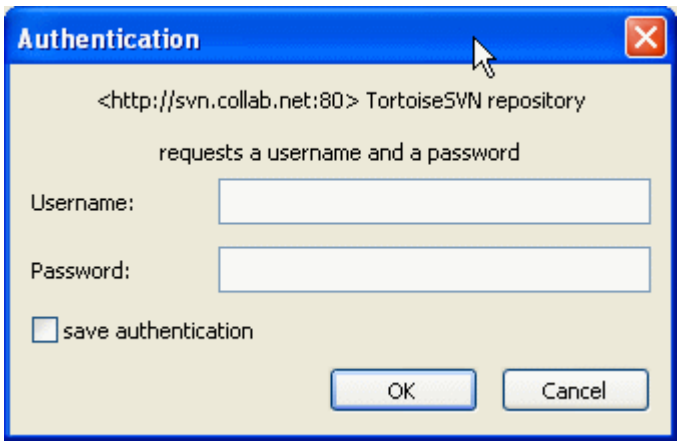


Figura 4.5. Janela de Autenticação

Entre seu nome de usuário e senha. A caixa de seleção (checkbox) fará com que o TortoiseSVN guarde suas credenciais no diretório padrão do Subversion. `%APPDATA%\Subversion\auth` na árvore de subdiretórios:

- `svn.simple` contains credentials for basic authentication (username/password).
- `svn.ssl.server` contém os certificados do servidor para SSL.
- `svn.username` contém credenciais para autenticações que necessitem apenas do nome de usuário (não é necessário senha).

Se você quiser limpar o cache de autenticação para todos os servidores, você poderá fazê-lo através da página Saved Data da página de configurões do TortoiseSVN. O botão limpa todos os dados de autenticação em cache dos diretórios `auth` do Subversion, assim como qualquer dados de autenticação armazenados no registro por versões anteriores do TortoiseSVN. Veja Seção 4.30.6, “Saved Data Settings”.

Algumas pessoas preferem que os dados de autenticação sejam excluídos quando fazem logoff do Windows ou desligam o computador. O meio de se fazer isto é executar um script ao desligar o computador que exclui o diretório `%APPDATA%\Subversion\auth`, e.g.

```
@echo off
rmdir /s /q "%APPDATA%\Subversion\auth"
```

Você poderá encontrar uma descrição de como instalar este tipo de script em *windows-help-central.com* [http://www.windows-help-central.com/windows-shutdown-script.html].

Para mais informações sobre como instalar seu servidor para autenticação e controle de acesso, veja Seção 3.5, “Acessando o Repositório”

4.1.6. Maximizando Janelas

Muitas das janelas do TortoiseSVN contém muita informação para mostrar, mas é frequentemente útil maximizar apenas a altura ou apenas a largura da janela, ao invés de maximizar na tela toda. Por conveniência, há atalhos para isto no botão Maximizar. Use o botão do meio do mouse para maximizar verticalmente, e o botão direito do mouse para maximizar horizontalmente.

4.2. Importando Dados Para Um Repositório

4.2.1. Importar

If you are importing into an existing repository which already contains some projects, then the repository structure will already have been decided. If are importing data into a new repository then it is worth taking the time to think about how it will be organised. Read Seção 3.1.5, “Leiaute do Repositório” for further advice.

Esta seção descreve o comando de importação do Subversion, o qual foi projetado para a importação de diretórios hierárquicos para o repositório em apenas um passo. Embora faça o trabalho, o comando de importação possui algumas deficiências:

- Não há qualquer maneira de selecionar os arquivos e pastas que deverão ser incluídos, em compensação poderá utilizar as configurações globais para ignorar alguns arquivos e pastas.
- A pasta importada não se torna uma cópia de trabalho. Você tem que fazer um checkout para copiar os arquivos desde o servidor.
- É fácil importar para uma pasta equivocada no repositório.

Por essas razões, recomendamos que você não use o comando de importação, mas sim que siga o método de duas etapas descrito em Seção 4.2.2, “Importando na Pasta”. Mas já que você está aqui, de forma básica é assim que funciona a importação ...

Antes de importar seu projeto para um repositório, você deveria:

1. Remova todos os arquivos que não são necessários para construir o projeto (arquivos temporários, arquivos que são gerados por um compilador como por exemplo *. obj, binários compilados, ...)
2. Organize os arquivos em pastas e sub-pastas. Embora posteriormente seja possível renomear/mover arquivos, é altamente recomendado que obtenha a estrutura correta do seu projeto antes de realizar a importação!

Agora selecione a pasta de nível superior da sua estrutura de diretório do projeto no Windows Explorer e com o botão direito clique para abrir o menu de contexto. Selecione o comando TortoiseSVN → Importar ...

Nesta janela de diálogo, você deve digitar a URL do local do repositório para onde você deseja importar o seu projeto. É muito importante que a pasta local que está importando não apareça no repositório, mas apenas o seu conteúdo. Por exemplo, se você tem uma estrutura:

```
C:\Projects\Widget\source
C:\Projects\Widget\doc
C:\Projects\Widget\images
```

e você está importando `C:\Projects\Widget` para `http://mydomain.com/svn/trunk` você se surpreenderá ao descobrir que seu subdiretórios irão diretamente para `trunk`, ao invés de irem para o subdiretório `Widget`. Você deverá especificar o subdiretório como parte do URL, `http://mydomain.com/svn/trunk/Widget-X`. Note que o comando de importação irá criar automaticamente subdiretórios dentro do repositório se eles não existirem.

A mensagem de importação é usada como uma mensagem para o histórico.

Por padrão, arquivos e pastas que correspondem aos padrões globais de pastas/arquivos ignorados, *não* são importados. Para contornar esse comportamento você pode utilizar o recurso Incluir arquivos ignorados. Consulte Seção 4.30.1, “Configurações Gerais” para obter mais informações sobre como definir um padrão global para ignorar pastas/arquivos.

Assim que clicar em OK TortoiseSVN inicia a importação da "árvore" de diretórios, incluindo todos os arquivos. O projeto estará armazenado no repositório sob controle de versão. Por favor note que a pasta que você importou *NÃO* está sob controle de versão! Para obter uma *cópia de trabalho* com versão controlada, você deverá fazer um Checkout da versão que você acabou de importar. Ou continue lendo sobre como proceder em "Importando na Pasta".

4.2.2. Importando na Pasta

Assumindo que você já possui um repositório e que deseja adicionar uma nova pasta na estrutura desse repositório, apenas siga estes passos:

1. Use o navegador de repositório para criar uma nova pasta de projeto diretamente no repositório.

2. Checkout a nova pasta em um nível superior à pasta que você deseja importar. Você receberá um aviso de que a pasta local não está vazia. Agora você tem uma pasta de nível superior com controle de versão, mas com conteúdo sem controle de versão.

3. Use TortoiseSVN → Adicionar... na pasta sob controle de versão para adicionar algum ou todo conteúdo. Você pode adicionar e remover arquivos, defina as configurações em `svn:ignore` para as pastas e faça outras alterações que você necessita.

4. Ao submeter a pasta de nível superior, você terá uma nova "árvore" sob controle de versão e uma cópia local de trabalho, criada a partir de sua pasta existente.

4.2.3. Arquivos Especiais

Às vezes, você precisa ter um arquivo sob controle de versão que contém dados específicos de usuário. Isso significa que você terá um arquivo que todo desenvolvedor/usuário precisará modificar para adequá-lo a sua configuração local. Portanto, é difícil manter um arquivo sob controle de versão porque cada usuário poderá submeter alterações no repositório constantemente.

Nesses casos, sugere-se a utilização de um arquivo *modelo*. Crie um arquivo que contém todos os dados que seus desenvolvedores necessitam, adicione e mantenha esse arquivo sob controle de versão e permita que os desenvolvedores façam o Checkout desse arquivo. Em seguida, cada desenvolvedor deverá fazer *uma cópiaemphasis> desse arquivo e renomear essa c*

Por exemplo, veja que script TortoiseSVN chama um arquivo chamado `TortoiseVars.bat` que não existe no repositório. Somente existe o arquivo `TortoiseVars.tmpl`. `TortoiseVars.tmpl` é o arquivo modelo que cada colaborador deve criar uma cópia e renomear a cópia para `TortoiseVars.bat`. Nesse arquivo, adicionamos comentários para que os usuários vejam as linhas que eles devem editar e mudar de acordo com sua instalação local.

Portanto, para não aborrecer usuários, também adicionamos o arquivo `TortoiseVars.bat` para a lista de "ignorados" da pasta que o contém, ou seja, configuramos `svn:ignore` para incluir esse arquivo. Dessa forma, ele não aparecerá como "não versionado" em cada submissão.

4.3. Obtendo Uma Cópia de Trabalho

Para obter uma cópia de trabalho você precisa realizar um *obter* do repositório.

Select a directory in windows explorer where you want to place your working copy. Right click to pop up the context menu and select the command TortoiseSVN → Checkout..., which brings up the following dialog box:

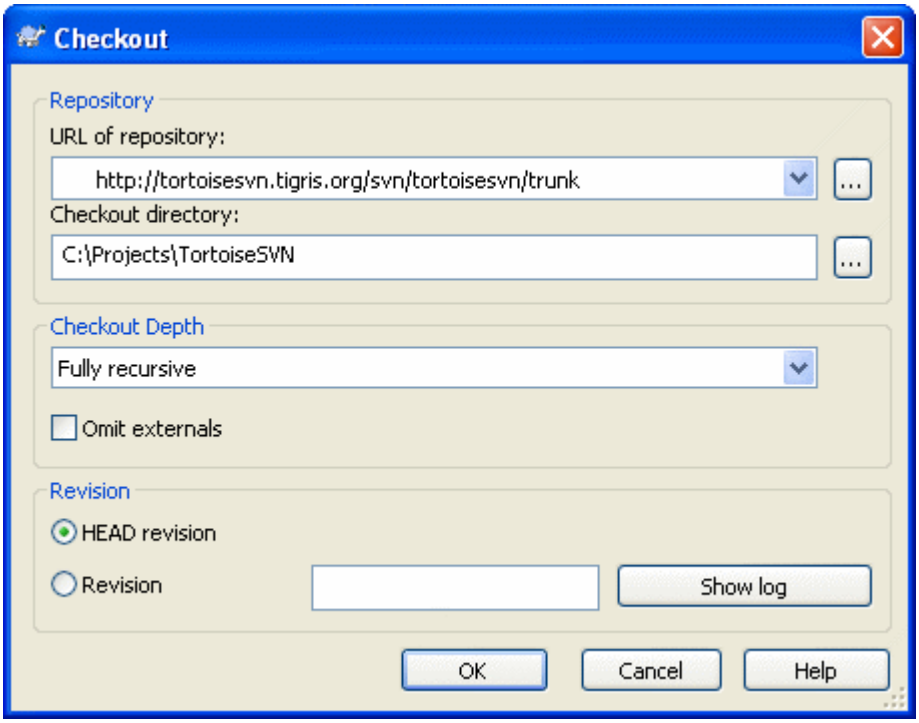


Figura 4.6. A janela de Obtenção

If you enter a folder name that does not yet exist, then a directory with that name is created.

4.3.1. Profundidade da Obtenção

You can choose the *depth* you want to checkout, which allows you to specify the depth of recursion into child folders. If you want just a few sections of a large tree, You can checkout the top level folder only, then update selected folders recursively.

Totalmente recursiva
: Checkout the entire tree, including all child folders and sub-folders.

Immediate children, including folders
: Checkout the specified directory, including all files and child folders, but do not populate the child folders.

Somente os arquivos filhos
: Checkout the specified directory, including all files but do not checkout any child folders.

Somente este item
: Checkout the directory only. Do not populate it with files or child folders.

Cópia de trabalho
: Retain the depth specified in the working copy. This option is not used in the checkout dialog, but it is the default in all other dialogs which have a depth setting.

Excluir
: Used to reduce working copy depth after a folder has already been populated. This option is only available in the Update to revision dialog.

If you check out a sparse working copy (i.e., by choosing something other than `fully recursive` for the checkout depth), you can fetch additional sub-folders by using the repository browser (Seção 4.24, “O Navegador de Repositório”) or the check for modifications dialog (Seção 4.7.3, “Estado Local e Remoto”).

In the repository browser, Right click on the checked out folder, then use TortoiseSVN → Repo-Browser to bring up the repository browser. Find the sub-folder you would like to add to your working copy, then use Context menu → Update item to revision... That menu will only be visible if the selected item does not exist yet in your working copy, but the parent item does exist.

In the check for modifications dialog, first click on the button Check repository. The dialog will show all the files and folders which are in the repository but which you have not checked out as `remotely added`. Right click on the folder(s) you would like to add to your working copy, then use Context menu → Update.

This feature is very useful when you only want to checkout parts of a large tree, but you want the convenience of updating a single working copy. Suppose you have a large tree which has sub-folders `Project01` to `Project99`, and you only want to checkout `Project03`, `Project25` and `Project76/SubProj`. Use these steps:

1. Checkout the parent folder with depth “Only this item” You now have an empty top level folder.

2. Select the new folder and use TortoiseSVN → Repo browser to display the repository content.

3. Right click on `Project03` and Context menu → Update item to revision.... Keep the default settings and click on OK. You now have that folder fully populated.

   Repeat the same process for `Project25`.

4. Navigate to `Project76/SubProj` and do the same. This time note that the `Project76` folder has no content except for `SubProj`, which itself is fully populated. Subversion has created the intermediate folders for you without populating them.

Mudando a profundidade da cópia de trabalho

Once you have checked out a working copy to a particular depth you can change that depth later to get more or less content using Context menu → Update item to revision....

Usando um servidor antigo

Pre-1.5 servers do not understand the working copy depth request, so they cannot always deal with requests efficiently. The command will still work, but an older server may send all the data, leaving the client to filter out what is not required, which may mean a lot of network traffic. If possible you should upgrade your server to 1.5.

If the project contains references to external projects which you do *not* want checked out at the same time, use the Omit externals checkbox.

Importante

If Omit externals is checked, or if you wish to increase the depth value, you will have to perform updates to your working copy using TortoiseSVN → Update to Revision... instead of TortoiseSVN → Update. The standard update will include all externals and keep the existing depth.

It is recommended that you check out only the `trunk` part of the directory tree, or lower. If you specify the parent path of the directory tree in the URL then you might end up with a full hard disk since you will get a copy of the entire repository tree including every branch and tag of your project!

Exportando

Sometimes you may want to create a local copy without any of those `.svn` directories, e.g. to create a zipped tarball of your source. Read Seção 4.26, “Exporting a Subversion Working Copy” to find out how to do that.

4.4. Submetendo Suas Alterações Para o Repositório

Enviar as alterações que você fez em sua cópia de trabalho é conhecido como *Submeter* as alterações. Mas antes de você submeter você deve ter certeza que sua cópia de trabalho está atualizada. Você pode usar tanto TortoiseSVN → Update diretamente. Ou você pode usar TortoiseSVN → Check for Modifications primeiro, para ver quais arquivos foram alterados localmente ou no servidor.

4.4.1. A Janela de Submissão

Se sua cópia de trabalho estiver atualizada e não houver conflitos, você está pronto para submeter suas alterações. Selecione qualquer arquivo e/ou pastas que você quiser submeter, então TortoiseSVN → Commit....

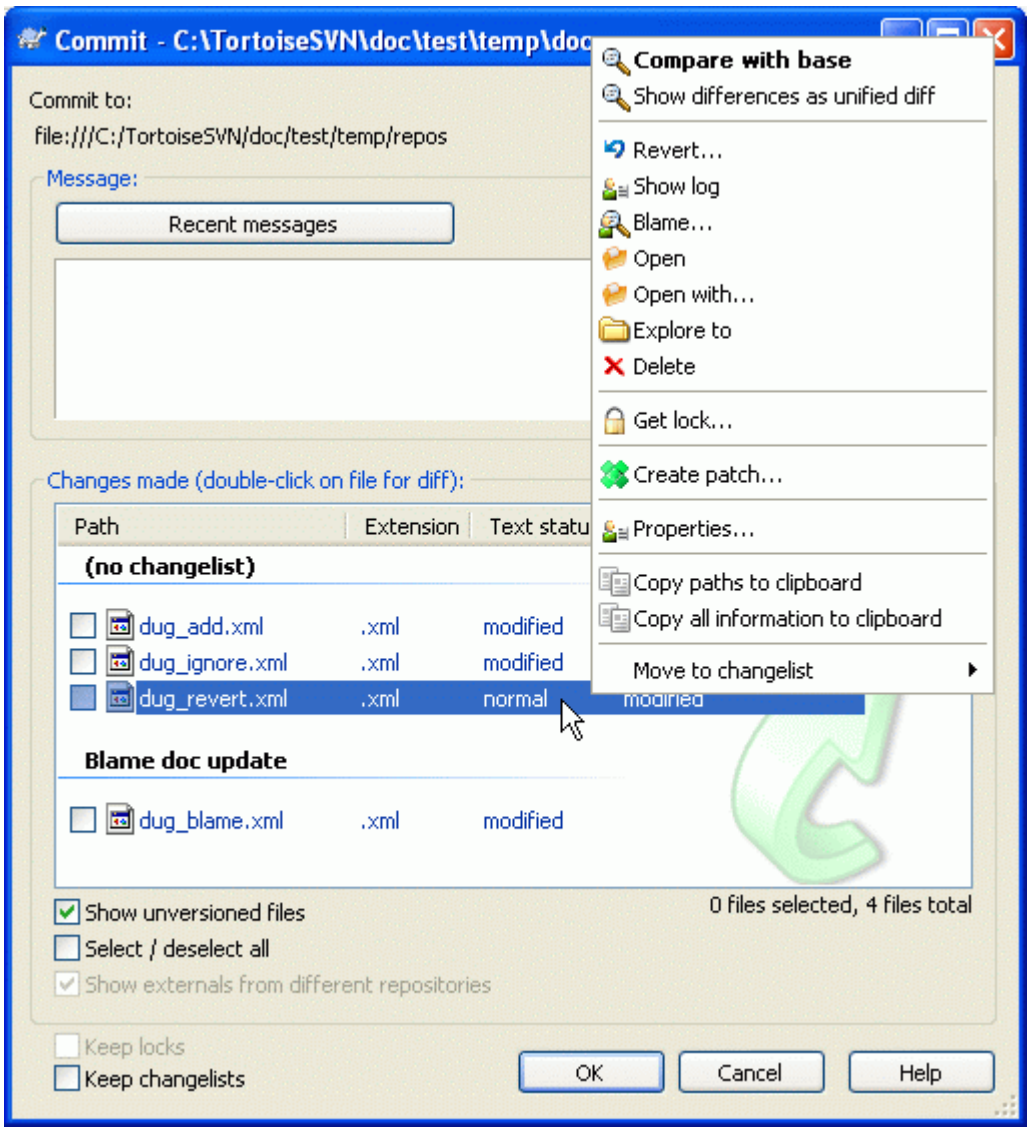


Figura 4.7. A janela de Submissão

A janela de submissão irá exibir todos os arquivos modificados, incluindo arquivos adicionados, deletados e não versionados. Se você não quiser que um arquivo alterado seja submetido, apenas desmarque o arquivo. Se você quiser incluir um arquivo não versionado, apenas marque o arquivo para adiciona-lo a submissão

Items que foram movidos para um caminho de repositório diferente são também indicados usando o marcador `(s)`. Você pode ter movido algo enquanto trabalhava em uma ramificação (branch) e se esqueceu de mover de volta para a versão principal (trunk). Esté é seu sinal de aviso!

Submeter arquivos ou diretórios?

Quando você submete arquivos, a janela de submissão exibe apenas os arquivos que você selecionou. Quando você submete uma pasta, a janela de submissão irá exibir selecionado os arquivos modificados automaticamente. Se você se esqueceu de um novo arquivo que criou, submeter a pasta irá encontra-lo de qualquer forma. Submeter uma pasta *não* significa que todos os arquivos serão marcados como modificados; Isso apenas fará sua vida mais fácil.

If you have modified files which have been included from a different repository using `svn:externals`, those changes cannot be included in the same atomic commit. A warning symbol below the file list tells you if this has happened, and the tooltip explains that those external files have to be committed separately.

Muitos arquivos não versionados na janela de submissão

Se você acha que a janela de submissão exibe muitos arquivos não versionados (arquivos gerados pelo compilador ou backups), existem muitas formas de lidar com isso. Você pode

- add the file (or a wildcard extension) to the list of files to exclude on the settings page. This will affect every working copy you have.
- add the file to the `svn:ignore` list using TortoiseSVN → Add to ignore list This will only affect the directory on which you set the `svn:ignore` property. Using the SVN Property Dialog, you can alter the `svn:ignore` property for a directory.

Read Seção 4.13, “Ignorando Arquivos e Diretórios” para mais informações

Click duplo em qualquer arquivo modificado na janela de submissão irá abrir um programa de diff exibindo suas modificações. O menu de contexto irá dar mais opções, como exibido na screenshot. Você também arrastar arquivos da janela de submissão para outros aplicativos como editores de textos ou IDEs.

Você pode selecionar ou deselecionar items clicando na caixa de seleção a esquerda do item. Para diretórios você pode usar **Shift**-select para fazer a ação recursiva.

The columns displayed in the bottom pane are customizable. If you right click on any column header you will see a context menu allowing you to select which columns are displayed. You can also change column width by using the drag handle which appears when you move the mouse over a column boundary. These customizations are preserved, so you will see the same headings next time.

By default when you commit changes, any locks that you hold on files are released automatically after the commit succeeds. If you want to keep those locks, make sure the Keep locks checkbox is checked. The default state of this checkbox is taken from the `no_unlock` option in the Subversion configuration file. Read Seção 4.30.1, “Configurações Gerais” for information on how to edit the Subversion configuration file.

Arrastar e Soltar

You can drag files into the commit dialog from elsewhere, so long as the working copies are checked out from the same repository. For example, you may have a huge working copy

with several explorer windows open to look at distant folders of the hierarchy. If you want to avoid committing from the top level folder (with a lengthy folder crawl to check for changes) you can open the commit dialog for one folder and drag in items from the other windows to include within the same atomic commit.

Você pode arrastar arquivos não versionados que residem em uma cópia de trabalho para uma janela de submissão e eles irão ser adicionados ao SVN automaticamente.

Reparando Referência Externas Renomeadas

Sometimes files get renamed outside of Subversion, and they show up in the file list as a missing file and an unversioned file. To avoid losing the history you need to notify Subversion about the connection. Simply select both the old name (missing) and the new name (unversioned) and use Context Menu → Repair Move to pair the two files as a rename.

4.4.2. Lista de Alterações

The commit dialog supports Subversion's changelist feature to help with grouping related files together. Find out about this feature in Seção 4.8, “Lista de Alterações”.

4.4.3. Excluindo Itens de uma Lista de Submissões

Sometimes you have versioned files that change frequently but that you really don't want to commit. Sometimes this indicates a flaw in your build process - why are those files versioned? should you be using template files? But occasionally it is inevitable. A classic reason is that your IDE changes a timestamp in the project file every time you build. The project file has to be versioned as it includes all the build settings, but it doesn't need to be committed just because the timestamp changed.

To help out in awkward cases like this, we have reserved a changelist called `ignore-on-commit`. Any file added to this changelist will automatically be unchecked in the commit dialog. You can still commit changes, but you have to select it manually in the commit dialog.

4.4.4. Registro de Mensagens de Submissão

Be sure to enter a log message which describes the changes you are committing. This will help you to see what happened and when, as you browse through the project log messages at a later date. The message can be as long or as brief as you like; many projects have guidelines for what should be included, the language to use, and sometimes even a strict format.

You can apply simple formatting to your log messages using a convention similar to that used within emails. To apply styling to `text`, use `*text*` for bold, `_text_` for underlining, and `^text^` for italics.

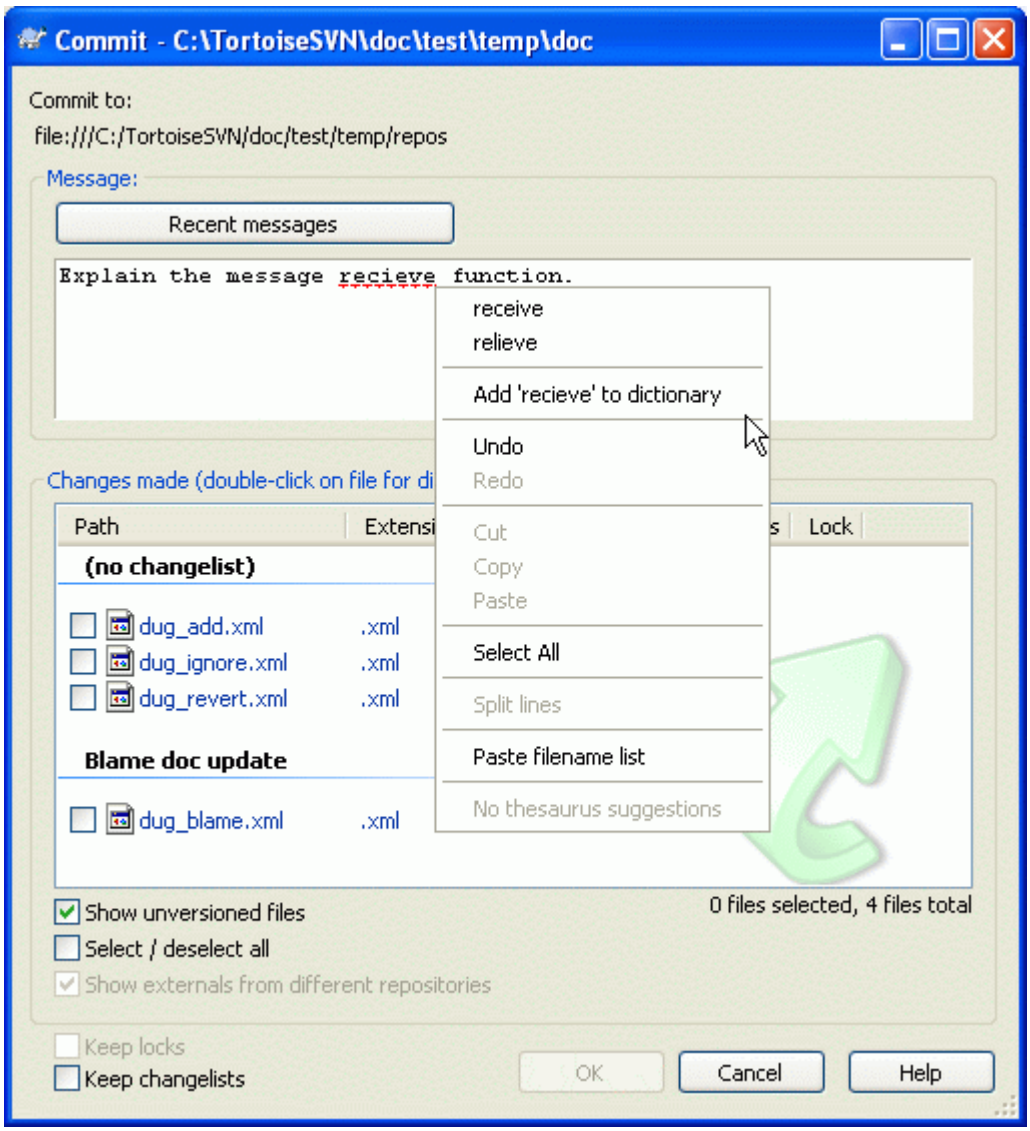


Figura 4.8. A Janela de Submissão Com Corretor Ortográfico

TortoiseSVN includes a spellchecker to help you get your log messages right. This will highlight any mis-spelled words. Use the context menu to access the suggested corrections. Of course, it doesn't know *every* technical term that you do, so correctly spelt words will sometimes show up as errors. But don't worry. You can just add them to your personal dictionary using the context menu.

The log message window also includes a filename and function auto-completion facility. This uses regular expressions to extract class and function names from the (text) files you are committing, as well as the filenames themselves. If a word you are typing matches anything in the list (after you have typed at least 3 characters, or pressed **Ctrl+Space**), a drop-down appears allowing you to select the full name. The regular expressions supplied with TortoiseSVN are held in the TortoiseSVN installation `bin` folder. You can also define your own regexes and store them in `%APPDATA%\TortoiseSVN\autolist.txt`. Of course your private autolist will not be overwritten when you update your installation of TortoiseSVN. If you are unfamiliar with regular expressions, take a look at the introduction at *http://en.wikipedia.org/wiki/Regular_expression*, and the online documentation and tutorial at *http://www.regular-expressions.info/*.

You can re-use previously entered log messages. Just click on Recent messages to view a list of the last few messages you entered for this working copy. The number of stored messages can be customized in the TortoiseSVN settings dialog.

You can clear all stored commit messages from the Saved data page of TortoiseSVN's settings, or you can clear individual messages from within the Recent messages dialog using the **Delete** key.

If you want to include the checked paths in your log message, you can use the command Context Menu → Paste filename list in the edit control.

Another way to insert the paths into the log message is to simply drag the files from the file list onto the edit control.

Propriedades de Pastas Especiais

There are several special folder properties which can be used to help give more control over the formatting of commit log messages and the language used by the spellchecker module. Read Seção 4.17, “Configurações do Projeto” for further information.

Integration with Bug Tracking Tools

If you have activated the bug tracking system, you can set one or more Issues in the Bug-ID / Issue-Nr: text box. Multiple issues should be comma separated. Alternatively, if you are using regex-based bug tracking support, just add your issue references as part of the log message. Learn more in Seção 4.28, “Integration with Bug Tracking Systems / Issue Trackers”.

4.4.5. Progresso da Submissão

After pressing OK, a dialog appears displaying the progress of the commit.

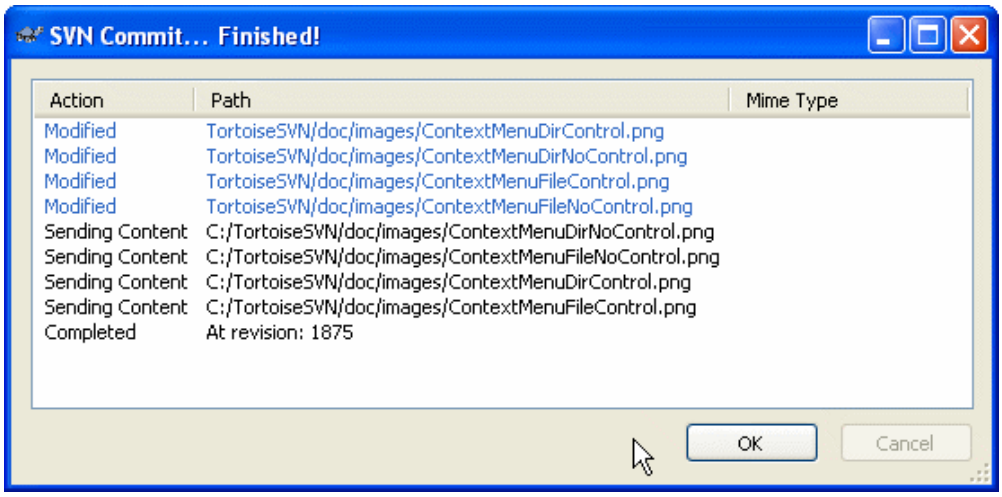


Figura 4.9. The Progress dialog showing a commit in progress

The progress dialog uses colour coding to highlight different commit actions

Azul
: Submetendo uma modificação.

Roxo
: Submetendo uma nova adição.

Vermelho escuro
: Submetendo um exclusão ou substituição.

Preto
: Todos os outros itens.

This is the default colour scheme, but you can customise those colours using the settings dialog. Read Seção 4.30.1.4, “TortoiseSVN Colour Settings” for more information.

4.5. Atualizar sua Cópia de Trabalho com mudanças feitas por outros

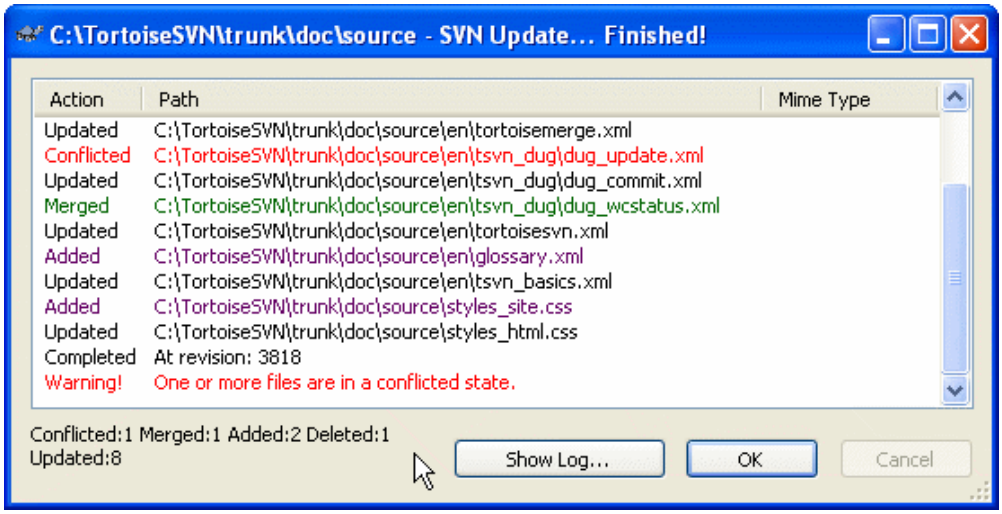


Figura 4.10. Janela de progresso mostrando atualização terminada

Periodicamente, você deve garantir que mudanças feitas por outros sejam incorporadas na sua cópia de trabalho. O processo de pegar as mudanças do servidor para sua cópia local é conhecido como *atualização*. A atualização pode ser feita em arquivos, uma seleção de arquivos, ou recursivamente na herarquia inteira de diretórios. Para atualizar, selecione os arquivos e/ou diretórios que quiser, clique com o botão direito e selecione TortoiseSVN → Atualizar no menu de contexto do explorer. Uma janela aparecerá mostrando o progresso da atualização em tempo real. As mudanças feitas por outros serão consolidadas com seus arquivos, mantendo quaisquer mudanças que você possa ter feito nos mesmos arquivos. O repositório *não* é afetado por uma atualização.

A janela de progresso faz uso de código de cores para sublinhar diferentes ações de atualização

Roxo
: Novo item adicionado à CT.

Vermelho escuro
: Item redundante apagado da sua CT, ou item substituído na sua CT.

Verde
: Mudanças no repositório consolidadas com suas mudanças locais com sucesso

Vermelho vivo
: Mudanças no repositório consolidadas com mudanças locais, resultando em conflitos que você precisa resolver.

Preto
: Item inalterado em sua CT atualizado com nova versão do repositório.

This is the default colour scheme, but you can customise those colours using the settings dialog. Read Seção 4.30.1.4, “TortoiseSVN Colour Settings” for more information.

Se você receber quaisquer *conflitos* durante uma atualização (isso pode acontecer se outras pessoas alteraram as mesmas linhas no mesmo arquivo que você e essas mudanças não são compatíveis) então a janela mostra esses conflitos em vermelho. Você pode efetuar um clique duplo nessas linhas para começar a ferramenta consolidação externa para resolver esses conflitos.

Quando a atualização estiver completa, a janela de progresso mostrará um sumário do número de itens atualizados, adicionados, removidos, em conflito, etc. abaixo da lista de arquivos. Esta informação pode ser copiada para a área de transferência usando **Ctrl+C**.

O Comando de Atualização padrão não possui opções e simplesmente atualizada sua cópia de trabalho para a revisão HEAD do repositório,que é o uso de caso mais comum. Se você deseja mais controle sobre o processo de atualização, você deve usar TortoiseSVN → Atualizar para revisão... ao invés. Isto permite que você atualize sua cópia de trabalho para uma revisão específica, não somente a mais recente. Suponha que sua cópia de trabalho está na revisão 100, mas você deseja refletir o estado que tinha na revisão 50 - então simplesmente atualize para a revisão 50. Na mesma janela você pode ainda escolher a *profundidade* na qual atualizar a pasta atual. Os termos usados estão descritos em Seção 4.3.1, “Profundidade da Obtenção”. A profundidade padrão é Cópia de trabalho, que preserva a configuração de profundidade existente. Você pode também escolher entre ignorar quaisquer projetos externos nesta atualização (ex: projetos referenciados usando `svn:externals`).

Cuidado

Se você atualizar um arquivo ou pasta para uma versõa específica, você não deve fazer mudanças nesses arquivos. Você receberá mensagens de erro do tipo “desatualizados” quando tentar submetê-las! Se você quiser desfazer mudanças feitas em um arquivo e começar novamente de uma versão anterior, você pode reverter para uma versão anterior a partir da janela de log de revisões. Dê uma olhada em Seção B.4, “Roll back (Undo) revisions in the repository” para mais instruções, e métodos alternativos.

Atualizar para Revisão pode ser ocasionalmente útil para ver como seu projeto parecia em um momento anterior de sua história. Mas de modo geral, atualizar arquivos individualmente para uma versão anterior não é uma boa idéia porque deixa sua cópia de trabalho em estado inconsistente. Se o arquivo que você está atualizando mudou de nome, você pode até descobrir que o arquivo simplesmente desaparece da sua cópia de trabalho porque não havia arquivo com aquele nome na revisão anterior. Você também deve notar que o item terá umrevestimento verde normal, para que seja indistinguível dos arquivos que estão atualizados com a última versão.

Se você simplesmente quer uma cópia local de uma versão anterior de um arquivo é melhor usar o comando Menu de Contexto → Salvar revisão para... a partir da janela de log daquele arquivo.

Múltiplos Arquivos/Diretórios

Se você selecionar múltiplos arquivos e pastas no explorer e depois selecionar Atualizar, todos estes arquivos/pastas serão atualizados um a um. O TortoiseSVN garantirá que todos arquivos/pastas que são do mesmo repositório serão atualizados para a mesma revisão! Mesmo se entre estas atualizações uma outra submissão tenha ocorrido.

Arquivo Já Existe Localmente

Algumas vezes quando você tenta atualizar, a atualização falha com uma mensagem dizendo que já existe um arquivo local de mesmo nome. Isto típicamente acontece quando o Subversion tenta baixar um arquivo que acabou de ser versionado, e desobre que um arquivo não versionado de mesmo nome já existe na sua pasta de trabalho. O Subversion nunca irá sobrescrever um arquivo não versionado - ele pode conter algo que você está trabalhando, que conhecidentemente tem o mesmo nome que outro desenvolvedor usou para seu novo arquivo submetido.

Se você receber esta mensagem de erro, a solução é simplesmente renomear o arquivo local não versionado. Após completar a atualização, você pode verificar se o arquivo renomeado ainda é necessário.

> Se você ainda está recebendo mensagens de erro, use TortoiseSVN → Verificar por Modificações ao invés de listar todos os arquivos com problema. Desta maneira você poderá lidar com todos eles ao mesmo tempo.

4.6. Resolvendo Conflitos

Em algum momento você poderá se deparar com um *conflito* quando for atualizar/unificar seus arquivos desde o repositório ou quando você for modificar sua cópia local para uma URL diferente. Existem dois tipos de conflitos:

conflitos de arquivo
: Um conflito de arquivo ocorre se dois (ou mais) desenvolvedores modificam as mesmas linhas de um arquivo.

conflitos de estrutura
: Um conflito de estrutura ocorre quando um desenvolvedor move/renomeia/elimina um arquivo ou uma pasta que outro desenvolvedor tenha também movido/renomeado/eliminado ou ainda apenas modificado.

4.6.1. Conflitos de Arquivo

A file conflict occurs when two or more developers have changed the same few lines of a file. As Subversion knows nothing of your project, it leaves resolving the conflicts to the developers. Whenever a conflict is reported, you should open the file in question, and search for lines starting with the string `<<<<<<<`. The conflicting area is marked like this:

```
<<<<<<< filename
 your changes
=======
 code merged from repository
>>>>>>> revision
```

Also, for every conflicted file Subversion places three additional files in your directory:

filename.ext.mine
: Este é seu arquivo tal como existia em sua cópia local antes de tê-la atualizado - ou seja, sem indicações de conflito. Este arquivo contém as modificações mais recentes e nada mais.

filename.ext.rREVANT
: Este é o arquivo que serviu de revisão BASE antes de atualizar sua cópia local. Ou seja, é o arquivo que você assinalou para controlar antes de efetuar as modificações.

filename.ext.rNOVAREV
: Este é o arquivo que o cliente Subversion acabou de receber do servidor quando você atualizou sua cópia local. Este arquivo corresponde à revisão HEAD do repositório.

Você pode tanto inicializar a ferramenta de inclusão externa / editor de conflitos com TortoiseSVN → Editar Conflitos como também pode utilizar qualquer outro editor para solucionar o conflito manualmente. Você deve decidir como o código deve permanecer, efetuar as modificações necessárias e gravar o arquivo.

Em seguida execute o comando TortoiseSVN → Resolvido e submeta suas modificações no repositório. Por favor, note que o comando Resolver não resolve o conflito. Ele apenas remove os arquivos `filename.ext.mine` e `filename.ext.r*`, para permitir que você submeta suas modificações.

Se tiver conflitos com arquivos binários, Subversion não irá incluir esses arquivos (propriamente dito). A cópia local permanece inalterada (exatamente como a modificação mais recente) e você terá arquivos

`filename.ext.r*`. Se desejar descartar as modificações e manter a versão do repositório, apenas utilize o comando Reverter. Se desejar manter sua versão e sobreescrever o repositório, utilize o comando Resolvido e submeta sua versão.

Você pode utilizar o comando Resolvido para múltiplos arquivos se clicar com o botão direito do mouse na pasta que contém os arquivos e selecionar TortoiseSVN → Resolvido... Isto fará aparecer uma caixa de diálogo de todos os arquivos em conflito naquela pasta e então poderá selecionar quais deverão ser assinalados como resolvido.

4.6.2. Conflitos de Estrutura

Um conflito de estrutura ocorre quando um desenvolvedor move/renomeia/elimina um arquivo ou uma pasta que outro desenvolvedor tenha também movido/renomeado/eliminado ou ainda apenas modificado. Existem diversas situações que podem resultar num conflito de estrutura e cada situação exigirá uma ação diferente para a solução do conflito.

Quando um arquivo é eliminado localmente na Subversão, o arquivo também é eliminado do sistema local de arquivos ou seja, mesmo que seja parte do conflito de estrutura isso não comprova ser uma sobreposição de conflito e você não poderá clicar nele com o botão direito do mouse, para resolver o conflito. Utilize a caixa de diálogo Verificar por Modificações ao invés de acessar a opção Editar conflitos.

TortoiseSVN pode auxiliar para encontrar o lugar correto para aplicar modificações mas poderão existir tarefas adicionais e requeridas para a solução de conflitos. Lembre-se que após uma atualização, a BASE de trabalho sempre conterá a revisão de cada item tal como era no repositório no momento da atualização. Se você reverter uma modificação após uma atualização, o repositório não retornará ao estado original, tal como era antes das modificações que você iniciou localmente.

4.6.2.1. Exclusão local, alteração na atualização

1. Desenvolvedor A modifica `Foo.c` e o submete para o repositório
2. Desenvolvedor B simultaneamente moveu `Foo.c` para `Bar.c` em sua cópia local ou simplesmente eliminou `Foo.c` ou a pasta que o continha.

Uma atualização da cópia de trabalho do desenvolvedor B resulta num conflito de estrutura:

- `Foo.c` foi eliminado da cópia de trabalho mas está marcado com um conflito de estrutura.
- Se o conflito resulta por ter renomeado ao invés de ter eliminado então `Bar.c` estará marcado como adicionado mas não irá conter as modificações do desenvolvedor A.

Desenvolvedor B agora deverá escolher se mantém ou não as modificações do desenvolvedor A. No caso de um arquivo ter sido renomeado, ele poderá submeter as modificações de `Foo.c` para o arquivo renomeado `Bar.c`. No caso de simples arquivo ou pasta deletada, ele poderá escolher entre manter o item com as modificações do desenvolvedor A e descartar o arquivo eliminado. Ou então, poderá marcar o conflito como resolvido sem tomar qualquer atitude e efeticamente irá descartar as modificações do desenvolvedor A.

A caixa de diálogo Editar Conflito possibilita submeter modificações se for possível encontrar o arquivo original do renomeado `Bar.c`. Dependendo de onde a atualização ocorreu, poderá não ser possível encontrar o arquivo fonte.

4.6.2.2. Alteração local, exclusão na atualização

1. Desenvolvedor A move `Foo.c` para `Bar.c` e submete para o repositório.
2. Desenvolvedor B modifica `Foo.c` na sua cópia de trabalho.

Ou no caso de um diretório movido ...

1. Desenvolvedor A move a pasta que o continha de `FooFolder` para `BarFolder` e submete para o repositório.

2. Desenvolvedor B modifica `Foo.c` na sua cópia de trabalho.

Uma atualização na cópia de trabalho do desenvolvedor B resulta num conflito de estrutura. Para um simples conflito de arquivo:

- `Bar.c` é adicionado na cópia de trabalho como um arquivo normal.
- `Foo.c` é assinalado como adicionado (com histórico) e indica um conflito de estrutura.

Para um conflito de pasta:

- `BarFolder` é adicionada para a cópia de trabalho como uma pasta normal.
- `FooFolder` é assinalada como adicionada (com histórico) e indica um conflito de estrutura.

  `Foo.c` é assinalado como modificado.

Desenvolvedor B agora terá que decidir o que fazer com a reorganização do desenvolvedor A e submeter suas modificações para o arquivo correspondente na nova estrutura ou simplesmente reverter as modificações do desenvolvedor A e manter o arquivo local.

Para submeter suas modificações locais com o RESHUFFLE, o desenvolvedor B primeiro precisará saber para qual nome o arquivo em conflito `Foo.c` foi renomeado ou movido no repositório. Isto pode ser feito fazendo uso da caixa de diálogo do Log. As modificações poderão ser então submetidas manualmente - uma vez que não há maneira de automatizar ou simplificar esse processo. Uma vez que as modificações tenham sido PORTED ACROSS, o PATH em conflito será redundante e poderá ser eliminado. Neste caso, utilize o botão Remover da caixa de diálogo do Editor de Conflitos para limpar e assinalar o conflito como resolvido.

Se o Desenvolvedor B decide que as modificações do Desenvolvedor A estão erradas então ele deverá escolher o botão Manter na caixa de diálogo do Editor de Conflitos. Desta forma, o conflito no arquivo ou pasta será assinalado como resolvido mas as modificações do Desenvolvedor A precisarão ser removidas manualmente. Mais uma vez, a caixa de diálogo do Log auxilia para identificar o que foi movido.

4.6.2.3. Exclusão local, exclusão na atualização

1. Desenvolvedor A move o arquivo `Foo.c` para `Bar.c` e submete para o repositório.
2. Desenvolvedor B move o arquivo `Foo.c` para `Bix.c`

Uma atualização da cópia de trabalho do desenvolvedor B resulta num conflito de estrutura:

- `Bix.c` é marcado como adicionado com histórico.
- `Bar.c` é adicionado na cópia local com status "normal".
- `Foo.c` é marcado como eliminado e possui um conflito de estrutura.

Para resolver este conflito, Desenvolvedor B terá que descobrir para qual nome o arquivo em conflito `Foo.c` foi renomeado/movido no repositório. Isto pode ser feito usando a caixa de diálogo do log.

Em seguida, o Desenvolvedor B terá que decidir se o novo nome do arquivo `Foo.c` será mantido - se aquele dado pelo Desenvolvedor A ou se aquele renomeado por ele mesmo.

Depois que o Desenvolvedor B tenha manualmente solucionado o conflito, o conflito de estrutura deve ser marcado como "resolvido", usando o botão apropriado na caixa de diálogo do Editor de Conflitos.

4.6.2.4. Inexistência local, alteração na atualização

1. Desenvolvedor A trabalhando no trunk, modifica o arquivo `Foo.c` e submete o arquivo para o repositório
2. Desenvolvedor B trabalhando com o branch, move e renomeia o arquivo `Foo.c` para `Bar.c` e submete a alteração para o repositório

A atualização do Desenvolvedor A no trunk e a do Desenvolvedor B no branch, resulta num conflito de estrutura:

- `Bar.c` já está na cópia de trabalho com status "norma'".
- `Foo.c` está assinalado como "desaparecido" e com um conflito de estrutura.

Para solucionar este conflito, o Desenvolvedor B terá que marcar esse arquivo como "resolvido" na caixa de diálogo do Editor de Conflitos - o qual irá removê-lo da lista de conflitos. O Desenvolvedor B terá então que decidir se deve copiar o arquivo `Foo.c` "desaparecido" do repositório para a cópia de trabalho ou então se deve atualizar as modificações do Desenvolvedor A no arquivo `Foo.c` para o arquivo renomeado como `Bar.c` ou então se deve ignorar as modificações marcando o conflito como resolvido e fazer nada mais.

Note que se você copiar o arquivo "desaparecido" do repositório e então marcá-lo como resolvido, esse arquivo copiado será removido novamente. Você deve primeiro, resolver o conflito.

4.6.2.5. Alteração local, exclusão na unificação

1. Desenvolvedor A trabalhando no trunk move o arquivo `Foo.c` para `Bar.c` e submete o arquivo para o repositório
2. Desenvolvedor B trabalhando no branch modifica o arquivo `Foo.c` e o submete para o repositório.

Existe uma situação equivalente para a movimentação de pastas, mas isto ainda não é detectado pela Subversão 1.6...

1. Desenvolvedor A trabalhando no trunk move a pasta `FooFolder` para `BarFolder` e submete a alteração para o repositório.
2. Desenvolvedor B trabalhando no branch modifica o arquivo `Foo.c` em sua cópia de trabalho.

A atualização do Desenvolvedor A no trunk e a do Desenvolvedor B no branch, resulta num conflito de estrutura:

- `Bar.c` é marcado como adicionado.
- `Foo.c` é marcado como modificado com um conflito de estrutura.

Desenvolvedor B agora terá que decidir o que fazer com a reorganização do desenvolvedor A e submeter suas modificações para o arquivo correspondente na nova estrutura ou simplesmente reverter as modificações do desenvolvedor A e manter o arquivo local.

Para submeter suas modificações locais com o RESHUFFLE, o Desenvolvedor B primeiro precisará saber para qual nome o arquivo em conflito `Foo.c` foi renomeado ou movido no repositório. Isto pode ser feito fazendo uso da caixa de diálogo do Log. O Editor de Conflitos somente exibirá o log da cópia de trabalho, uma vez que ele não sabe qual PATH foi usado na submissão - por isso você deverá descobrir este detalhe. As modificações deverão ser submetidas manualmente uma vez que não há maneira de automatizar ou mesmo simplificar esse processo. Uma vez que as alterações tenham sido realizadas, o PATH em conflito é redundante e poderá ser eliminado. Neste caso, utilize o botão Remover da caixa de diálogo do Editor de Conflitos para limpar e marcar o conflito como resolvido.

Se o Desenvolvedor B decide que as modificações do Desenvolvedor A estão erradas então ele deverá escolher o botão Manter na caixa de diálogo do Editor de Conflitos. Desta forma, o conflito no arquivo ou pasta será assinalado como resolvido mas as modificações do Desenvolvedor A precisarão ser removidas manualmente. Mais uma vez, a caixa de diálogo de log do MERGE SOURCE auxilia para identificar o que foi movido.

4.6.2.6. Exclusão local, exclusão na unificação

1. Desenvolvedor A trabalhando no trunk move o arquivo `Foo.c` para `Bar.c` e submete o arquivo para o repositório

2. Desenvolvedor B trabalhando com o branch, move e renomeia o arquivo `Foo.c` para `Bix.c` e submete a alteração para o repositório

A atualização do Desenvolvedor A no trunk e a do Desenvolvedor B no branch, resulta num conflito de estrutura:

- `Bix.c` é marcado com Status de normal (não modificado).
- `Bar.c` é marcado como adicionado com histórico.
- `Foo.c` é marcado como inexistente e como tendo um conflito de estrutura.

Para resolver este conflito, Desenvolvedor B terá que descobrir para qual nome o arquivo em conflito `Foo.c` foi renomeado/movido no repositório. Isto pode ser feito usando a caixa de diálogo do MERGE SOURCE. O Editor de Conflitos apenas exibirá o log da cópia de trabalho, uma vez que não sabe qual PATH foi usado na submissão e portanto você deverá descobrir esse detalhe.

Em seguida, o Desenvolvedor B terá que decidir se o novo nome do arquivo `Foo.c` será mantido - se aquele dado pelo Desenvolvedor A ou se aquele renomeado por ele mesmo.

Depois que o Desenvolvedor B tenha manualmente solucionado o conflito, o conflito de estrutura deve ser marcado como "resolvido", usando o botão apropriado na caixa de diálogo do Editor de Conflitos.

4.7. Obtendo Informações de Estado

Enquanto você estiver trabalhando na sua cópia de trabalho você frequentemente precisa saber quais arquivos foram alterados/adicionados/removidos ou renomeados, ou ainda quais arquivos foram alterados por outros.

4.7.1. Sobreposição dos Ícones

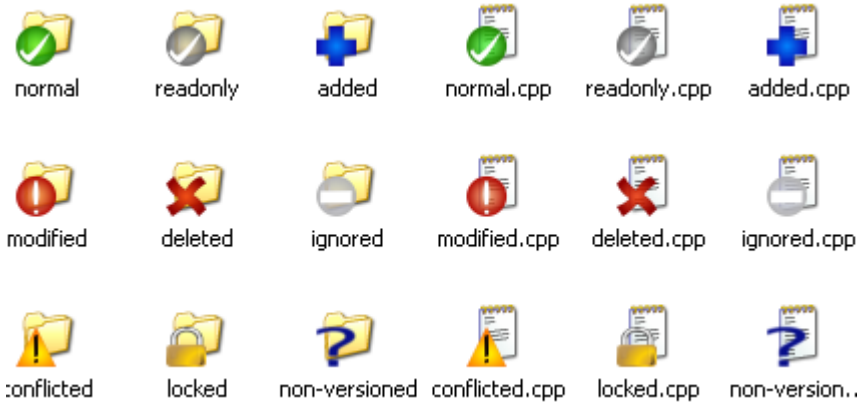


Figura 4.11. Explorer mostra os ícones sobrepostos

Agora que você obteve uma cópia de trabalho do repositório Subversion você pode ver seus arquivos no windows explorer com ícones modificados. Esta é uma das razões para que o TortoiseSVN seja tão popular. O TortoiseSVN adiciona um ícone de revestimento sobre o ícone de cada arquivo. Dependendo do status do arquivo no Subversion o revestimento é diferente.

Uma cópia de trabalho recém obtida possui uma camada contendo uma marca verde. Isto significa que o estado é *normal*.

Assim que você começar a editar um arquivo, o estado muda para *modificado* e a camada de revestimento então muda para um ponto de exclamação vermelho. Dessa maneira você pode facilmente ver quais arquivos foram alterados desde que você atualizou sua cópia de trabalho e que precisam ser submetidos.

Se durante uma atualização ocorrer um *conflito* então o ícone muda para um ponto de exclamação amarelo.

Se você marcou a propriedade `svn:needs-lock` em um arquivo, o Subversion faz com que aquele arquivo se torne somente-leitura até que você obtenha uma trava naquele arquivo. Tais arquivos tem este revestimento para indicar que você precisa obter uma trava antes que você possa editar aquele arquivo.

Se você mantém uma trava em um arquivo, e o estado do Subversion é *normal*, o revestimento do ícone lembra você de que você deve liberar a trava se não estiver usando-o para permitir que outros possam submeter suas mudanças para quele arquivo.

Este ícone mostra a você que alguns arquivos ou pastas dentro da pasta atual foram agendados para serem *deletados* do controle de versionamento ou um arquivo dentro do controle de versionamento está faltando em uma pasta.

O sinal de mais significa que um arquivo ou pasta foi agendado para ser *addicionado* ao controle de versões.

O sinal de barra lhe diz que um arquivo ou pasta é *ignorado* do controle de versões. Esse revestimento é opcional.

Este ícone mostra arquivos e pastas que não estão sob o controle de versionamento, mas não foram ignorados. Este revestimento é opcional.

De fato, você pode descobrir que nem todos estes ícones são usados em seu sistema. Isto é porque o número de camadas permitidas pelo Windows é bem limitado e se você também estiver usando uma versão antiga do TortoiseCVS, então não haverá espaço suficiente. O TortoiseSVN tenta ser um “Good Citizen (TM) - Bom Cidadão” e limita seu uso de camadas para dar outros aplicativos uma chance também.

Agora que existem mais clientes Tortoise (TortoiseCVS, TortoiseHG, ...) o limite de ícones se torna um problema real. Para contornar este problema, o projeto TortoiseSVN introduziu um conjunto compartilhado comum de ícones, carregado como uma DLL, que pode ser usada por todos os clientes Tortoise. Verifique com o fornecedor do seu cliente para verificar se isto já foi integrado à ele :-)

Para uma descrição de como as camadas de ícones correspondem ao estado do Subversion e outros detalhes técnicos, leia Seção F.1, “Sobreposição dos Ícones”.

4.7.2. Colunas do TortoiseSVN no Windows Explorer

A mesma informação que está disponível das camadas de ícones (e muito mais) pode ser exibida como colunas adicionais na Visualização em Detalhes do Windows Explorer.

Simplesmente clique com o botão direito em um dos títulos de uma coluna, e escolha Mais... do menu de contexto mostrado. Uma janela aparecerá onde você pode especificar as colunas e sua ordem, que é mostrada na "Visualização em Detalhes". Role para baixo até que as entradas começando com SVN comecem a aparecer. Marque aquelas que você gostaria de ver e feche a janela pressionando OK. As colunas serão concatenadas à direita daquelas já exibidas. Você pode reordená-las arrastando e soltando-as, ou redimensioná-las, para que encaixem às suas necessidades.

Importante

As colunas adicionais no Windows Explorer não estão disponíveis no Vista, já que a Microsoft decidiu não permitir tais colunas para *todos* os arquivos mas somente para tipos de arquivo específicos.

Dica

Se você deseja que o layout atual seja exibido em todas suas cópias de trabalho, você pode querer definir este como a visualização padrão.

4.7.3. Estado Local e Remoto

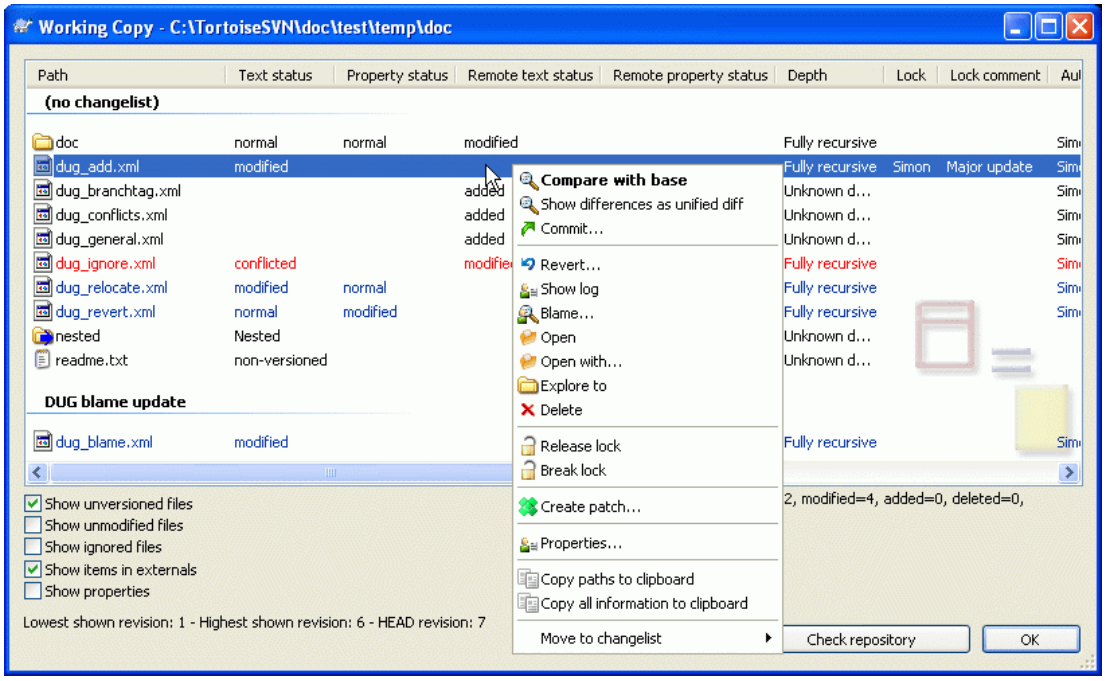


Figura 4.12. Procurar por Modificações

É frequentemente muito útil saber quais arquivos você alterou e também quais arquivos foram alterados e submetidos por outros. É aí que o comando TortoiseSVN → Verificar por Modificações... se torna útil. Esta janela lhe mostrará cada arquivo que foi alterado de qualquer maneira dentro da sua cópia de tabalho, assim como quaisquer arquivos não-versionados que você possa possuir.

Se você clicar no botão Verificar Repositório então você também pode olhar as mudanças no repositório. Dessa maneira você pode checar antes de uma atualização se pode haver um possível conflito. Você também pode atualizar arquivos selecionados do repositório sem atualizar toda a pasta. Por padrão, o botão Verificar Repositório somente busca o estado remoto com a profundidade de obtenção da sua cópia de trabalho. Se você deseja ver todos os arquivos e pastas no repositório, mesmo aquelas que você não obteve, então você deve manter pressionada a tecla **Shift** quando clicar no botão Verificar Repositório.

A janela usa codificação de cores para demarcar o estado.

Azul
: Itens modificados localmente.

Roxo
: Itens adicionados. Itens que foram adicionados com histórico tem um sinal + na coluna de Estado de Texto, e uma janela mostra de onde o item foi copiado.

Vermelho escuro
: Itens excluídos ou perdidos.

Verde
: Itens modificados localmente e no repositório. As mudanças serão juntadas ao atualizar. Isto *pode* produzir conflitos ao atualizar.

Vermelho vivo
: Itens modificados localmente e apagados no repositório, ou modificados no repositório e apagados localmente. Isto *irá* produzir conflitos durante a atualização.

Preto
: Itens não modificados ou não controlados.

This is the default colour scheme, but you can customise those colours using the settings dialog. Read Seção 4.30.1.4, “TortoiseSVN Colour Settings” for more information.

Items que foram movidos para um caminho de repositório diferente são também indicados usando o marcador `(s)`. Você pode ter movido algo enquanto trabalhava em uma ramificação (branch) e se esqueceu de mover de volta para a versão principal (trunk). Esté é seu sinal de aviso!

Do menu de contexto da janela você pode ver um diff de mudanças. Marque as mudanças localis que *você* fez usando Menu de Contexto → Comparar com Base. Marque as mudanças no repositório feitas por outros usando Menu de Contexto → Mostrar Diferenças como Diff Unificado.

Você também pode reverter as mudanças em arquivos individuais. Se você apagou um arquivo acidentalmente, ele aparecerá como *Faltando* e você pode usar *Reverter* para recuperá-lo.

Arquivos não-versionados e ignorados podem ser enviados para a lixeira daqui usando Menu de Contexto → Apagar. Se você deseja apagar arquivos permanentemente (ignorando a lixeira) matenha pressionada a tecla **Shift** enquanto cliclar em Apagar.

Se você deseja examinar o arquivo em detalhes, você pode arrastá-lo daqui para um outro aplicativo como um editor de texto ou uma IDE.

As colunas são personalizáveis. Se você clicar com o botão direito em qualquer cabeçalho de coluna você verá um menu de contexto permitindo-o selecionar quais colunas serão exibidas. Você pode também alterar a largura das colunas usando o ícone de arrastar que aparece quando você passa o mouse em cima de uma borda de coluna. Essas personalizações são preservadas, então você verá os mesmos cabeçalhos da próxima vez.

Se você está trabalhando em diversas tarefas independentes ao mesmo tempo, você pode também agrupar arquivos em changelists. Leia Seção 4.4.2, “Lista de Alterações” para maiores informações.

Na parte inferior da janela você pode ver um sumário da variação das revisões do repositório em uso na sua cópia de trabalho. Essas são as revisões *submetidas*, não as revisões *atualizadas*; elas representam a variação das revisões onde estes arquivos foram submetidos pela última vez, não as revisões nas quais foram atualizados. Note que a variação de revisões mostrada aplica-se somente aos itens exibidos, não à cópia de trabalho inteira. Se você deseja ver informações sobre a cópia de trabalho inteira você deve marcar a caixa Mostrar arquivos não modificados.

Dica

Se você deseja uma exibição simples da sua cópia de tabalho, p. ex. mostrando todos os arquivos e pastas em cada nível da hierarquia de pastas, então a janela Verificar por Modificações é a maneira mais fácil de conseguir isso. Simplesmente marque a caixa Mostrar arquivos não modificados para mostrar todos os arquivos na sua cópia de trabalho.

Reparando Referência Externas Renomeadas

Sometimes files get renamed outside of Subversion, and they show up in the file list as a missing file and an unversioned file. To avoid losing the history you need to notify Subversion about the connection. Simply select both the old name (missing) and the new name (unversioned) and use Context Menu → Repair Move to pair the two files as a rename.

4.7.4. Visualizar diferenças

Frequentemente você deseja olhar dentro de seus arquivos, para observar o que foi alterado. Você pode fazê-lo selecionando o arquivo que foi alterado, e selecionando Diff do menu de contexto do TortoiseSVN. Isto inicia o visualizador-diff externo, que irá então comparar o arquivo atual com a cópia pristina (revisão BASE), que foi armazenada após a última submissão ou atualização.

Dica

Mesmo quando não não se está dentro de uma cópia de trabalho ou quando você tiver múltiplas versões do arquivo em disco, você ainda pode visualizar diffs:

Selecione os dois arquivos que você deseja comparar no explorer (p. ex. usando **Ctrl** e o mouse) e escolha Diff do menu de contexto do TortoiseSVN. O arquivo clicado por último (o que possui foco, ou seja, um retângulo pontilhado) será compreendido como o último.

4.8. Lista de Alterações

Em um mundo ideal, você sempre estará tabalhando em uma coisa de cada vez, e sua cópia de trabalho contém apenas um conjunto de alterações lógicas. OK, devolta a realidade. Frequentemente acontece que você precisa trabalhar em diversas tarefas não relacionadas ao mesmo tempo, e quando você olhar na janela de submissão, todas as mudanças estão misturadas. A funcionalidade das *changelists* ajuda-o a agrupar os arquivos, tornando mais fácil de ver o que você está fazendo. É claro que isto só pode acontecer se as mudanças não se sobrepuserem. Se duas tarefas diferentes afetarem o mesmo arquivo, não há outra maneira de separar as mudanças.

Importante

A funcionalidade das listas de alterações no TortoiseSVN está apenas disponível no Windows XP e posteriores, porque depende de uma capacidade da interface do Windows

> que não está presente no Windows 2000. Desculpe-nos, mas o Win2K é muito antigo hoje em dia, então por favor não reclame.

Você pode ver listas de alterações em diversos lugares, mas os mais importantes são na janela de submissão e na janela de verificação-de-modificações. Vamos começar na janela de verificação-de-modificações após você tiver trabalhado em diversas funcionalidades e muitos arquivos. Quando você abrir esta janela pela primeira vez, todos os arquivos alterados estarão listados juntos. Suponha que agora você queira organizar as coisas e agrupar os arquivos de acordo com a funcionalidade.

Selecione um ou mais arquivos e use Menu de Contexto → Mover para a changelist para adicionar um item à lista de alterações. Inicialmente não haverão listas de alterações, então a primeira vez que você fizer isto você criará uma nova changelist. Dê o nome que melhor descreve para que à está usando, e clique OK. A janela irá agora mudar para mostrar os grupos de itens.

Uma vez que você tiver criado a lista de alterações você pode arrastar e soltar itens nela, tanto de outra lista de alterações, como do Windows Explorer. Arrastar do Explorer pode ser útil já que permite-o adicionar itens para a lista de modificações antes que o arquivo seja modificado. Você pode fazê-lo da janela de verificar-por-modificações, mas somente através da exibição de todos os arquivos não modificados.

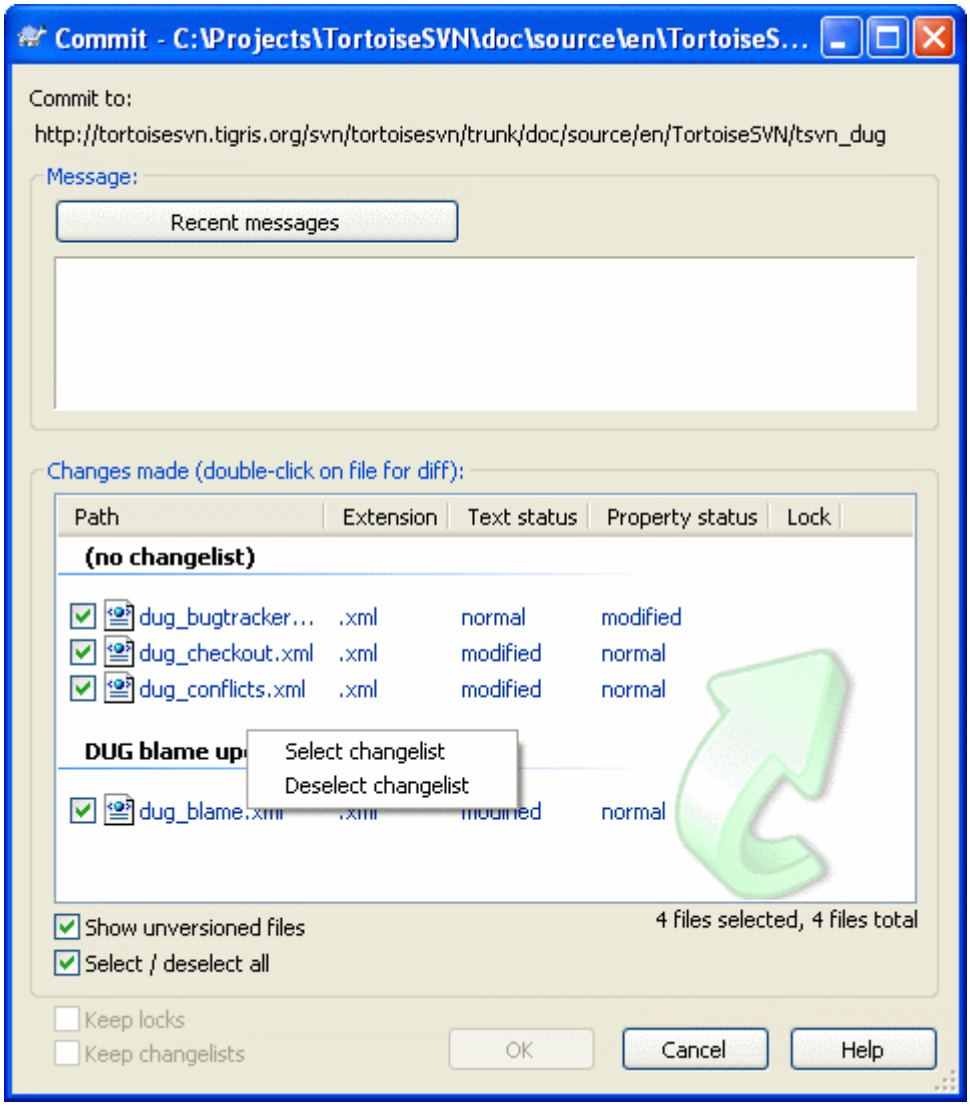


Figura 4.13. Janela de Submissão com Lista de Alterações

In the commit dialog you can see those same files, grouped by changelist. Apart from giving an immediate visual indication of groupings, you can also use the group headings to select which files to commit.

No XP, existe um menu de contexto quando você clica com o botão direito em um grupo de títulos que lhe fornece uma escolha para marcar ou desmarcar todas as entradas desse grupo. Entratanto no Vista o menu de contexto não é necessário. Clique no grupo de títulos para selecionar todas as entradas, então marque uma das entradas selecionadas para marcar todas.

O TortoiseSVN reserva um nome de lista de alterações para uso pessoal, chamado de `ignore-on-commit`. Isto é usado para marcar arquivos versionados que você geralmente não deseja submeter mesmo que tenham mudanças locais. Esta funcionalidade está desrita em Seção 4.4.3, “Excluindo Itens de uma Lista de Submissões”.

Quando você submete arquivos pertencendo à uma lista de mudanças então normalmente você espera que a pertinência da lista de mudanças não seja mais necessária. Então por padrão, arquivos são removidos das listas automaticamente na submissão. Se você deseja reter o arquivo em sua lista, marque a caixa Guardar changelists na parte inferior da janela de submissão.

Dica

Listas de Alterações são puramente uma funcionalidade local do cliente. Criar e remover Listas de Alterações não afetará o repositório, nem a cópia de tabalho de mais ninguém. Elas são simplesmente uma maneira conveniente para vocÇê organizar seus arquivos.

4.9. Janela de Revisão de Registro

For every change you make and commit, you should provide a log message for that change. That way you can later find out what changes you made and why, and you have a detailed log for your development process.

The Revision Log Dialog retrieves all those log messages and shows them to you. The display is divided into 3 panes.

- The top pane shows a list of revisions where changes to the file/folder have been committed. This summary includes the date and time, the person who committed the revision and the start of the log message.

  Lines shown in blue indicate that something has been copied to this development line (perhaps from a branch).

- The middle pane shows the full log message for the selected revision.

- The bottom pane shows a list of all files and folders that were changed as part of the selected revision.

But it does much more than that - it provides context menu commands which you can use to get even more information about the project history.

4.9.1. Invocando a Janela de Histórico de Revisão

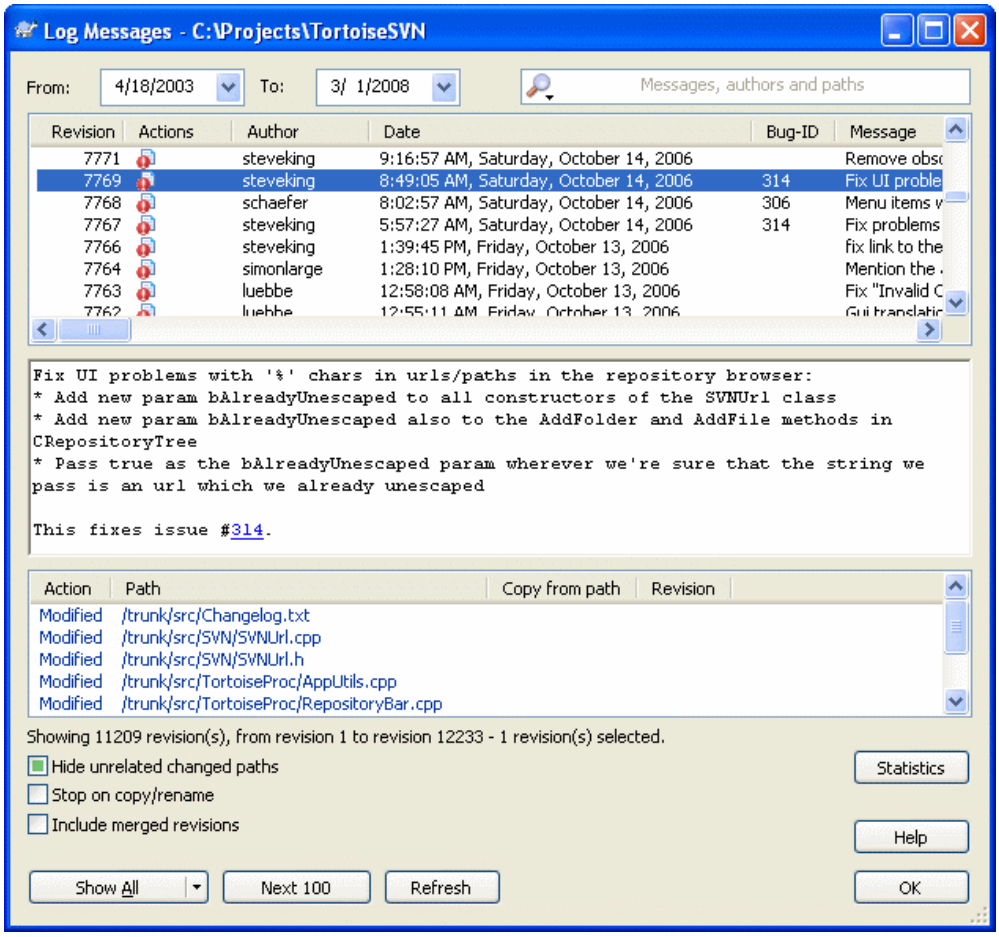


Figura 4.14. A Janela de Histórico de Revisão

There are several places from where you can show the Log dialog:

- Do TortoiseSVN contexto do submenu
- Da página de propriedades
- From the Progress dialog after an update has finished. Then the Log dialog only shows those revisions which were changed since your last update

If the repository is unavailable you will see the Want to go offline? dialog, described in Seção 4.9.10, “Modo desconectado”.

4.9.2. Histórico de Ações de Revisão

The top pane has an Actions column containing icons that summarize what has been done in that revision. There are four different icons, each shown in its own column.

If a revision modified a file or directory, the *modified* icon is shown in the first column.

If a revision added a file or directory, the *added* icon is shown in the second column.

If a revision deleted a file or directory, the *deleted* icon is shown in the third column.

If a revision replaced a file or directory, the *replaced* icon is shown in the fourth column.

4.9.3. Recuperando Informações Adicionais

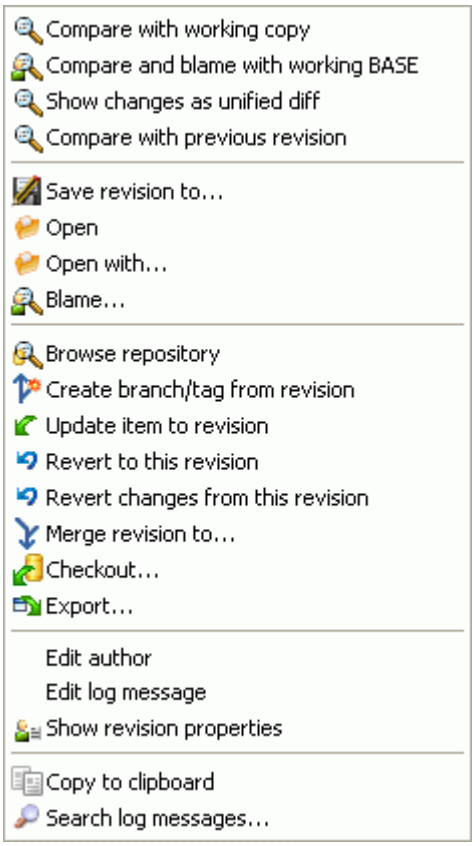


Figura 4.15. The Revision Log Dialog Top Pane with Context Menu

The top pane of the Log dialog has a context menu that allows you to access much more information. Some of these menu entries appear only when the log is shown for a file, and some only when the log is shown for a folder.

Comparar com a cópia de trabalho
: Compare the selected revision with your working copy. The default Diff-Tool is TortoiseMerge which is supplied with TortoiseSVN. If the log dialog is for a folder, this will show you a list of changed files, and allow you to review the changes made to each file individually.

Compare and blame with working BASE
: Blame the selected revision, and the file in your working BASE and compare the blame reports using a visual diff tool. Read Seção 4.23.2, “Diferenças de Autoria” for more detail. (files only).

Mostrar alterações com as diferenças unificadas
: View the changes made in the selected revision as a Unified-Diff file (GNU patch format). This shows only the differences with a few lines of context. It is harder to read than a visual file compare, but will show all file changes together in a compact format.

Comparar com a revisão anterior
: Compare the selected revision with the previous revision. This works in a similar manner to comparing with your working copy. For folders this option will first show the changed files dialog allowing you to select files to compare.

Compare and blame with previous revision
: Show the changed files dialog allowing you to select files. Blame the selected revision, and the previous revision, and compare the results using a visual diff tool. (folders only).

Save revisão para...
: Save the selected revision to a file so you have an older version of that file. (files only).

Abrir / Abrir com...
: Open the selected file, either with the default viewer for that file type, or with a program you choose. (files only).

Autoria...
: Autorias do arquivo até a revisão selecionada (apenas arquivos).

Navegar no repositório
: Open the repository browser to examine the selected file or folder in the repository as it was at the selected revision.

Criar ramificação/tórulo de uma revisão
: Create a branch or tag from a selected revision. This is useful e.g. if you forgot to create a tag and already committed some changes which weren't supposed to get into that release.

Atualizar item para a revisão
: Update your working copy to the selected revision. Useful if you want to have your working copy reflect a time in the past, or if there have been further commits to the repository and you want to update your working copy one step at a time. It is best to update a whole directory in your working copy, not just one file, otherwise your working copy could be inconsistent.

  If you want to undo an earlier change permanently, use Revert to this revision instead.

Reverter para esta revisão
: Revert to an earlier revision. If you have made several changes, and then decide that you really want to go back to how things were in revision N, this is the command you need. The changes are undone in your working copy so this operation does *not* affect the repository until you commit the changes. Note that this will undo *all* changes made after the selected revision, replacing the file/folder with the earlier version.

  If your working copy is in an unmodified state, after you perform this action your working copy will show as modified. If you already have local changes, this command will merge the *undo* changes into your working copy.

  What is happening internally is that Subversion performs a reverse merge of all the changes made after the selected revision, undoing the effect of those previous commits.

  If after performing this action you decide that you want to *undo the undo* and get your working copy back to its previous unmodified state, you should use TortoiseSVN → Revert from within Windows Explorer, which will discard the local modifications made by this reverse merge action.

  If you simply want to see what a file or folder looked like at an earlier revision, use Update to revision or Save revision as... instead.

Reverter alterações de uma revisão
: Undo changes from which were made in the selected revision. The changes are undone in your working copy so this operation does *not* affect the repository at all! Note that this will undo the changes made in that revision only; it does not replace your working copy with the entire file at the earlier revision. This is very useful for undoing an earlier change when other unrelated changes have been made since.

  If your working copy is in an unmodified state, after you perform this action your working copy will show as modified. If you already have local changes, this command will merge the *undo* changes into your working copy.

What is happening internally is that Subversion performs a reverse merge of that one revision, undoing its effect from a previous commit.

You can *undo the undo* as described above in Revert to this revision.

Unificar revisão para...
: Merge the selected revision(s) into a different working copy. A folder selection dialog allows you to choose the working copy to merge into, but after that there is no confirmation dialog, nor any opportunity to try a test merge. It is a good idea to merge into an unmodified working copy so that you can revert the changes if it doesn't work out! This is a useful feature if you want to merge selected revisions from one branch to another.

Obter...
: Make a fresh checkout of the selected folder at the selected revision. This brings up a dialog for you to confirm the URL and revision, and select a location for the checkout.

Exportar...
: Export the selected file/folder at the selected revision. This brings up a dialog for you to confirm the URL and revision, and select a location for the export.

Editar autor / mensagem de log
: Edit the log message or author attached to a previous commit. Read Seção 4.9.7, “Changing the Log Message and Author” to find out how this works.

Mostrar propriedades da revisão
: View and edit any revision property, not just log message and author. Refer to Seção 4.9.7, “Changing the Log Message and Author”.

Copiar para área de transferência
: Copy the log details of the selected revisions to the clipboard. This will copy the revision number, author, date, log message and the list of changed items for each revision.

Procurar mensagens de log...
: Search log messages for the text you enter. This searches the log messages that you entered and also the action summaries created by Subversion (shown in the bottom pane). The search is not case sensitive.

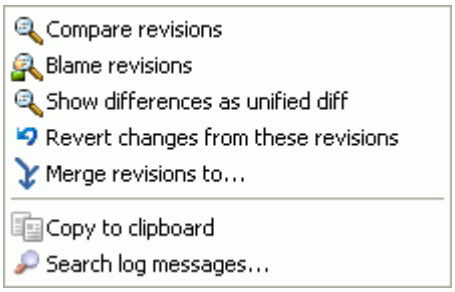


Figura 4.16. Top Pane Context Menu for 2 Selected Revisions

If you select two revisions at once (using the usual **Ctrl**-modifier), the context menu changes and gives you fewer options:

Comparar revisões
: Compare the two selected revisions using a visual difference tool. The default Diff-Tool is TortoiseMerge which is supplied with TortoiseSVN.

If you select this option for a folder, a further dialog pops up listing the changed files and offering you further diff options. Read more about the Compare Revisions dialog in Seção 4.10.3, “Comparando Diretórios”.

Autoria das revisões
: Blame the two revisions and compare the blame reports using a visual difference tool. Read Seção 4.23.2, “Diferenças de Autoria” for more detail.

Mostrar diferenças como diferenças unificadas
: View the differences between the two selected revisions as a Unified-Diff file. This works for files and folders.

Copiar para área de transferência
: Copy log messages to clipboard as described above.

Procurar mensagens de log...
: Search log messages as described above.

If you select two or more revisions (using the usual **Ctrl** or **Shift** modifiers), the context menu will include an entry to Revert all changes which were made in the selected revisions. This is the easiest way to rollback a group of revisions in one go.

You can also choose to merge the selected revisions to another working copy, as described above.

If all selected revisions have the same author, you can edit the author of all those revisions in one go.

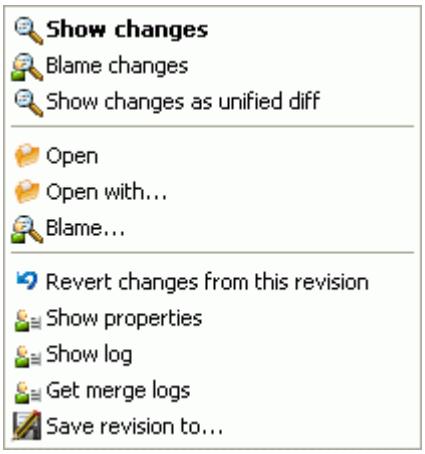


Figura 4.17. The Log Dialog Bottom Pane with Context Menu

The bottom pane of the Log dialog also has a context menu that allows you to

Mostrar alterações
: Show changes made in the selected revision for the selected file. This context menu is only available for files shown as *modified*.

Autoria das alterações
: Blame the selected revision and the previous revision for the selected file, and compare the blame reports using a visual diff tool. Read Seção 4.23.2, “Diferenças de Autoria” for more detail.

Mostra como arquivo de diferenças unificado
: Show file changes in unified diff format. This context menu is only available for files shown as *modified*.

Abrir / Abrir com...
: Open the selected file, either with the default viewer for that file type, or with a program you choose.

Autoria...
: Opens the Blame dialog, allowing you to blame up to the selected revision.

Reverter alterações de uma revisão
: Revert the changes made to the selected file in that revision.

Mostrar propriedades
: View the Subversion properties for the selected item.

Mostrar log
: Show the revision log for the selected single file.

Recuperar log das unificações
: Show the revision log for the selected single file, including merged changes. Find out more in Seção 4.9.6, “Merge Tracking Features”.

Save revisão para...
: Save the selected revision to a file so you have an older version of that file.

Dica

You may notice that sometimes we refer to changes and other times to differences. What's the difference?

Subversion uses revision numbers to mean 2 different things. A revision generally represents the state of the repository at a point in time, but it can also be used to represent the changeset which created that revision, eg. “Done in r1234” means that the changes committed in r1234 implement feature X. To make it clearer which sense is being used, we use two different terms.

If you select two revisions N and M, the context menu will offer to show the *difference* between those two revisions. In Subversion terms this is `diff -r M:N`.

If you select a single revision N, the context menu will offer to show the *changes* made in that revision. In Subversion terms this is `diff -r N-1:N` or `diff -c N`.

The bottom pane shows the files changed in all selected revisions, so the context menu always offers to show *changes*.

4.9.4. Obtendo mais mensagens de log

The Log dialog does not always show all changes ever made for a number of reasons:

- For a large repository there may be hundreds or even thousands of changes and fetching them all could take a long time. Normally you are only interested in the more recent changes. By default, the number of log messages fetched is limited to 100, but you can change this value in TortoiseSVN → Settings (Seção 4.30.1.2, “TortoiseSVN Dialog Settings 1”),
- When the Stop on copy/rename box is checked, Show Log will stop at the point that the selected file or folder was copied from somewhere else within the repository. This can be useful when looking at branches (or tags) as it stops at the root of that branch, and gives a quick indication of changes made in that branch only.

  Normally you will want to leave this option unchecked. TortoiseSVN remembers the state of the checkbox, so it will respect your preference.

  When the Show Log dialog is invoked from within the Merge dialog, the box is always checked by default. This is because merging is most often looking at changes on branches, and going back beyond the root of the branch does not make sense in that instance.

  Note that Subversion currently implements renaming as a copy/delete pair, so renaming a file or folder will also cause the log display to stop if this option is checked.

If you want to see more log messages, click the Next 100 to retrieve the next 100 log messages. You can repeat this as many times as needed.

Next to this button there is a multi-function button which remembers the last option you used it for. Click on the arrow to see the other options offered.

Use Show Range ... if you want to view a specific range of revisions. A dialog will then prompt you to enter the start and end revision.

Use Show All if you want to see *all* log messages from HEAD right back to revision 1.

4.9.5. Revisão da Cópia de Trabalho Atual

Because the log dialog shows you the log from HEAD, not from the current working copy revision, it often happens that there are log messages shown for content which has not yet been updated in your working copy. To help make this clearer, the commit message which corresponds to the revision you have in your working copy is shown in bold.

When you show the log for a folder the revision highlighted is the highest revision found anywhere within that folder, which requires a crawl of the working copy. This can be a slow operation for large working copies, and the log messages are not displayed until the crawl completes. If you want to disable or limit this feature you need to set a registry key `HKCU\Software\TortoiseSVN\RecursiveLogRev` as described in Seção 4.30.10, “Registry Settings”.

4.9.6. Merge Tracking Features

Subversion 1.5 and later keeps a record of merges using properties. This allows us to get a more detailed history of merged changes. For example, if you develop a new feature on a branch and then merge that branch back to trunk, the feature development will show up on the trunk log as a single commit for the merge, even though there may have been 1000 commits during branch development.

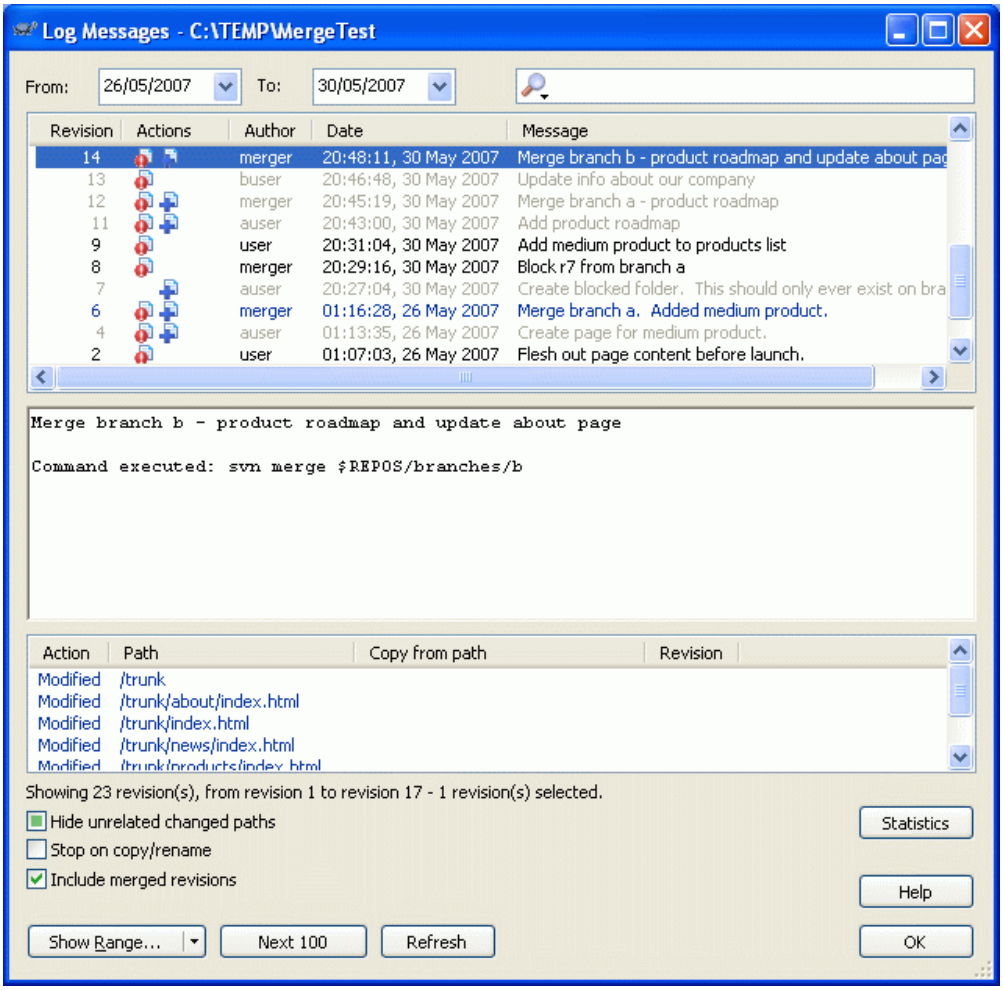


Figura 4.18. The Log Dialog Showing Merge Tracking Revisions

If you want to see the detail of which revisions were merged as part of that commit, use the Include merged revisions checkbox. This will fetch the log messages again, but will also interleave the log messages from revisions which were merged. Merged revisions are shown in grey because they represent changes made on a different part of the tree.

Of course, merging is never simple! During feature development on the branch there will probably be occasional merges back from trunk to keep the branch in sync with the main line code. So the merge history of the branch will also include another layer of merge history. These different layers are shown in the log dialog using indentation levels.

4.9.7. Changing the Log Message and Author

Revision properties are completely different from the Subversion properties of each item. Revprops are descriptive items which are associated with one specific revision number in the repository, such as log message, commit date and committer name (author).

Sometimes you might want to change a log message you once entered, maybe because there's a spelling error in it or you want to improve the message or change it for other reasons. Or you want to change the author of the commit because you forgot to set up authentication or...

Subversion lets you change revision properties any time you want. But since such changes can't be undone (those changes are not versioned) this feature is disabled by default. To make this work, you must set up a pre-revprop-change hook. Please refer to the chapter on *Hook Scripts* [http://svnbook.red-

bean.com/en/1.5/svn.reposadmin.create.html#svn.reposadmin.create.hooks] in the Subversion Book for details about how to do that. Read Seção 3.3, “Rotinas de eventos no servidor” to find some further notes on implementing hooks on a Windows machine.

Once you've set up your server with the required hooks, you can change the author and log message (or any other revprop) of any revision, using the context menu from the top pane of the Log dialog. You can also edit a log message using the context menu for the middle pane.

Atenção

Because Subversion's revision properties are not versioned, making modifications to such a property (for example, the `svn:log` commit message property) will overwrite the previous value of that property *forever*.

4.9.8. Filtrando Mensagens de Log

If you want to restrict the log messages to show only those you are interested in rather than scrolling through a list of hundreds, you can use the filter controls at the top of the Log Dialog. The start and end date controls allow you to restrict the output to a known date range. The search box allows you to show only messages which contain a particular phrase.

Click on the search icon to select which information you want to search in, and to choose *regex* mode. Normally you will only need a simple text search, but if you need to more flexible search terms, you can use regular expressions. If you hover the mouse over the box, a tooltip will give hints on how to use the regex functions. You can also find online documentation and a tutorial at *http://www.regular-expressions.info/*. The filter works by checking whether your filter string matches the log entries, and then only those entries which *match* the filter string are shown.

To make the filter show all log entries that do *not* match the filter string, start the string with an exclamation mark ('!'). For example, a filter string `!username` will only show those entries which were not committed by `username`.

Note that these filters act on the messages already retrieved. They do not control downloading of messages from the repository.

You can also filter the path names in the bottom pane using the Hide unrelated changed paths checkbox. Related paths are those which contain the path used to display the log. If you fetch the log for a folder, that means anything in that folder or below it. For a file it means just that one file. The checkbox is tristate: you can show all paths, grey out the unrelated ones, or hide the unrelated paths completely.

Sometimes your working practices will require log messages to follow a particular format, which means that the text describing the changes is not visible in the abbreviated summary shown in the top pane. The property `tsvn:logsummary` can be used to extract a portion of the log message to be shown in the top pane. Read Seção 4.17.2, “TortoiseSVN Project Properties” to find out how to use this property.

No Log Formatting from Repository Browser

Because the formatting depends upon accessing subversion properties, you will only see the results when using a checked out working copy. Fetching properties remotely is a slow operation, so you will not see this feature in action from the repo browser.

4.9.9. Statistical Information

The Statistics button brings up a box showing some interesting information about the revisions shown in the Log dialog. This shows how many authors have been at work, how many commits they have made,

progress by week, and much more. Now you can see at a glance who has been working hardest and who is slacking ;-)

4.9.9.1. Página de Estatísticas

This page gives you all the numbers you can think of, in particular the period and number of revisions covered, and some min/max/average values.

4.9.9.2. Submissões por Página do Autor

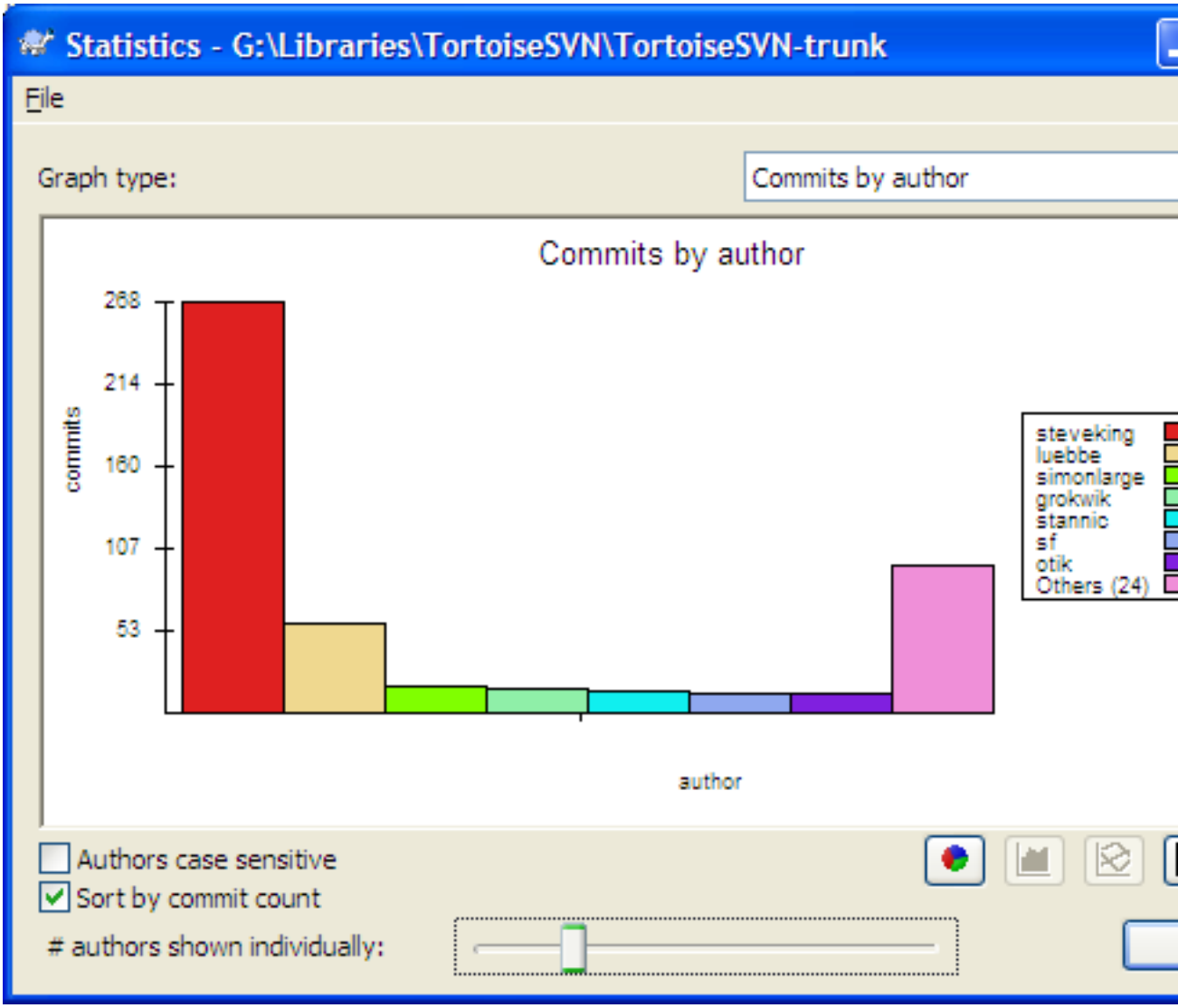


Figura 4.19. Histograma de Submissões-por-autor

This graph shows you which authors have been active on the project as a simple histogram, stacked histogram or pie chart.

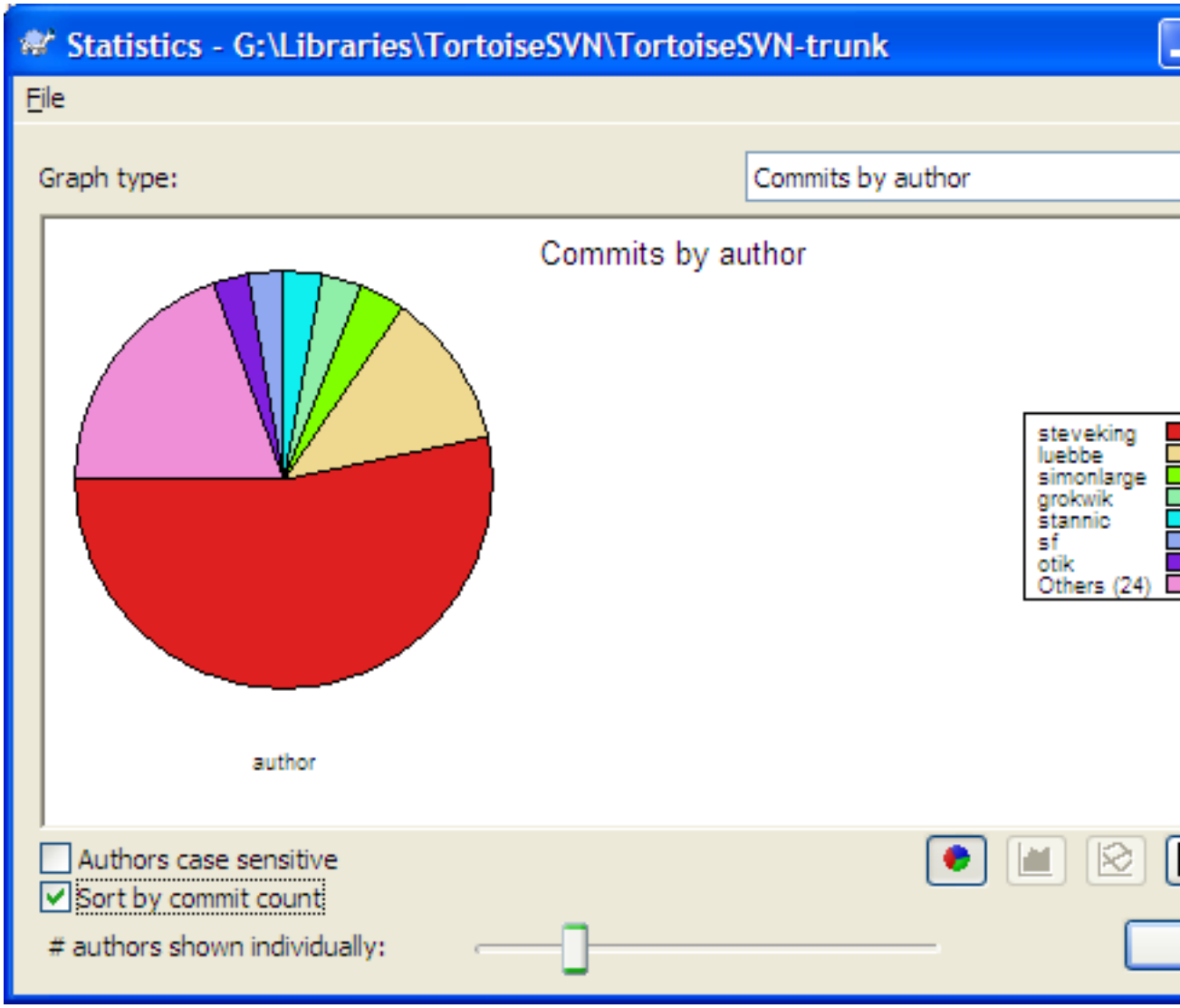


Figura 4.20. Gráfico de Pizza das Submissões-por-Autor

Where there are a few major authors and many minor contributors, the number of tiny segments can make the graph more difficult to read. The slider at the bottom allows you to set a threshold (as a percentage of total commits) below which any activity is grouped into an *Others* category.

4.9.9.3. Submissões por Página de Data

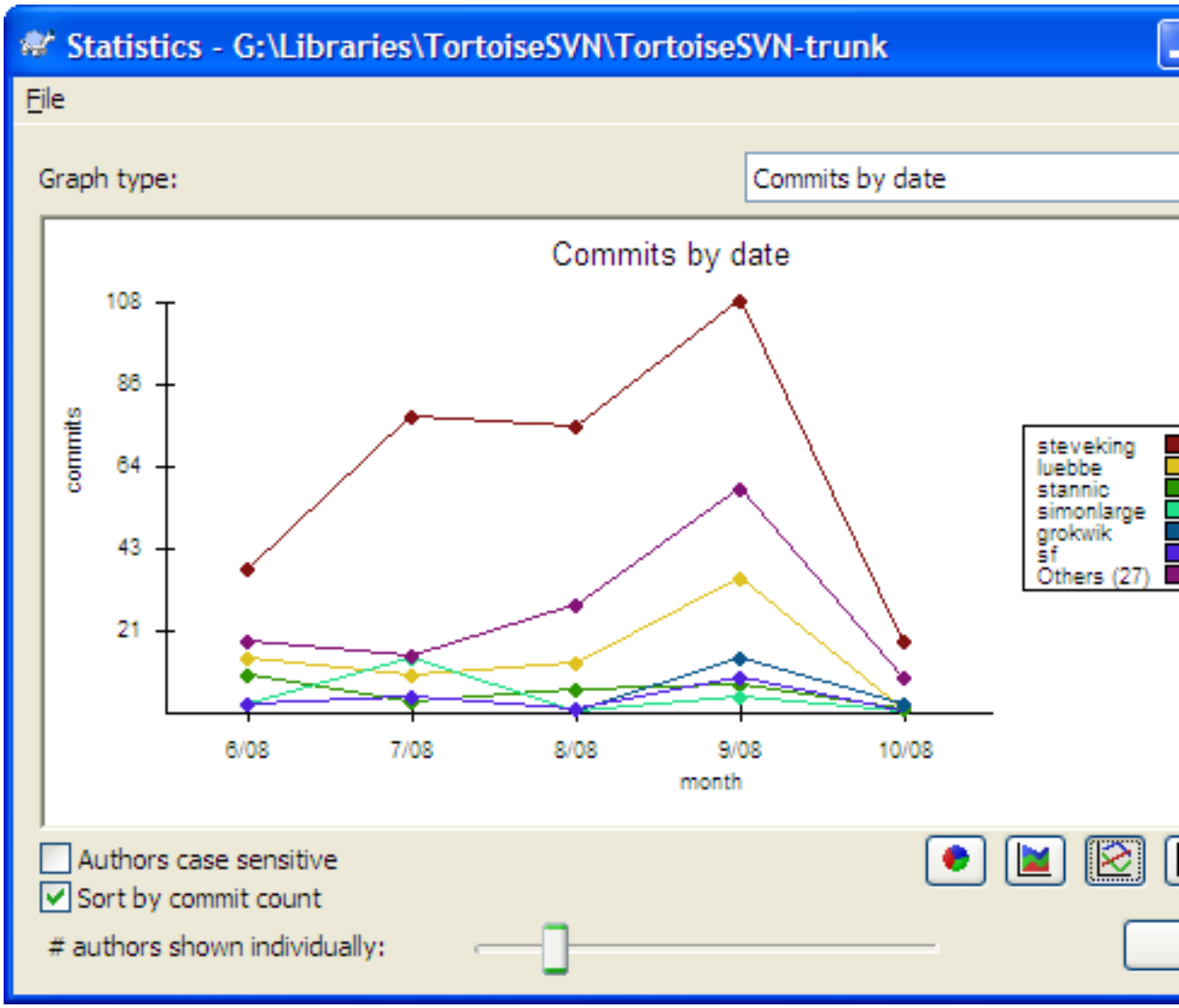


Figura 4.21. Gráfico de Submissões-por-data

This page gives you a graphical representation of project activity in terms of number of commits *and* author. This gives some idea of when a project is being worked on, and who was working at which time.

When there are several authors, you will get many lines on the graph. There are two views available here: *normal*, where each author's activity is relative to the base line, and *stacked*, where each author's activity is relative to the line underneath. The latter option avoids the lines crossing over, which can make the graph easier to read, but less easy to see one author's output.

By default the analysis is case-sensitive, so users `PeterEgan` and `PeteRegan` are treated as different authors. However, in many cases user names are not case-sensitive, and are sometimes entered inconsistently, so you may want `DavidMorgan` and `davidmorgan` to be treated as the same person. Use the Authors case insensitive checkbox to control how this is handled.

Note that the statistics cover the same period as the Log dialog. If that is only displaying one revision then the statistics will not tell you very much.

4.9.10. Modo desconectado

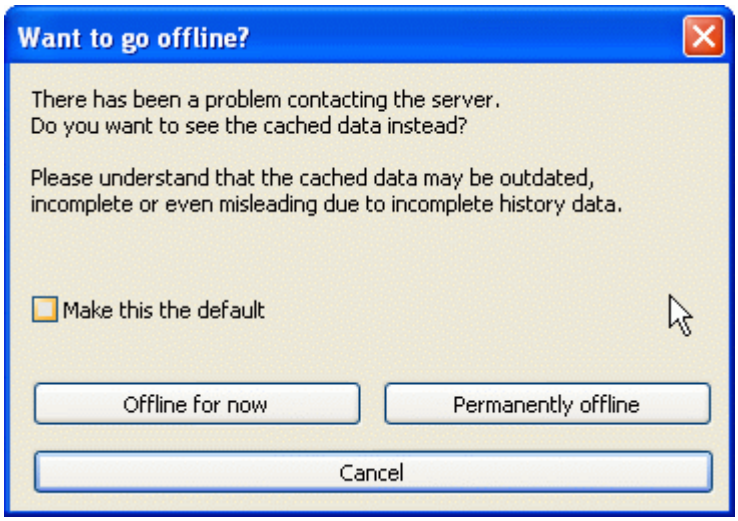


Figura 4.22. Ir para Janela de Desconectado

If the server is not reachable, and you have log caching enabled you can use the log dialog and revision graph in offline mode. This uses data from the cache, which allows you to continue working although the information may not be up-to-date or even complete.

Here you have three options:

Desconectado agora
: Complete the current operation in offline mode, but retry the repository next time log data is requested.

Permanentemente desconectado
: Remain in offline mode until a repository check is specifically requested. See Seção 4.9.11, “Atualizando a Visualização”.

Cancelar
: If you don't want to continue the operation with possibly stale data, just cancel.

The Make this the default checkbox prevents this dialog from re-appearing and always picks the option you choose next. You can still change (or remove) the default after doing this from TortoiseSVN → Settings.

4.9.11. Atualizando a Visualização

If you want to check the server again for newer log messages, you can simply refresh the view using **F5**. If you are using the log cache (enabled by default), this will check the repository for newer messages and fetch only the new ones. If the log cache was in offline mode, this will also attempt to go back online.

If you are using the log cache and you think the message content or author may have changed, you can use **Shift-F5** or **Ctrl-F5** to re-fetch the displayed messages from the server and update the log cache. Note that this only affects messages currently shown and does not invalidate the entire cache for that repository.

4.10. Visualizando as Diferenças

One of the commonest requirements in project development is to see what has changed. You might want to look at the differences between two revisions of the same file, or the differences between two separate

files. TortoiseSVN provides a built-in tool named TortoiseMerge for viewing differences of text files. For viewing differences of image files, TortoiseSVN also has a tool named TortoiseIDiff. Of course, you can use your own favourite diff program if you like.

4.10.1. Diferenças do Arquivo

Alterações Locais
: If you want to see what changes *you* have made in your working copy, just use the explorer context menu and select TortoiseSVN → Diff.

Difference to another branch/tag
: If you want to see what has changed on trunk (if you are working on a branch) or on a specific branch (if you are working on trunk), you can use the explorer context menu. Just hold down the **Shift** key while you right click on the file. Then select TortoiseSVN → Diff with URL. In the following dialog, specify the URL in the repository with which you want to compare your local file to.

  You can also use the repository browser and select two trees to diff, perhaps two tags, or a branch/tag and trunk. The context menu there allows you to compare them using Compare revisions. Read more in Seção 4.10.3, “Comparando Diretórios”.

Diferenças da revisão anterior
: If you want to see the difference between a particular revision and your working copy, use the Revision Log dialog, select the revision of interest, then select Compare with working copy from the context menu.

  If you want to see the difference between the last committed revision and your working copy, assuming that the working copy hasn't been modified, just right click on the file. Then select TortoiseSVN → Diff with previous version. This will perform a diff between the revision before the last-commit-date (as recorded in your working copy) and the working BASE. This shows you the last change made to that file to bring it to the state you now see in your working copy. It will not show changes newer than your working copy.

Difference between two previous revisions
: If you want to see the difference between two revisions which are already committed, use the Revision Log dialog and select the two revisions you want to compare (using the usual **Ctrl**-modifier). Then select Compare revisions from the context menu.

  If you did this from the revision log for a folder, a Compare Revisions dialog appears, showing a list of changed files in that folder. Read more in Seção 4.10.3, “Comparando Diretórios”.

Todas as alterações feitas em uma submissão
: If you want to see the changes made to all files in a particular revision in one view, you can use Unified-Diff output (GNU patch format). This shows only the differences with a few lines of context. It is harder to read than a visual file compare, but will show all the changes together. From the Revision Log dialog select the revision of interest, then select Show Differences as Unified-Diff from the context menu.

Diferença entre arquivos
: If you want to see the differences between two different files, you can do that directly in explorer by selecting both files (using the usual **Ctrl**-modifier). Then from the explorer context menu select TortoiseSVN → Diff.

Difference between WC file/folder and a URL
: If you want to see the differences between a file in your working copy, and a file in any Subversion repository, you can do that directly in explorer by selecting the file then holding down the **Shift** key whilst right clicking to obtain the context menu. Select TortoiseSVN → Diff with URL. You can do the same thing for a working copy folder. TortoiseMerge shows these differences in the same way as it shows a patch file - a list of changed files which you can view one at a time.

Difference with blame information
: If you want to see not only the differences but also the author, revision and date that changes were made, you can combine the diff and blame reports from within the revision log dialog. Read Seção 4.23.2, “Diferenças de Autoria” for more detail.

Diferença entre diretórios
: The built-in tools supplied with TortoiseSVN do not support viewing differences between directory hierarchies. But if you have an external tool which does support that feature, you can use that instead. In Seção 4.10.5, “External Diff/Merge Tools” we tell you about some tools which we have used.

If you have configured a third party diff tool, you can use **Shift** when selecting the Diff command to use the alternate tool. Read Seção 4.30.5, “External Program Settings” to find out about configuring other diff tools.

4.10.2. Line-end and Whitespace Options

Sometimes in the life of a project you might change the line endings from CRLF to LF, or you may change the indentation of a section. Unfortunately this will mark a large number of lines as changed, even though there is no change to the meaning of the code. The options here will help to manage these changes when it comes to comparing and applying differences. You will see these settings in the Merge and Blame dialogs, as well as in the settings for TortoiseMerge.

Ignore line endings excludes changes which are due solely to difference in line-end style.

Compare whitespaces includes all changes in indentation and inline whitespace as added/removed lines.

Ignore whitespace changes excludes changes which are due solely to a change in the amount or type of whitespace, eg. changing the indentation or changing tabs to spaces. Adding whitespace where there was none before, or removing a whitespace completely is still shown as a change.

Ignore all whitespaces excludes all whitespace-only changes.

Naturally, any line with changed content is always included in the diff.

4.10.3. Comparando Diretórios

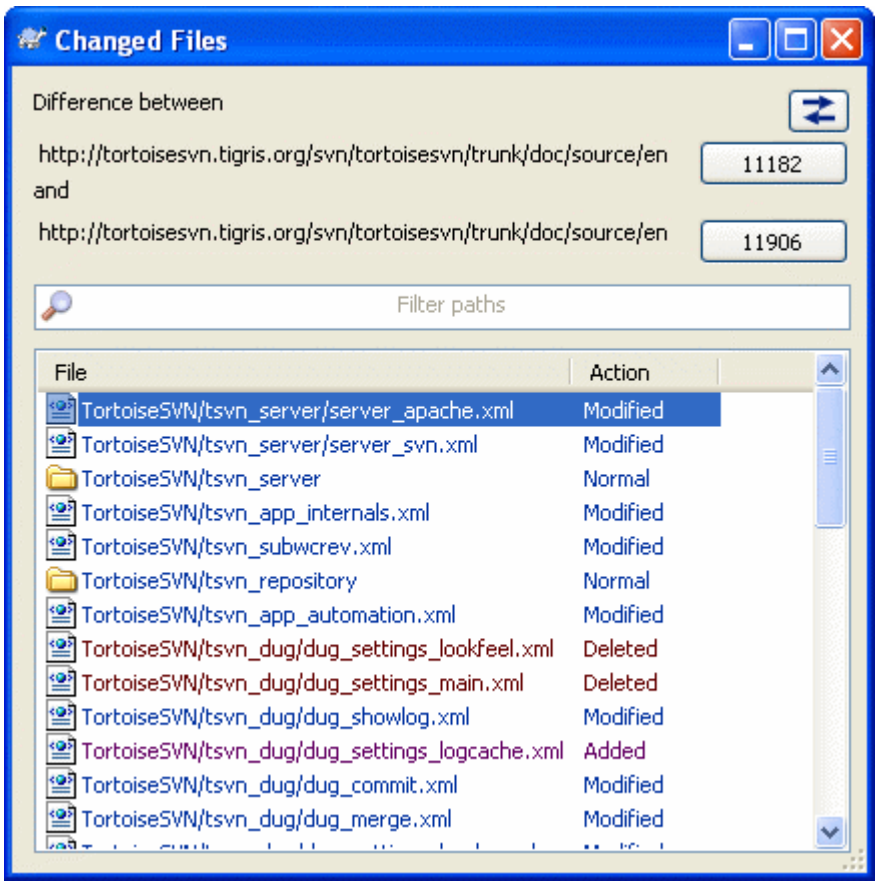


Figura 4.23. A Janela de Comparação de Revisões

When you select two trees within the repository browser, or when you select two revisions of a folder in the log dialog, you can Context menu → Compare revisions.

This dialog shows a list of all files which have changed and allows you to compare or blame them individually using context menu.

You can export a *change tree*, which is useful if you need to send someone else your project tree structure, but containing only the files which have changed. This operation works on the selected files only, so you need to select the files of interest - usually that means all of them - and then Context menu → Export selection to.... You will be prompted for a location to save the change tree.

You can also export the *list* of changed files to a text file using Context menu → Save list of selected files to....

If you want to export the list of files *and* the actions (modified, added, deleted) as well, you can do that using Context menu → Copy selection to clipboard.

The button at the top allows you to change the direction of comparison. You can show the changes need to get from A to B, or if you prefer, from B to A.

The buttons with the revision numbers on can be used to change to a different revision range. When you change the range, the list of items which differ between the two revisions will be updated automatically.

If the list of filenames is very long, you can use the search box to reduce the list to filenames containing specific text. Note that a simple text search is used, so if you want to restrict the list to C source files you should enter `.c` rather than `*.c`.

4.10.4. Diffing Images Using TortoiseIDiff

There are many tools available for diffing text files, including our own TortoiseMerge, but we often find ourselves wanting to see how an image file has changed too. That's why we created TortoiseIDiff.

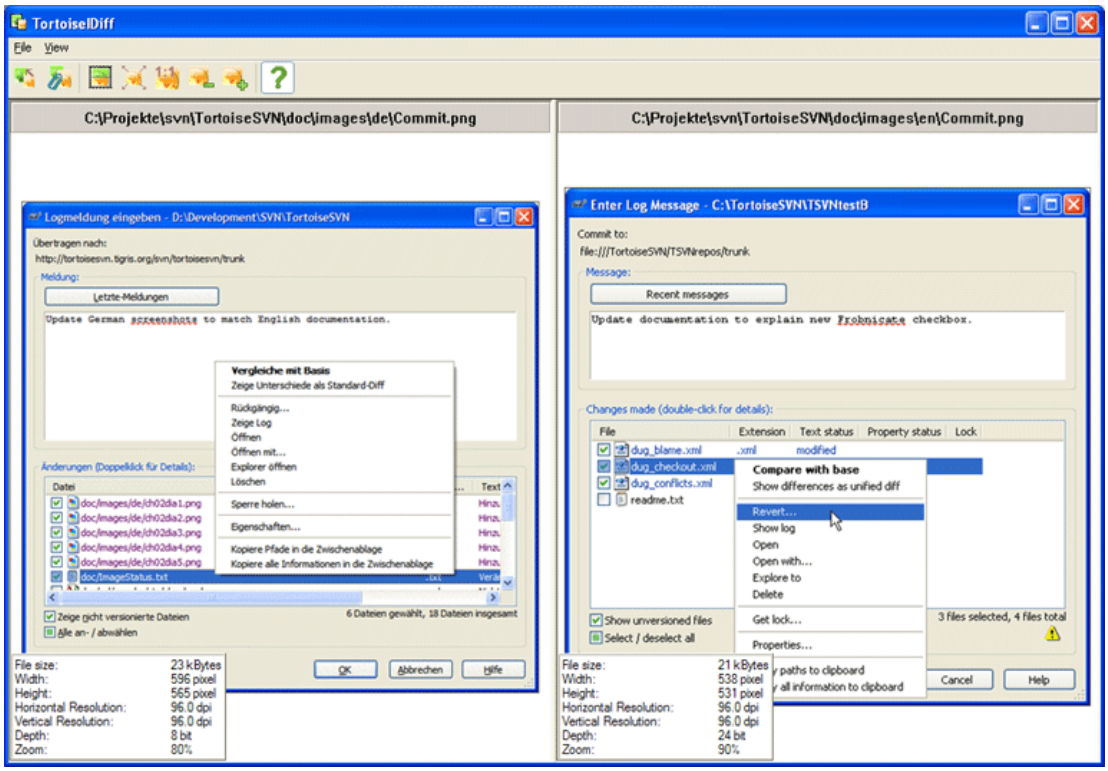


Figura 4.24. The image difference viewer

TortoiseSVN → Diff for any of the common image file formats will start TortoiseIDiff to show image differences. By default the images are displayed side-by-side but you can use the View menu or toolbar to switch to a top-bottom view instead, or if you prefer, you can overlay the images and pretend you are using a lightbox.

Naturally you can also zoom in and out and pan around the image. You can also pan the image simply by left-dragging it. If you select the Link images together option, then the pan controls (scrollbars, mousewheel) on both images are linked.

An image info box shows details about the image file, such as the size in pixels, resolution and colour depth. If this box gets in the way, use View → Image Info to hide it. You can get the same information in a tooltip if you hover the mouse over the image title bar.

When the images are overlaid, the relative intensity of the images (alpha blend) is controlled by a slider control at the left side. You can click anywhere in the slider to set the blend directly, or you can drag the slider to change the blend interactively. **Ctrl+Shift**-Wheel to change the blend.

The button above the slider toggles between 0% and 100% blends, and if you double click the button, the blend toggles automatically every second until you click the button again. This can be useful when looking for multiple small changes.

Sometimes you want to see a difference rather than a blend. You might have the image files for two revisions of a printed circuit board and want to see which tracks have changed. If you disable alpha blend mode, the difference will be shown as an *XOR* of the pixel colour values. Unchanged areas will be plain white and changes will be coloured.

4.10.5. External Diff/Merge Tools

If the tools we provide don't do what you need, try one of the many open-source or commercial programs available. Everyone has their own favourites, and this list is by no means complete, but here are a few that you might consider:

WinMerge
: *WinMerge* [http://winmerge.sourceforge.net/] is a great open-source diff tool which can also handle directories.

Forçar Unificação
: Perforce is a commercial RCS, but you can download the diff/merge tool for free. Get more information from *Perforce* [http://www.perforce.com/perforce/products/merge.html].

KDiff3
: KDiff3 is a free diff tool which can also handle directories. You can download it from *here* [http://kdiff3.sf.net/].

ExaminarDiferenças
: ExamDiff Standard is freeware. It can handle files but not directories. ExamDiff Pro is shareware and adds a number of goodies including directory diff and editing capability. In both flavours, version 3.2 and above can handle unicode. You can download them from *PrestoSoft* [http://www.prestosoft.com/].

Beyond Compare
: Similar to ExamDiff Pro, this is an excellent shareware diff tool which can handle directory diffs and unicode. Download it from *Scooter Software* [http://www.scootersoftware.com/].

Araxis Merge
: Araxis Merge is a useful commercial tool for diff and merging both files and folders. It does three-way comparison in merges and has synchronization links to use if you've changed the order of functions. Download it from *Araxis* [http://www.araxis.com/merge/index.html].

SciTE
: This text editor includes syntax colouring for unified diffs, making them much easier to read. Download it from *Scintilla* [http://www.scintilla.org/SciTEDownload.html].

Notepad2
: Notepad2 is designed as a replacement for the standard Windows Notepad program, and is based on the Scintilla open-source edit control. As well as being good for viewing unified diffs, it is much better than the Windows notepad for most jobs. Download it for free *here* [http://www.flos-freeware.ch/notepad2.html].

Read Seção 4.30.5, “External Program Settings” for information on how to set up TortoiseSVN to use these tools.

4.11. Adding New Files And Directories

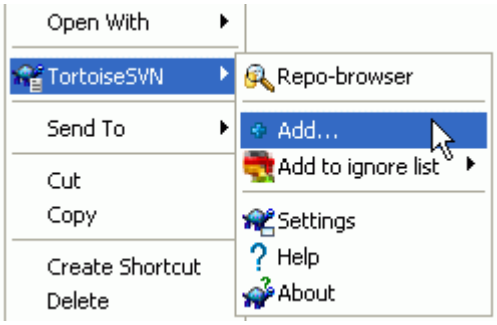


Figura 4.25. Explorer context menu for unversioned files

If you created new files and/or directories during your development process then you need to add them to source control too. Select the file(s) and/or directory and use TortoiseSVN → Add.

After you added the files/directories to source control the file appears with a added icon overlay which means you first have to commit your working copy to make those files/directories available to other developers. Adding a file/directory does *not* affect the repository!

Muitas Adições

You can also use the Add command on already versioned folders. In that case, the add dialog will show you all unversioned files inside that versioned folder. This helps if you have many new files and need to add them all at once.

To add files from outside your working copy you can use the drag-and-drop handler:

1. select the files you want to add
2. right-drag them to the new location inside the working copy
3. release the right mouse button
4. select Context Menu → SVN Add files to this WC. The files will then be copied to the working copy and added to version control.

You can also add files within a working copy simply by left-dragging and dropping them onto the commit dialog.

If you add a file or folder by mistake, you can undo the addition before you commit using TortoiseSVN → Undo add....

4.12. Copying/Moving/Renaming Files and Folders

It often happens that you already have the files you need in another project in your repository, and you simply want to copy them across. You could simply copy the files and add them as described above, but that would not give you any history. And if you subsequently fix a bug in the original files, you can only merge the fix automatically if the new copy is related to the original in Subversion.

The easiest way to copy files and folders from within a working copy is to use the right-drag menu. When you right-drag a file or folder from one working copy to another, or even within the same folder, a context menu appears when you release the mouse.

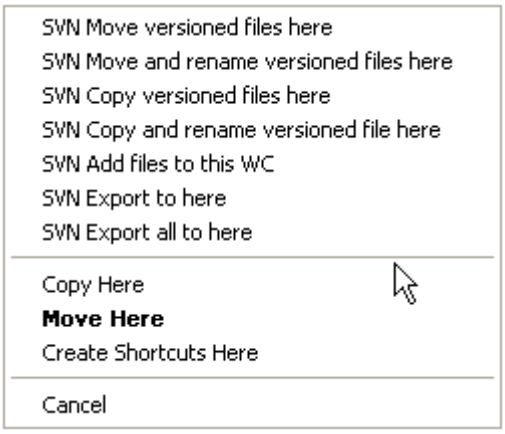


Figura 4.26. Menu de quando se clica com o botão direito e se arrasta um diretório que está sob o controle de versão.

Now you can copy existing versioned content to a new location, possibly renaming it at the same time.

You can also copy or move versioned files within a working copy, or between two working copies, using the familiar cut-and-paste method. Use the standard Windows Copy or Cut to copy one or more versioned items to the clipboard. If the clipboard contains such versioned items, you can then use TortoiseSVN → Paste (note: not the standard Windows Paste) to copy or move those items to the new working copy location.

You can copy files and folders from your working copy to another location in the repository using TortoiseSVN → Branch/Tag. Refer to Seção 4.19.1, “Criando um Ramo ou Rótulo” to find out more.

You can locate an older version of a file or folder in the log dialog and copy it to a new location in the repository directly from the log dialog using Context menu → Create branch/tag from revision. Refer to Seção 4.9.3, “Recuperando Informações Adicionais” to find out more.

You can also use the repository browser to locate content you want, and copy it into your working copy directly from the repository, or copy between two locations within the repository. Refer to Seção 4.24, “O Navegador de Repositório” to find out more.

Cannot copy between repositories

Whilst you can copy and files and folders *within* a repository, you *cannot* copy or move from one repository to another while preserving history using TortoiseSVN. Not even if the repositories live on the same server. All you can do is copy the content in its current state and add it as new content to the second repository.

If you are uncertain whether two URLs on the same server refer to the same or different repositories, use the repo browser to open one URL and find out where the repository root is. If you can see both locations in one repo browser window then they are in the same repository.

4.13. Ignorando Arquivos e Diretórios

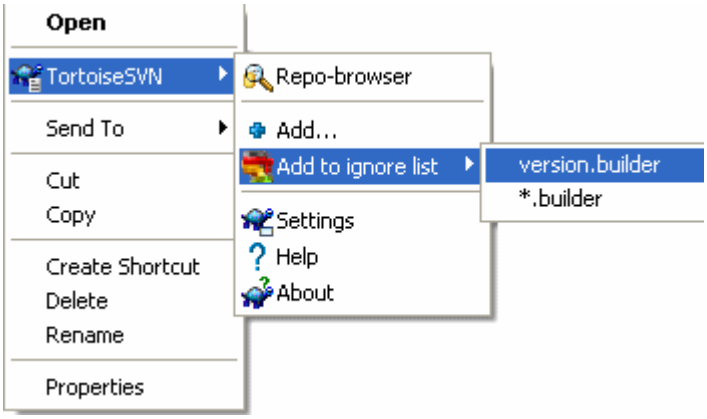


Figura 4.27. Explorer context menu for unversioned files

Na maioria dos projetos você terá arquivos e diretórios que não deverão ser controlados. Entre eles estão arquivos criados pelo compilador,
<nome_do_arquivo>*.obj, *.lst</nome_do_arquivo>
, talvez um diretório de saída usado para gravar o executável. Não importa que você submeta alterações, TortoiseSVN mostrará seus arquivos não controlados, os quais serão apresentados na lista da janela de

submissão. Claro, você pode optar por não mostrá-los, mas então você poderá esquecer de adicionar alguma novo arquivo de código.

A melhor maneira de se livrar desses problemas é adicionar esses arquivos na lista de arquivos ignorados do projeto. Dessa forma eles nunca serão mostrados na lista da janela de submissão, mas arquivos genuinos de código não controlados aparecerão para serem adicionados.

Se você clicar com o botão direito em um único arquivo não controlado, e selecionar o comando TortoiseSVN → Adicionar à lista de ignorados do menu de contexto, um submenu aparecerá permitindo que você selecione apenas o arquivo, ou todos os arquivos com a mesma extensão. Se você selecionar vários arquivos, não aparecerá o submenu e você poderá somente adicionar os arquivos/diretórios selecionados.

Se você quer remover um ou mais itens da lista de arquivos ignorados, clique com o botão direito sobre os itens e selecione TortoiseSVN → Remover da lista de ignorados. Você pode também acessar a propriedade `svn:ignore` do diretório diretamente. Isto lhe permite especificar regras gerais usando o nome dos arquivos globais, descrito na próxima seção. Leia Seção 4.17, “Configurações do Projeto” para mais informações sobre como definir as propriedades diretamente. Por favor tenha o cuidado de colocar cada regra em uma linha separada. Separá-las por espaço não funciona.

A Lista Global de Arquivos Ignorados

Outra forma de ignorar arquivos é adicionar eles para a *lista global de arquivo*. A grande diferença é que a lista global de arquivos ignorados é uma propriedade no cliente. Isto é aplicado parar *todos* projetos no Subversion, but somente no PC cliente. Em geral o melhor é usar a propriedade `svn:ignore` onde possível, porque pode ser aplicado para áreas específicas do projeto, e isto funciona para todos que obterem o projeto. Leia Seção 4.30.1, “Configurações Gerais” para mais informações.

Ignorando Arquivos Controlados

Arquivos e diretórios controlados nunca devem ser ignorados - esta é uma característica do Subversion. Se você está controlando um arquivo mas não deveria, leia Seção B.8, “Ignore files which are already versioned” para instruções de como “não controlar” o arquivo.

4.13.1. Padrões de Filtro na Lista de Arquivos Ignorados

Os padrões para ingorar arquivos do Subversion fazem uso do nome de arquivo global, uma técnica originalmente usada no Unix para especificar arquivos usando meta-caracteres como coringas. Os caracteres a seguir tem um significado especial:

*
: Filtra por qualquer texto de caracteres, incluindo um texto vazio (nenhum caracter).

?
: Filtra por qualquer caracter simples.

[...]
: Filtra por qualquer um dos caracteres dentro dos colchetes. Dentro de colchetes, um par de caracteres separados por “-” filtra qualquer caracter léxico entre os dois. Por exemplo `[AGm-p]` flitra por qualquer um dos caracteres `A`, `G`, `m`, `n`, `o` or `p`.

Padrões de filtro são sensíveis à caixa, o que pode causar problemas no Windows. Você pode forçar o uso não sensível da maneira difícil usando pares de caracteres, por exemplo para ignorar `*.tmp` independente do caso, você poderá usar um padrão como `*.[Tt][Mm][Pp]`.

Se você quer uma definição oficial para nomes de arquivos globais, você pode encontrar nas especificações IEEE para linguagem de comando de integração *Pattern Matching Notation* [http://www.opengroup.org/onlinepubs/009695399/utilities/xcu_chap02.html#tag_02_13].

Nenhum Caminho na Lista Global de Arquivos Ignorados

Você não deverá incluir caminhos no seu padrão. O padrão de filtro tem a intensão de ser usado apenas para nomes de arquivos e diretórios. Se você quer ignorar todos os diretórios `CVS`, apenas adicione o diretório `CVS` na lista de arquivos ignorados. Não é necessário especificar `CVS */CVS` como você faria em versões anteriores. Se você quer ignorar todos os diretórios `tmp` quando existirem dentro de um diretório `prog` mas não quer quando estiverem dentro de um diretório

4.14. Apagando, Movendo e Renomeando

Ao contrário do CVS, Subversion permite renomear e mover arquivos e diretórios. Assim há menu para excluir e renomear no submenu do TortoiseSVN.

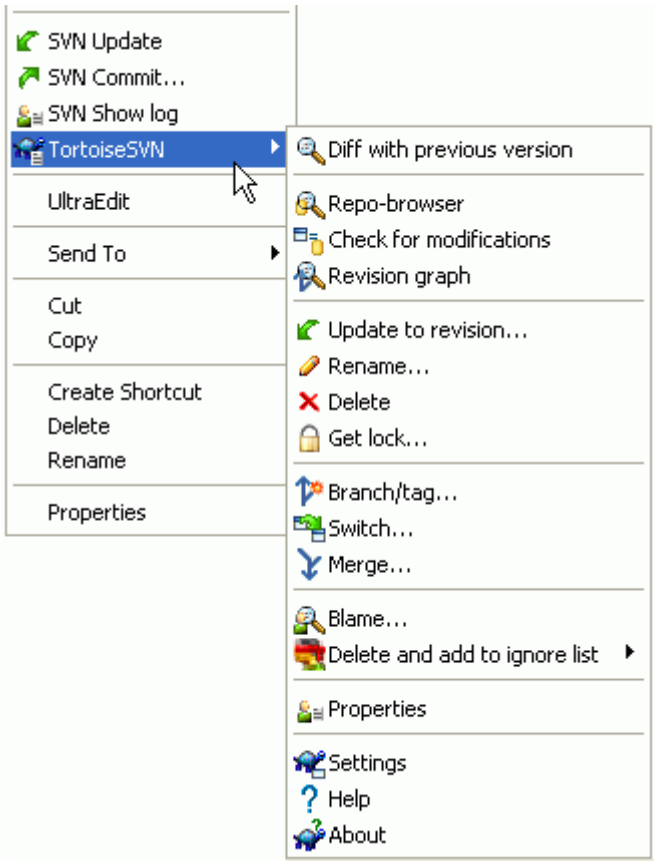


Figura 4.28. Menu de contexto do Explorerer para arquivos controlados

4.14.1. Apagando arquivos e diretórios

Use TortoiseSVN → Excluir para remover arquivos ou diretórios do repositório.

Quando você TortoiseSVN

Quando você TortoiseSVN → Excluir um diretório, ele não será apagado da sua cópia de trabalho, mas a exclusão será apresentada através de um ícone no diretório. Até que você faça a submissão da alteração,

você poderá recuperar o diretório usando TortoiseSVN → Reverter no próprio diretório. A diferença entre os comportamentos de arquivos e diretórios faz parte do Subversion, e não do TortoiseSVN.

Se você quer apagar um item do repositório, mas manter uma versão local não versionada do arquivo/diretório, use Menu de Contexto Estendido → Excluir (manter local). Você deve manter a tecla **Shift** pressionada enquando clica com o botão direito no item desejado no painel direito do "explorer" para ver este menu de contexto estendido.

Se um *arquivo* é excluido usando o "explorer" ao invés do menu de contexto do TortoiseSVN, a janela de submissão apresentará estes arquivos e permitirá a você removê-los do controle antes da submissão. Entretanto, se você fizer uma atualização da sua cópia de trabalho, Subversion marcará o arquivo perdido e o substituirá com a versão mais recente do repositório. Se você precisa apagar um arquivo controlado, deve usar sempre TortoiseSVN → Excluir para que o Subversion não tenha que adivinhar o que você realmente quer fazer.

Se um *diretório* é excluido através do "explorer" ao invés do menu de contexto do TortoiseSVN, sua cópia de trabalho estará corrompida e você não conseguirá fazer uma submissão. Se você atualizar sua cópia de trabalho, Subversion substituirá o diretório perdido com a última revisão do repositório e você então precisará excluir da maneira correta usando <menuchoise>TortoiseSVNExcluir.</menuchoise>

Recuperando arquivos e diretórios excluidos

Se você apagou um arquivo ou diretório e já submeteu essa exclusão para o repositório, então a maneira normal TortoiseSVN → Reverter não recuperará nada. Mas o arquivo ou diretório não estão totalmente perdidos. Se você sabe a revisão em que o arquivo ou diretório foi excluido (se você não sabe, use a janela de log para descobrir) abra o navegador de repositório e alterne para a revisão. Então selecione o arquivo ou diretório que você apagou, clique com o botão direito sobre e escolha Menu de Contexto → Copiar para... e como destino da c

4.14.2. Movendo arquivos e diretórios

Se você quer fazer uma simples renomeação de arquivo ou diretório, use Menu de Contexto → Renomear... e entre com o novo nome para o item e estará feito.

Se você quer mover arquivos dentro da sua cópia de trabalho, talvez para um sub-diretório, use o manipulador de arrasto do botão direito:

1. selecione os arquivos ou diretórios que você quer mover
2. right-drag them to the new location inside the working copy
3. release the right mouse button
4. no menu que se abre escolha Menu de Contexto → SVN Mover itens sob controle de versão para cá

Submetendo o diretório pai

Já que renomear e mover arquivos está feito assim como a exclusão como complemento você deve submeter o diretório pai dos arquivos renomeados/movidos assim como os excluídos/movidos que aparecerão na janela de submissão. Se você não submeteu a parte removida da ação de renomear/mover, ela continuará no repositório e quando algum outro

participante do projeto atualizar sua cópia, os velhos arquivos não serão removidos. Por exemplo eles *ambas* as cópias, velhas e novas.

Você *deve* submeter um diretório renomeado antes de alterar qualquer arquivo dentro dele, caso contrário sua cópia de trabalho pode realmente se corromper.

Você pode também usar o navegador de repositório para mover itens. Leia Seção 4.24, “O Navegador de Repositório” para descobrir mais.

Não Mova Arquivos Externos

Você *não* deverá usar os comandos do TortoiseSVN Mover ou Renomear em um diretório que tenha sido criado usando `svn:externals`. Esta ação poderá causar a exclusão de um item externo em seu repositório, provavelmente chateando muitas outras pessoas. Se você precisa mover um diretório externo você deverá usar uma outra ferramenta para mover, e então ajustar as propriedades `svn:externals` do diretório de origem e de destino.

4.14.3. Alterando a caixa do nome do arquivo

Alterando apenas a caixa do nome do arquivo é difícil no Subversion em ambiente Windows, porque por um curto período de tempo durante a renomeação, ambos os arquivos devem existir. Como o Windows utiliza um sistema de arquivos não sensível à caixa, ele simplesmente não permite utilizar o comando padrão Renomear.

Felismente existem (pelo menos) dois possiveis meios de renomear um arquivo sem perder sua histórico de log. Por isso é importante renomear usando o subversion. Apenas renomear no "explorer" vai corrompear sua cópia de trabalho!

Solução A) (recomendada)

1. Submetendo as alterações da sua cópia de trabalho
2. Renomear o arquivo de CAIXAalta para caixaALTA diretamente no repositório usando o navegador do repositório.
3. Atualizar sua cópia de trabalho.

Solução B)

1. Renomear de CAIXAalta para CAIXAalta_ com o comando de renomeação do submenu do TortoiseSVN.
2. Submeter as alterações.
3. Renomear de CAIXAalta_ para caixaALTA.
4. Submeter as alterações.

4.14.4. Procedimento em caso de conflito com o nome do arquivo

Se o repositório atualmente já contém dois arquivos com o mesmo nome diferenciados apenas pela caixa (exemplo: `TEST.TXT` e `test.txt`, você não poderá atualizar ou obter o diretório em um cliente Windows.Embora o Subversion suporte diferenciação na caixa no nome dos arquivos, o Windows não suporta.

Isto algumas vezes acontece quando duas pessoas submetem, de diferentes cópias de trabalho, arquivos salvos com o mesmo nome, mas com caixa diferente. Isto pode também acontecer quando os arquivos forem submetidos de um sistema de arquivos que trata a caixa do nome dos arquivos, como Linux.

Nestes casos, você precisa decidir qual dos arquivos você quer manter e então apagar (ou renomear) o outro arquivo no repositório.

Prevenindo dois arquivos com o mesmo nome

Existe um script para servidor disponível em: *http://svn.collab.net/repos/svn/trunk/contrib/hook-scripts/* que previne submissões que resultem em conflito de caixa.

4.14.5. Reparando Renomeação de Arquivos

Algumas vezes seu amigável IDE vai renomear arquivos para você como parte de uma refatoração, e é claro isto não é o Subversion. Se você tentar submeter suas alterações, Subversion verá que há velhos arquivos perdidos e que há novos arquivos não controlados. Você poderá apenas verificar o novo nome para adicioná-lo, mas você então perderá o histórico, já que o Subversion não conhece a relação dos arquivos.

Uma maneira melhor de notificar o Subversion é identificar a alteração como uma renomeação, e você pode fazer isso usando as janelas de Submeter e Verificar alterações. Apenas selecione ambos os arquivos (perdido e não controlado) e use
<menuchocie>Menu de ContextoReparar movimento para identificar os dois arquivos como uma renomea</menuchocie>

4.14.6. Apagando Arquivos não Controlados

Normalmente você define sua lista de arquivos ignorados para que todos os arquivos criados sejam ignorados pelo Subversion. Mas e se você quer remover todos os arquivos ignorados para produzir uma compilação limpa? Normalmente você definie isso no arquivo makefile, mas e se você está depurando o arquivo makefile, ou alterando o sistema de compilação é útil ter uma forma de limpar tudo.

TortoiseSVN provê justamente uma opção usando Menu de Contexto Estendido → Apagar arquivos não controlados.... VocShift enquando clica o bot

Quando tais arquivos são removidos, a lixeira é usada, então se você cometeu algum erro ao excluir um arquivo que não estava sendo controlado, você ainda poderá restaurar o arquivo.

4.15. Desfazendo Alterações

Se você quer desfazer todas as alterações feitas em um arquivo desde a última atualização você precisa selecionar o arquivo, clique com o botão direito para abrir o menu de contexto e então selecione a opção TortoiseSVN → Reverter. Uma janela será apresentada com os arquivos que você alterou e que podem ser revertidos. Selecione os arquivos que você reverter e clique em Ok.

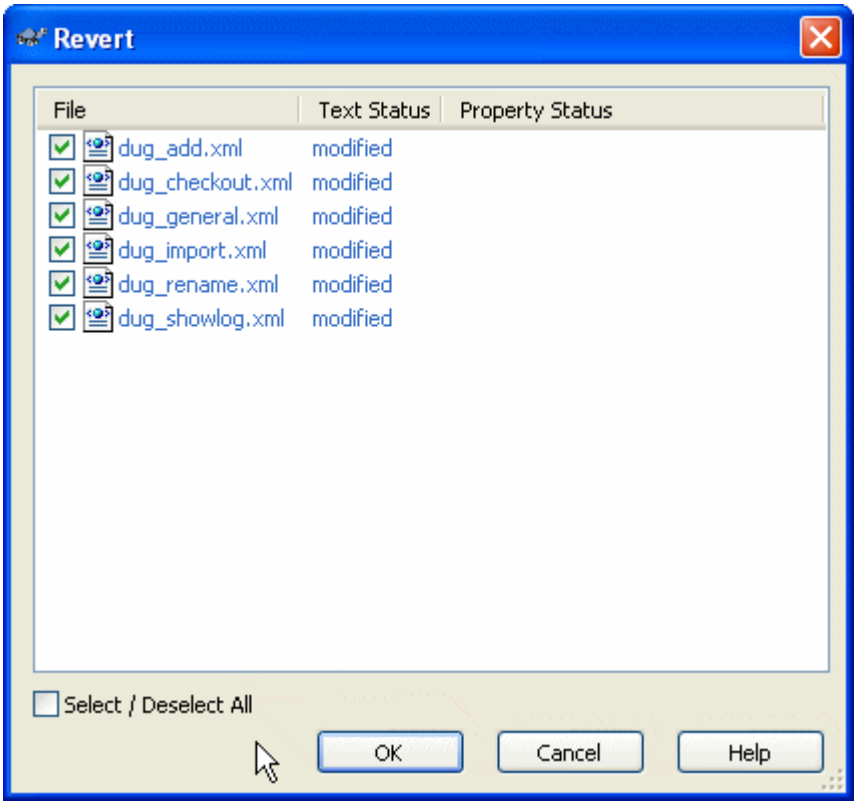


Figura 4.29. Janela de Reversão

Se você quer desfazer uma exclusão ou uma renomeação, você precisa usar o Reverter no diretório em que o arquivo foi excluido/renomeado já que não existe mais para usar o clique com o botão direito.

Se você quer desfazer a adição de um item, ele aparece no menu de contexto como TortoiseSVN → Desfazer adição.... Esta op

As colunas nesta janela podem ser personalizadas da mesma forma como as colunas da janela Verificar alterações. Leia Seção 4.7.3, “Estado Local e Remoto” para mais detalhes.

Desfazendo Alterações que já foram Submetidas

Reverter irá somente desfazer alterações locais. A opção reverter *não* desfará alterações que já foram submetidas. Se você quer desfazer todas as alterações que foram submetidas em uma revisão em particular, leia Seção 4.9, “Janela de Revisão de Registro” para informações adicionais.

Reversão está Lenta

Quando você reverter alteracões você pode achar que a operação levou mais tempo que o esperado. Isto acontece porque a versão da modificação do arquivo foi enviado para a lixeira, então você pode recuperar suas alterações se você reverter algo por engano. Entretanto, se sua lixeira estiver cheia, Windows vai levar um longo tempo para encontrar um lugar para colocar o seu arquivo. A solução é simples: mantenha a lixeira vazia ou desative a opção Usar lixeira para fazer reversão nas configurações do TortoiseSVN.

4.16. Limpar

Se um comando do Subversion não pode ser completado com sucesso, talvez por causa de problemas com o servidor, sua cópia de trabalho pode ficar em um estado inconsistente. Neste caso você precisa usar o comando TortoiseSVN → Limpar no diretório. É uma boa idéia fazer isso no diretório de nivel mais alto da sua cópia de trabalho.

Limpar tem outra utilidade. Se a data de um arquivo foi alterada mas não o seu conteúdo, Subversion não poderá lhe avisar que ele foi alterado exceto através de um comparação byte-por-byte com a cópia anterior. Se você tem muitos arquivos nesta situação isto fará a aquisição da situação muito lenta, o qual fará muitas janelas lentas para responder. Ao executar o comando Limpar na sua cópia de trabalho ela irá reparar essas datas “inválidas” e irá restaurar a verificação da situação para a velocidade máxima.

Usar a Data da Submissão

Algumas liberações recentes do Subversion foram afetadas por um erro do qual causou problemas com a data quando você obteve arquivos com a opção Usar a data da submissão selecionada. Use o comando Limpar para aumentar a velocidade dessas cópias de trabalho.

4.17. Configurações do Projeto

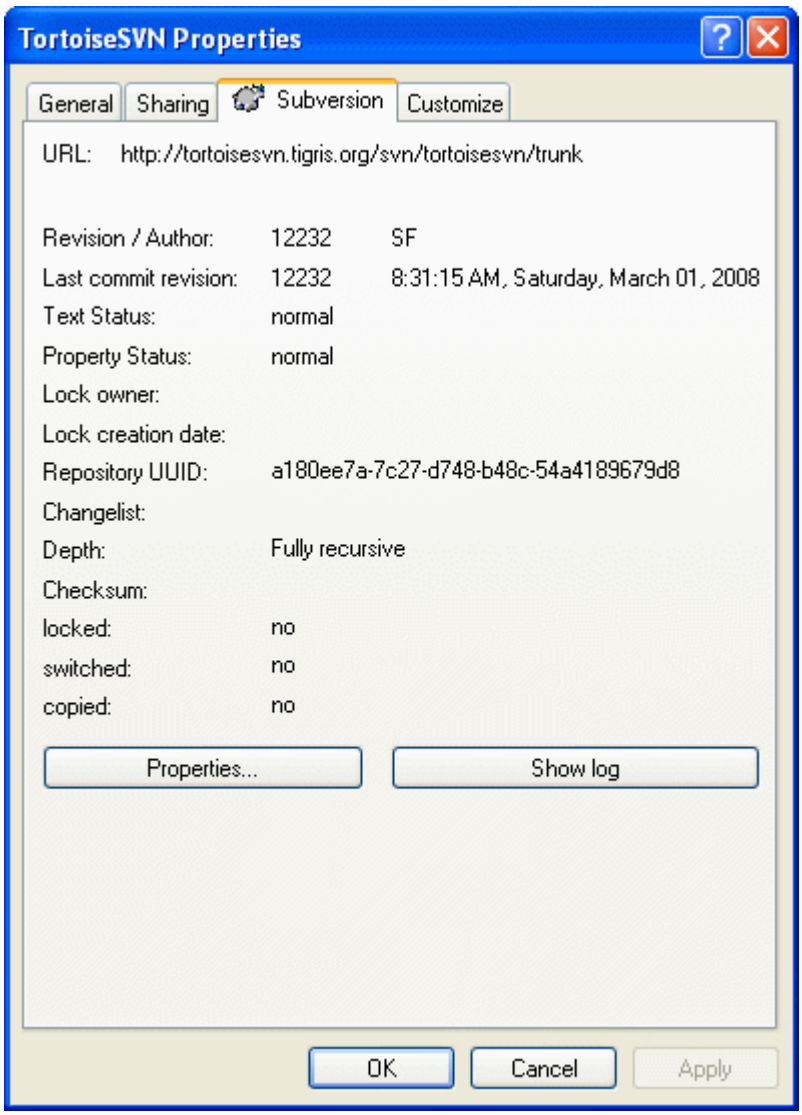


Figura 4.30. Explorer property page, Subversion tab

Sometimes you want to have more detailed information about a file/directory than just the icon overlay. You can get all the information Subversion provides in the explorer properties dialog. Just select the file or directory and select Windows Menu → properties in the context menu (note: this is the normal properties menu entry the explorer provides, not the one in the TortoiseSVN submenu!). In the properties dialog box TortoiseSVN has added a new property page for files/folders under Subversion control, where you can see all relevant information about the selected file/directory.

4.17.1. Propriedades do Subversion

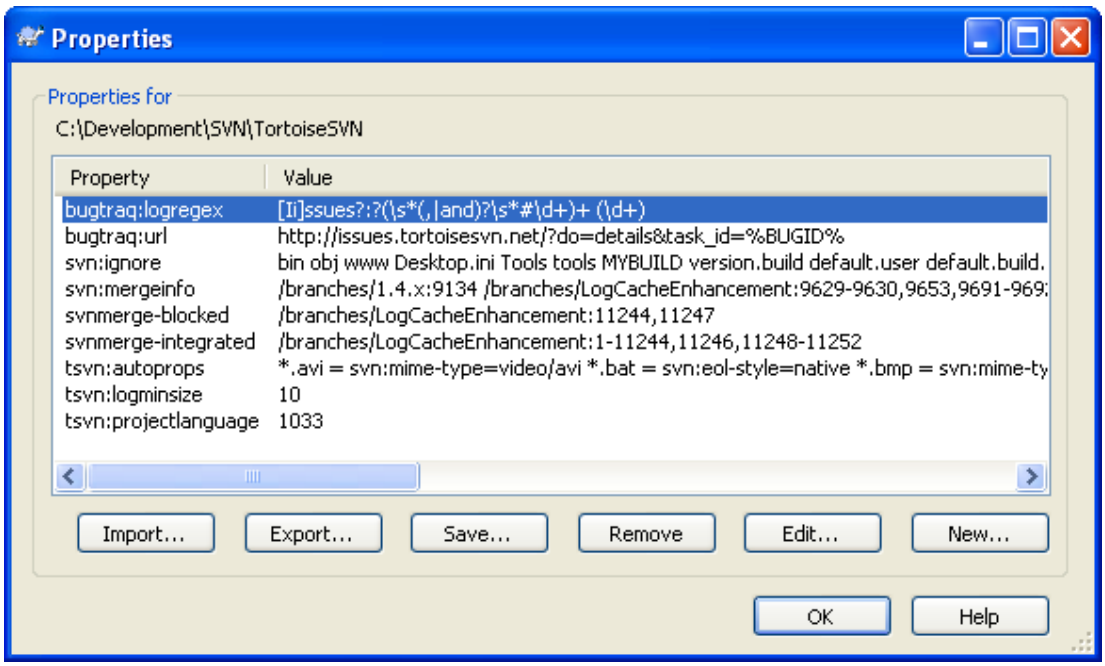


Figura 4.31. página de propriedades do Subversion

You can read and set the Subversion properties from the Windows properties dialog, but also from TortoiseSVN → properties and within TortoiseSVN's status lists, from Context menu → properties.

You can add your own properties, or some properties with a special meaning in Subversion. These begin with `svn:`. `svn:externals` is such a property; see how to handle externals in Seção 4.18, “Itens Externos”.

4.17.1.1. svn:keywords

Subversion supports CVS-like keyword expansion which can be used to embed filename and revision information within the file itself. Keywords currently supported are:

$Date$
: Date of last known commit. This is based on information obtained when you update your working copy. It does *not* check the repository to find more recent changes.

$Revision$
: Revisão da última submissão conhecida

$Author$
: Author who made the last known commit.

$HeadURL$
: The full URL of this file in the repository.

Id
: A compressed combination of the previous four keywords.

To find out how to use these keywords, look at the *svn:keywords section* [http://svnbook.red-bean.com/en/1.5/svn.advanced.props.special.keywords.html] in the Subversion book, which gives a full description of these keywords and how to enable and use them.

For more information about properties in Subversion see the *Special Properties* [http://svnbook.red-bean.com/en/1.5/svn.advanced.props.html].

4.17.1.2. Adding and Editing Properties

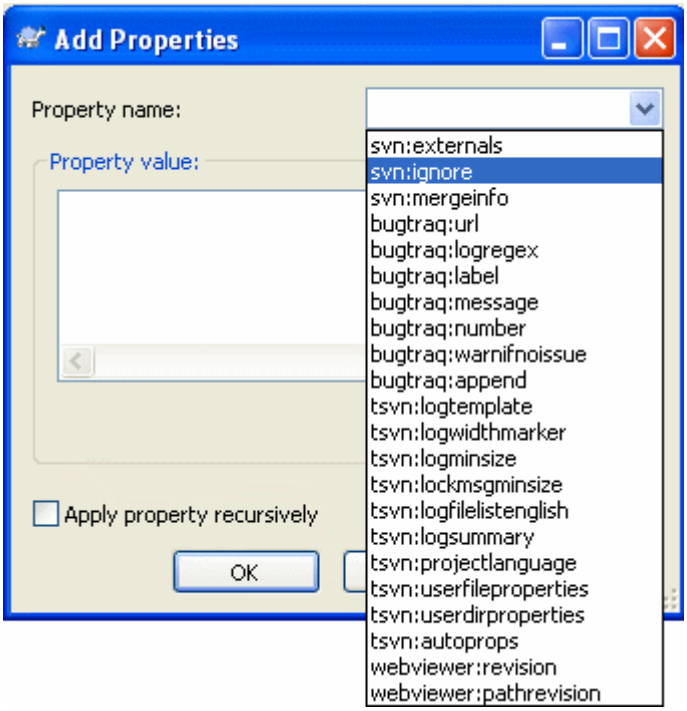


Figura 4.32. Adicionando propriedades

To add a new property, first click on Add.... Select the required property name from the combo box, or type in a name of your own choice, then enter a value in the box below. Properties which take multiple values, such as an ignore list, can be entered on multiple lines. Click on OK to add that property to the list.

If you want to apply a property to many items at once, select the files/folders in explorer, then select Context menu → properties

If you want to apply the property to *every* file and folder in the hierarchy below the current folder, check the Recursive checkbox.

Some properties, for example `svn:needs-lock`, can only be applied to files, so the property name doesn't appear in the drop down list for folders. You can still apply such a property recursively to all files in a hierarchy, but you have to type in the property name yourself.

If you wish to edit an existing property, select that property from the list of existing properties, then click on Edit....

If you wish to remove an existing property, select that property from the list of existing properties, then click on Remove.

The `svn:externals` property can be used to pull in other projects from the same repository or a completely different repository. For more information, read Seção 4.18, “Itens Externos”.

4.17.1.3. Exporting and Importing Properties

Often you will find yourself applying the same set of properties many times, for example `bugtraq:logregex`. To simplify the process of copying properties from one project to another, you can use the Export/Import feature.

From the file or folder where the properties are already set, use TortoiseSVN → properties, select the properties you wish to export and click on Export.... You will be prompted for a filename where the property names and values will be saved.

From the folder(s) where you wish to apply these properties, use TortoiseSVN → properties and click on Import.... You will be prompted for a filename to import from, so navigate to the place you saved the export file previously and select it. The properties will be added to the folders non-recursively.

If you want to add properties to a tree recursively, follow the steps above, then in the property dialog select each property in turn, click on Edit..., check the Apply property recursively box and click on OK.

The Import file format is binary and proprietary to TortoiseSVN. Its only purpose is to transfer properties using Import and Export, so there is no need to edit these files.

4.17.1.4. Propriedades Binárias

TortoiseSVN can handle binary property values using files. To read a binary property value, Save... to a file. To set a binary value, use a hex editor or other appropriate tool to create a file with the content you require, then Load... from that file.

Although binary properties are not often used, they can be useful in some applications. For example if you are storing huge graphics files, or if the application used to load the file is huge, you might want to store a thumbnail as a property so you can obtain a preview quickly.

4.17.1.5. Automatic property setting

You can configure Subversion and TortoiseSVN to set properties automatically on files and folders when they are added to the repository. There are two ways of doing this.

You can edit the subversion configuration file to enable this feature on your client. The General page of TortoiseSVN's settings dialog has an edit button to take you there directly. The config file is a simple text file which controls some of subversion's workings. You need to change two things: firstly in the section headed `miscellany` uncomment the line `enable-auto-props = yes`. Secondly you need to edit the section below to define which properties you want added to which file types. This method is a standard subversion feature and works with any subversion client. However it has to be defined on each client individually - there is no way to propagate these settings from the repository.

An alternative method is to set the `tsvn:autoprops` property on folders, as described in the next section. This method only works for TortoiseSVN clients, but it does get propagated to all working copies on update.

Whichever method you choose, you should note that auto-props are only applied to files at the time they are added to the repository. Auto-props will never change the properties of files which are already versioned.

If you want to be absolutely sure that new files have the correct properties applied, you should set up a repository pre-commit hook to reject commits where the required properties are not set.

Submeter propriedades

Subversion properties are versioned. After you change or add a property you have to commit your changes.

Conflicts on properties

If there's a conflict on committing the changes, because another user has changed the same property, Subversion generates a `.prej` file. Delete this file after you have resolved the conflict.

4.17.2. TortoiseSVN Project Properties

TortoiseSVN has a few special properties of its own, and these begin with `tsvn:`.

- `tsvn:logminsize` sets the minimum length of a log message for a commit. If you enter a shorter message than specified here, the commit is disabled. This feature is very useful for reminding you to supply a proper descriptive message for every commit. If this property is not set, or the value is zero, empty log messages are allowed.

  `tsvn:lockmsgminsize` sets the minimum length of a lock message. If you enter a shorter message than specified here, the lock is disabled. This feature is very useful for reminding you to supply a proper descriptive message for every lock you get. If this property is not set, or the value is zero, empty lock messages are allowed.

- `tsvn:logwidthmarker` is used with projects which require log messages to be formatted with some maximum width (typically 80 characters) before a line break. Setting this property to a non-zero will do 2 things in the log message entry dialog: it places a marker to indicate the maximum width, and it disables word wrap in the display, so that you can see whether the text you entered is too long. Note: this feature will only work correctly if you have a fixed-width font selected for log messages.

- `tsvn:logtemplate` is used with projects which have rules about log message formatting. The property holds a multi-line text string which will be inserted in the commit message box when you start a commit. You can then edit it to include the required information. Note: if you are also using `tsvn:logminsize`, be sure to set the length longer than the template or you will lose the protection mechanism.

- Subversion allows you to set “autoprops” which will be applied to newly added or imported files, based on the file extension. This depends on every client having set appropriate autoprops in their subversion configuration file. `tsvn:autoprops` can be set on folders and these will be merged with the user's local autoprops when importing or adding files. The format is the same as for subversion autoprops, e.g. `*.sh = svn:eol-style=native;svn:executable` sets two properties on files with the `.sh` extension.

  If there is a conflict between the local autoprops and `tsvn:autoprops`, the project settings take precedence because they are specific to that project.

- In the Commit dialog you have the option to paste in the list of changed files, including the status of each file (added, modified, etc). `tsvn:logfilelistenglish` defines whether the file status is inserted in English or in the localized language. If the property is not set, the default is `true`.

- TortoiseSVN can use spell checker modules which are also used by OpenOffice and Mozilla. If you have those installed this property will determine which spell checker to use, i.e. in which language the log messages for your project should be written. `tsvn:projectlanguage` sets the language module the spell checking engine should use when you enter a log message. You can find the values for your language on this page: *MSDN: Language Identifiers* [http://msdn2.microsoft.com/en-us/library/ms776260.aspx].

  You can enter this value in decimal, or in hexadecimal if prefixed with `0x`. For example English (US) can be entered as `0x0409` or `1033`.

- The property `tsvn:logsummary` is used to extract a portion of the log message which is then shown in the log dialog as the log message summary.

  The value of the `tsvn:logsummary` property must be set to a one line regex string which contains one regex group. Whatever matches that group is used as the summary.

  An example: `\[SUMMARY\]:\s+(.*)` Will catch everything after “[SUMMARY]” in the log message and use that as the summary.

- When you want to add a new property, you can either pick one from the list in the combo box, or you can enter any property name you like. If your project uses some custom properties, and you want

those properties to appear in the list in the combo box (to avoid typos when you enter a property name), you can create a list of your custom properties using `tsvn:userfileproperties` and `tsvn:userdirproperties`. Apply these properties to a folder. When you go to edit the properties of any child item, your custom properties will appear in the list of pre-defined property names.

Some `tsvn:` properties require a `true/false` value. TortoiseSVN also understands `yes` as a synonym for `true` and `no` as a synonym for `false`.

TortoiseSVN can integrate with some bug tracking tools. This uses project properties that start with `bugtraq:`. Read Seção 4.28, “Integration with Bug Tracking Systems / Issue Trackers” for further information.

It can also integrate with some web-based repository browsers, using project properties that start with `webviewer:`. Read Seção 4.29, “Integration with Web-based Repository Viewers” for further information.

Set the project properties on folders

These special project properties must be set on *folders* for the system to work. When you commit a file or folder the properties are read from that folder. If the properties are not found there, TortoiseSVN will search upwards through the folder tree to find them until it comes to an unversioned folder, or the tree root (eg. `C:\`) is found. If you can be sure that each user checks out only from e.g `trunk/` and not some sub-folder, then it is sufficient to set the properties on `trunk/`. If you can't be sure, you should set the properties recursively on each sub-folder. A property setting deeper in the project hierarchy overrides settings on higher levels (closer to `trunk/`).

For project properties *only* you can use the Recursive checkbox to set the property to all sub-folders in the hierarchy, without also setting it on all files.

When you add new sub-folders using TortoiseSVN, any project properties present in the parent folder will automatically be added to the new child folder too.

Cuidado

Although TortoiseSVN's project properties are extremely useful, they only work with TortoiseSVN, and some will only work in newer versions of TortoiseSVN. If people working on your project use a variety of Subversion clients, or possibly have old versions of TortoiseSVN, you may want to use repository hooks to enforce project policies. project properties can only help to implement a policy, they cannot enforce it.

4.18. Itens Externos

Por vezes é útil construir uma cópia de trabalho feita a partir de diferentes checkouts. Por exemplo, você poderá querer diferentes arquivos ou sub-diretórios de diferentes locais de um repositório ou talvez, de diferentes repositórios. Se você quiser que todo usuário tenha o mesmo layout, você pode definir as propriedades `svn:externals` para incluir o recurso especificado nos locais onde são necessários.

4.18.1. Diretórios Externos

Vamos assumir que tenha exportado uma cópia de trabalho do `/project1` para `D:\dev\project1`. Selecione a pasta `D:\dev\project1`, clique com o botão direito e escolha Menu do Windows → Propriedades a partir do menu de contexto. Aparecerá a caixa de diálogo Propriedades. Em seguida vá à aba Subversion, onde poderá ajustar as propriedades. Clique Adicionar.... Selecione a propriedade

`svn:externals` a partir da combobox e digite na caixa de edição o URL do repositório no formato `url folder` ou, se quiser especificar uma revisão em particular, digite `-rREV url folder`. Você pode adicionar múltiplos projetos externos, 1 por linha. Suponha que tenha ajustado essas propriedades no `D:\dev\project1`:

```
http://sounds.red-bean.com/repos  sounds
http://graphics.red-bean.com/repos/fast%20graphics  "quick graphs"
-r21 http://svn.red-bean.com/repos/skin-maker  skins/toolkit
```

Agora clique Fixar e submeta suas alterações. Quando você (ou outro usuário) atualizar a cópia de trabalho, o Subversion irá criar um sub-diretório `D:\dev\project1\sounds` e exportará o projeto "sounds", outro sub-diretório `D:\dev\project1\quick_graphs` contendo o projeto "graphics" e finalmente um sub-diretório encadeado `D:\dev\project1\skins\toolkit` contendo a revisão 21 do projeto "skin-maker".

As URLs teem que ser "escapados" apropriadamente ou não irão funcionar - por exemplo, deverá substituir cada "espaço" com `%20`, como exibido no segundo exemplo acima.

Se você quiser que o caminho local inclua "espaços" ou outro caracter especial, terá que incluí-lo entre aspas ou usar o caracter `\` (barra invertida) como um caracter de escape, ao estilo da linha de comandos do Linux, precedendo cada caracter especial. É claro que isto também significa que terá que necessariamente usar `/` (barra inclinada) como delimitador de caminho. Note que este comportamento é inusitado no Subversion 1.6 e não funcionará em versões mais antigas.

Usar números explícitos de revisão

Você deve considerar, seriamente, o uso de números de revisão explícitos em todas as suas definições externas, como descrito acima. Assim fazendo, significa que terá que decidir qual será o momento para puxar uma diferente "cópia" da informação externa e exatamente qual "cópia" deverá puxar. Além do aspecto do senso comum de não ser surpreendido por alterações nos repositórios de terceiros sobre os quais não pode controlar, ao usar números de revisões explícitos também significa que ao reverter sua cópia de trabalho para uma revisão anterior, suas definições externas também irão reverter para o estado que tinham nessa prévia revisão que por sua vez, significa que as cópias de trabalho externas serão atualizadas para corresponder o estado que *elas* tinham quando seu repositório estava naquela prévia revisão. Para projetos de software isto pode ser a diferença entre uma versão com falhas ou não, de uma "cópia" antiga de sua complexa base de códigos.

Antigas definições de svn:externals

O formato mostrado aqui foi introduzido no Subversion 1.5. Você também pode ver no formato antigo, que tem a mesma informação, numa ordem diferente. O novo formato é preferível uma vez que suporta várias funcionalidades úteis descritas abaixo, mas não irá funcionar em versões antigas. As diferenças são exibidas em *Subversion Book* [http://svnbook.red-bean.com/en/1.5/svn.advanced.externals.html].

If the external project is in the same repository, any changes you make there there will be included in the commit list when you commit your main project.

Se o projeto externo estiver num repositório diferente, quaisquer alterações que fizer serão notificadas quando submeter o projeto principal mas terá que submeter essas alterações separadamente.

Se usar URLs absolutas nas definições `svn:externals` e tiver que realocar sua cópia de trabalho (ou seja, a URL do seu repositório é alterada), então seus "externos" não serão alterados e poderão deixar de funcionar.

Para evitar tais problemas, a versão 1.5 ou superior do cliente Subversion, suporta URLs externas relativas. Quatro diferentes métodos de especificar uma URL relativa são suportados. Nos exemplos seguintes assume-se que temos dois repositórios: um em `http://example.com/svn/repos-1` e outro em `http://example.com/svn/repos-2`. Temos um SVN exportado do `http://example.com/svn/repos-1/project/trunk` em `C:\Working` e a propriedade `svn:externals` está configurada no trunk.

Relativo ao diretório pai
: Essas URLs sempre iniciam com `../`, por exemplo:

```
../../widgets/foo  common/foo-widget
```

. Esse comando irá extrair `http://example.com/svn/repos-1/widgets/foo` em `C:\Working\common\foo-widget`.

Note que a URL é relativa a URL do diretório com a propriedade `svn:externals` e não a URL do diretório onde o arquivo externo está armazenado.

Relativo ao diretório principal do repositório
: Essas URLs sempre iniciam com `^/`, por exemplo:

```
^/widgets/foo  common/foo-widget
```

. Esse comando irá extrair `http://example.com/svn/repos-1/widgets/foo` em `C:\Working\common\foo-widget`.

Você poderá facilmente fazer referência a outros repositórios com o mesmo `SVNParentPath` (um diretório comum que contém vários repositórios). Por exemplo:

```
^/../repos-2/hammers/claw  common/claw-hammer
```

Esse comando irá extrair `http://example.com/svn/repos-2/hammers/claw` em `C:\Working\common\claw-hammer`.

Relativo ao esquema
: URLs que iniciam com `//` copiam só a parte do esquema da URL. Isto é útil quando o mesmo "hostname" necessita ser acessado com esquemas diferentes - dependendo da localização na rede; por exemplo, clientes na intranet usam `http://`, enquanto clientes externos usam `svn+ssh://`. Por exemplo:

```
//example.com/svn/repos-1/widgets/foo  common/foo-widget
```

. Esse comando irá extrair `http://example.com/svn/repos-1/widgets/foo` ou `svn+ssh://example.com/svn/repos-1/widgets/foo` dependendo qual método foi utilizado para exportar (checkout) `C:\Working`.

Relativo ao nome do servidor
: URLs que iniciam com `/` copiam a parte da URL do esquema e do "hostname", por exemplo:

```
/svn/repos-1/widgets/foo  common/foo-widget
```

. Esse comando irá extrair `http://example.com/svn/repos-1/widgets/foo` em `C:\Working\common\foo-widget`. Mas se você exportar sua cópia de trabalho desde outro servidor em `svn+ssh://another.mirror.net/svn/repos-1/project1/`

`trunk` então, a referência externa irá extrair `svn+ssh://another.mirror.net/svn/repos-1/widgets/foo`.

Você também pode especificar uma revisão "peg" depois da URL, se necessário; por exemplo, `http://sounds.red-bean.com/repos@19`.

Caso necessite mais informações sobre como TortoiseSVN lida com "Propriedades", consulte Seção 4.17, “Configurações do Projeto”.

Para saber mais sobre diferentes métodos de acesso a subprojetos comuns, consulte Seção B.6, “Include a common sub-project”.

4.18.2. Arquivos Externos

Desde a versão 1.6 do Subversion você pode incluir um único arquivo externo em sua cópia de trabalho usando a mesma sintaxe que usaria para diretórios. Entretanto existem algumas restrições.

- O caminho para o arquivo externo deverá colocar o arquivo num diretório existente e versionado. Em geral, faz mais sentido colocar o arquivo diretamente no diretório configurado com `svn:externals` mas se necessário, também poderá ser colocado num sub-diretório versionado. Por contraste, diretórios externos irão criar automaticamente qualquer pasta intermediária não versionada (se assim for requerido).
- A URL para um arquivo externo deverá estar no mesmo repositório que a URL na qual o arquivo externo será incluído; arquivos externos inter-repositórios não são suportados.

Em vários aspectos um arquivo externo comporta-se como qualquer outro arquivo versionado, mas eles não podem ser movidos ou eliminados usando os comandos normais; é obrigatório que a propriedade `svn:externals` seja modificada.

O suporte para arquivos externos é incompleto na versão 1.6 do Subversion

Na versão 1.6 do Subversion, uma vez que tenha adicionado um arquivo externo, não é possível removê-lo da sua cópia de trabalho - ainda que também remova a propriedade `svn:externals`. Você precisará realizar a obtenção (checkout) de uma nova cópia de trabalho para remover o arquivo.

4.19. Ramificando / Rotulando

One of the features of version control systems is the ability to isolate changes onto a separate line of development. This line is known as a *branch*. Branches are often used to try out new features without disturbing the main line of development with compiler errors and bugs. As soon as the new feature is stable enough then the development branch is *merged* back into the main branch (trunk).

Another feature of version control systems is the ability to mark particular revisions (e.g. a release version), so you can at any time recreate a certain build or environment. This process is known as *tagging*.

Subversion does not have special commands for branching or tagging, but uses so-called “cheap copies” instead. Cheap copies are similar to hard links in Unix, which means that instead of making a complete copy in the repository, an internal link is created, pointing to a specific tree/revision. As a result branches and tags are very quick to create, and take up almost no extra space in the repository.

4.19.1. Criando um Ramo ou Rótulo

If you have imported your project with the recommended directory structure, creating a branch or tag version is very simple:

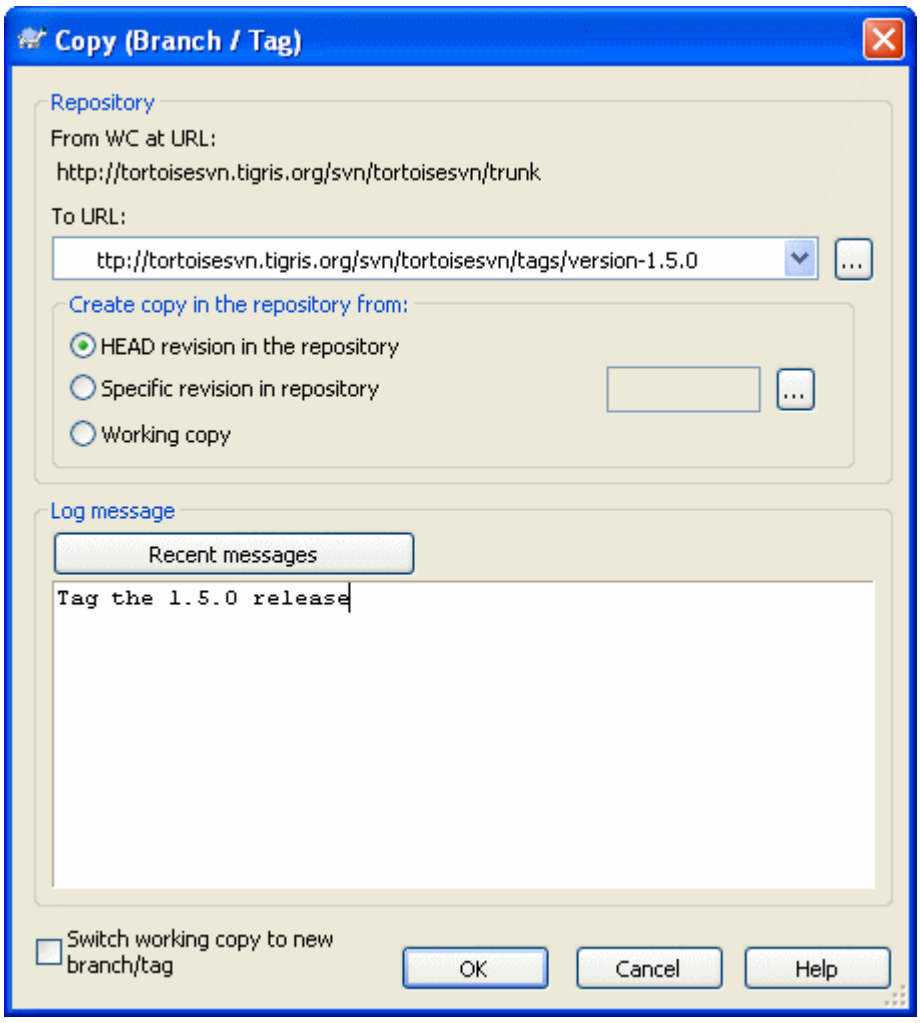


Figura 4.33. A Janela de Ramificação/Rotulação

Select the folder in your working copy which you want to copy to a branch or tag, then select the command TortoiseSVN → Branch/Tag....

The default destination URL for the new branch will be the source URL on which your working copy is based. You will need to edit that URL to the new path for your branch/tag. So instead of

```
http://svn.collab.net/repos/ProjectName/trunk
```

you might now use something like

```
http://svn.collab.net/repos/ProjectName/tags/Release_1.10
```

If you can't remember the naming convention you used last time, click the button on the right to open the repository browser so you can view the existing repository structure.

Now you have to select the source of the copy. Here you have three options:

Última revisão no repositório
: The new branch is copied directly in the repository from the HEAD revision. No data needs to be transferred from your working copy, and the branch is created very quickly.

Revisão específica no repositório
: The new branch is copied directly in the repository but you can choose an older revision. This is useful if you forgot to make a tag when you released your project last week. If you can't remember the revision number, click the button on the right to show the revision log, and select the revision number from there. Again no data is transferred from your working copy, and the branch is created very quickly.

Cópia de trabalho
: The new branch is an identical copy of your local working copy. If you have updated some files to an older revision in your WC, or if you have made local changes, that is exactly what goes into the copy. Naturally this sort of complex tag may involve transferring data from your WC back to the repository if it does not exist there already.

If you want your working copy to be switched to the newly created branch automatically, use the Switch working copy to new branch/tag checkbox. But if you do that, first make sure that your working copy does not contain modifications. If it does, those changes will be merged into the branch WC when you switch.

Press OK to commit the new copy to the repository. Don't forget to supply a log message. Note that the copy is created *inside the repository*.

Note that unless you opted to switch your working copy to the newly created branch, creating a Branch or Tag does *not* affect your working copy. Even if you create the branch from your WC, those changes are committed to the new branch, not to the trunk, so your WC may still be marked as modified with respect to the trunk.

4.19.2. Para Obter ou Alternar

...that is (not really) the question. While a checkout downloads everything from the desired branch in the repository to your working directory, TortoiseSVN → Switch... only transfers the changed data to your working copy. Good for the network load, good for your patience. :-)

To be able to work with your freshly generated branch or tag you have several ways to handle it. You can:

- TortoiseSVN → Checkout to make a fresh checkout in an empty folder. You can check out to any location on your local disk and you can create as many working copies from your repository as you like.
- Switch your current working copy to the newly created copy in the repository. Again select the top level folder of your project and use TortoiseSVN → Switch... from the context menu.

  In the next dialog enter the URL of the branch you just created. Select the Head Revision radio button and click on OK. Your working copy is switched to the new branch/tag.

  Switch works just like Update in that it never discards your local changes. Any changes you have made to your working copy which have not yet been committed will be merged when you do the Switch. If you do not want this to happen then you must either commit the changes before switching, or revert your working copy to an already-committed revision (typically HEAD).
- If you want to work on trunk and branch, but don't want the expense of a fresh checkout, you can use Windows Explorer to make a copy of your trunk checkout in another folder, then TortoiseSVN → Switch... that copy to your new branch.

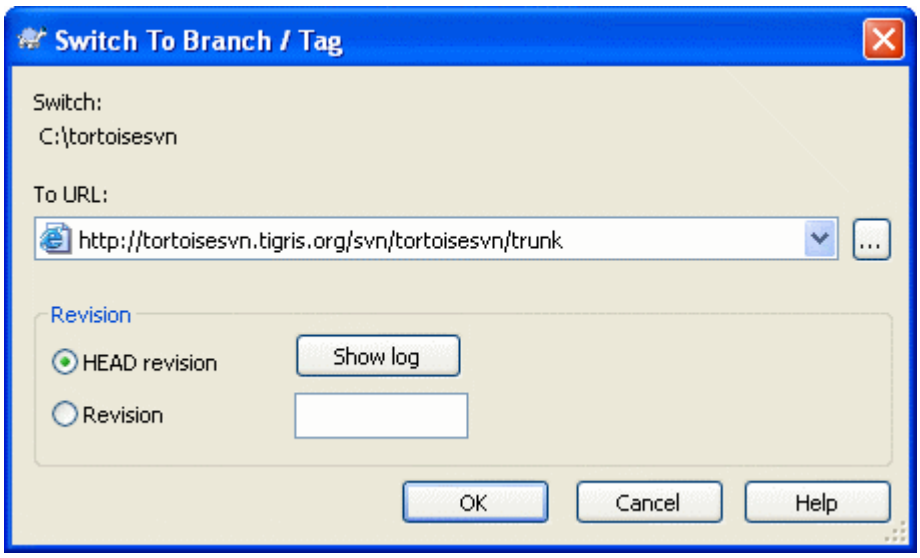


Figura 4.34. A Janela de Troca

Although Subversion itself makes no distinction between tags and branches, the way they are typically used differs a bit.

- Tags are typically used to create a static snapshot of the project at a particular stage. As such they not normally used for development - that's what branches are for, which is the reason we recommended the `/trunk /branches /tags` repository structure in the first place. Working on a tag revision is *not a good idea*, but because your local files are not write protected there is nothing to stop you doing this by mistake. However, if you try to commit to a path in the repository which contains `/tags/`, TortoiseSVN will warn you.
- It may be that you need to make further changes to a release which you have already tagged. The correct way to handle this is to create a new branch from the tag first and commit the branch. Do your Changes on this branch and then create a new tag from this new branch, e.g. `Version_1.0.1`.
- If you modify a working copy created from a branch and commit, then all changes go to the new branch and *not* the trunk. Only the modifications are stored. The rest remains a cheap copy.

4.20. Unificando

Where branches are used to maintain separate lines of development, at some stage you will want to merge the changes made on one branch back into the trunk, or vice versa.

It is important to understand how branching and merging works in Subversion before you start using it, as it can become quite complex. It is highly recommended that you read the chapter *Branching and Merging* [http://svnbook.red-bean.com/en/1.5/svn.branchmerge.html] in the Subversion book, which gives a full description and many examples of how it is used.

The next point to note is that merging *always* takes place within a working copy. If you want to merge changes *into* a branch, you have to have a working copy for that branch checked out, and invoke the merge wizard from that working copy using TortoiseSVN → Merge....

In general it is a good idea to perform a merge into an unmodified working copy. If you have made other changes in your WC, commit those first. If the merge does not go as you expect, you may want to revert the changes, and the Revert command will discard *all* changes including any you made before the merge.

There are three common use cases for merging which are handled in slightly different ways, as described below. The first page of the merge wizard asks you to select the method you need.

Merge a range of revisions
: This method covers the case when you have made one or more revisions to a branch (or to the trunk) and you want to port those changes across to a different branch.

  What you are asking Subversion to do is this: “Calculate the changes necessary to get [FROM] revision 1 of branch A [TO] revision 7 of branch A, and apply those changes to my working copy (of trunk or branch B).”

Reintegrar um ramo
: This method covers the case when you have made a feature branch as discussed in the Subversion book. All trunk changes have been ported to the feature branch, week by week, and now the feature is complete you want to merge it back into the trunk. Because you have kept the feature branch synchronized with the trunk, the latest versions of branch and trunk will be absolutely identical except for your branch changes.

  This is a special case of the tree merge described below, and it requires only the URL to merge from (normally) your development branch. It uses the merge-tracking features of Subversion to calculate the correct revision ranges to use, and perform additional checks which ensure that the branch has been fully updated with trunk changes. This ensures that you don't accidentally undo work that others have committed to trunk since you last synchronized changes.

  After the merge, all branch development has been completely merged back into the main development line. The branch is now redundant and can be deleted.

  Once you have performed a reintegrate merge you should not continue to use it for development. The reason for this is that if you try to resynchronize your existing branch from trunk later on, merge tracking will see your reintegration as a trunk change that has not yet been merged into the branch, and will try to merge the branch-to-trunk merge back into the branch! The solution to this is simply to create a new branch from trunk to continue the next phase of your development.

Unificar duas árvores diferentes
: This is a more general case of the reintegrate method. What you are asking Subversion to do is: “Calculate the changes necessary to get [FROM] the head revision of the trunk [TO] the head revision of the branch, and apply those changes to my working copy (of the trunk).” The net result is that trunk now looks exactly like the branch.

  If your server/repository does not support merge-tracking then this is the only way to merge a branch back to trunk. Another use case occurs when you are using vendor branches and you need to merge the changes following a new vendor drop into your trunk code. For more information read the chapter on *vendor branches* [http://svnbook.red-bean.com/en/1.5/svn.advanced.vendorbr.html] in the Subversion Book.

4.20.1. Unificar um Intervalo de Revisões

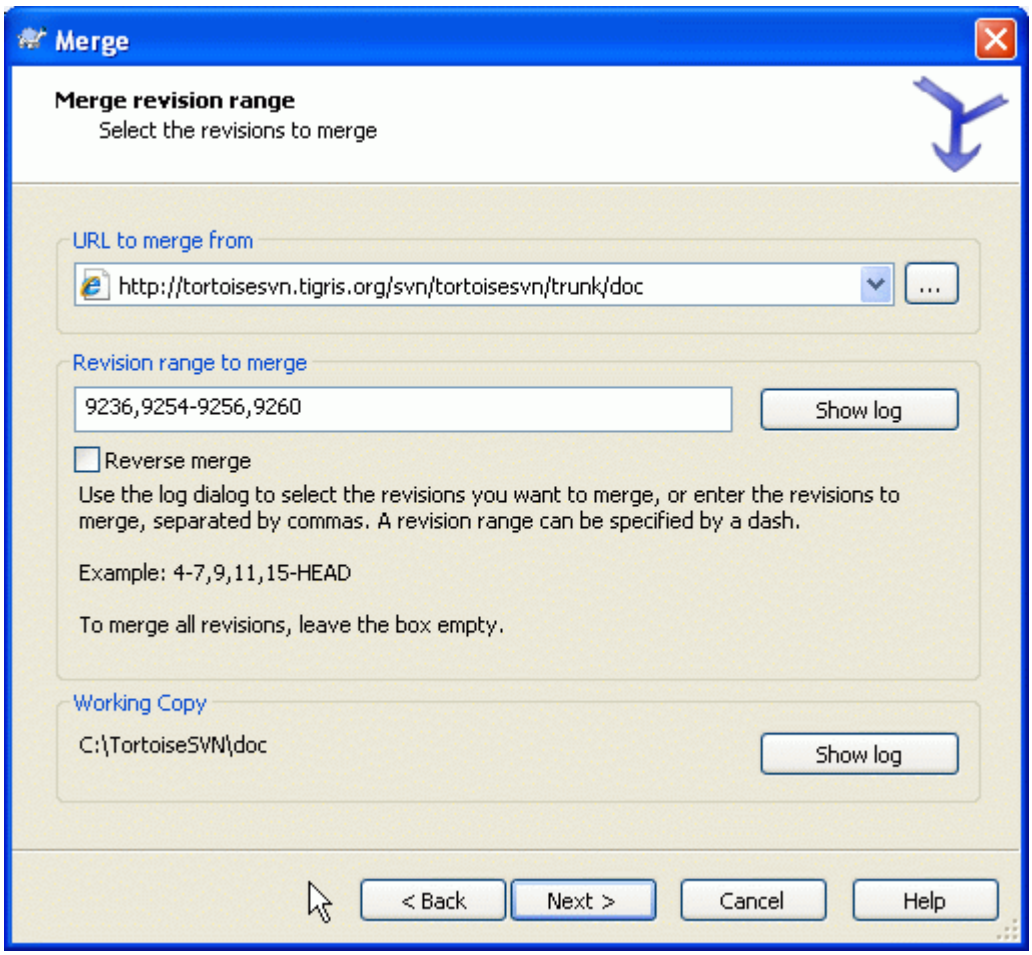


Figura 4.35. The Merge Wizard - Select Revision Range

In the From: field enter the full folder URL of the branch or tag containing the changes you want to port into your working copy. You may also click ... to browse the repository and find the desired branch. If you have merged from this branch before, then just use the drop down list which shows a history of previously used URLs.

In the Revision range to merge field enter the list of revisions you want to merge. This can be a single revision, a list of specific revisions separated by commas, or a range of revisions separated by a dash, or any combination of these.

Importante

There is an important difference in the way a revision range is specified with TortoiseSVN compared to the command line client. The easiest way to visualise it is to think of a fence with posts and fence panels.

With the command line client you specify the changes to merge using two “fence post” revisions which specify the *before* and *after* points.

With TortoiseSVN you specify the changeset to merge using “fence panels”. The reason for this becomes clear when you use the log dialog to specify revisions to merge, where each revision appears as a changeset.

If you are merging revisions in chunks, the method shown in the subversion book will have you merge 100-200 this time and 200-300 next time. With TortoiseSVN you would merge 100-200 this time and 201-300 next time.

> This difference has generated a lot of heat on the mailing lists. We acknowledge that there is a difference from the command line client, but we believe that for the majority of GUI users it is easier to understand the method we have implemented.

The easiest way to select the range of revisions you need is to click on Show Log, as this will list recent changes with their log comments. If you want to merge the changes from a single revision, just select that revision. If you want to merge changes from several revisions, then select that range (using the usual **Shift**-modifier). Click on OK and the list of revision numbers to merge will be filled in for you.

If you want to merge changes back *out* of your working copy, to revert a change which has already been committed, select the revisions to revert and make sure the Reverse merge box is checked.

If you have already merged some changes from this branch, hopefully you will have made a note of the last revision merged in the log message when you committed the change. In that case, you can use Show Log for the Working Copy to trace that log message. Remembering that we are thinking of revisions as changesets, you should Use the revision after the end point of the last merge as the start point for this merge. For example, if you have merged revisions 37 to 39 last time, then the start point for this merge should be revision 40.

If you are using the merge tracking features of Subversion, you do not need to remember which revisions have already been merged - Subversion will record that for you. If you leave the revision range blank, all revisions which have not yet been merged will be included. Read Seção 4.20.6, “Histórico de combinações” to find out more.

If other people may be committing changes then be careful about using the HEAD revision. It may not refer to the revision you think it does if someone else made a commit after your last update.

Click Next and go to Seção 4.20.4, “Opções de Combinação”

4.20.2. Reintegrar um ramo

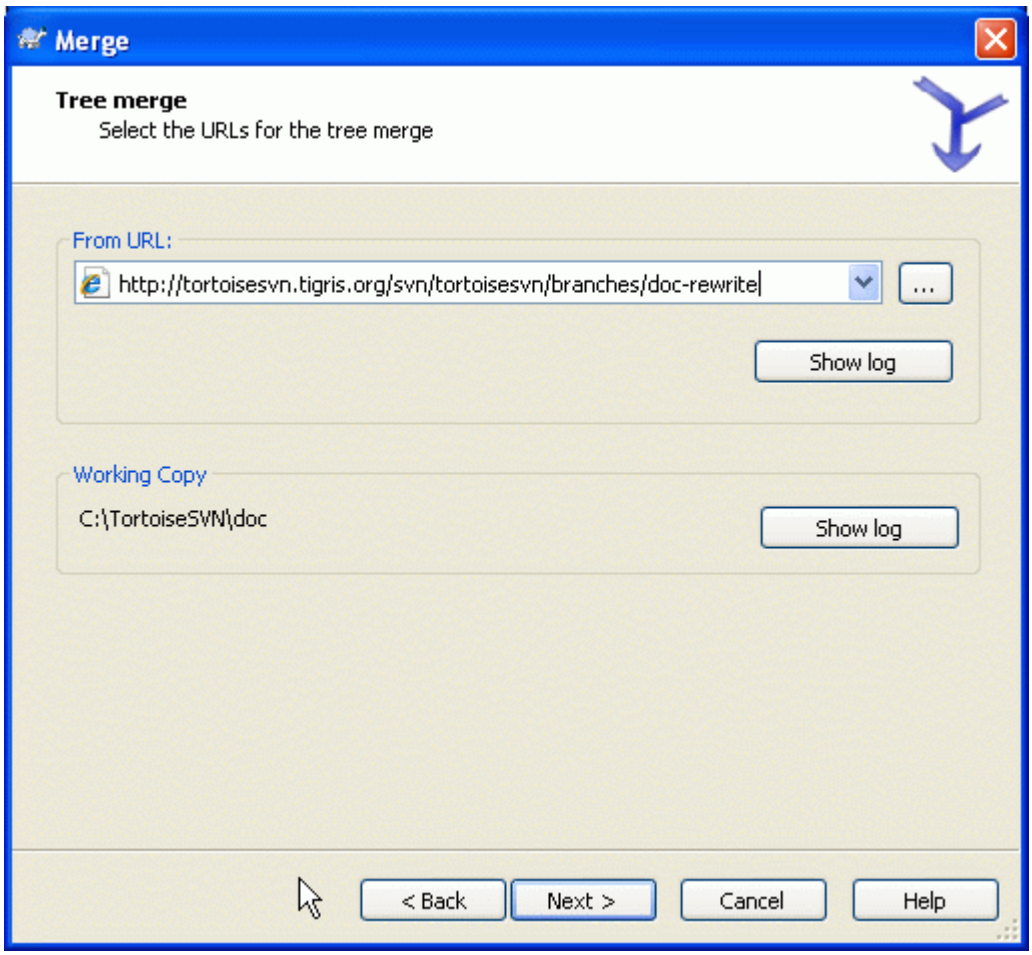


Figura 4.36. The Merge Wizard - Reintegrate Merge

To merge a feature branch back into the trunk you must start the merge wizard from within a working copy of the trunk.

In the From URL: field enter the full folder URL of the branch that you want to merge back. You may also click ... to browse the repository.

There are some conditions which apply to a reintegrate merge. Firstly, the server must support merge tracking. The working copy must be of depth infinite (no sparse checkouts), and it must not have any local modifications, switched items or items that have been updated to revisions other than HEAD. All changes to trunk made during branch development must have been merged across to the branch (or marked as having been merged). The range of revisions to merge will be calculated automatically.

4.20.3. Combinando Duas Árvores Diferentes

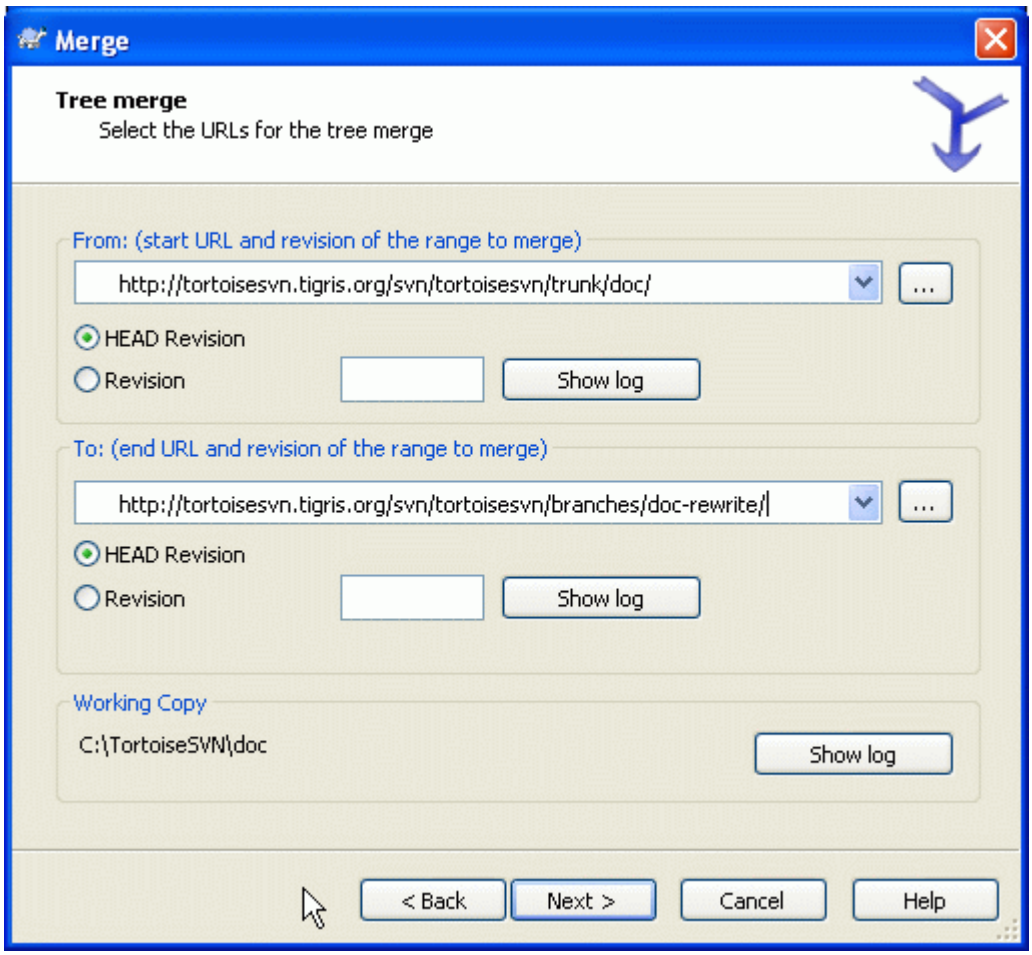


Figura 4.37. The Merge Wizard - Tree Merge

If you are using this method to merge a feature branch back to trunk, you need to start the merge wizard from within a working copy of trunk.

In the From: field enter the full folder URL of the *trunk*. This may sound wrong, but remember that the trunk is the start point to which you want to add the branch changes. You may also click ... to browse the repository.

In the To: field enter the full folder URL of the feature branch.

In both the From Revision field and the To Revision field, enter the last revision number at which the two trees were synchronized. If you are sure no-one else is making commits you can use the HEAD revision in both cases. If there is a chance that someone else may have made a commit since that synchronization, use the specific revision number to avoid losing more recent commits.

You can also use Show Log to select the revision.

4.20.4. Opções de Combinação

This page of the wizard lets you specify advanced options, before starting the merge process. Most of the time you can just use the default settings.

You can specify the depth to use for the merge, i.e. how far down into your working copy the merge should go. The depth terms used are described in Seção 4.3.1, “Profundidade da Obtenção”. The default depth is Working copy, which uses the existing depth setting, and is almost always what you want.

Most of the time you want merge to take account of the file's history, so that changes relative to a common ancestor are merged. Sometimes you may need to merge files which are perhaps related, but not in

your repository. For example you may have imported versions 1 and 2 of a third party library into two separate directories. Although they are logically related, Subversion has no knowledge of this because it only sees the tarballs you imported. If you attempt to merge the difference between these two trees you would see a complete removal followed by a complete add. To make Subversion use only path-based differences rather than history-based differences, check the Ignore ancestry box. Read more about this topic in the Subversion book, *Noticing or Ignoring Ancestry* [http://svnbook.red-bean.com/en/1.5/svn.branchmerge.advanced.html#svn.branchmerge.advanced.ancestry]

You can specify the way that line ending and whitespace changes are handled. These options are described in Seção 4.10.2, “Line-end and Whitespace Options”. The default behaviour is to treat all whitespace and line-end differences as real changes to be merged.

If you are using merge tracking and you want to mark a revision as having been merged, without actually doing the merge here, check the Only record the merge checkbox. There are two possible reasons you might want to do this. It may be that the merge is too complicated for the merge algorithms, so you code the changes by hand, then mark the change as merged so that the merge tracking algorithm is aware of it. Or you might want to prevent a particular revision from being merged. Marking it as already merged will prevent the merge occurring with merge-tracking-aware clients.

Now everything is set up, all you have to do is click on the Merge button. If you want to preview the results Test Merge performs the merge operation, but does *not* modify the working copy at all. It shows you a list of the files that will be changed by a real merge, and notes those areas where conflicts will occur.

The merge progress dialog shows each stage of the merge, with the revision ranges involved. This may indicate one more revision than you were expecting. For example if you asked to merge revision 123 the progress dialog will report “Merging revisions 122 through 123”. To understand this you need to remember that Merge is closely related to Diff. The merge process works by generating a list of differences between two points in the repository, and applying those differences to your working copy. The progress dialog is simply showing the start and end points for the diff.

4.20.5. Reviewing the Merge Results

The merge is now complete. It's a good idea to have a look at the merge and see if it's as expected. Merging is usually quite complicated. Conflicts often arise if the branch has drifted far from the trunk.

For Subversion clients and servers prior to 1.5, no merge information is stored and merged revisions have to be tracked manually. When you have tested the changes and come to commit this revision, your commit log message should *always* include the revision numbers which have been ported in the merge. If you want to apply another merge at a later time you will need to know what you have already merged, as you do not want to port a change more than once. For more information about this, refer to *Best Practices for Merging* [http://svnbook.red-bean.com/en/1.4/svn.branchmerge.copychanges.html#svn.branchmerge.copychanges.bestprac] in the Subversion book.

If your server and all clients are running Subversion 1.5 or higher, the merge tracking facility will record the revisions merged and avoid a revision being merged more than once. This makes your life much simpler as you can simply merge the entire revision range each time and know that only new revisions will actually be merged.

Branch management is important. If you want to keep this branch up to date with the trunk, you should be sure to merge often so that the branch and trunk do not drift too far apart. Of course, you should still avoid repeated merging of changes, as explained above.

Dica

If you have just merged a feature branch back into the trunk, the trunk now contains all the new feature code, and the branch is obsolete. You can now delete it from the repository if required.

Importante

Subversion can't merge a file with a folder and vice versa - only folders to folders and files to files. If you click on a file and open up the merge dialog, then you have to give a path to a file in that dialog. If you select a folder and bring up the dialog, then you must specify a folder URL for the merge.

4.20.6. Histórico de combinações

Subversion 1.5 introduced facilities for merge tracking. When you merge changes from one tree into another, the revision numbers merged are stored and this information can be used for several different purposes.

- You can avoid the danger of merging the same revision twice (repeated merge problem). Once a revision is marked as having been merged, future merges which include that revision in the range will skip over it.
- When you merge a branch back into trunk, the log dialog can show you the branch commits as part of the trunk log, giving better traceability of changes.
- When you show the log dialog from within the merge dialog, revisions already merged are shown in grey.
- When showing blame information for a file, you can choose to show the original author of merged revisions, rather than the person who did the merge.
- You can mark revisions as *do not merge* by including them in the list of merged revisions without actually doing the merge.

Merge tracking information is stored in the `svn:mergeinfo` property by the client when it performs a merge. When the merge is committed the server stores that information in a database, and when you request merge, log or blame information, the server can respond appropriately. For the system to work properly you must ensure that the server, the repository and all clients are upgraded. Earlier clients will not store the `svn:mergeinfo` property and earlier servers will not provide the information requested by new clients.

Find out more about merge tracking from Subversion's *Merge tracking documentation* [http://subversion.tigris.org/merge-tracking/index.html].

4.20.7. Handling Conflicts during Merge

Merging does not always go smoothly. Sometimes there is a conflict, and if you are merging multiple ranges, you generally want to resolve the conflict before merging of the next range starts. TortoiseSVN helps you through this process by showing the *merge conflict callback* dialog.

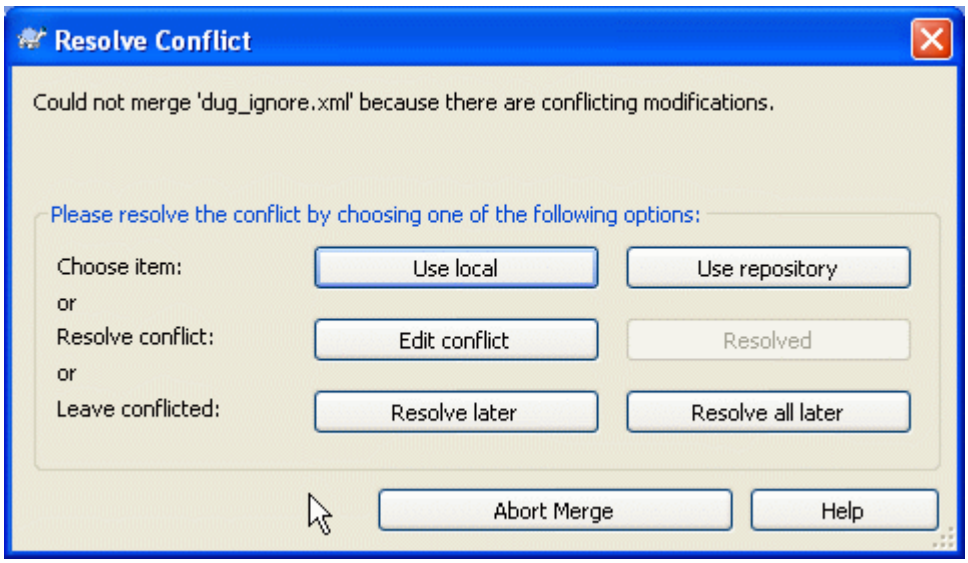


Figura 4.38. The Merge Conflict Callback Dialog

When a conflict occurs during the merge, you have three ways to handle it.

1. You may decide that your local changes are much more important, so you want to discard the version from the repository and keep your local version. Or you might discard your local changes in favour of the repository version. Either way, no attempt is made to merge the changes - you choose one or the other.

2. Normally you will want to look at the conflicts and resolve them. In that case, choose the Edit Conflict which will start up your merge tool. When you are satisfied with the result, click Resolved.

3. The last option is to postpone resolution and continue with merging. You can choose to do that for the current conflicted file, or for all files in the rest of the merge. However, if there are further changes in that file, it will not be possible to complete the merge.

If you do not want to use this interactive callback, there is a checkbox in the merge progress dialog Merge non-interactive. If this is set for a merge and the merge would result in a conflict, the file is marked as in conflict and the merge goes on. You will have to resolve the conflicts after the whole merge is finished. If it is not set, then before a file is marked as conflicted you get the chance to resolve the conflict *during* the merge. This has the advantage that if a file gets multiple merges (multiple revisions apply a change to that file), subsequent merges might succeed depending on which lines are affected. But of course you can't walk away to get a coffee while the merge is running ;)

4.20.8. Merge a Completed Branch

If you want to merge all changes from a feature branch back to trunk, then you can use the TortoiseSVN → Merge reintegrate... from the extended context menu (hold down the **Shift** key while you right click on the file).

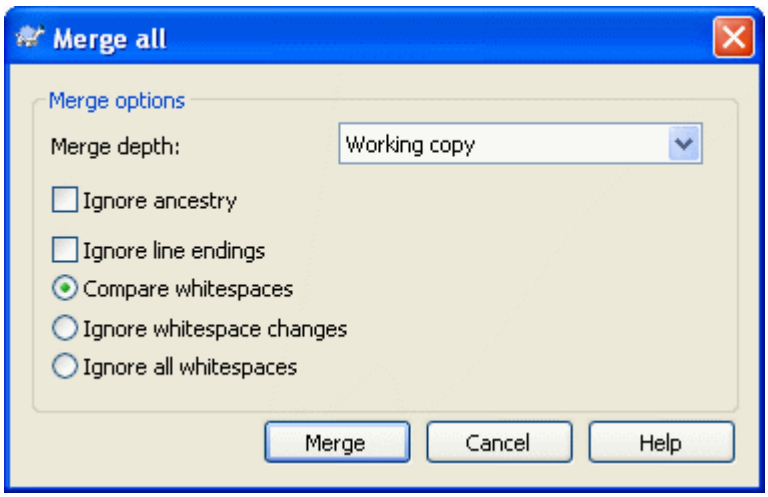


Figura 4.39. The Merge reintegrate Dialog

This dialog is very easy. All you have to do is set the options for the merge, as described in Seção 4.20.4, “Opções de Combinação”. The rest is done by TortoiseSVN automatically using merge tracking.

4.20.9. Feature Branch Maintenance

When you develop a new feature on a separate branch it is a good idea to work out a policy for re-integration when the feature is complete. If other work is going on in `trunk` at the same time you may find that the differences become significant over time, and merging back becomes a nightmare.

If the feature is relatively simple and development will not take long then you can adopt a simple approach, which is to keep the branch entirely separate until the feature is complete, then merge the branch changes back into trunk. In the merge wizard this would be a simple Merge a range of revisions, with the revision range being the revision span of the branch.

If the feature is going to take longer and you need to account for changes in `trunk`, then you need to keep the branch synchronised. This simply means that periodically you merge trunk changes into the branch, so that the branch contains all the trunk changes *plus* the new feature. The synchronisation process uses Merge a range of revisions. When the feature is complete then you can merge it back to `trunk` using either Reintegrate a branch or Merge two different trees.

4.21. Bloqueando

Subversion generally works best without locking, using the “Copy-Modify-Merge” methods described earlier in Seção 2.2.3, “A solução Copiar-Modificar-Unificar”. However there are a few instances when you may need to implement some form of locking policy.

- You are using “unmergeable” files, for example, graphics files. If two people change the same file, merging is not possible, so one of you will lose their changes.
- Your company has always used a locking revision control system in the past and there has been a management decision that “locking is best”.

Firstly you need to ensure that your Subversion server is upgraded to at least version 1.2. Earlier versions do not support locking at all. If you are using `file://` access, then of course only your client needs to be updated.

4.21.1. How Locking Works in Subversion

By default, nothing is locked and anyone who has commit access can commit changes to any file at any time. Others will update their working copies periodically and changes in the repository will be merged with local changes.

If you *Get a Lock* on a file, then only you can commit that file. Commits by all other users will be blocked until you release the lock. A locked file cannot be modified in any way in the repository, so it cannot be deleted or renamed either, except by the lock owner.

However, other users will not necessarily know that you have taken out a lock. Unless they check the lock status regularly, the first they will know about it is when their commit fails, which in most cases is not very useful. To make it easier to manage locks, there is a new Subversion property `svn:needs-lock`. When this property is set (to any value) on a file, whenever the file is checked out or updated, the local copy is made read-only *unless* that working copy holds a lock for the file. This acts as a warning that you should not edit that file unless you have first acquired a lock. Files which are versioned and read-only are marked with a special overlay in TortoiseSVN to indicate that you need to acquire a lock before editing.

Locks are recorded by working copy location as well as by owner. If you have several working copies (at home, at work) then you can only hold a lock in *one* of those working copies.

If one of your co-workers acquires a lock and then goes on holiday without releasing it, what do you do? Subversion provides a means to force locks. Releasing a lock held by someone else is referred to as *Breaking* the lock, and forcibly acquiring a lock which someone else already holds is referred to as *Stealing* the lock. Naturally these are not things you should do lightly if you want to remain friends with your co-workers.

Locks are recorded in the repository, and a lock token is created in your local working copy. If there is a discrepancy, for example if someone else has broken the lock, the local lock token becomes invalid. The repository is always the definitive reference.

4.21.2. Obtendo uma trava

Select the file(s) in your working copy for which you want to acquire a lock, then select the command TortoiseSVN → Get Lock....

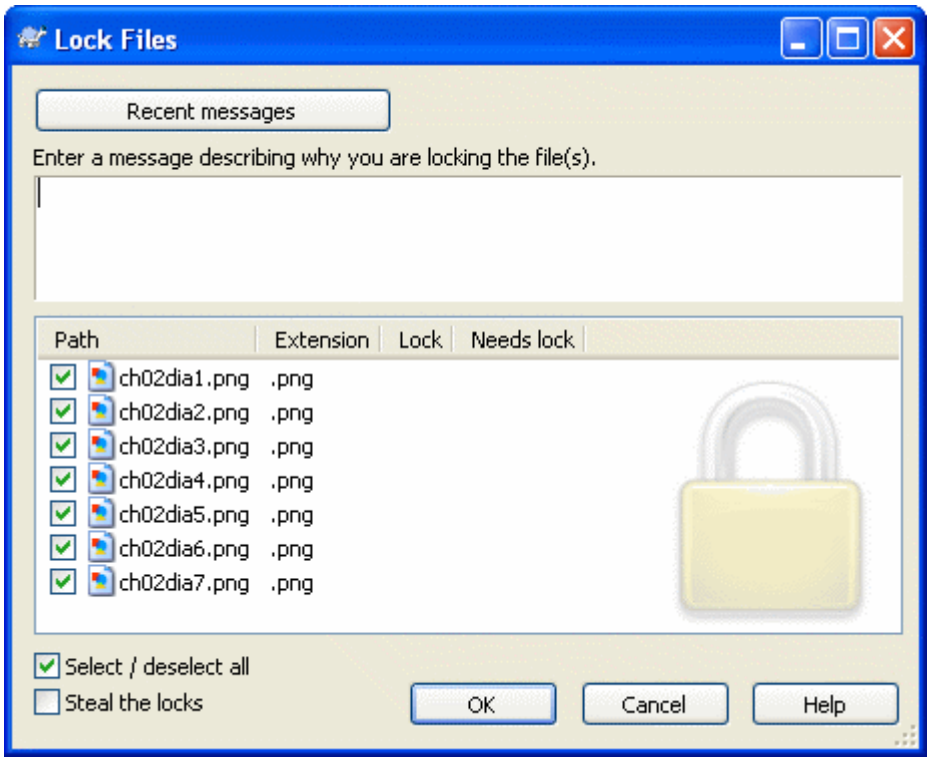


Figura 4.40. The Locking Dialog

A dialog appears, allowing you to enter a comment, so others can see why you have locked the file. The comment is optional and currently only used with Svnserve based repositories. If (and *only* if) you need to steal the lock from someone else, check the Steal lock box, then click on OK.

If you select a folder and then use TortoiseSVN → Get Lock... the lock dialog will open with *every* file in *every* sub-folder selected for locking. If you really want to lock an entire hierarchy, that is the way to do it, but you could become very unpopular with your co-workers if you lock them out of the whole project. Use with care ...

4.21.3. Liberando uma trava

To make sure you don't forget to release a lock you don't need any more, locked files are shown in the commit dialog and selected by default. If you continue with the commit, locks you hold on the selected files are removed, even if the files haven't been modified. If you don't want to release a lock on certain files, you can uncheck them (if they're not modified). If you want to keep a lock on a file you've modified, you have to enable the Keep locks checkbox before you commit your changes.

To release a lock manually, select the file(s) in your working copy for which you want to release the lock, then select the command TortoiseSVN → Release Lock There is nothing further to enter so TortoiseSVN will contact the repository and release the locks. You can also use this command on a folder to release all locks recursively.

4.21.4. Checking Lock Status

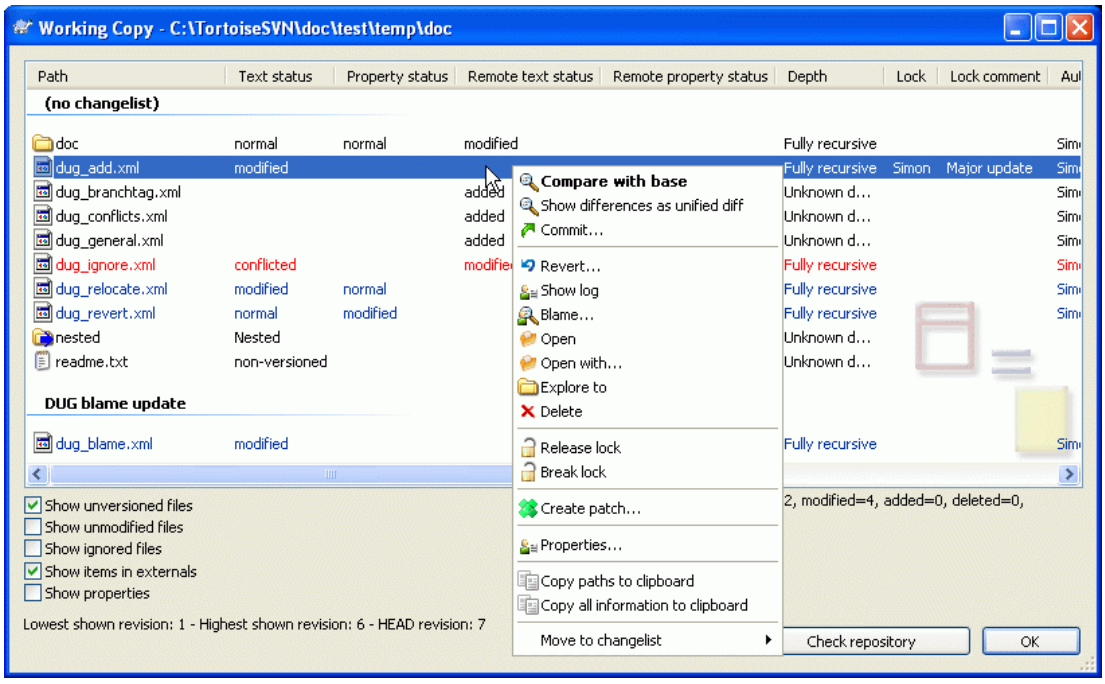


Figura 4.41. The Check for Modifications Dialog

To see what locks you and others hold, you can use TortoiseSVN → Check for Modifications.... Locally held lock tokens show up immediately. To check for locks held by others (and to see if any of your locks are broken or stolen) you need to click on Check Repository.

From the context menu here, you can also get and release locks, as well as breaking and stealing locks held by others.

Avoid Breaking and Stealing Locks

If you break or steal someone else's lock without telling them, you could potentially cause loss of work. If you are working with unmergeable file types and you steal someone else's

> lock, once you release the lock they are free to check in their changes and overwrite yours. Subversion doesn't lose data, but you have lost the team-working protection that locking gave you.

4.21.5. Making Non-locked Files Read-Only

As mentioned above, the most effective way to use locking is to set the `svn:needs-lock` property on files. Refer to Seção 4.17, “Configurações do Projeto” for instructions on how to set properties. Files with this property set will always be checked out and updated with the read-only flag set unless your working copy holds a lock.

As a reminder, TortoiseSVN uses a special overlay to indicate this.

If you operate a policy where every file has to be locked then you may find it easier to use Subversion's auto-props feature to set the property automatically every time you add new files. Read Seção 4.17.1.5, “Automatic property setting” for further information.

4.21.6. The Locking Hook Scripts

When you create a new repository with Subversion 1.2 or higher, four hook templates are created in the repository `hooks` directory. These are called before and after getting a lock, and before and after releasing a lock.

It is a good idea to install a `post-lock` and `post-unlock` hook script on the server which sends out an email indicating the file which has been locked. With such a script in place, all your users can be notified if someone locks/unlocks a file. You can find an example hook script `hooks/post-lock.tmpl` in your repository folder.

You might also use hooks to disallow breaking or stealing of locks, or perhaps limit it to a named administrator. Or maybe you want to email the owner when one of their locks is broken or stolen.

Read Seção 3.3, “Rotinas de eventos no servidor” to find out more.

4.22. Creating and Applying Patches

For open source projects (like this one) everyone has read access to the repository, and anyone can make a contribution to the project. So how are those contributions controlled? If just anyone could commit changes, the project would be permanently unstable and probably permanently broken. In this situation the change is managed by submitting a *patch* file to the development team, who do have write access. They can review the patch first, and then either submit it to the repository or reject it back to the author.

Patch files are simply Unified-Diff files showing the differences between your working copy and the base revision.

4.22.1. Creating a Patch File

First you need to make *and test* your changes. Then instead of using TortoiseSVN → Commit... on the parent folder, you select TortoiseSVN → Create Patch...

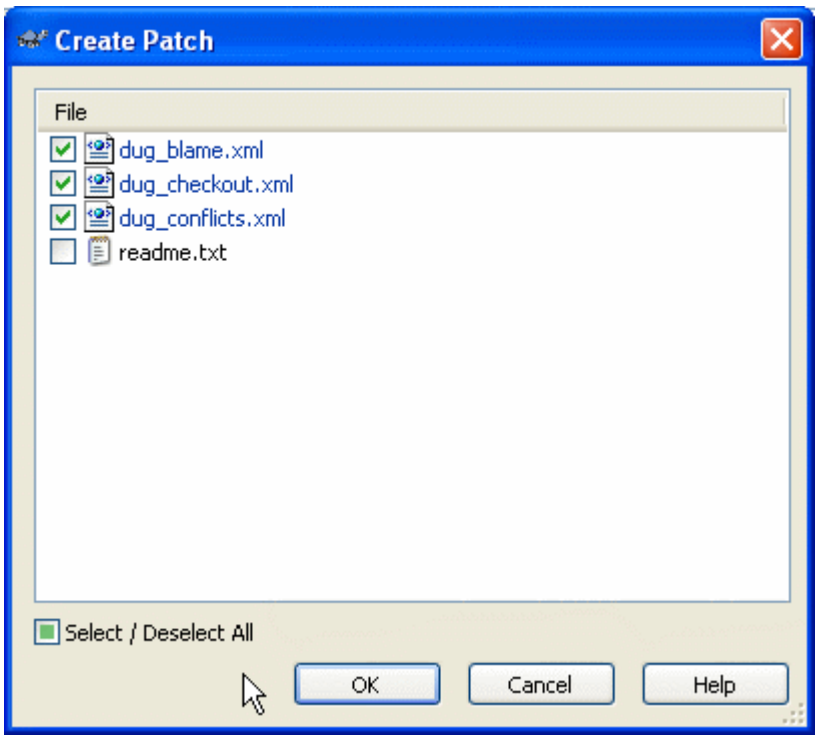


Figura 4.42. The Create Patch dialog

you can now select the files you want included in the patch, just as you would with a full commit. This will produce a single file containing a summary of all the changes you have made to the selected files since the last update from the repository.

As colunas nesta janela podem ser personalizadas da mesma forma como as colunas da janela Verificar alterações. Leia Seção 4.7.3, “Estado Local e Remoto” para mais detalhes.

You can produce separate patches containing changes to different sets of files. Of course, if you create a patch file, make some more changes to the *same* files and then create another patch, the second patch file will include *both* sets of changes.

Just save the file using a filename of your choice. Patch files can have any extension you like, but by convention they should use the `.patch` or `.diff` extension. You are now ready to submit your patch file.

You can also save the patch to the clipboard instead of to a file. You might want to do this so that you can paste it into an email for review by others. Or if you have two working copies on one machine and you want to transfer changes from one to the other, a patch on the clipboard is a convenient way of doing this.

4.22.2. Applying a Patch File

Patch files are applied to your working copy. This should be done from the same folder level as was used to create the patch. If you are not sure what this is, just look at the first line of the patch file. For example, if the first file being worked on was `doc/source/english/chapter1.xml` and the first line in the patch file is `Index: english/chapter1.xml` then you need to apply the patch to the `doc/source/` folder. However, provided you are in the correct working copy, if you pick the wrong folder level, TortoiseSVN will notice and suggest the correct level.

In order to apply a patch file to your working copy, you need to have at least read access to the repository. The reason for this is that the merge program must reference the changes back to the revision against which they were made by the remote developer.

From the context menu for that folder, click on TortoiseSVN → Apply Patch... This will bring up a file open dialog allowing you to select the patch file to apply. By default only `.patch` or `.diff` files

are shown, but you can opt for “All files”. If you previously saved a patch to the clipboard, you can use Open from clipboard... in the file open dialog.

Alternatively, if the patch file has a `.patch` or `.diff` extension, you can right click on it directly and select TortoiseSVN → Apply Patch.... In this case you will be prompted to enter a working copy location.

These two methods just offer different ways of doing the same thing. With the first method you select the WC and browse to the patch file. With the second you select the patch file and browse to the WC.

Once you have selected the patch file and working copy location, TortoiseMerge runs to merge the changes from the patch file with your working copy. A small window lists the files which have been changed. Double click on each one in turn, review the changes and save the merged files.

The remote developer's patch has now been applied to your working copy, so you need to commit to allow everyone else to access the changes from the repository.

4.23. Who Changed Which Line?

Sometimes you need to know not only what lines have changed, but also who exactly changed specific lines in a file. That's when the TortoiseSVN → Blame... command, sometimes also referred to as *annotate* command comes in handy.

This command lists, for every line in a file, the author and the revision the line was changed.

4.23.1. Blame for Files

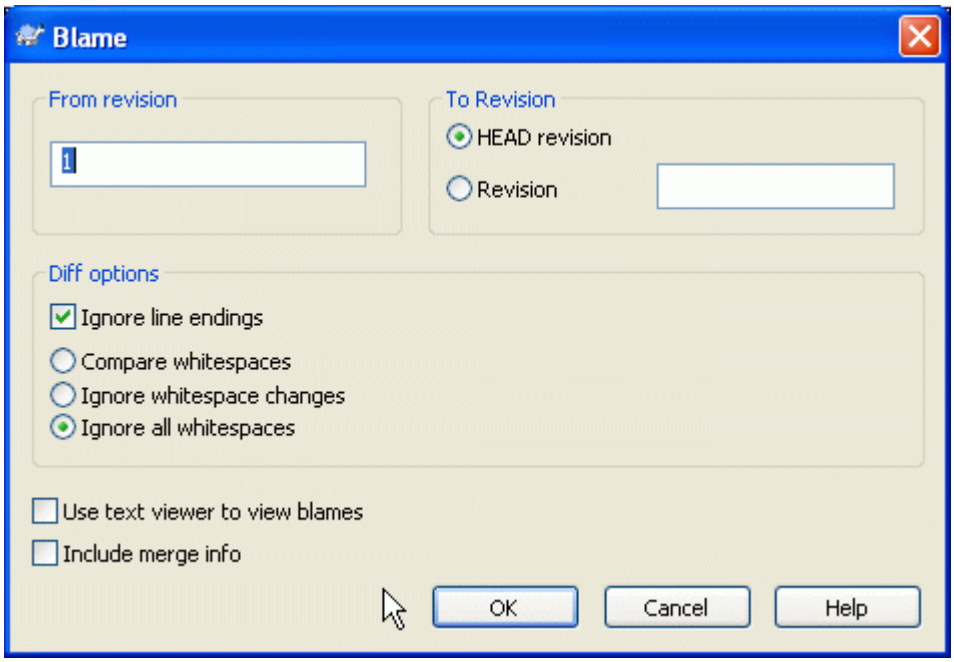


Figura 4.43. The Annotate / Blame Dialog

If you're not interested in changes from earlier revisions you can set the revision from which the blame should start. Set this to `1`, if you want the blame for *every* revision.

By default the blame file is viewed using *TortoiseBlame*, which highlights the different revisions to make it easier to read. If you wish to print or edit the blame file, select Use Text viewer to view blames

You can specify the way that line ending and whitespace changes are handled. These options are described in Seção 4.10.2, “Line-end and Whitespace Options”. The default behaviour is to treat all whitespace and line-end differences as real changes, but if you want to ignore an indentation change and find the original author, you can choose an appropriate option here.

Once you press OK TortoiseSVN starts retrieving the data to create the blame file. Please note: This can take several minutes to finish, depending on how much the file has changed and of course your network connection to the repository. Once the blame process has finished the result is written into a temporary file and you can view the results.

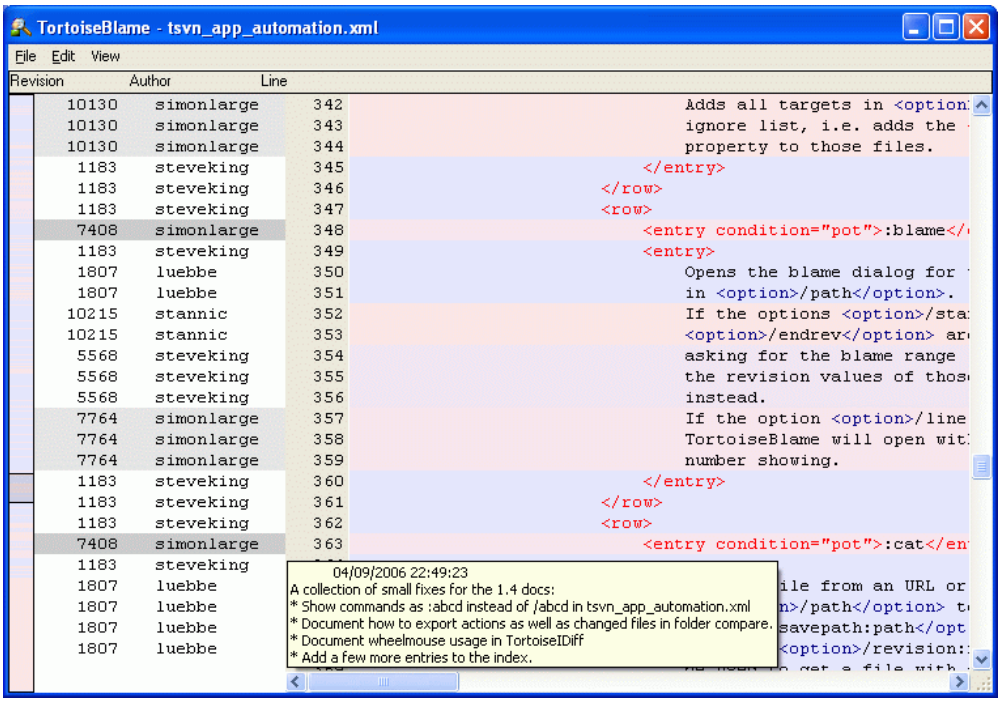


Figura 4.44. TortoiseBlame

TortoiseBlame, which is included with TortoiseSVN, makes the blame file easier to read. When you hover the mouse over a line in the blame info column, all lines with the same revision are shown with a darker background. Lines from other revisions which were changed by the same author are shown with a light background. The colouring may not work as clearly if you have your display set to 256 colour mode.

If you left click on a line, all lines with the same revision are highlighted, and lines from other revisions by the same author are highlighted in a lighter colour. This highlighting is sticky, allowing you to move the mouse without losing the highlights. Click on that revision again to turn off highlighting.

The revision comments (log message) are shown in a hint box whenever the mouse hovers over the blame info column. If you want to copy the log message for that revision, use the context menu which appears when you right click on the blame info column.

You can search within the Blame report using Edit → Find.... This allows you to search for revision numbers, authors and the content of the file itself. Log messages are not included in the search - you should use the Log Dialog to search those.

You can also jump to a specific line number using Edit → Go To Line....

When the mouse is over the blame info columns, a context menu is available which helps with comparing revisions and examining history, using the revision number of the line under the mouse as a reference. Context menu → Blame previous revision generates a blame report for the same file, but using the

previous revision as the upper limit. This gives you the blame report for the state of the file just before the line you are looking at was last changed. Context menu → Show changes starts your diff viewer, showing you what changed in the referenced revision. Context menu → Show log displays the revision log dialog starting with the referenced revision.

If you need a better visual indicator of where the oldest and newest changes are, select View → Color age of lines. This will use a colour gradient to show newer lines in red and older lines in blue. The default colouring is quite light, but you can change it using the TortoiseBlame settings.

If you are using Merge Tracking, where lines have changed as a result of merging from another path, TortoiseBlame will show the revision and author of the last change in the original file rather than the revision where the merge took place. These lines are indicated by showing the revision and author in italics. If you do not want merged lines shown in this way, uncheck the Include merge info checkbox.

If you want to see the paths involved in the merge, select View → Merge paths.

The settings for TortoiseBlame can be accessed using TortoiseSVN → Settings... on the TortoiseBlame tab. Refer to Seção 4.30.9, “Configurações TortoiseBlame”.

4.23.2. Diferenças de Autoria

One of the limitations of the Blame report is that it only shows the file as it was in a particular revision, and shows the last person to change each line. Sometimes you want to know what change was made, as well as who made it. What you need here is a combination of the diff and blame reports.

The revision log dialog includes several options which allow you to do this.

Revisões de Autoria

In the top pane, select 2 revisions, then select Context menu → Blame revisions. This will fetch the blame data for the 2 revisions, then use the diff viewer to compare the two blame files.

Alterações de Autoria

Select one revision in the top pane, then pick one file in the bottom pane and select Context menu → Blame changes. This will fetch the blame data for the selected revision and the previous revision, then use the diff viewer to compare the two blame files.

Compare and Blame with Working BASE

Show the log for a single file, and in the top pane, select a single revision, then select Context menu → Compare and Blame with Working BASE. This will fetch the blame data for the selected revision, and for the file in the working BASE, then use the diff viewer to compare the two blame files.

4.24. O Navegador de Repositório

Sometimes you need to work directly on the repository, without having a working copy. That's what the *Repository Browser* is for. Just as the explorer and the icon overlays allow you to view your working copy, so the Repository Browser allows you to view the structure and status of the repository.

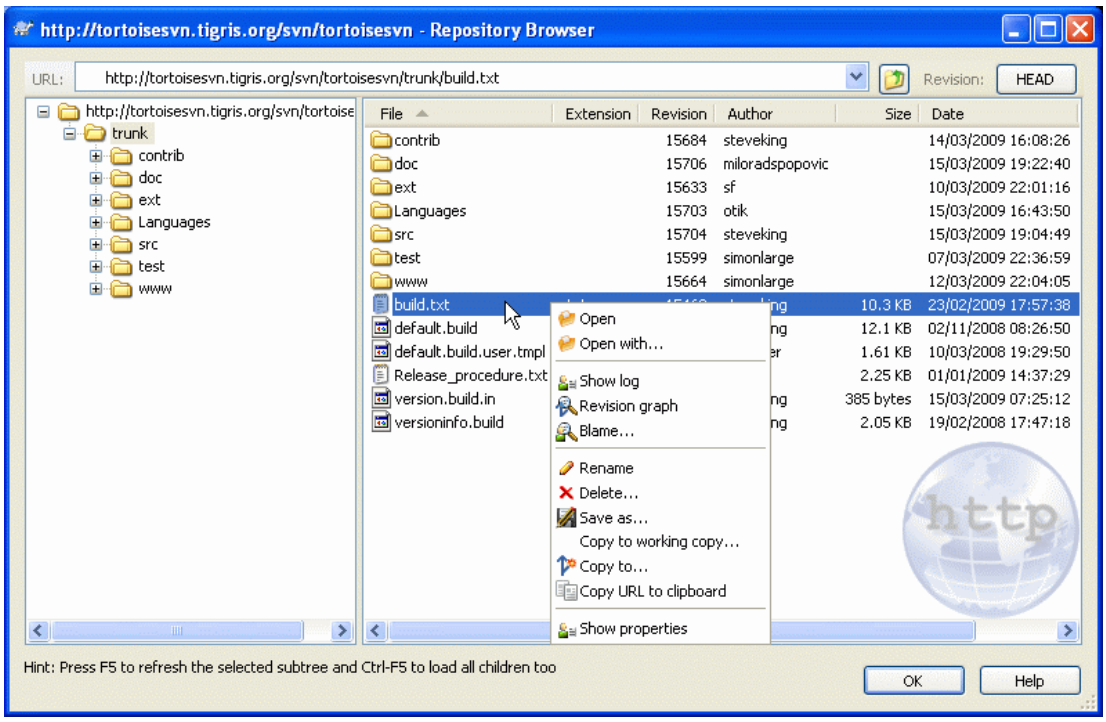


Figura 4.45. O Navegador de Repositório

With the Repository Browser you can execute commands like copy, move, rename, ... directly on the repository.

The repository browser looks very similar to the Windows explorer, except that it is showing the content of the repository at a particular revision rather than files on your computer. In the left pane you can see a directory tree, and in the right pane are the contents of the selected directory. At the top of the Repository Browser Window you can enter the URL of the repository and the revision you want to browse.

Just like Windows explorer, you can click on the column headings in the right pane if you want to set the sort order. And as in explorer there are context menus available in both panes.

The context menu for a file allows you to:

- Open the selected file, either with the default viewer for that file type, or with a program you choose.
- Save an unversioned copy of the file to your hard drive.
- Show the revision log for that file, or show a graph of all revisions so you can see where the file came from.
- Blame the file, to see who changed which line and when.
- Delete or rename the file.
- Make a copy of the file, either to a different part of the repository, or to a working copy rooted in the same repository.
- View/Edit the file's properties.

The context menu for a folder allows you to:

- Show the revision log for that folder, or show a graph of all revisions so you can see where the folder came from.

- Export the folder to a local unversioned copy on your hard drive.
- Checkout the folder to produce a local working copy on your hard drive.
- Create a new folder in the repository.
- Add files or folders directly to the repository.
- Delete or rename the folder.
- Make a copy of the folder, either to a different part of the repository, or to a working copy rooted in the same repository.
- View/Edit the folder's properties.
- Mark the folder for comparison. A marked folder is shown in bold.
- Compare the folder with a previously marked folder, either as a unified diff, or as a list of changed files which can then be visually diffed using the default diff tool. This can be particularly useful for comparing two tags, or trunk and branch to see what changed.

If you select two folders in the right pane, you can view the differences either as a unified-diff, or as a list of files which can be visually diffed using the default diff tool.

If you select multiple folders in the right pane, you can checkout all of them at once into a common parent folder.

If you select 2 tags which are copied from the same root (typically `/trunk/`), you can use Context Menu → Show Log... to view the list of revisions between the two tag points.

You can use **F5** to refresh the view as usual. This will refresh everything which is currently displayed. If you want to pre-fetch or refresh the information for nodes which have not been opened yet, use **Ctrl-F5**. After that, expanding any node will happen instantly without a network delay while the information is fetched.

You can also use the repository browser for drag-and-drop operations. If you drag a folder from explorer into the repo-browser, it will be imported into the repository. Note that if you drag multiple items, they will be imported in separate commits.

If you want to move an item within the repository, just left drag it to the new location. If you want to create a copy rather than moving the item, **Ctrl**-left drag instead. When copying, the cursor has a “plus” symbol on it, just as it does in Explorer.

If you want to copy/move a file or folder to another location and also give it a new name at the same time, you can right drag or **Ctrl**-right drag the item instead of using left drag. In that case, a rename dialog is shown where you can enter a new name for the file or folder.

Whenever you make changes in the repository using one of these methods, you will be presented with a log message entry dialog. If you dragged something by mistake, this is also your chance to cancel the action.

Sometimes when you try to open a path you will get an error message in place of the item details. This might happen if you specified an invalid URL, or if you don't have access permission, or if there is some other server problem. If you need to copy this message to include it in an email, just right click on it and use Context Menu → Copy error message to clipboard, or simply use **Ctrl+C**.

4.25. Gráfico de Revisões

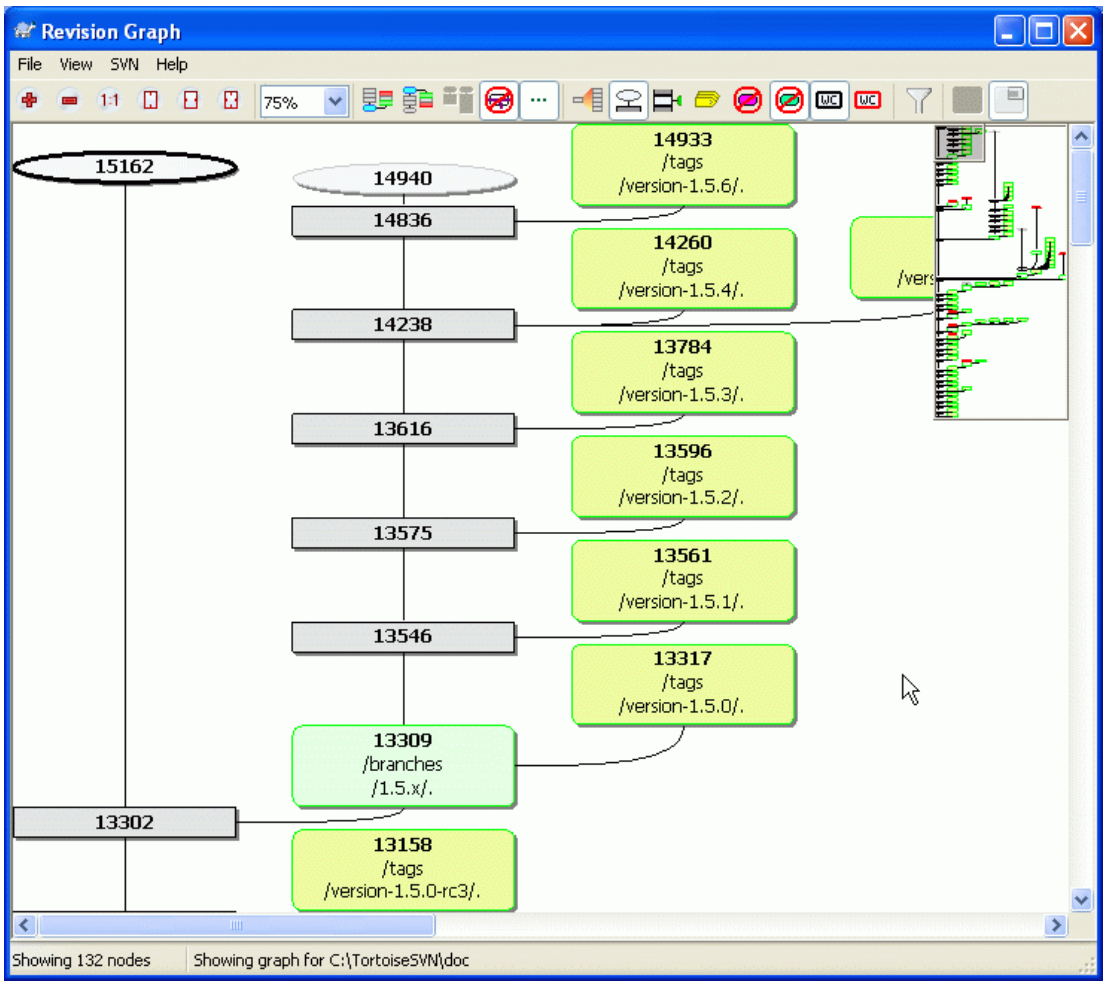


Figura 4.46. Um Gráfico de Revisão

Sometimes you need to know where branches and tags were taken from the trunk, and the ideal way to view this sort of information is as a graph or tree structure. That's when you need to use TortoiseSVN → Revision Graph...

This command analyses the revision history and attempts to create a tree showing the points at which copies were taken, and when branches/tags were deleted.

Importante

In order to generate the graph, TortoiseSVN must fetch all log messages from the repository root. Needless to say this can take several minutes even with a repository of a few thousand revisions, depending on server speed, network bandwidth, etc. If you try this with something like the *Apache* project which currently has over 500,000 revisions you could be waiting for some time.

The good news is that if you are using log caching, you only have to suffer this delay once. After that, log data is held locally. Log caching is enabled in TortoiseSVN's settings.

4.25.1. Nós do Gráfico de Revisões

Each revision graph node represents a revision in the repository where something changed in the tree you are looking at. Different types of node can be distinguished by shape and colour. The shapes are fixed, but colours can be set using TortoiseSVN → Settings

Added or copied items
: Items which have been added, or created by copying another file/folder are shown using a rounded rectangle. The default colour is green. Tags and trunks are treated as a special case and use a different shade, depending on the TortoiseSVN → Settings

Itens excluídos
: Deleted items eg. a branch which is no longer required, are shown using an octagon (rectangle with corners cut off). The default colour is red.

Itens renomeados
: Renamed items are also shown using an octagon, but the default colour is blue.

Branch tip revision
: The graph is normally restricted to showing branch points, but it is often useful to be able to see the respective HEAD revision for each branch too. If you select Show HEAD revisions, each HEAD revision nodes will be shown as an ellipse. Note that HEAD here refers to the last revision committed on that path, not to the HEAD revision of the repository.

Working copy revision
: If you invoked the revision graph from a working copy, you can opt to show the BASE revision on the graph using Show WC revision, which marks the BASE node with a bold outline.

Modified working copy
: If you invoked the revision graph from a working copy, you can opt to show an additional node representing your modified working copy using Show WC modifications. This is an elliptical node with a bold outline in red by default.

Item normal
: All other items are shown using a plain rectangle.

Note that by default the graph only shows the points at which items were added, copied or deleted. Showing every revision of a project will generate a very large graph for non-trivial cases. If you really want to see *all* revisions where changes were made, there is an option to do this in the View menu and on the toolbar.

The default view (grouping off) places the nodes such that their vertical position is in strict revision order, so you have a visual cue for the order in which things were done. Where two nodes are in the same column the order is very obvious. When two nodes are in adjacent columns the offset is much smaller because there is no need to prevent the nodes from overlapping, and as a result the order is a little less obvious. Such optimisations are necessary to keep complex graphs to a reasonable size. Please note that this ordering uses the *edge* of the node on the *older* side as a reference, i.e. the bottom edge of the node when the graph is shown with oldest node at the bottom. The reference edge is significant because the node shapes are not all the same height.

4.25.2. Changing the View

Because a revision graph is often quite complex, there are a number of features which can be used to tailor the view the way you want it. These are available in the View menu and from the toolbar.

Group branches
: The default behavior (grouping off) has all rows sorted strictly by revision. As a result, long-living branches with sparse commits occupy a whole column for only a few changes and the graph becomes very broad.

  This mode groups changes by branch, so that there is no global revision ordering: Consecutive revisions on a branch will be shown in (often) consecutive lines. Sub-branches, however, are arranged in such a way that later branches will be shown in the same column above older branches to keep the graph slim. As a result, a given row may contain changes from different revisions.

Mais antigo primeiro
: Normally the graph shows the oldest revision at the bottom, and the tree grows upwards. Use this option to grow down from the top instead.

Align trees on top
: When a graph is broken into several smaller trees, the trees may appear either in natural revision order, or aligned at the bottom of the window, depending on whether you are using the Group Branches option. Use this option to grow all trees down from the top instead.

Reduce cross lines
: If the layout of the graph has produced a lot of crossing lines, use this option to clean it up. This may make the layout columns appear in less logical places, for example in a diagonal line rather than a column, and the graph may require a larger area to draw.

Differential path names
: Long path names can take a lot of space and make the node boxes very large. Use this option to show only the changed part of a path, replacing the common part with dots. E.g. if you create a branch `/branches/1.2.x/doc/html` from `/trunk/doc/html` the branch could be shown in compact form as `/branches/1.2.x/..` because the last two levels, `doc` and `html`, did not change.

Show all revisions
: This does just what you expect and shows every revision where something (in the tree that you are graphing) has changed. For long histories this can produce a truly huge graph.

Show HEAD revisions
: This ensures that the latest revision on every branch is always shown on the graph.

Exact copy sources
: When a branch/tag is made, the default behaviour is to show the branch as taken from the last node where a change was made. Strictly speaking this is inaccurate since the branches are often made from the current HEAD rather than a specific revision. So it is possible to show the more correct (but less useful) revision that was used to create the copy. Note that this revision may be younger than the HEAD revision of the source branch.

Fold tags
: When a project has many tags, showing every tag as a separate node on the graph takes a lot of space and obscures the more interesting development branch structure. At the same time you may need to be able to access the tag content easily so that you can compare revisions. This option hides the nodes for tags and shows them instead in the tooltip for the node that they were copied from. A tag icon on the right side of the source node indicates that tags were made.

Hide deleted paths
: Hides paths which are no longer present at the HEAD revision of the repository, e.g. deleted branches.

Hide unchanged branches
: Hides branches where no changes were committed to the respective file or sub-folder. This does not necessarily indicate that the branch was not used, just that no changes were made to *this* part of it.

Show WC revision
: Marks the revision on the graph which corresponds to the update revision of the item you fetched the graph for. If you have just updated, this will be HEAD, but if others have committed changes since your last update your WC may be a few revisions lower down. The node is marked by giving it a bold outline.

Show WC modifications
: If your WC contains local changes, this option draws it as a separate elliptical node, linked back to the node that your WC was last updated to. The default outline colour is red. You may need to refresh the graph using **F5** to capture recent changes.

Filtro
: Sometimes the revision graph contains more revisions than you want to see. This option opens a dialog which allows you to restrict the range of revisions displayed, and to hide particular paths by name.

Tree stripes
: Where the graph contains several trees, it is sometimes useful to use alternating colours on the background to help distinguish between trees.

Show overview
: Shows a small picture of the entire graph, with the current view window as a rectangle which you can drag. This allows you to navigate the graph more easily. Note that for very large graphs the overview may become useless due to the extreme zoom factor and will therefore not be shown in such cases.

4.25.3. Using the Graph

To make it easier to navigate a large graph, use the overview window. This shows the entire graph in a small window, with the currently displayed portion highlighted. You can drag the highlighted area to change the displayed region.

The revision date, author and comments are shown in a hint box whenever the mouse hovers over a revision box.

If you select two revisions (Use **Ctrl**-left click), you can use the context menu to show the differences between these revisions. You can choose to show differences as at the branch creation points, but usually you will want to show the differences at the branch end points, i.e. at the HEAD revision.

You can view the differences as a Unified-Diff file, which shows all differences in a single file with minimal context. If you opt to Context Menu → Compare Revisions you will be presented with a list of changed files. Double click on a file name to fetch both revisions of the file and compare them using the visual difference tool.

If you right click on a revision you can use Context Menu → Show Log to view the history.

You can also merge changes in the selected revision(s) into a different working copy. A folder selection dialog allows you to choose the working copy to merge into, but after that there is no confirmation dialog, nor any opportunity to try a test merge. It is a good idea to merge into an unmodified working copy so that you can revert the changes if it doesn't work out! This is a useful feature if you want to merge selected revisions from one branch to another.

Learn to Read the Revision Graph

First-time users may be surprised by the fact that the revision graph shows something that does not match the user's mental model. If a revision changes multiple copies or branches of a file or folder, for instance, then there will be multiple nodes for that single revision. It is a good practice to start with the leftmost options in the toolbar and customize the graph step-by-step until it comes close to your mental model.

All filter options try lose as little information as possible. That may cause some nodes to change their color, for instance. Whenever the result is unexpected, undo the last filter operation and try to understand what is special about that particular revision or branch. In most cases, the initially expected outcome of the filter operation would either be inaccurate or misleading.

4.25.4. Atualizando a Visualização

If you want to check the server again for newer information, you can simply refresh the view using **F5**. If you are using the log cache (enabled by default), this will check the repository for newer commits and fetch only the new ones. If the log cache was in offline mode, this will also attempt to go back online.

If you are using the log cache and you think the message content or author may have changed, you should use the log dialog to refresh the messages you need. Since the revision graph works from the repository root, we would have to invalidate the entire log cache, and refilling it could take a *very* long time.

4.25.5. Pruning Trees

A large tree can be difficult to navigate and sometimes you will want to hide parts of it, or break it down into a forest of smaller trees. If you hover the mouse over the point where a node link enters or leaves the node you will see one or more popup buttons which allow you to do this.

Click on the minus button to collapse the attached sub-tree.

Click on the plus button to expand a collapsed tree. When a tree has been collapsed, this button remains visible to indicate the hidden sub-tree.

Click on the cross button to split the attached sub-tree and show it as a separate tree on the graph.

Click on the circle button to reattach a split tree. When a tree has been split away, this button remains visible to indicate that there is a separate sub-tree.

Click on the graph background for the main context menu, which offers options to Expand all and Join all. If no branch has been collapsed or split, the context menu will not be shown.

4.26. Exporting a Subversion Working Copy

Sometimes you may want a copy of your working tree without any of those `.svn` directories, e.g. to create a zipped tarball of your source, or to export to a web server. Instead of making a copy and then deleting all those `.svn` directories manually, TortoiseSVN offers the command TortoiseSVN → Export.... Exporting from a URL and exporting from a working copy are treated slightly differently.

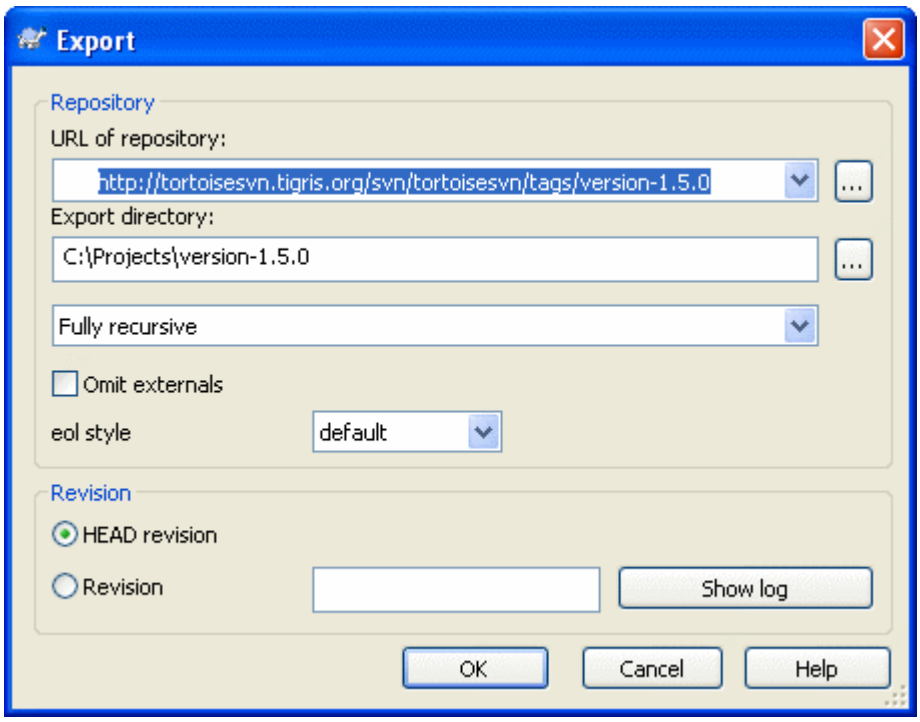


Figura 4.47. The Export-from-URL Dialog

If you execute this command on an unversioned folder, TortoiseSVN will assume that the selected folder is the target, and open a dialog for you to enter the URL and revision to export from. This dialog has options to export only the top level folder, to omit external references, and to override the line end style for files which have the `svn:eol-style` property set.

Of course you can export directly from the repository too. Use the Repository Browser to navigate to the relevant subtree in your repository, then use Context Menu → Export. You will get the Export from URL dialog described above.

If you execute this command on your working copy you'll be asked for a place to save the *clean* working copy without the `.svn` folders. By default, only the versioned files are exported, but you can use the Export unversioned files too checkbox to include any other unversioned files which exist in your WC and not in the repository. External references using `svn:externals` can be omitted if required.

Another way to export from a working copy is to right drag the working copy folder to another location and choose Context Menu → SVN Export here or Context Menu → SVN Export all here. The second option includes the unversioned files as well.

When exporting from a working copy, if the target folder already contains a folder of the same name as the one you are exporting, you will be given the option to overwrite the existing content, or to create a new folder with an automatically generated name, eg. `Target (1)`.

Exporting single files

The export dialog does not allow exporting single files, even though Subversion can.

To export single files with TortoiseSVN, you have to use the repository browser (Seção 4.24, “O Navegador de Repositório”). Simply drag the file(s) you want to export from the repository browser to where you want them in the explorer, or use the context menu in the repository browser to export the files.

Exporting a Change Tree

If you want to export a copy of your project tree structure but containing only the files which have changed in a particular revision, or between any two revisions, use the compare revisions feature described in Seção 4.10.3, “Comparando Diretórios”.

4.26.1. Removing a working copy from version control

Sometimes you have a working copy which you want to convert back to a normal folder without the `.svn` directories. What you really need is an export-in-place command, that just removes the control directories rather than generating a new clean directory tree.

The answer is surprisingly simple - export the folder to itself! TortoiseSVN detects this special case and asks if you want to make the working copy unversioned. If you answer *yes* the control directories will be removed and you will have a plain, unversioned directory tree.

4.27. Relocating a working copy

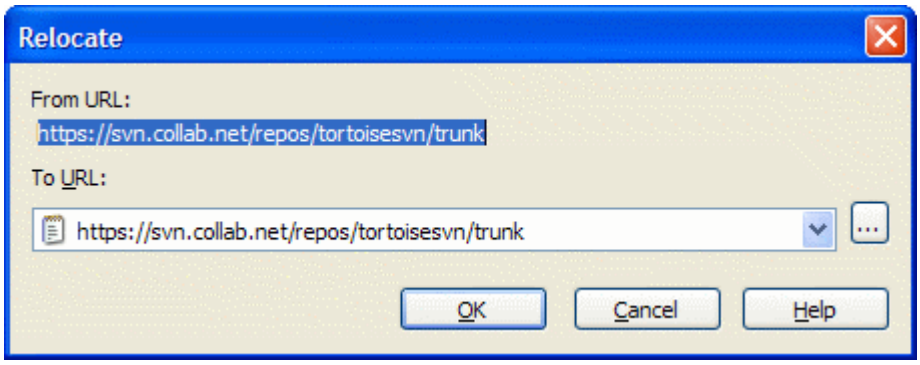


Figura 4.48. The Relocate Dialog

If your repository has for some reason changed it's location (IP/URL). Maybe you're even stuck and can't commit and you don't want to checkout your working copy again from the new location and to move all your changed data back into the new working copy, TortoiseSVN → Relocate is the command you are looking for. It basically does very little: it scans all `entries` files in the `.svn` folder and changes the URL of the entries to the new value.

You may be surprised to find that TortoiseSVN contacts the repository as part of this operation. All it is doing is performing some simple checks to make sure that the new URL really does refer to the same repository as the existing working copy.

Atenção

This is a very infrequently used operation. The relocate command is *only* used if the URL of the repository root has changed. Possible reasons are:

- The IP address of the server has changed.
- The protocol has changed (e.g. http:// to https://).
- The repository root path in the server setup has changed.

Put another way, you need to relocate when your working copy is referring to the same location in the same repository, but the repository itself has moved.

It does not apply if:

- You want to move to a different Subversion repository. In that case you should perform a clean checkout from the new repository location.
- You want to switch to a different branch or directory within the same repository. To do that you should use TortoiseSVN → Switch.... Read Seção 4.19.2, “Para Obter ou Alternar” for more information.

If you use relocate in either of the cases above, it *will corrupt your working copy* and you will get many unexplainable error messages while updating, committing, etc. Once that has happened, the only fix is a fresh checkout.

4.28. Integration with Bug Tracking Systems / Issue Trackers

It is very common in Software Development for changes to be related to a specific bug or issue ID. Users of bug tracking systems (issue trackers) would like to associate the changes they make in Subversion with a specific ID in their issue tracker. Most issue trackers therefore provide a pre-commit hook script which parses the log message to find the bug ID with which the commit is associated. This is somewhat error prone since it relies on the user to write the log message properly so that the pre-commit hook script can parse it correctly.

TortoiseSVN can help the user in two ways:

1. When the user enters a log message, a well defined line including the issue number associated with the commit can be added automatically. This reduces the risk that the user enters the issue number in a way the bug tracking tools can't parse correctly.

   Or TortoiseSVN can highlight the part of the entered log message which is recognized by the issue tracker. That way the user knows that the log message can be parsed correctly.

2. When the user browses the log messages, TortoiseSVN creates a link out of each bug ID in the log message which fires up the browser to the issue mentioned.

4.28.1. Adding Issue Numbers to Log Messages

You can integrate a bug tracking tool of your choice in TortoiseSVN. To do this, you have to define some properties, which start with `bugtraq:`. They must be set on Folders: (Seção 4.17, “Configurações do Projeto”)

There are two ways to integrate TortoiseSVN with issue trackers. One is based on simple strings, the other is based on *regular expressions*. The properties used by both approaches are:

bugtraq:url
: Set this property to the URL of your bug tracking tool. It must be properly URI encoded and it has to contain `%BUGID%`. `%BUGID%` is replaced with the Issue number you entered. This allows TortoiseSVN to display a link in the log dialog, so when you are looking at the revision log you can jump directly to your bug tracking tool. You do not have to provide this property, but then TortoiseSVN shows only the issue number and not the link to it. e.g the TortoiseSVN project is using `http://issues.tortoisesvn.net/?do=details&id=%BUGID%`

  You can also use relative URLs instead of absolute ones. This is useful when your issue tracker is on the same domain/server as your source repository. In case the domain name ever changes, you don't have to adjust the `bugtraq:url` property. There are two ways to specify a relative URL:

If it begins with the string `^/` it is assumed to be relative to the repository root. For example, `^/../?do=details&id=%BUGID%` will resolve to `http://tortoisesvn.net/?do=details&id=%BUGID%` if your repository is located on `http://tortoisesvn.net/svn/trunk/`.

A URL beginning with the string `/` is assumed to be relative to the server's hostname. For example `/?do=details&id=%BUGID%` will resolve to `http://tortoisesvn.net/?do=details&id=%BUGID%` if your repository is located anywhere on `http://tortoisesvn.net`.

bugtraq:warnifnoissue
: Set this to `true`, if you want TortoiseSVN to warn you because of an empty issue-number text field. Valid values are `true/false`. *If not defined, `false` is assumed.*

4.28.1.1. Issue Number in Text Box

In the simple approach, TortoiseSVN shows the user a separate input field where a bug ID can be entered. Then a separate line is appended/prepended to the log message the user entered.

bugtraq:message
: This property activates the bug tracking system in *Input field* mode. If this property is set, then TortoiseSVN will prompt you to enter an issue number when you commit your changes. It's used to add a line at the end of the log message. It must contain `%BUGID%`, which is replaced with the issue number on commit. This ensures that your commit log contains a reference to the issue number which is always in a consistent format and can be parsed by your bug tracking tool to associate the issue number with a particular commit. As an example you might use `Issue : %BUGID%`, but this depends on your Tool.

bugtraq:append
: This property defines if the bug-ID is appended (true) to the end of the log message or inserted (false) at the start of the log message. Valid values are `true/false`. *If not defined, `true` is assumed, so that existing projects don't break.*

bugtraq:label
: This text is shown by TortoiseSVN on the commit dialog to label the edit box where you enter the issue number. If it's not set, `Bug-ID / Issue-Nr:` will be displayed. Keep in mind though that the window will not be resized to fit this label, so keep the size of the label below 20-25 characters.

bugtraq:number
: If set to `true` only numbers are allowed in the issue-number text field. An exception is the comma, so you can comma separate several numbers. Valid values are `true/false`. *If not defined, `true` is assumed.*

4.28.1.2. Issue Numbers Using Regular Expressions

In the approach with *regular expressions*, TortoiseSVN doesn't show a separate input field but marks the part of the log message the user enters which is recognized by the issue tracker. This is done while the user writes the log message. This also means that the bug ID can be anywhere inside a log message! This method is much more flexible, and is the one used by the TortoiseSVN project itself.

bugtraq:logregex
: This property activates the bug tracking system in *Regex* mode. It contains either a single regular expressions, or two regular expressions separated by a newline.

  If two expressions are set, then the first expression is used as a pre-filter to find expressions which contain bug IDs. The second expression then extracts the bare bug IDs from the result of the first regex. This allows you to use a list of bug IDs and natural language expressions if you wish. e.g. you might fix several bugs and include a string something like this: “This change resolves issues #23, #24 and #25”

If you want to catch bug IDs as used in the expression above inside a log message, you could use the following regex strings, which are the ones used by the TortoiseSVN project: `[Ii]ssues?:?(\s*(,|and)?\s*#\d+)+` and `(\d+)`

The first expression picks out “issues #23, #24 and #25” from the surrounding log message. The second regex extracts plain decimal numbers from the output of the first regex, so it will return “23”, “24” and “25” to use as bug IDs.

Breaking the first regex down a little, it must start with the word “issue”, possibly capitalised. This is optionally followed by an “s” (more than one issue) and optionally a colon. This is followed by one or more groups each having zero or more leading whitespace, an optional comma or “and” and more optional space. Finally there is a mandatory “#” and a mandatory decimal number.

If only one expression is set, then the bare bug IDs must be matched in the groups of the regex string. Example: `[Ii]ssue(?:s)? #?(\d+)` This method is required by a few issue trackers, e.g. trac, but it is harder to construct the regex. We recommend that you only use this method if your issue tracker documentation tells you to.

If you are unfamiliar with regular expressions, take a look at the introduction at *http://en.wikipedia.org/wiki/Regular_expression*, and the online documentation and tutorial at *http://www.regular-expressions.info/*.

If both the `bugtraq:message` and `bugtraq:logregex` properties are set, `logregex` takes precedence.

Dica

Even if you don't have an issue tracker with a pre-commit hook parsing your log messages, you still can use this to turn the issues mentioned in your log messages into links!

And even if you don't need the links, the issue numbers show up as a separate column in the log dialog, making it easier to find the changes which relate to a particular issue.

Some `tsvn:` properties require a `true/false` value. TortoiseSVN also understands `yes` as a synonym for `true` and `no` as a synonym for `false`.

Set the Properties on Folders

These properties must be set on folders for the system to work. When you commit a file or folder the properties are read from that folder. If the properties are not found there, TortoiseSVN will search upwards through the folder tree to find them until it comes to an unversioned folder, or the tree root (eg. `C:\`) is found. If you can be sure that each user checks out only from e.g `trunk/` and not some sub-folder, then it's enough if you set the properties on `trunk/`. If you can't be sure, you should set the properties recursively on each sub-folder. A property setting deeper in the project hierarchy overrides settings on higher levels (closer to `trunk/`).

For `tsvn:` properties *only* you can use the Recursive checkbox to set the property to all sub-folders in the hierarchy, without also setting it on all files.

No Issue Tracker Information from Repository Browser

Because the issue tracker integration depends upon accessing subversion properties, you will only see the results when using a checked out working copy. Fetching properties remotely is a slow operation, so you will not see this feature in action from the repo browser.

This issue tracker integration is not restricted to TortoiseSVN; it can be used with any Subversion client. For more information, read the full *Issue Tracker Integration Specification* [http://

tortoisesvn.googlecode.com/svn/trunk/doc/issuetrackers.txt] in the TortoiseSVN source repository. (Seção 3, “TortoiseSVN é grátis!” explains how to access the repository).

4.28.2. Getting Information from the Issue Tracker

The previous section deals with adding issue information to the log messages. But what if you need to get information from the issue tracker? The commit dialog has a COM interface which allows integration an external program that can talk to your tracker. Typically you might want to query the tracker to get a list of open issues assigned to you, so that you can pick the issues that are being addressed in this commit.

Any such interface is of course highly specific to your issue tracker system, so we cannot provide this part, and describing how to create such a program is beyond the scope of this manual. The interface definition and sample plugins in C# and C++/ATL can be obtained from the `contrib` folder in the *TortoiseSVN repository* [http://tortoisesvn.googlecode.com/svn/trunk/contrib/issue-tracker-plugins]. (Seção 3, “TortoiseSVN é grátis!” explains how to access the repository). A summary of the API is also given in Capítulo 6, *IBugtraqProvider interface* Another (working) example plugin in C# is *Gurtle* [http://code.google.com/p/gurtle/] which implements the required COM interface to interact with the *Google Code* [http://code.google.com/hosting/] issue tracker.

For illustration purposes, let's suppose that your system administrator has provided you with an issue tracker plugin which you have installed, and that you have set up some of your working copies to use the plugin in TortoiseSVN's settings dialog. When you open the commit dialog from a working copy to which the plugin has been assigned, you will see a new button at the top of the dialog.

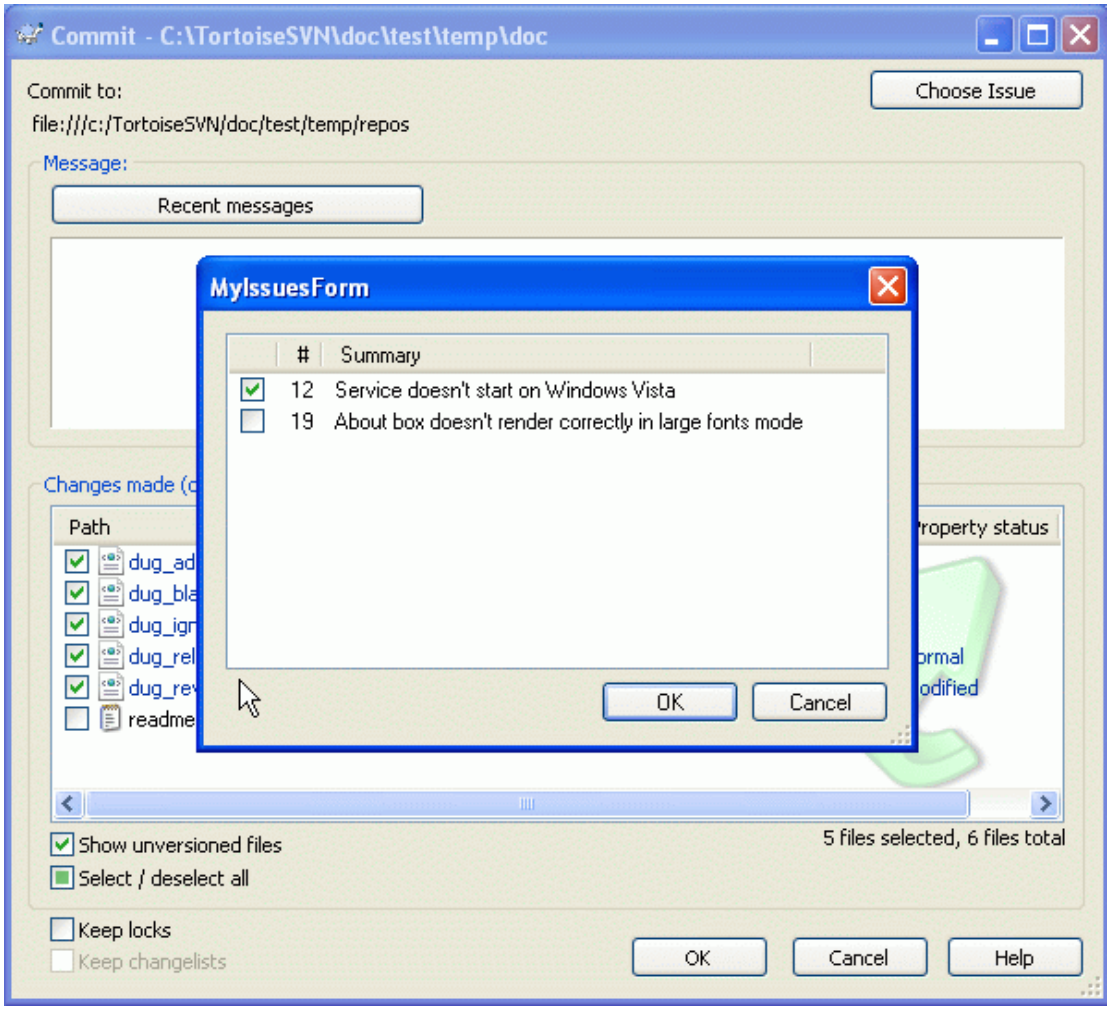


Figura 4.49. Example issue tracker query dialog

In this example you can select one or more open issues. The plugin can then generate specially formatted text which it adds to your log message.

4.29. Integration with Web-based Repository Viewers

There are several web-based repository viewers available for use with Subversion such as *ViewVC* [http://www.viewvc.org/] and *WebSVN* [http://websvn.tigris.org/]. TortoiseSVN provides a means to link with these viewers.

You can integrate a repo viewer of your choice in TortoiseSVN. To do this, you have to define some properties which define the linkage. They must be set on Folders: (Seção 4.17, “Configurações do Projeto”)

webviewer:revision
: Set this property to the URL of your repo viewer to view all changes in a specific revision. It must be properly URI encoded and it has to contain `%REVISION%`. `%REVISION%` is replaced with the revision number in question. This allows TortoiseSVN to display a context menu entry in the log dialog Context Menu → View revision in webviewer

webviewer:pathrevision
: Set this property to the URL of your repo viewer to view changes to a specific file in a specific revision. It must be properly URI encoded and it has to contain `%REVISION%` and `%PATH%`. `%PATH%` is replaced with the path relative to the repository root. This allows TortoiseSVN to display a context menu entry in the log dialog Context Menu → View revision and path in webviewer For example, if you right-click in the log dialog bottom pane on a file entry `/trunk/src/file` then the `%PATH%` in the URL will be replaced with `/trunk/src/file`.

You can also use relative URLs instead of absolute ones. This is useful in case your web viewer is on the same domain/server as your source repository. In case the domain name ever changes, you don't have to adjust the `webviewer:revision` and `webviewer:pathrevision` property. The format is the same as for the `bugtraq:url` property. See Seção 4.28, “Integration with Bug Tracking Systems / Issue Trackers”.

Set the Properties on Folders

These properties must be set on folders for the system to work. When you commit a file or folder the properties are read from that folder. If the properties are not found there, TortoiseSVN will search upwards through the folder tree to find them until it comes to an unversioned folder, or the tree root (eg. `C:\`) is found. If you can be sure that each user checks out only from e.g `trunk/` and not some sub-folder, then it's enough if you set the properties on `trunk/`. If you can't be sure, you should set the properties recursively on each sub-folder. A property setting deeper in the project hierarchy overrides settings on higher levels (closer to `trunk/`).

For `tsvn:` properties *only* you can use the Recursive checkbox to set the property to all sub-folders in the hierarchy, without also setting it on all files.

No Repo Viewer Links from Repository Browser

Because the repo viewer integration depends upon accessing subversion properties, you will only see the results when using a checked out working copy. Fetching properties remotely is a slow operation, so you will not see this feature in action from the repo browser.

4.30. Configurações do TortoiseSVN

To find out what the different settings are for, just leave your mouse pointer a second on the editbox/checkbox... and a helpful tooltip will popup.

4.30.1. Configurações Gerais

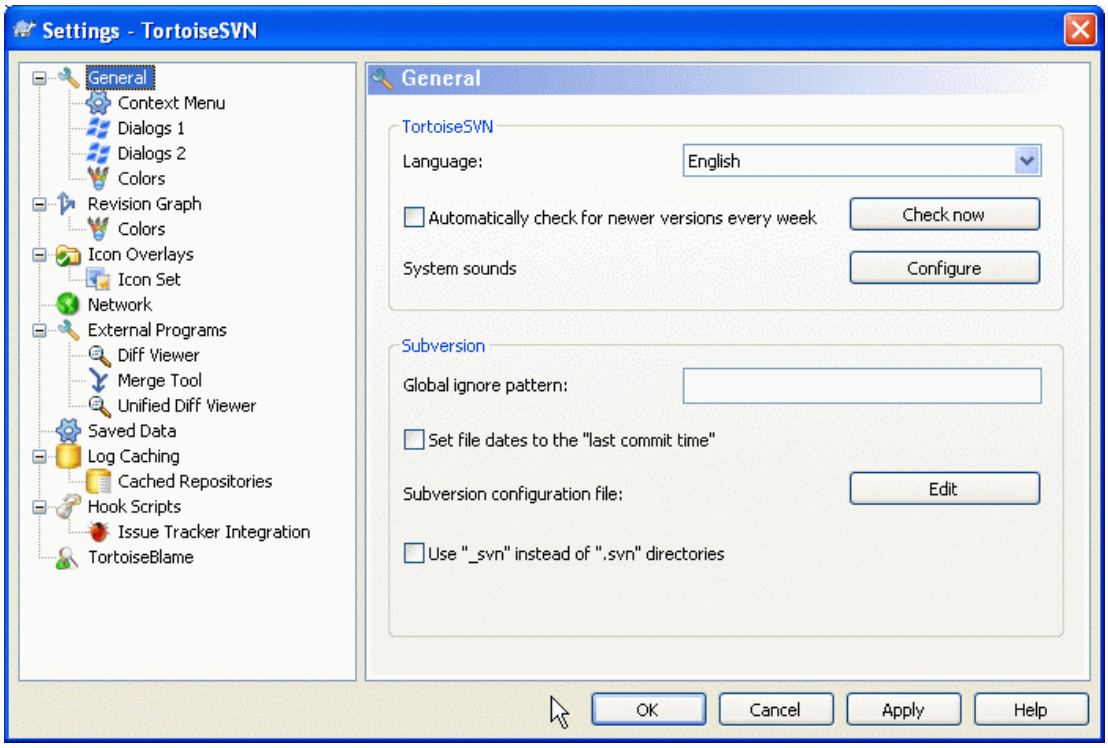


Figura 4.50. The Settings Dialog, General Page

This dialog allows you to specify your preferred language, and the Subversion-specific settings.

Idioma
: Selects your user interface language. What else did you expect?

Automatically check for newer versions every week
: If checked, TortoiseSVN will contact its download site once a week to see if there is a newer version of the program available. Use Check now if you want an answer right away. The new version will not be downloaded; you simply receive an information dialog telling you that the new version is available.

Sons do sistema
: TortoiseSVN has three custom sounds which are installed by default.

  - Error
  - Notificação
  - Aviso

  You can select different sounds (or turn these sounds off completely) using the Windows Control Panel. Configure is a shortcut to the Control Panel.

Padrão global para ignorar

: Global ignore patterns are used to prevent unversioned files from showing up e.g. in the commit dialog. Files matching the patterns are also ignored by an import. Ignore files or directories by typing

in the names or extensions. Patterns are separated by spaces e.g. `bin obj *.bak *.~?? *.jar *.[Tt]mp`. These patterns should not include any path separators. Note also that there is no way to differentiate between files and directories. Read Seção 4.13.1, “Padrões de Filtro na Lista de Arquivos Ignorados” for more information on the pattern-matching syntax.

Note that the ignore patterns you specify here will also affect other Subversion clients running on your PC, including the command line client.

Cuidado

If you use the Subversion configuration file to set a `global-ignores` pattern, it will override the settings you make here. The Subversion configuration file is accessed using the Edit as described below.

This ignore pattern will affect all your projects. It is not versioned, so it will not affect other users. By contrast you can also use the versioned `svn:ignore` property to exclude files or directories from version control. Read Seção 4.13, “Ignorando Arquivos e Diretórios” for more information.

Set file dates to the “last commit time”
: This option tells TortoiseSVN to set the file dates to the last commit time when doing a checkout or an update. Otherwise TortoiseSVN will use the current date. If you are developing software it is generally best to use the current date because build systems normally look at the date stamps to decide which files need compiling. If you use “last commit time” and revert to an older file revision, your project may not compile as you expect it to.

Subversion configuration file
: Use Edit to edit the Subversion configuration file directly. Some settings cannot be modified directly by TortoiseSVN, and need to be set here instead. For more information about the Subversion `config` file see the *Runtime Configuration Area* [http://svnbook.red-bean.com/en/1.5/svn.advanced.confarea.html]. The section on *Automatic Property Setting* [http://svnbook.red-bean.com/en/1.5/svn.advanced.props.html#svn.advanced.props.auto] is of particular interest, and that is configured here. Note that Subversion can read configuration information from several places, and you need to know which one takes priority. Refer to *Configuration and the Windows Registry* [http://svnbook.red-bean.com/en/1.5/svn.advanced.confarea.html#svn.advanced.confarea.windows-registry] to find out more.

Use `_svn` instead of `.svn` directories
: VS.NET when used with web projects can't handle the `.svn` folders that Subversion uses to store its internal information. This is not a bug in Subversion. The bug is in VS.NET and the frontpage extensions it uses. Read Seção 4.30.11, “Pastas de Trabalho do Subversion” to find out more about this issue.

  If you want to change the behaviour of Subversion and TortoiseSVN, you can use this checkbox to set the environment variable which controls this.

  You should note that changing this option will not automatically convert existing working copies to use the new admin directory. You will have to do that yourself using a script (See our FAQ) or simply check out a fresh working copy.

4.30.1.1. Configurações do Menu de Contexto

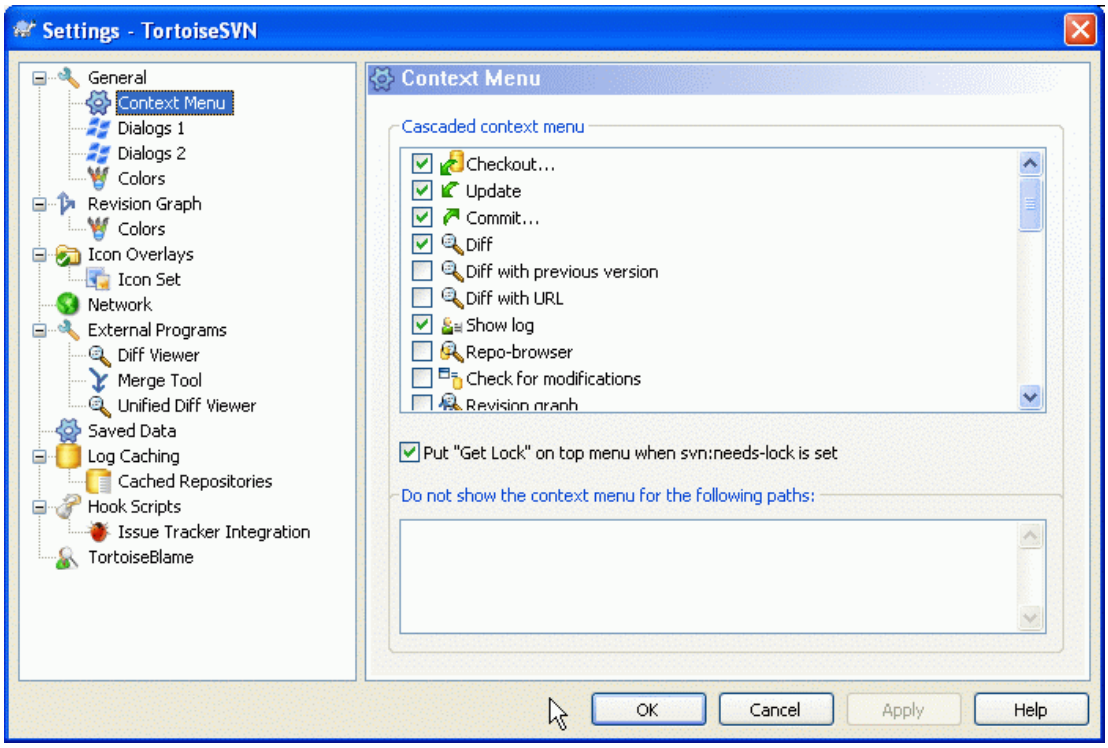


Figura 4.51. The Settings Dialog, Context Menu Page

This page allows you to specify which of the TortoiseSVN context menu entries will show up in the main context menu, and which will appear in the TortoiseSVN submenu. By default most items are unchecked and appear in the submenu.

There is a special case for Get Lock. You can of course promote it to the top level using the list above, but as most files don't need locking this just adds clutter. However, a file with the `svn:needs-lock` property needs this action every time it is edited, so in that case it is very useful to have at the top level. Checking the box here means that when a file is selected which has the `svn:needs-lock` property set, Get Lock will always appear at the top level.

If there are some paths on your computer where you just don't want TortoiseSVN's context menu to appear at all, you can list them in the box at the bottom.

4.30.1.2. TortoiseSVN Dialog Settings 1

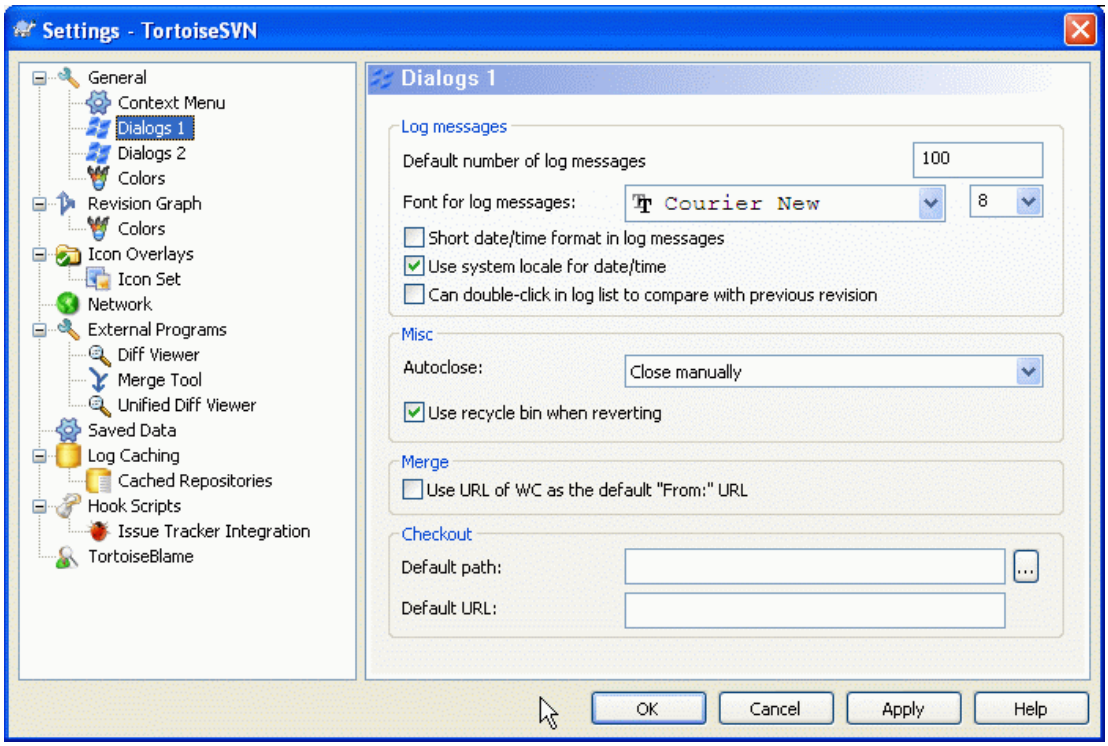


Figura 4.52. The Settings Dialog, Dialogs 1 Page

This dialog allows you to configure some of TortoiseSVN's dialogs the way you like them.

Default number of log messages
: Limits the number of log messages that TortoiseSVN fetches when you first select TortoiseSVN → Show Log Useful for slow server connections. You can always use Show All or Next 100 to get more messages.

Font for log messages
: Selects the font face and size used to display the log message itself in the middle pane of the Revision Log dialog, and when composing log messages in the Commit dialog.

Short date / time format in log messages
: If the standard long messages use up too much space on your screen use the short format.

Can double-click in log list to compare with previous revision
: If you frequently find yourself comparing revisions in the top pane of the log dialog, you can use this option to allow that action on double-click. It is not enabled by default because fetching the diff is often a long process, and many people prefer to avoid the wait after an accidental double-click, which is why this option is not enabled by default.

Janela de Progresso
: TortoiseSVN can automatically close all progress dialogs when the action is finished without error. This setting allows you to select the conditions for closing the dialogs. The default (recommended) setting is Close manually which allows you to review all messages and check what has happened. However, you may decide that you want to ignore some types of message and have the dialog close automatically if there are no critical changes.

  Auto-close if no merges, adds or deletes means that the progress dialog will close if there were simple updates, but if changes from the repository were merged with yours, or if any files were added or deleted, the dialog will remain open. It will also stay open if there were any conflicts or errors during the operation.

Auto-close if no merges, adds or deletes for local operations means that the progress dialog will close as for Auto-close if no merges, adds or deletes but only for local operations like adding files or reverting changes. For remote operations the dialog will stay open.

Auto-close if no conflicts relaxes the criteria further and will close the dialog even if there were merges, adds or deletes. However, if there were any conflicts or errors, the dialog remains open.

Auto-close if no errors always closes the dialog even if there were conflicts. The only condition that keeps the dialog open is an error condition, which occurs when Subversion is unable to complete the task. For example, an update fails because the server is inaccessible, or a commit fails because the working copy is out-of-date.

Use recycle bin when reverting
: When you revert local modifications, your changes are discarded. TortoiseSVN gives you an extra safety net by sending the modified file to the recycle bin before bringing back the pristine copy. If you prefer to skip the recycle bin, uncheck this option.

Use URL of WC as the default “From:” URL
: In the merge dialog, the default behaviour is for the From: URL to be remembered between merges. However, some people like to perform merges from many different points in their hierarchy, and find it easier to start out with the URL of the current working copy. This can then be edited to refer to a parallel path on another branch.

Default checkout path
: You can specify the default path for checkouts. If you keep all your checkouts in one place, it is useful to have the drive and folder pre-filled so you only have to add the new folder name to the end.

Default checkout URL
: You can also specify the default URL for checkouts. If you often checkout sub-projects of some very large project, it can be useful to have the URL pre-filled so you only have to add the sub-project name to the end.

4.30.1.3. TortoiseSVN Dialog Settings 2

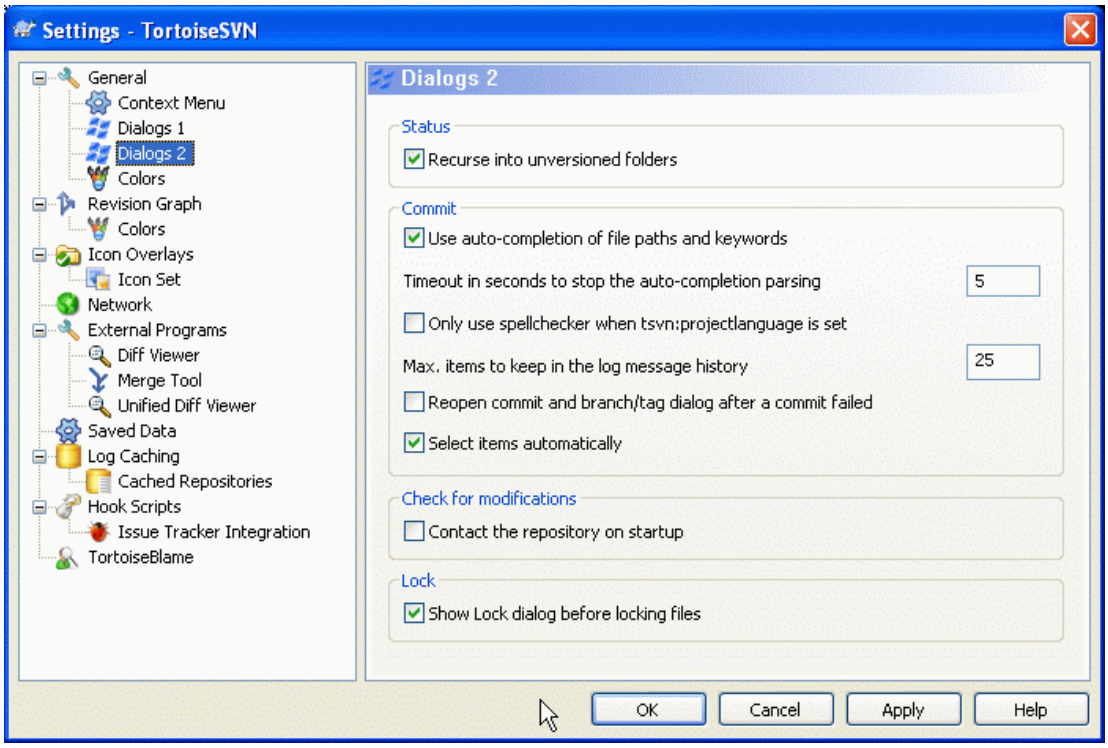


Figura 4.53. The Settings Dialog, Dialogs 2 Page

Recurse into unversioned folders
: If this box is checked (default state), then whenever the status of an unversioned folder is shown in the Add, Commit or Check for Modifications dialog, every child file and folder is also shown. If you uncheck this box, only the unversioned parent is shown. Unchecking reduces clutter in these dialogs. In that case if you select an unversioned folder for Add, it is added recursively.

Use auto-completion of file paths and keywords
: The commit dialog includes a facility to parse the list of filenames being committed. When you type the first 3 letters of an item in the list, the auto-completion box pops up, and you can press Enter to complete the filename. Check the box to enable this feature.

Timeout in seconds to stop the auto-completion parsing
: The auto-completion parser can be quite slow if there are a lot of large files to check. This timeout stops the commit dialog being held up for too long. If you are missing important auto-completion information, you can extend the timeout.

Only use spellchecker when `tsvn:projectlanguage` is set
: If you don't wish to use the spellchecker for all commits, check this box. The spellchecker will still be enabled where the project properties require it.

Max. items to keep in the log message history
: When you type in a log message in the commit dialog, TortoiseSVN stores it for possible re-use later. By default it will keep the last 25 log messages for each repository, but you can customize that number here. If you have many different repositories, you may wish to reduce this to avoid filling your registry.

  Note that this setting applies only to messages that you type in on this computer. It has nothing to do with the log cache.

Re-open commit and branch/tag dialog after a commit failed
: When a commit fails for some reason (working copy needs updating, pre-commit hook rejects commit, network error, etc), you can select this option to keep the commit dialog open ready to try again. However, you should be aware that this can lead to problems. If the failure means you need to update your working copy, and that update leads to conflicts you must resolve those first.

Select items automatically
: The normal behaviour in the commit dialog is for all modified (versioned) items to be selected for commit automatically. If you prefer to start with nothing selected and pick the items for commit manually, uncheck this box.

Contact the repository on startup
: The Check for Modifications dialog checks the working copy by default, and only contacts the repository when you click Check repository. If you always want to check the repository, you can use this setting to make that action happen automatically.

Show Lock dialog before locking files
: When you select one or more files and then use TortoiseSVN → Lock to take out a lock on those files, on some projects it is customary to write a lock message explaining why you have locked the files. If you do not use lock messages, you can uncheck this box to skip that dialog and lock the files immediately.

  If you use the lock command on a folder, you are always presented with the lock dialog as that also gives you the option to select files for locking.

  If your project is using the `tsvn:lockmsgminsize` property, you will see the lock dialog regardless of this setting because the project *requires* lock messages.

4.30.1.4. TortoiseSVN Colour Settings

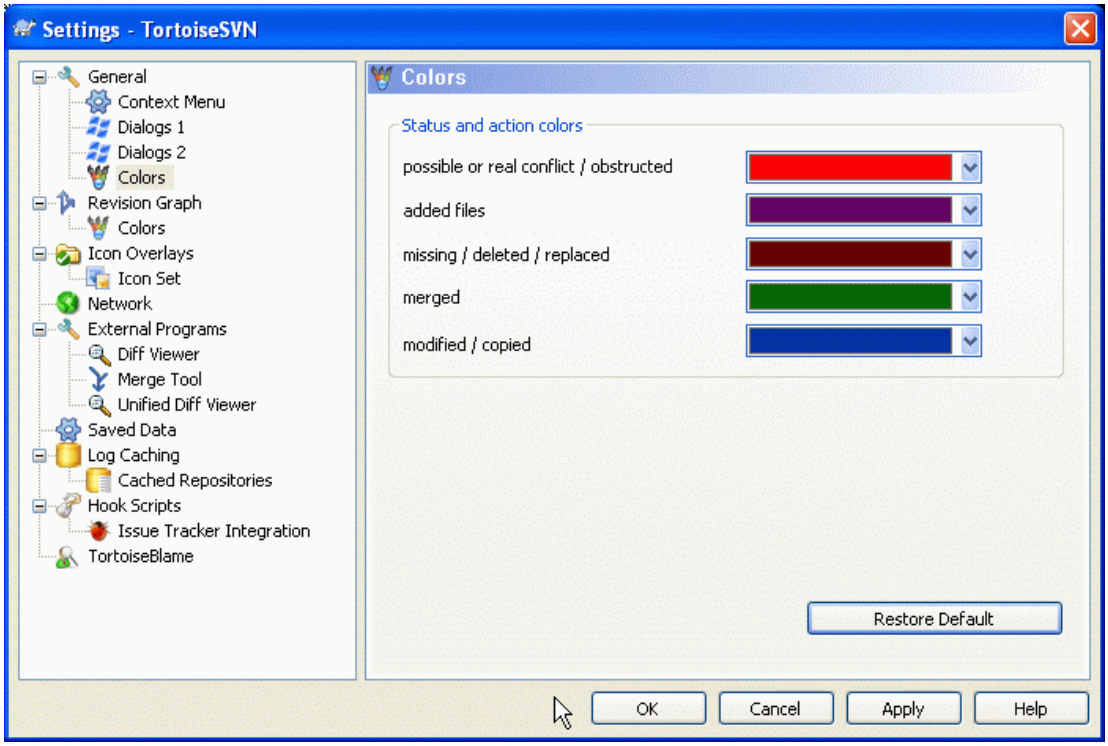


Figura 4.54. The Settings Dialog, Colours Page

This dialog allows you to configure the text colours used in TortoiseSVN's dialogs the way you like them.

Possible or real conflict / obstructed
: A conflict has occurred during update, or may occur during merge. Update is obstructed by an existing unversioned file/folder of the same name as a versioned one.

  This colour is also used for error messages in the progress dialogs.

Arquivos adicionados
: Items added to the repository.

Missing / deleted / replaced
: Items deleted from the repository, missing from the working copy, or deleted from the working copy and replaced with another file of the same name.

Combinado
: Changes from the repository successfully merged into the WC without creating any conflicts.

Modificado / copiado
: Add with history, or paths copied in the repository. Also used in the log dialog for entries which include copied items.

Nó excluído
: An item which has been deleted from the repository.

Nó adicionado
: An item which has been added to the repository, by an add, copy or move operation.

Nó renomeado
: An item which has been renamed within the repository.

Nó substituido
: The original item has been deleted and a new item with the same name replaces it.

4.30.2. Revision Graph Settings

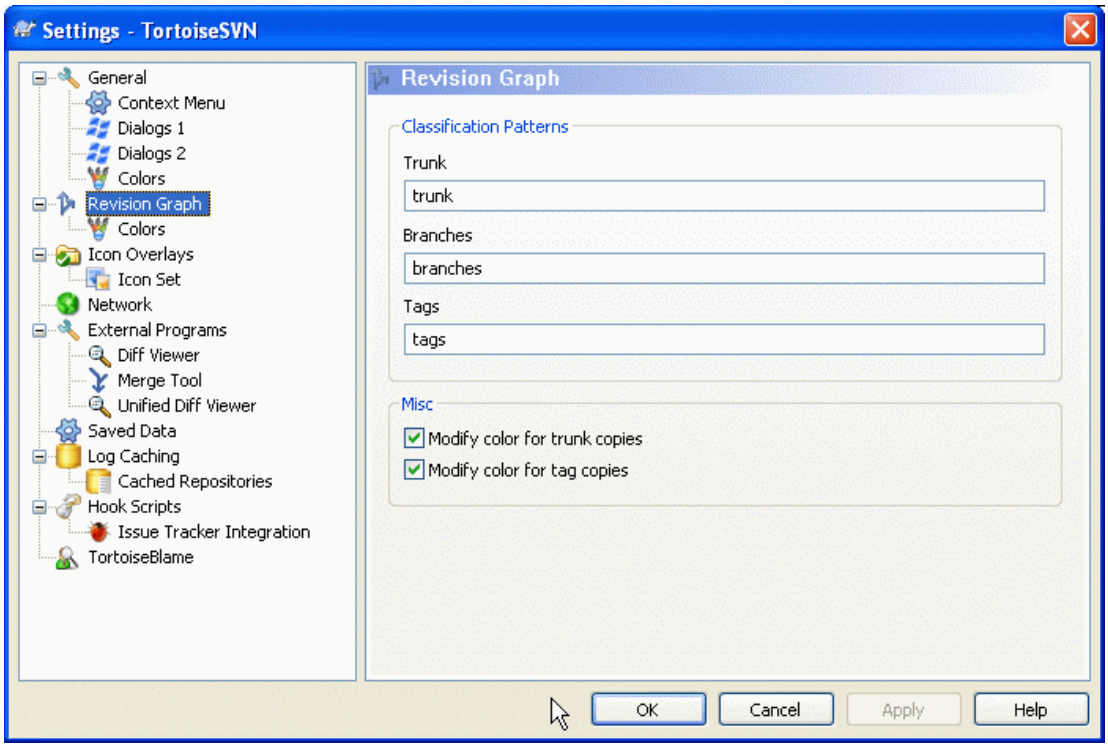


Figura 4.55. The Settings Dialog, Revision Graph Page

Classification Patterns
: The revision graph attempts to show a clearer picture of your repository structure by distinguishing between trunk, branches and tags. As there is no such classification built into Subversion, this information is extracted from the path names. The default settings assume that you use the conventional English names as suggested in the Subversion documentation, but of course your usage may vary.

  Specify the patterns used to recognise these paths in the three boxes provided. The patterns will be matched case-insensitively, but you must specify them in lower case. Wild cards `*` and `?` will work as usual, and you can use `;` to separate multiple patterns. Do not include any extra white space as it will be included in the matching specification.

Cores Modificadas
: Colors are used in the revision graph to indicate the node type, i.e. whether a node is added, deleted, renamed. In order to help pick out node classifications, you can allow the revision graph to blend colors to give an indication of both node type and classification. If the box is checked, blending is used. If the box is unchecked, color is used to indicate node type only. Use the color selection dialog to allocate the specific colors used.

4.30.2.1. Cores do Gráfico de Revisões

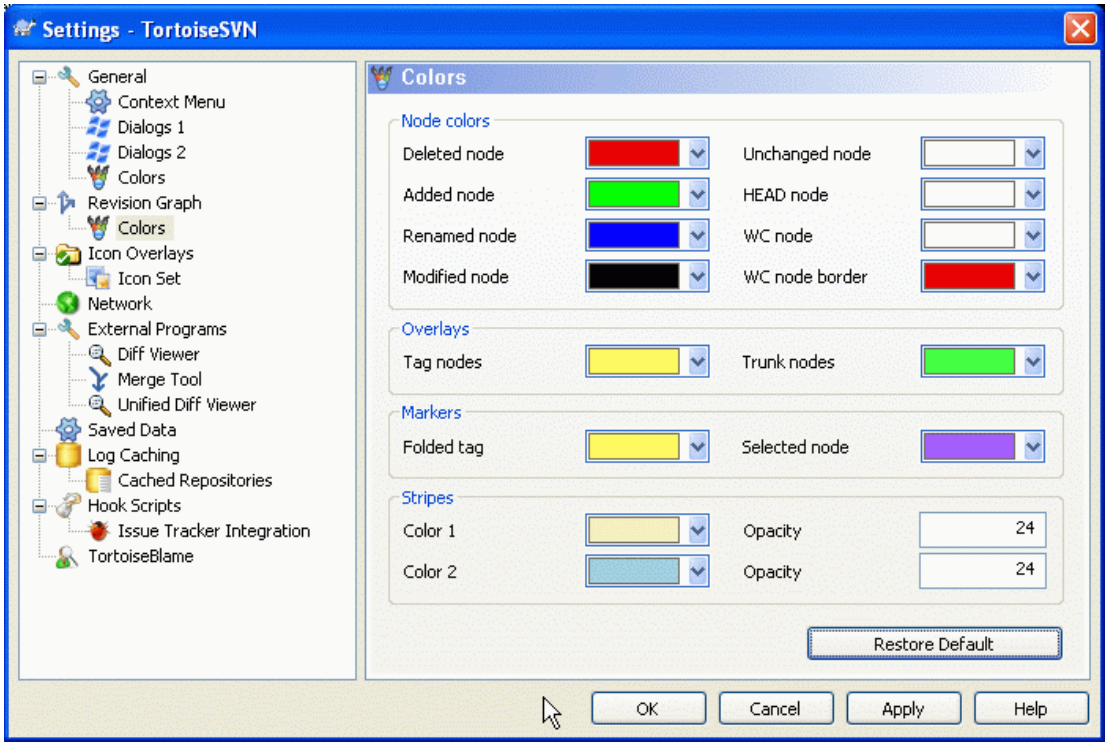


Figura 4.56. The Settings Dialog, Revision Graph Colors Page

This page allows you to configure the colors used. Note that the color specified here is the solid color. Most nodes are colored using a blend of the node type color, the background color and optionally the classification color.

Nó Excluído
: Items which have been deleted and not copied anywhere else in the same revision.

Nó Adicionado
: Items newly added, or copied (add with history).

Nó Renomeado
: Items deleted from one location and added in another in the same revision.

Nó Modificado
: Simple modifications without any add or delete.

Unchanged Node
: May be used to show the revision used as the source of a copy, even when no change (to the item being graphed) took place in that revision.

Nó Principal
: Current HEAD revision in the repository.

Nó CT
: If you opt to show an extra node for your modified working copy, attached to its last-commit revision on the graph, use this color.

Limite do nó CT
: If you opt to show whether the working copy is modified, use this color border on the WC node when modifications are found.

Nós de rótulos
: Nodes classified as tags may be blended with this color.

Nós do tronco
: Nodes classified as trunk may be blended with this color.

Folded Tag Markers
: If you use tag folding to save space, tags are marked on the copy source using a block in this color.

Selected Node Markers
: When you left click on a node to select it, the marker used to indicate selection is a block in this color.

Stripes
: These colors are used when the graph is split into sub-trees and the background is colored in alternating stripes to help pick out the separate trees.

4.30.3. Icon Overlay Settings

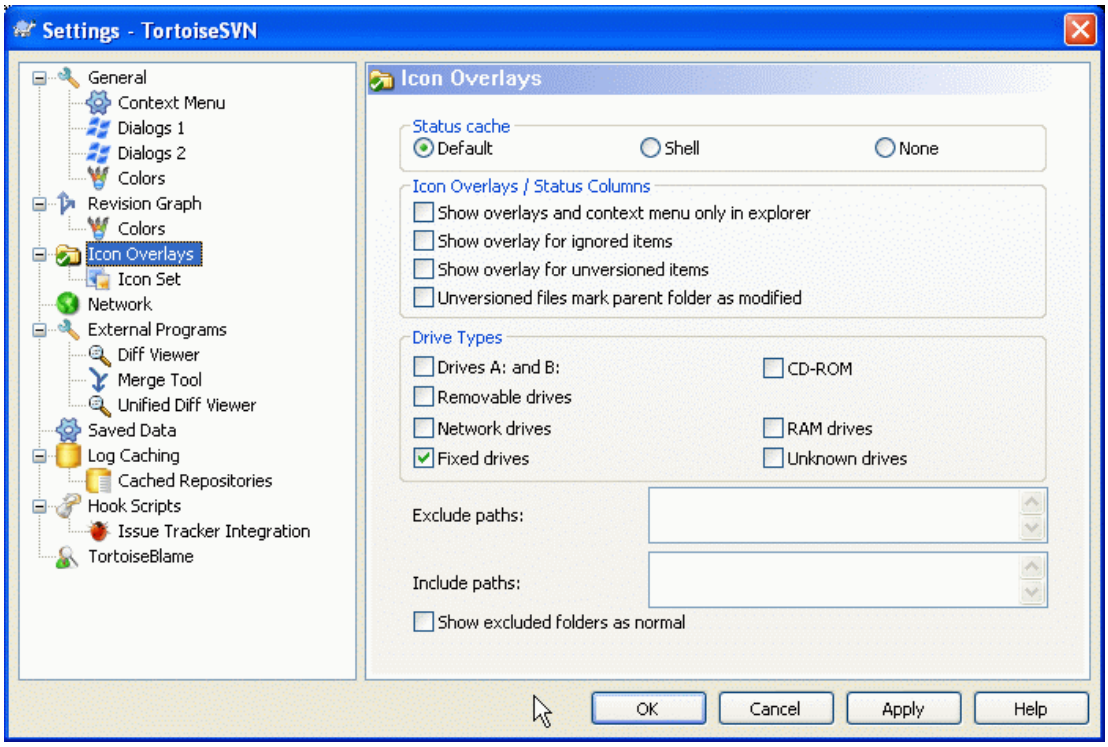


Figura 4.57. The Settings Dialog, Icon Overlays Page

This page allows you to choose the items for which TortoiseSVN will display icon overlays.

By default, overlay icons and context menus will appear in all open/save dialogs as well as in Windows Explorer. If you want them to appear *only* in Windows Explorer, check the Show overlays and context menu only in explorer box.

Ignored items and Unversioned items are not usually given an overlay. If you want to show an overlay in these cases, just check the boxes.

You can also choose to mark folders as modified if they contain unversioned items. This could be useful for reminding you that you have created new files which are not yet versioned. This option is only available when you use the *default* status cache option (see below).

Since it takes quite a while to fetch the status of a working copy, TortoiseSVN uses a cache to store the status so the explorer doesn't get hogged too much when showing the overlays. You can choose which type of cache TortoiseSVN should use according to your system and working copy size here:

Padrão
: Caches all status information in a separate process (`TSVNCache.exe`). That process watches all drives for changes and fetches the status again if files inside a working copy get modified. The process runs with the least possible priority so other programs don't get hogged because of it. That also means that the status information is not *real time* but it can take a few seconds for the overlays to change.

  Advantage: the overlays show the status recursively, i.e. if a file deep inside a working copy is modified, all folders up to the working copy root will also show the modified overlay. And since the process can send notifications to the shell, the overlays on the left tree view usually change too.

  Disadvantage: the process runs constantly, even if you're not working on your projects. It also uses around 10-50 MB of RAM depending on number and size of your working copies.

Shell
: Caching is done directly inside the shell extension dll, but only for the currently visible folder. Each time you navigate to another folder, the status information is fetched again.

  Advantage: needs only very little memory (around 1 MB of RAM) and can show the status in *real time*.

  Disadvantage: Since only one folder is cached, the overlays don't show the status recursively. For big working copies, it can take more time to show a folder in explorer than with the default cache. Also the mime-type column is not available.

Nenhum
: With this setting, the TortoiseSVN does not fetch the status at all in Explorer. Because of that, files don't get an overlay and folders only get a 'normal' overlay if they're versioned. No other overlays are shown, and no extra columns are available either.

  Advantage: uses absolutely no additional memory and does not slow down the Explorer at all while browsing.

  Disadvantage: Status information of files and folders is not shown in Explorer. To see if your working copies are modified, you have to use the “Check for modifications” dialog.

The next group allows you to select which classes of storage should show overlays. By default, only hard drives are selected. You can even disable all icon overlays, but where's the fun in that?

Network drives can be very slow, so by default icons are not shown for working copies located on network shares.

USB Flash drives appear to be a special case in that the drive type is identified by the device itself. Some appear as fixed drives, and some as removable drives.

The Exclude Paths are used to tell TortoiseSVN those paths for which it should *not* show icon overlays and status columns. This is useful if you have some very big working copies containing only libraries which you won't change at all and therefore don't need the overlays. For example:

`f:\development\SVN\Subversion` will disable the overlays *only* on that specific folder. You still can see the overlays on all files and folder inside that folder.

`f:\development\SVN\Subversion*` will disable the overlays on *all* files and folders whose path starts with `f:\development\SVN\Subversion`. That means you won't see overlays for any files and folders below that path.

The same applies to the Include Paths. Except that for those paths the overlays are shown even if the overlays are disabled for that specific drive type, or by an exclude path specified above.

Users sometimes ask how these three settings interact, and the definitive answer is:

```
if (path is in include list)
  show overlays
if (path is allowed drive type) AND (path is not in exclude list)
  show overlays
```

The include list *always* makes the overlays show. Otherwise, overlays are shown for all marked drive types *unless* the path is excluded.

TSVNCache.exe also uses these paths to restrict its scanning. If you want it to look only in particular folders, disable all drive types and include only the folders you specifically want to be scanned.

Exclude SUBST Drives

It is often convenient to use a SUBST drive to access your working copies, e.g. using the command

```
subst T: C:\TortoiseSVN\trunk\doc
```

However this can cause the overlays not to update, as TSVNCache will only receive one notification when a file changes, and that is normally for the original path. This means that your overlays on the subst path may never be updated.

An easy way to work around this is to exclude the original path from showing overlays, so that the overlays show up on the subst path instead.

Sometimes you will exclude areas that contain working copies, which saves TSVNCache from scanning and monitoring for changes, but you still want a visual indication that such folders are versioned. The Show excluded folders as 'normal' checkbox allows you to do this. With this option, versioned folders in any excluded area (drive type not checked, or specifically excluded) will show up as normal and up-to-date, with a green check mark. This reminds you that you are looking at a working copy, even though the folder overlays may not be correct. Files do not get an overlay at all. Note that the context menus still work, even though the overlays are not shown.

As a special exception to this, drives A: and B: are never considered for the Show excluded folders as 'normal' option. This is because Windows is forced to look on the drive, which can result in a delay of several seconds when starting Explorer, even if your PC does have a floppy drive.

4.30.3.1. Icon Set Selection

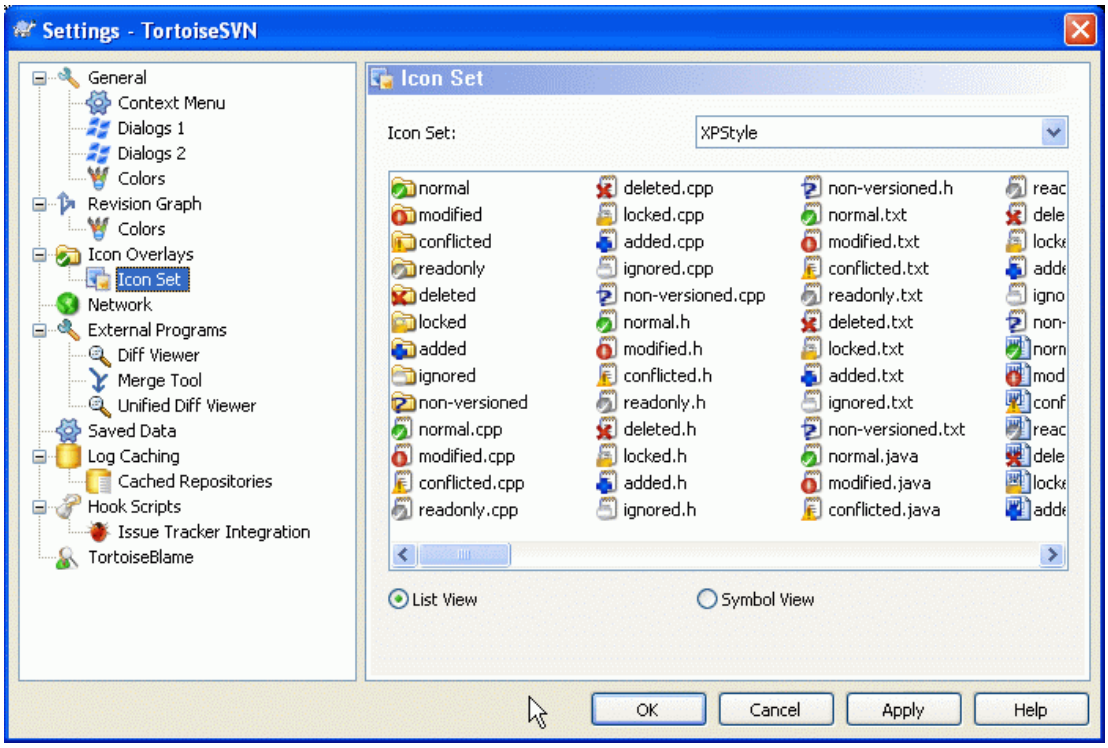


Figura 4.58. The Settings Dialog, Icon Set Page

You can change the overlay icon set to the one you like best. Note that if you change overlay set, you may have to restart your computer for the changes to take effect.

4.30.4. Network Settings

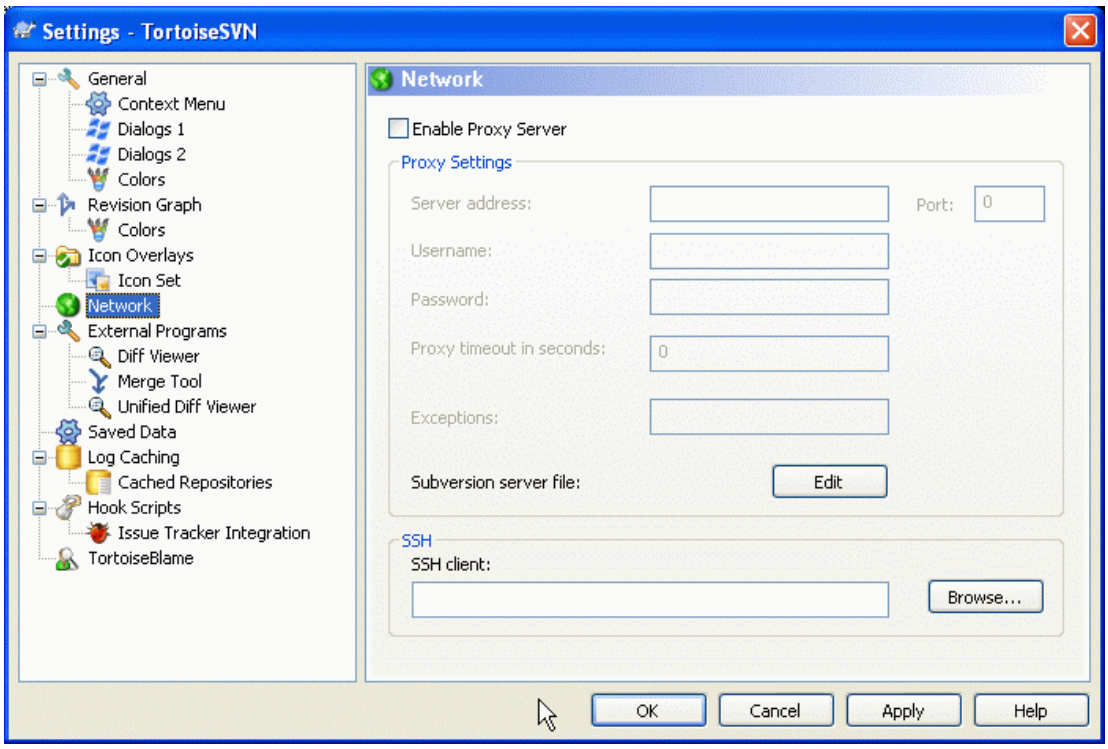


Figura 4.59. The Settings Dialog, Network Page

Here you can configure your proxy server, if you need one to get through your company's firewall.

If you need to set up per-repository proxy settings, you will need to use the Subversion `servers` file to configure this. Use Edit to get there directly. Consult the *Runtime Configuration Area* [http://svnbook.red-bean.com/en/1.5/svn.advanced.confarea.html] for details on how to use this file.

You can also specify which program TortoiseSVN should use to establish a secure connection to a svn +ssh repository. We recommend that you use TortoisePlink.exe. This is a version of the popular Plink program, and is included with TortoiseSVN, but it is compiled as a Windowless app, so you don't get a DOS box popping up every time you authenticate.

You must specify the full path to the executable. For TortoisePlink.exe this is the standard TortoiseSVN bin directory. Use the Browse button to help locate it. Note that if the path contains spaces, you must enclose it in quotes, e.g.

```
"C:\Program Files\TortoiseSVN\bin\TortoisePlink.exe"
```

One side-effect of not having a window is that there is nowhere for any error messages to go, so if authentication fails you will simply get a message saying something like “Unable to write to standard output”. For this reason we recommend that you first set up using standard Plink. When everything is working, you can use TortoisePlink with exactly the same parameters.

TortoisePlink does not have any documentation of its own because it is just a minor variant of Plink. Find out about command line parameters from the *PuTTY website* [http://www.chiark.greenend.org.uk/~sgtatham/putty/]

To avoid being prompted for a password repeatedly, you might also consider using a password caching tool such as Pageant. This is also available for download from the PuTTY website.

Finally, setting up SSH on server and clients is a non-trivial process which is beyond the scope of this help file. However, you can find a guide in the TortoiseSVN FAQ listed under *Subversion/TortoiseSVN SSH How-To* [http://tortoisesvn.net/ssh_howto].

4.30.5. External Program Settings

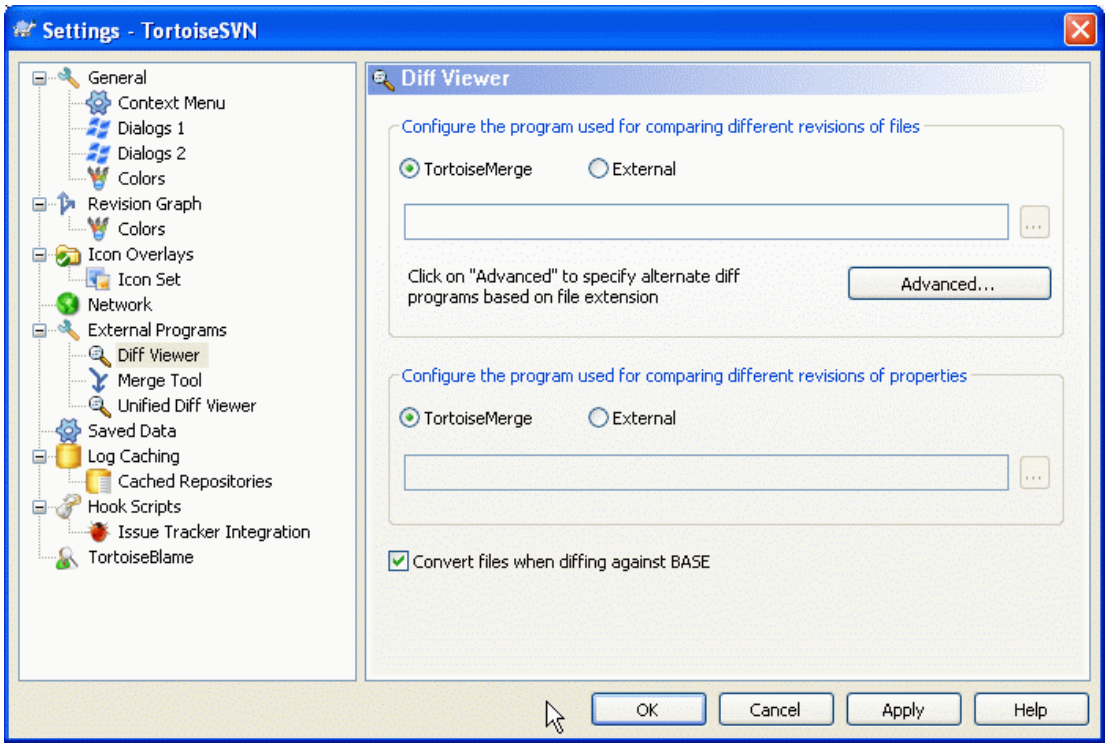


Figura 4.60. The Settings Dialog, Diff Viewer Page

Here you can define your own diff/merge programs that TortoiseSVN should use. The default setting is to use TortoiseMerge which is installed alongside TortoiseSVN.

Read Seção 4.10.5, “External Diff/Merge Tools” for a list of some of the external diff/merge programs that people are using with TortoiseSVN.

4.30.5.1. Visualizador de Diferenças

An external diff program may be used for comparing different revisions of files. The external program will need to obtain the filenames from the command line, along with any other command line options. TortoiseSVN uses substitution parameters prefixed with %. When it encounters one of these it will substitute the appropriate value. The order of the parameters will depend on the Diff program you use.

%base
: The original file without your changes

%bname
: The window title for the base file

%mine
: Your own file, with your changes

%yname
: The window title for your file

The window titles are not pure filenames. TortoiseSVN treats that as a name to display and creates the names accordingly. So e.g. if you're doing a diff from a file in revision 123 with a file in your working copy, the names will be `filename : revision 123` and `filename : working copy`

For example, with ExamDiff Pro:

```
C:\Path-To\ExamDiff.exe %base %mine --left_display_name:%bname
    --right_display_name:%yname
```

or with KDiff3:

```
C:\Path-To\kdiff3.exe %base %mine --L1 %bname --L2 %yname
```

or with WinMerge:

```
C:\Path-To\WinMerge.exe -e -ub -dl %bname -dr %yname %base %mine
```

or with Araxis:

```
C:\Path-To\compare.exe /max /wait /title1:%bname /title2:%yname
    %base %mine
```

If you use the `svn:keywords` property to expand keywords, and in particular the *revision* of a file, then there may be a difference between files which is purely due to the current value of the keyword. Also if you use `svn:eol-style = native` the BASE file will have pure `LF` line endings whereas your file will have `CR-LF` line endings. TortoiseSVN will normally hide these differences automatically by first parsing the BASE file to expand keywords and line endings before doing the diff operation. However, this can take a long time with large files. If Convert files when diffing against BASE is unchecked then TortoiseSVN will skip pre-processing the files.

You can also specify a different diff tool to use on Subversion properties. Since these tend to be short simple text strings, you may want to use a simpler more compact viewer.

If you have configured an alternate diff tool, you can access TortoiseMerge *and* the third party tool from the context menus. Context menu → Diff uses the primary diff tool, and **Shift**+ Context menu → Diff uses the secondary diff tool.

4.30.5.2. Ferramente de Unificação

An external merge program used to resolve conflicted files. Parameter substitution is used in the same way as with the Diff Program.

%base
: the original file without your or the others changes

%bname
: The window title for the base file

%mine
: your own file, with your changes

%yname
: The window title for your file

%theirs
: the file as it is in the repository

%tname
: The window title for the file in the repository

%merged
: the conflicted file, the result of the merge operation

%mname
The window title for the merged file

For example, with Perforce Merge:

```
C:\Path-To\P4Merge.exe %base %theirs %mine %merged
```

or with KDiff3:

```
C:\Path-To\kdiff3.exe %base %mine %theirs -o %merged
    --L1 %bname --L2 %yname --L3 %tname
```

or with Araxis:

```
C:\Path-To\compare.exe /max /wait /3 /title1:%tname /title2:%bname
    /title3:%yname %theirs %base %mine %merged /a2
```

or with WinMerge (2.8 or later):

```
C:\Path-To\WinMerge.exe %merged
```

4.30.5.3. Diff/Merge Advanced Settings

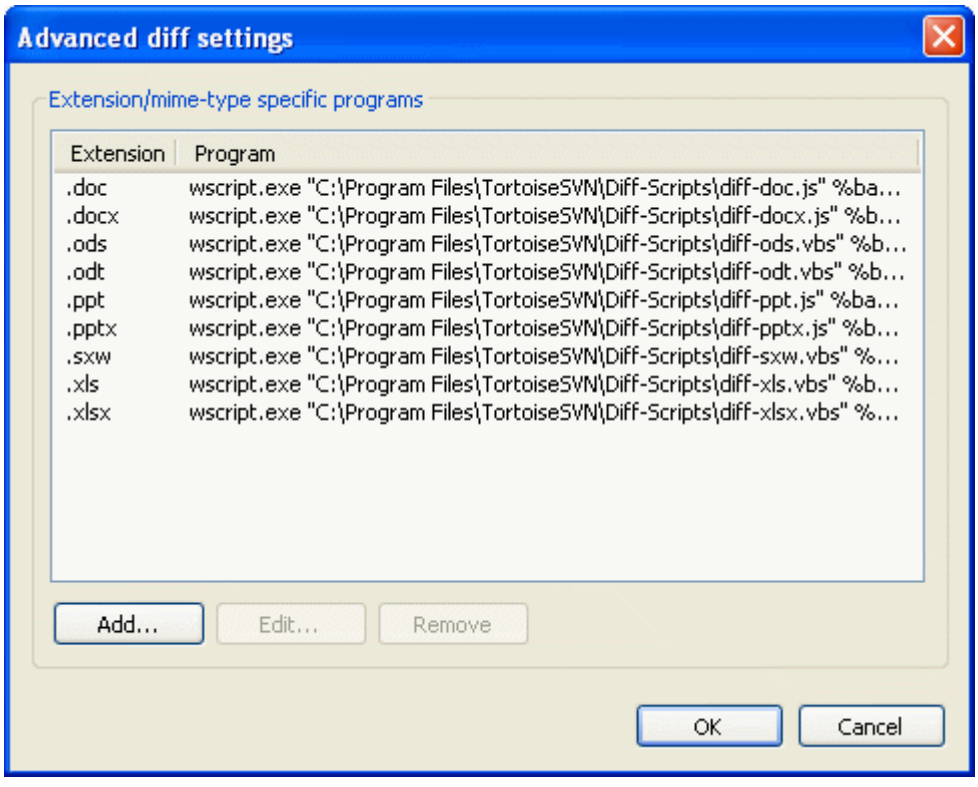


Figura 4.61. The Settings Dialog, Diff/Merge Advanced Dialog

In the advanced settings, you can define a different diff and merge program for every file extension. For instance you could associate Photoshop as the “Diff” Program for `.jpg` files :-) You can also associate the `svn:mime-type` property with a diff or merge program.

To associate using a file extension, you need to specify the extension. Use `.bmp` to describe Windows bitmap files. To associate using the `svn:mime-type` property, specify the mime type, including a slash, for example `text/xml`.

4.30.5.4. Unified Diff Viewer

A viewer program for unified-diff files (patch files). No parameters are required. The Default option is to check for a file association for `.diff` files, and then for `.txt` files. If you don't have a viewer for `.diff` files, you will most likely get NotePad.

The original Windows NotePad program does not behave well on files which do not have standard CR-LF line-endings. Since most unified diff files have pure LF line-endings, they do not view well in NotePad. However, you can download a free NotePad replacement *Notepad2* [http://www.flos-freeware.ch/notepad2.html] which not only displays the line-endings correctly, but also colour codes the added and removed lines.

4.30.6. Saved Data Settings

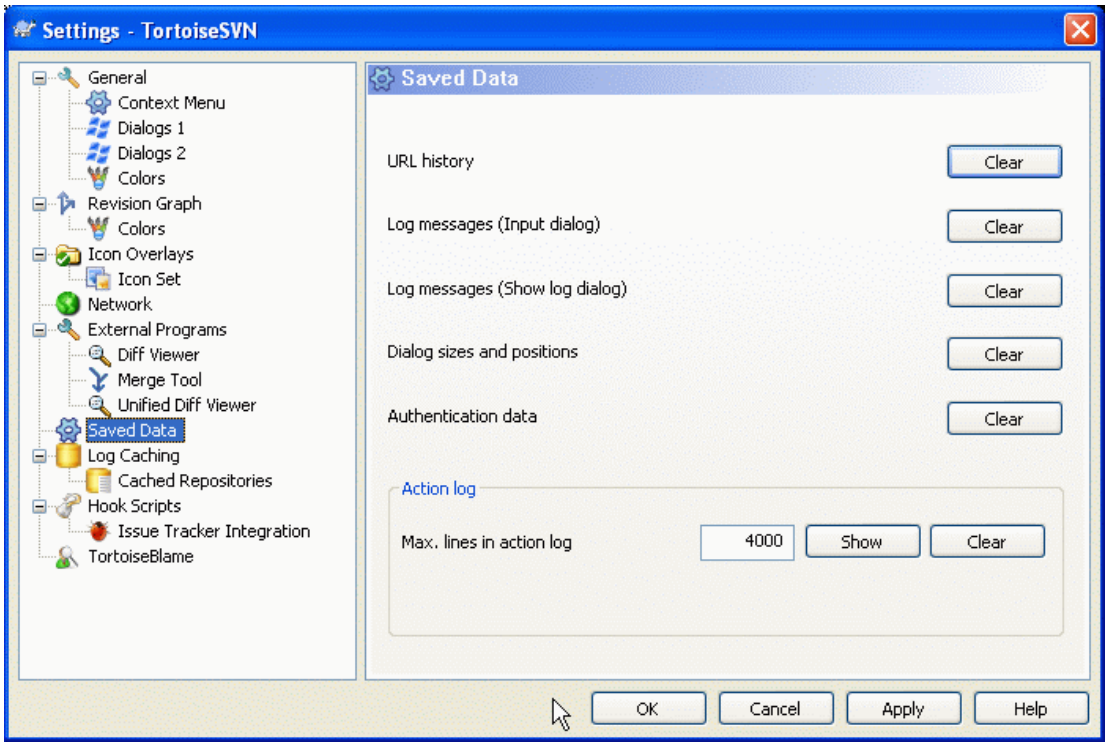


Figura 4.62. The Settings Dialog, Saved Data Page

For your convenience, TortoiseSVN saves many of the settings you use, and remembers where you have been lately. If you want to clear out that cache of data, you can do it here.

URL history
: Whenever you checkout a working copy, merge changes or use the repository browser, TortoiseSVN keeps a record of recently used URLs and offers them in a combo box. Sometimes that list gets cluttered with outdated URLs so it is useful to flush it out periodically.

  If you want to remove a single item from one of the combo boxes you can do that in-place. Just click on the arrow to drop the combo box down, move the mouse over the item you want to remove and type **Shift+Del**.

Log messages (Input dialog)
: TortoiseSVN stores recent commit log messages that you enter. These are stored per repository, so if you access many repositories this list can grow quite large.

Log messages (Show log dialog)
: TortoiseSVN caches log messages fetched by the Show Log dialog to save time when you next show the log. If someone else edits a log message and you already have that message cached, you will not see the change until you clear the cache. Log message caching is enabled on the Log Cache tab.

Dialog sizes and positions
: Many dialogs remember the size and screen position that you last used.

Authentication data
: When you authenticate with a Subversion server, the username and password are cached locally so you don't have to keep entering them. You may want to clear this for security reasons, or because you want to access the repository under a different username ... does John know you are using his PC?

  If you want to clear authentication data for one particular server only, read Seção 4.1.5, “Autenticação” for instructions on how to find the cached data.

Action log
: TortoiseSVN keeps a log of everything written to its progress dialogs. This can be useful when, for example, you want to check what happened in a recent update command.

  The log file is limited in length and when it grows too big the oldest content is discarded. By default 4000 lines are kept, but you can customize that number.

  From here you can view the log file content, and also clear it.

4.30.7. Log Caching

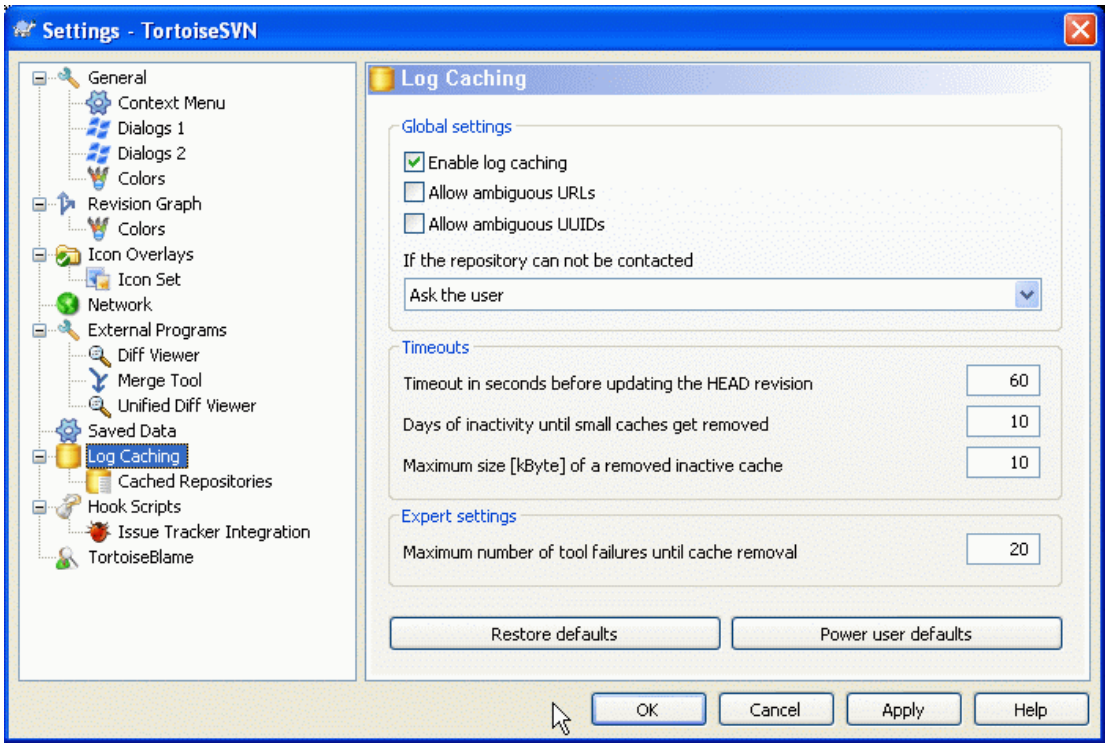


Figura 4.63. The Settings Dialog, Log Cache Page

This dialog allows you to configure the log caching feature of TortoiseSVN, which retains a local copy of log messages and changed paths to avoid time-consuming downloads from the server. Using the log cache can dramatically speed up the log dialog and the revision graph. Another useful feature is that the log messages can still be accessed when offline.

Enable log caching
: Enables log caching whenever log data is requested. If checked, data will be retrieved from the cache when available, and any messages not in the cache will be retrieved from the server and added to the cache.

  If caching is disabled, data will always be retrieved directly from the server and not stored locally.

Allow ambiguous URLs
: Occasionally you may have to connect to a server which uses the same URL for all repositories. Older versions of `svnbridge` would do this. If you need to access such repositories you will have to check this option. If you don't, leave it unchecked to improve performance.

Allow ambiguous UUIDs
: Some hosting services give all their repositories the same UUID. You may even have done this yourself by copying a repository folder to create a new one. For all sorts of reasons this is a bad idea - a UUID should be *unique*. However, the log cache will still work in this situation if you check this box. If you don't need it, leave it unchecked to improve performance.

If the repository cannot be contacted
: If you are working offline, or if the repository server is down, the log cache can still be used to supply log messages already held in the cache. Of course the cache may not be up-to-date, so there are options to allow you to select whether this feature should be used.

  When log data is being taken from the cache without contacting the server, the dialog using those message will show the offline state in its title bar.

Timeout before updating the HEAD revision
: When you invoke the log dialog you will normally want to contact the server to check for any newer log messages. If the timeout set here is non-zero then the server will only be contacted when the timeout has elapsed since the last time contact. This can reduce server round-trips if you open the log dialog frequently and the server is slow, but the data shown may not be completely up-to-date. If you want to use this feature we suggest using a value of 300 (5 minutes) as a compromise.

Days of inactivity until small caches get removed
: If you browse around a lot of repositories you will accumulate a lot of log caches. If you're not actively using them, the cache will not grow very big, so TortoiseSVN purges them after a set time by default. Use this item to control cache purging.

Maximum size of removed inactive caches
: Larger caches are more expensive to reacquire, so TortoiseSVN only purges small caches. Fine tune the threshold with this value.

Maximum number of tool failures before cache removal
: Occasionally something goes wrong with the caching and causes a crash. If this happens the cache is normally deleted automatically to prevent a recurrence of the problem. If you use the less stable nightly build you may opt to keep the cache anyway.

4.30.7.1. Cached Repositories

On this page you can see a list of the repositories that are cached locally, and the space used for the cache. If you select one of the repositories you can then use the buttons underneath.

Click on the Update to completely refresh the cache and fill in any holes. For a large repository this could be very time consuming, but useful if you are about to go offline and want the best available cache.

Click on the Export button to export the entire cache as a set of CSV files. This could be useful if you want to process the log data using an external program, although it is mainly useful to the developers.

Click on Delete to remove all cached data for the selected repositories. This does not disable caching for the repository so the next time you request log data, a new cache will be created.

4.30.7.2. Log Cache Statistics

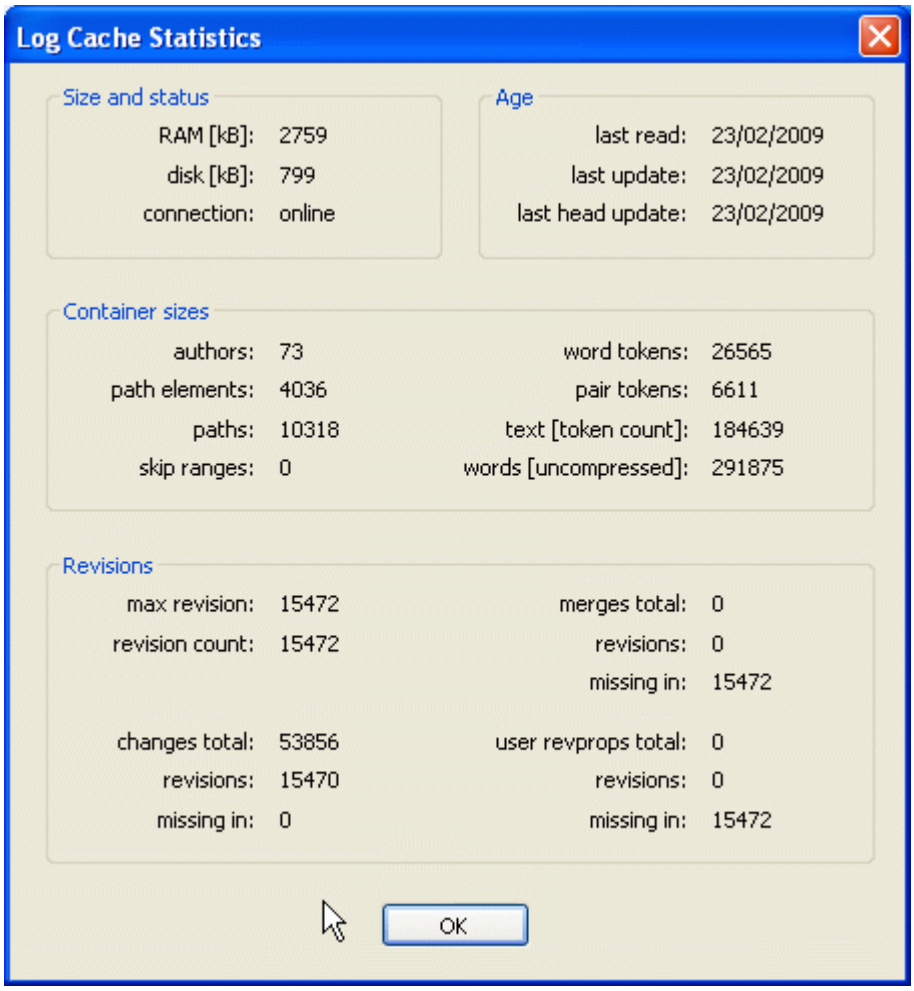


Figura 4.64. The Settings Dialog, Log Cache Statistics

Click on the Details button to see detailed statistics for a particular cache. Many of the fields shown here are mainly of interest to the developers of TortoiseSVN, so they are not all described in detail.

RAM
: The amount of memory required to service this cache.

Disco
: The amount of disk space used for the cache. Data is compressed, so disk usage is generally fairly modest.

Conexão
: Shows whether the repository was available last time the cache was used.

Última atualização
: The last time the cache content was changed.

Última atualização principal
: The last time we requested the HEAD revision from the server.

Autores
: The number of different authors with messages recorded in the cache.

Caminhos
: The number of paths listed, as you would see using `svn log -v`.

INtervalos omitidos
: The number of revision ranges which we have not fetched, simply because they haven't been requested. This is a measure of the number of holes in the cache.

Revisão máxima
: The highest revision number stored in the cache.

Número de revisões
: The number of revisions stored in the cache. This is another measure of cache completeness.

4.30.8. Client Side Hook Scripts

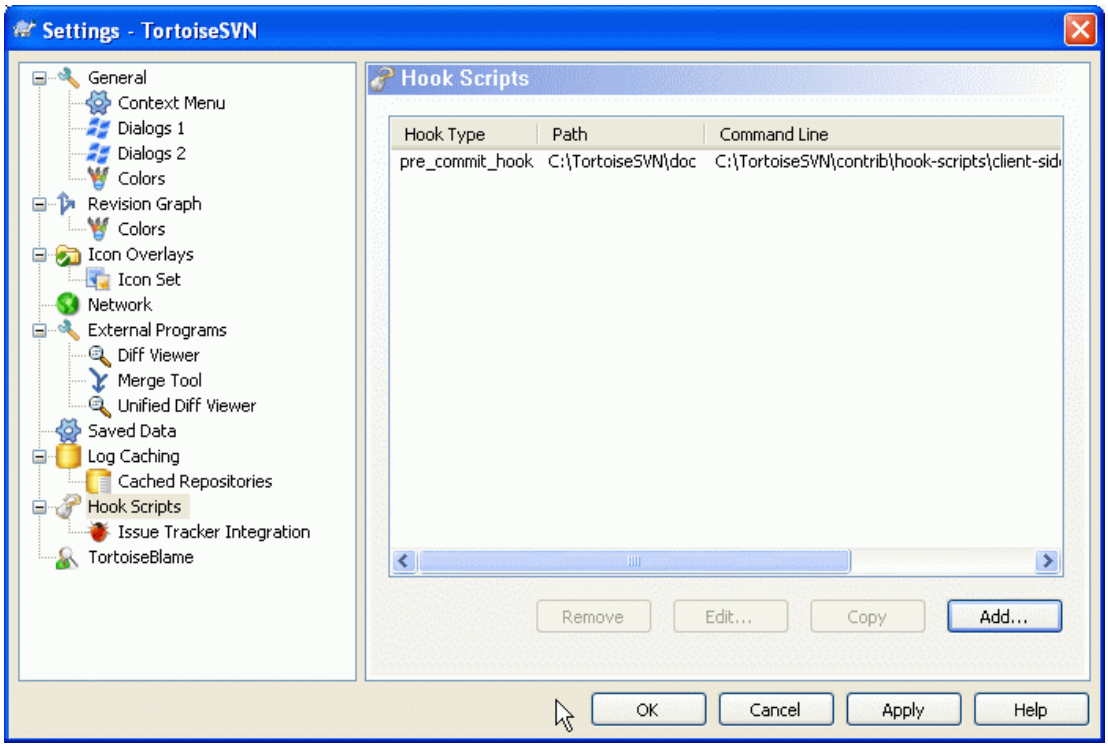


Figura 4.65. The Settings Dialog, Hook Scripts Page

This dialog allows you to set up hook scripts which will be executed automatically when certain Subversion actions are performed. As opposed to the hook scripts explained in Seção 3.3, “Rotinas de eventos no servidor”, these scripts are executed locally on the client.

One application for such hooks might be to call a program like `SubWCRev.exe` to update version numbers after a commit, and perhaps to trigger a rebuild.

For various security and implementation reasons, hook scripts are defined locally on a machine, rather than as project properties. You define what happens, no matter what someone else commits to the repository. Of course you can always choose to call a script which is itself under version control.

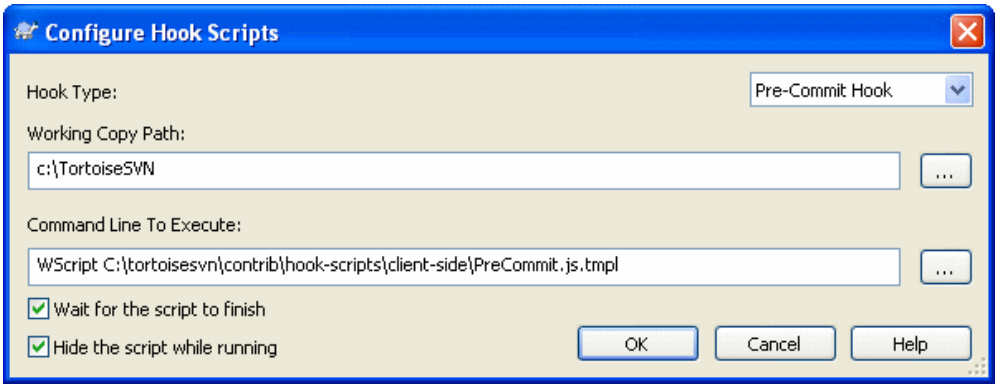


Figura 4.66. The Settings Dialog, Configure Hook Scripts

To add a new hook script, simply click Add and fill in the details.

There are currently six types of hook script available

Start-commit
: Called before the commit dialog is shown. You might want to use this if the hook modifies a versioned file and affects the list of files that need to be committed and/or commit message. However you should note that because the hook is called at an early stage, the full list of objects selected for commit is not available.

Pre-commit
: Called after the user clicks OK in the commit dialog, and before the actual commit begins. This hook has a list of exactly what will be committed.

Post-commit
: Called after the commit finishes (whether successful or not).

Start-update
: Called before the update-to-revision dialog is shown.

Pre-update
: Called before the actual Subversion update begins.

Post-update
: Called after the update finishes (whether successful or not).

A hook is defined for a particular working copy path. You only need to specify the top level path; if you perform an operation in a sub-folder, TortoiseSVN will automatically search upwards for a matching path.

Next you must specify the command line to execute, starting with the path to the hook script or executable. This could be a batch file, an executable file or any other file which has a valid windows file association, eg. a perl script.

The command line includes several parameters which get filled in by TortoiseSVN. The parameters passed depend upon which hook is called. Each hook has its own parameters which are passed in the following order:

Start-commit
: `PATHMESSAGEFILECWD`

Pre-commit
: `PATHDEPTHMESSAGEFILECWD`

Post-commit
: `PATHDEPTHMESSAGEFILEREVISIONERRORCWD`

Start-update
`PATHCWD`

Pre-update
`PATHDEPTHREVISIONCWD`

Post-update
`PATHDEPTHREVISIONERRORCWD`

The meaning of each of these parameters is described here:

PATH
A path to a temporary file which contains all the paths for which the operation was started. Each path is on a separate line in the temp file.

DEPTH
The depth with which the commit/update is done.

Possible values are:

-2
`svn_depth_unknown`

-1
`svn_depth_exclude`

0
`svn_depth_empty`

1
`svn_depth_files`

2
`svn_depth_immediates`

3
`svn_depth_infinity`

MESSAGEFILE
Path to a file containing the log message for the commit. The file contains the text in UTF-8 encoding. After successful execution of the start-commit hook, the log message is read back, giving the hook a chance to modify it.

REVISION
The repository revision to which the update should be done or after a commit completes.

ERROR
Path to a file containing the error message. If there was no error, the file will be empty.

CWD
The current working directory with which the script is run. This is set to the common root directory of all affected paths.

Note that although we have given these parameters names for convenience, you do not have to refer to those names in the hook settings. All parameters listed for a particular hook are always passed, whether you want them or not ;-)

If you want the Subversion operation to hold off until the hook has completed, check Wait for the script to finish.

Normally you will want to hide ugly DOS boxes when the script runs, so Hide the script while running is checked by default.

Sample client hook scripts can be found in the `contrib` folder in the *TortoiseSVN repository* [http://tortoisesvn.googlecode.com/svn/trunk/contrib/hook-scripts]. (Seção 3, "TortoiseSVN é grátis!" explains how to access the repository).

4.30.8.1. Issue Tracker Integration

TortoiseSVN can use a COM plugin to query issue trackers when in the commit dialog. The use of such plugins is described in Seção 4.28.2, "Getting Information from the Issue Tracker". If your system administrator has provided you with a plugin, which you have already installed and registered, this is the place to specify how it integrates with your working copy.

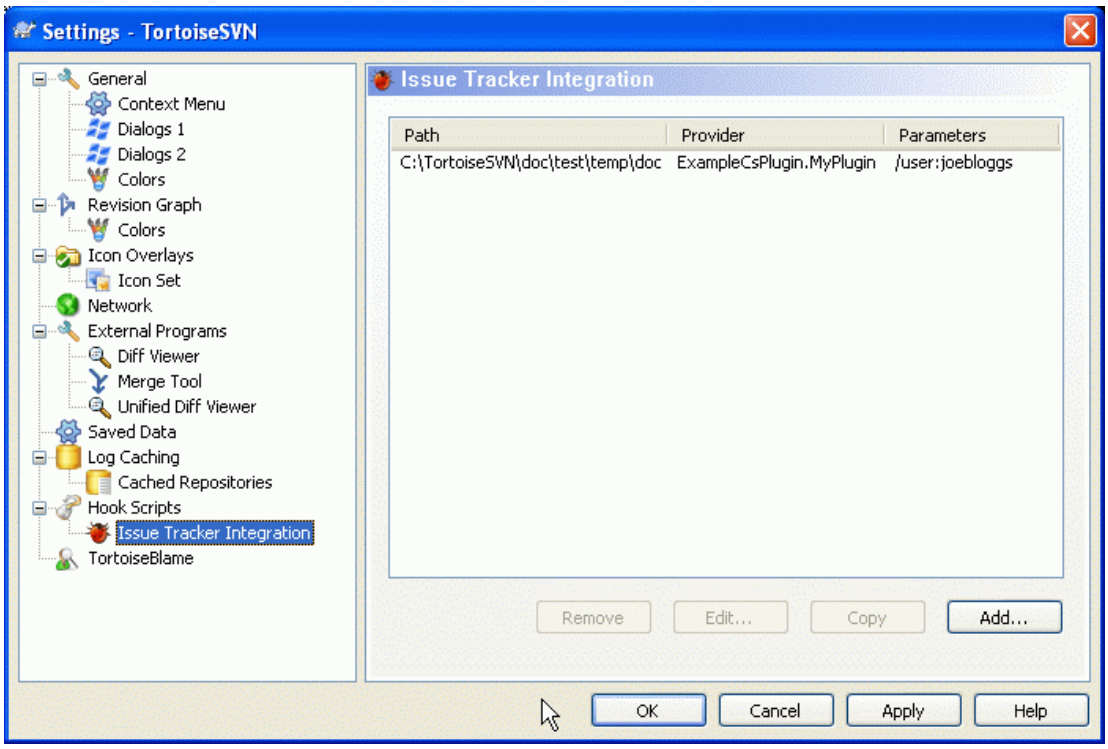


Figura 4.67. The Settings Dialog, Issue Tracker Integration Page

Click on Add... to use the plugin with a particular working copy. Here you can specify the working copy path, choose which plugin to use from a drop down list of all registered issue tracker plugins, and any parameters to pass. The parameters will be specific to the plugin, but might include your user name on the issue tracker so that the plugin can query for issues which are assigned to you.

If you want all users to use the same COM plugin for your project, you can specify the plugin also with the properties `bugtraq:provideruuid` and `bugtraq:providerparams`.

bugtraq:provideruuid
: This property specifies the COM UUID of the IBugtraqProvider, for example `{91974081-2DC7-4FB1-B3BE-0DE1C8D6CE4E}`. (this example is the UUID of the *Gurtle bugtraq provider* [http://code.google.com/p/gurtle/], which is a provider for the *Google Code* [http://code.google.com/hosting/] issue tracker).

bugtraq:providerparams
: This property specifies the parameters passed to the IBugtraqProvider.

Please check the documentation of your IBugtraqProvider plugin to find out what to specify in these two properties.

4.30.9. Configurações TortoiseBlame

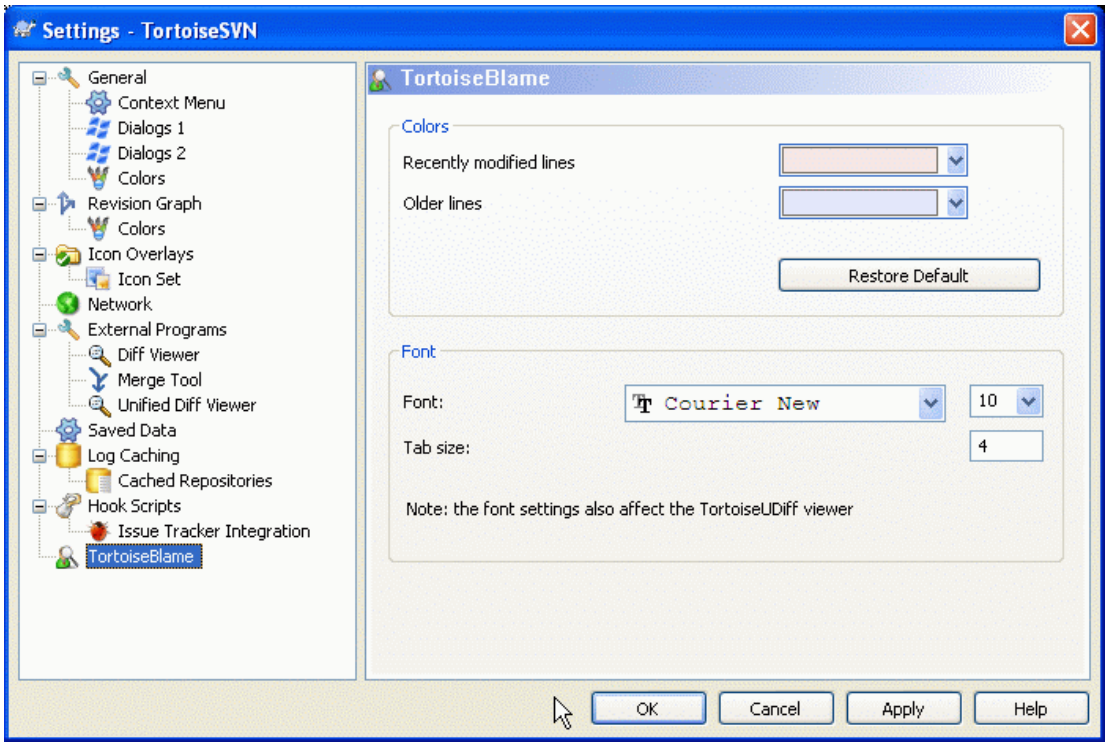


Figura 4.68. The Settings Dialog, TortoiseBlame Page

The settings used by TortoiseBlame are controlled from the main context menu, not directly with TortoiseBlame itself.

Cores
: TortoiseBlame can use the background colour to indicate the age of lines in a file. You set the endpoints by specifying the colours for the newest and oldest revisions, and TortoiseBlame uses a linear interpolation between these colours according to the repository revision indicated for each line.

Fonte
: You can select the font used to display the text, and the point size to use. This applies both to the file content, and to the author and revision information shown in the left pane.

Abas
: Defines how many spaces to use for expansion when a tab character is found in the file content.

4.30.10. Registry Settings

A few infrequently used settings are available only by editing the registry directly. It goes without saying that you should only edit registry values if you know what you are doing.

Configurações
: You can specify a different location for the Subversion configuration file using registry location `HKCU\Software\TortoiseSVN\ConfigDir`. This will affect all TortoiseSVN operations.

Cache tray icon
: To add a cache tray icon for the TSVNCache program, create a `DWORD` key with a value of 1 at `HKCU\Software\TortoiseSVN\CacheTrayIcon`. This is really only useful for developers as it allows you to terminate the program gracefully.

Debug
: To show the command line parameters passed from the shell extension to TortoiseProc.exe create a `DWORD` key with a value of 1 at `HKCU\Software\TortoiseSVN\Debug`.

Context Menu Icons
: This can be useful if you use something other than the windows explorer or if you get problems with the context menu displaying correctly. create a `DWORD` key with a value of 0 at `HKCU\Software\TortoiseSVN\ShowContextMenuIcons` if you don't want TortoiseSVN to not show icons for the shell context menu items. Set this value to 1 to show the icons again.

Block Overlay Status
: If you don't want the explorer to update the status overlays while another TortoiseSVN command is running (e.g. Update, Commit, ...) then create a `DWORD` key with a value of 1 at `HKCU\Software\TortoiseSVN\BlockStatus`.

Update Check URL
: `HKCU\Software\TortoiseSVN\UpdateCheckURL` contains the URL from which TortoiseSVN tries to download a text file to find out if there are updates available. You can also set this under `HKLM` instead of `HKCU` if you want, but `HKCU` overwrites the setting in `HKLM`. This might be useful for company admins who don't want their users to update TortoiseSVN until they approve it.

Filenames without extensions in auto-completion list
: The auto-completion list shown in the commit message editor displays the names of files listed for commit. To also include these names with extensions removed, create a `DWORD` key with a value of 1 at `HKCU\Software\TortoiseSVN\AutocompleteRemovesExtensions`.

Explorer columns everywhere
: The extra columns the TortoiseSVN adds to the details view in Windows Explorer are normally only active in a working copy. If you want those to be accessible everywhere, not just in working copies, create a `DWORD` key with a value of 1 at `HKCU\Software\TortoiseSVN\ColumnsEveryWhere`.

Merge log separator
: When you merge revisions from another branch, and merge tracking information is available, the log messages from the revisions you merge will be collected to make up a commit log message. A pre-defined string is used to separate the individual log messages of the merged revisions. If you prefer, you can create a `SZ` key at `HKCU\Software\TortoiseSVN\MergeLogSeparator` containing a separator string of your choice.

Always blame changes with TortoiseMerge
: TortoiseSVN allows you to assign external diff viewer. Most such viewers, however, are not suited for change blaming (Seção 4.23.2, “Diferenças de Autoria”), so you might wish to fall back to TortoiseMerge in this case. To do so, create a `DWORD` key with a value of 1 at `HKCU\Software\TortoiseSVN\DiffBlamesWithTortoiseMerge`.

Current revision highlighting for folders in log dialog
: The log dialog highlights the current working copy revision when the log is shown for a file. To do the same thing for a folder requires a working copy crawl, which is the default action, but it can be a slow operation for large working copies. If you want to change the operation of this feature you must create a `DWORD` registry key at `HKCU\Software\TortoiseSVN\RecursiveLogRev`. A value of 0 disables the feature (no highlighting for folders), a value of 1 (default) will fetch the status recursively (find the highest revision in the working copy tree), and a value of 2 will check the revision of the selected folder itself, but will not check any child items.

Make checkout fail if an item of the same name exists
: By default, if you checkout a working copy over an existing unversioned folder structure, as you might do after import, then any existing which differ from the repository content will be left unchanged and marked as modified. When you come to commit, it is your local copy which will then be sent back to the repository. Some people would prefer the checkout to fail if the

existing content differs, so that if two people add the same file the second person's version does not overwrite the original version by mistake. If you want to force checkouts to fail in this instance you must create a `DWORD` registry key with value 0 at `HKCU\Software\TortoiseSVN\AllowUnversionedObstruction`.

4.30.11. Pastas de Trabalho do Subversion

VS.NET 2003 when used with web projects can't handle the `.svn` folders that Subversion uses to store its internal information. This is not a bug in Subversion. The bug is in VS.NET 2003 and the frontpage extensions it uses.

> Note that the bug is fixed in VS2005 and later versions.

As of Version 1.3.0 of Subversion and TortoiseSVN, you can set the environment variable `SVN_ASP_DOT_NET_HACK`. If that variable is set, then Subversion will use `_svn` folders instead of `.svn` folders. You must restart your shell for that environment variable to take effect. Normally that means rebooting your PC. To make this easier, you can now do this from the general settings page using a simple checkbox - refer to Seção 4.30.1, “Configurações Gerais”.

For more information, and other ways to avoid this problem in the first place, check out the article about this in our *FAQ* [http://tortoisesvn.net/aspdotnethack].

4.31. Final Step

Doe!

Even though TortoiseSVN and TortoiseMerge are free, you can support the developers by sending in patches and play an active role in the development. You can also help to cheer us up during the endless hours we spend in front of our computers.

While working on TortoiseSVN we love to listen to music. And since we spend many hours on the project we need a *lot* of music. Therefore we have set up some wish-lists with our favourite music CDs and DVDs: *http://tortoisesvn.tigris.org/donate.html* Please also have a look at the list of people who contributed to the project by sending in patches or translations.

Capítulo 5. The SubWCRev Program

SubWCRev is Windows console program which can be used to read the status of a Subversion working copy and optionally perform keyword substitution in a template file. This is often used as part of the build process as a means of incorporating working copy information into the object you are building. Typically it might be used to include the revision number in an “About” box.

5.1. The SubWCRev Command Line

SubWCRev reads the Subversion status of all files in a working copy, excluding externals by default. It records the highest commit revision number found, and the commit timestamp of that revision, It also records whether there are local modifications in the working copy, or mixed update revisions. The revision number, update revision range and modification status are displayed on stdout.

SubWCRev.exe is called from the command line or a script, and is controlled using the command line parameters.

```
SubWCRev WorkingCopyPath [SrcVersionFile DstVersionFile] [-nmdfe]
```

`WorkingCopyPath` is the path to the working copy being checked. You can only use SubWCRev on working copies, not directly on the repository. The path may be absolute or relative to the current working directory.

If you want SubWCRev to perform keyword substitution, so that fields like repository revision and URL are saved to a text file, you need to supply a template file `SrcVersionFile` and an output file `DstVersionFile` which contains the substituted version of the template.

There are a number of optional switches which affect the way SubWCRev works. If you use more than one, they must be specified as a single group, eg. `-nm`, not `-n -m`.

Switch	Description
-n	If this switch is given, SubWCRev will exit with `ERRORLEVEL 7` if the working copy contains local modifications. This may be used to prevent building with uncommitted changes present.
-m	If this switch is given, SubWCRev will exit with `ERRORLEVEL 8` if the working copy contains mixed revisions. This may be used to prevent building with a partially updated working copy.
-d	If this switch is given, SubWCRev will exit with `ERRORLEVEL 9` if the destination file already exists.
-f	If this switch is given, SubWCRev will include the last-changed revision of folders. The default behaviour is to use only files when getting the revision numbers.
-e	If this switch is given, SubWCRev will examine directories which are included with `svn:externals`, but only if they are from the same repository. The default behaviour is to ignore externals.
-x	If this switch is given, SubWCRev will output the revision numbers in HEX.
-X	If this switch is given, SubWCRev will output the revision numbers in HEX, with '0X' prepended.

Tabela 5.1. List of available command line switches

5.2. Keyword Substitution

If a source and destination files are supplied, SubWCRev copies source to destination, performing keyword substitution as follows:

Keyword	Description
$WCREV$	Replaced with the highest commit revision in the working copy.
$WCDATE$	Replaced with the commit date/time of the highest commit revision. By default, international format is used: `yyyy-mm-dd hh:mm:ss`. Alternatively, you can specify a custom format which will be used with `strftime()`, for example: `$WCDATE=%a %b %d %I:%M:%S %p$`. For a list of available formatting characters, look at the *online reference* [http://www.cppreference.com/stddate/strftime.html].
$WCNOW$	Replaced with the current system date/time. This can be used to indicate the build time. Time formatting can be used as described for `$WCDATE$`.
$WCRANGE$	Replaced with the update revision range in the working copy. If the working copy is in a consistent state, this will be a single revision. If the working copy contains mixed revisions, either due to being out of date, or due to a deliberate update-to-revision, then the range will be shown in the form 100:200
$WCMIXED$	`$WCMIXED?TText:FText$` is replaced with `TText` if there are mixed update revisions, or `FText` if not.
$WCMODS$	`$WCMODS?TText:FText$` is replaced with `TText` if there are local modifications, or `FText` if not.
$WCURL$	Replaced with the repository URL of the working copy path passed to SubWCRev.
$WCINSVN$	`$WCINSVN?TText:FText$` is replaced with `TText` if the entry is versioned, or `FText` if not.
$WCNEEDSLOCK$	`$WCNEEDSLOCK?TText:FText$` is replaced with `TText` if the entry has the `svn:needs-lock` property set, or `FText` if not.
$WCISLOCKED$	`$WCISLOCKED?TText:FText$` is replaced with `TText` if the entry is locked, or `FText` if not.
$WCLOCKDATE$	Replaced with the lock date. Time formatting can be used as described for `$WCDATE$`.
$WCLOCKOWNER$	Replaced with the name of the lock owner.
$WCLOCKCOMMENT$	Replaced with the comment of the lock.

Tabela 5.2. List of available command line switches

Dica

Some of these keywords apply to single files rather than to an entire working copy, so it only makes sense to use these when SubWCRev is called to scan a single file. This applies to `$WCINSVN$`, `$WCNEEDSLOCK$`, `$WCISLOCKED$`, `$WCLOCKDATE$`, `$WCLOCKOWNER$` and `$WCLOCKCOMMENT$`.

5.3. Keyword Example

The example below shows how keywords in a template file are substituted in the output file.

```
// Test file for SubWCRev: testfile.tmpl

char *Revision = "$WCREV$";
char *Modified = "$WCMODS?Modified:Not modified$";
```

```
char *Date     = "$WCDATE$";
char *Range    = "$WCRANGE$";
char *Mixed    = "$WCMIXED?Mixed revision WC:Not mixed$";
char *URL      = "$WCURL$";

#if $WCMODS?1:0$
#error Source is modified
#endif

// End of file
```

After running `SubWCRev.exe path\to\workingcopy testfile.tmpl testfile.txt`, the output file `testfile.txt` would looks like this:

```
// Test file for SubWCRev: testfile.txt

char *Revision = "3701";
char *Modified = "Modified";
char *Date     = "2005/06/15 11:15:12";
char *Range    = "3699:3701";
char *Mixed    = "Mixed revision WC";
char *URL      = "http://project.domain.org/svn/trunk/src";

#if 1
#error Source is modified
#endif

// End of file
```

Dica

A file like this will be included in the build so you would expect it to be versioned. Be sure to version the template file, not the generated file, otherwise each time you regenerate the version file you need to commit the change, which in turn means the version file needs to be updated.

5.4. COM interface

If you need to access Subversion revision information from other programs, you can use the COM interface of SubWCRev. The object to create is `SubWCRev.object`, and the following methods are supported:

Método	Description
.GetWCInfo	This method traverses the working copy gathering the revision information. Naturally you must call this before you can access the information using the remaining methods. The first parameter is the path. The second parameter should be true if you want to include folder revisions. Equivalent to the `-f` command line switch. The third parameter should be true if you want to include svn:externals. Equivalent to the `-e` command line switch.
.Revision	The highest commit revision in the working copy. Equivalent to `$WCREV$`
.Date	The commit date/time of the highest commit revision. Equivalent to `$WCDATE$`

Método	Description
.Author	The author of the highest commit revision, that is, the last person to commit changes to the working copy.
.MinRev	The minimum update revision, as shown in `$WCRANGE$`
.MaxRev	The maximum update revision, as shown in `$WCRANGE$`
.HasModifications	True if there are local modifications
.Url	Replaced with the repository URL of the working copy path used in `GetWCInfo`. Equivalent to `$WCURL$`
.IsSvnItem	True if the item is versioned.
.NeedsLocking	True if the item has the `svn:needs-lock` property set.
.IsLocked	True if the item is locked.
.LockCreationDate	String representing the date when the lock was created, or an empty string if the item is not locked.
.LockOwner	String representing the lock owner, or an empty string if the item is not locked.
.LockComment	The message entered when the lock was created.

Tabela 5.3. COM/automation methods supported

The following example shows how the interface might be used.

```
// testCOM.js - javascript file
// test script for the SubWCRev COM/Automation-object

filesystem = new ActiveXObject("Scripting.FileSystemObject");

revObject1 = new ActiveXObject("SubWCRev.object");
revObject2 = new ActiveXObject("SubWCRev.object");
revObject3 = new ActiveXObject("SubWCRev.object");
revObject4 = new ActiveXObject("SubWCRev.object");

revObject1.GetWCInfo(
    filesystem.GetAbsolutePathName("."), 1, 1);
revObject2.GetWCInfo(
    filesystem.GetAbsolutePathName(".."), 1, 1);
revObject3.GetWCInfo(
    filesystem.GetAbsolutePathName("SubWCRev.cpp"), 1, 1);
revObject4.GetWCInfo(
    filesystem.GetAbsolutePathName("..\\.."), 1, 1);

wcInfoString1 = "Revision = " + revObject1.Revision +
    "\nMin Revision = " + revObject1.MinRev +
    "\nMax Revision = " + revObject1.MaxRev +
    "\nDate = " + revObject1.Date +
    "\nURL = " + revObject1.Url + "\nAuthor = " +
    revObject1.Author + "\nHasMods = " +
    revObject1.HasModifications + "\nIsSvnItem = " +
    revObject1.IsSvnItem + "\nNeedsLocking = " +
    revObject1.NeedsLocking + "\nIsLocked = " +
    revObject1.IsLocked + "\nLockCreationDate = " +
    revObject1.LockCreationDate + "\nLockOwner = " +
    revObject1.LockOwner + "\nLockComment = " +
```

```
    revObject1.LockComment;
wcInfoString2 = "Revision = " + revObject2.Revision +
    "\nMin Revision = " + revObject2.MinRev +
    "\nMax Revision = " + revObject2.MaxRev +
    "\nDate = " + revObject2.Date +
    "\nURL = " + revObject2.Url + "\nAuthor = " +
    revObject2.Author + "\nHasMods = " +
    revObject2.HasModifications + "\nIsSvnItem = " +
    revObject2.IsSvnItem + "\nNeedsLocking = " +
    revObject2.NeedsLocking + "\nIsLocked = " +
    revObject2.IsLocked + "\nLockCreationDate = " +
    revObject2.LockCreationDate + "\nLockOwner = " +
    revObject2.LockOwner + "\nLockComment = " +
    revObject2.LockComment;
wcInfoString3 = "Revision = " + revObject3.Revision +
    "\nMin Revision = " + revObject3.MinRev +
    "\nMax Revision = " + revObject3.MaxRev +
    "\nDate = " + revObject3.Date +
    "\nURL = " + revObject3.Url + "\nAuthor = " +
    revObject3.Author + "\nHasMods = " +
    revObject3.HasModifications + "\nIsSvnItem = " +
    revObject3.IsSvnItem + "\nNeedsLocking = " +
    revObject3.NeedsLocking + "\nIsLocked = " +
    revObject3.IsLocked + "\nLockCreationDate = " +
    revObject3.LockCreationDate + "\nLockOwner = " +
    revObject3.LockOwner + "\nLockComment = " +
    revObject3.LockComment;
wcInfoString4 = "Revision = " + revObject4.Revision +
    "\nMin Revision = " + revObject4.MinRev +
    "\nMax Revision = " + revObject4.MaxRev +
    "\nDate = " + revObject4.Date +
    "\nURL = " + revObject4.Url + "\nAuthor = " +
    revObject4.Author + "\nHasMods = " +
    revObject4.HasModifications + "\nIsSvnItem = " +
    revObject4.IsSvnItem + "\nNeedsLocking = " +
    revObject4.NeedsLocking + "\nIsLocked = " +
    revObject4.IsLocked + "\nLockCreationDate = " +
    revObject4.LockCreationDate + "\nLockOwner = " +
    revObject4.LockOwner + "\nLockComment = " +
    revObject4.LockComment;

WScript.Echo(wcInfoString1);
WScript.Echo(wcInfoString2);
WScript.Echo(wcInfoString3);
WScript.Echo(wcInfoString4);
```

Capítulo 6. IBugtraqProvider interface

To get a tighter integration with issue trackers than by simply using the `bugtraq:` properties, TortoiseSVN can make use of COM plugins. With such plugins it is possible to fetch information directly from the issue tracker, interact with the user and provide information back to TortoiseSVN about open issues, verify log messages entered by the user and even run actions after a successful commit to e.g, close an issue.

We can't provide information and tutorials on how you have to implement a COM object in your preferred programming language, but we have example plugins in C++/ATL and C# in our repository in the `contrib/issue-tracker-plugins` folder. In that folder you can also find the required include files you need to build your plugin. (Seção 3, “TortoiseSVN é grátis!” explains how to access the repository).

6.1. The IBugtraqProvider interface

TortoiseSVN 1.5 can use plugins which implement the IBugtraqProvider interface. The interface provides a few methods which plugins can use to interact with the issue tracker.

```
HRESULT ValidateParameters (
  // Parent window for any UI that needs to be
  // displayed during validation.
  [in] HWND hParentWnd,

  // The parameter string that needs to be validated.
  [in] BSTR parameters,

  // Is the string valid?
  [out, retval] VARIANT_BOOL *valid
);
```

This method is called from the settings dialog where the user can add and configure the plugin. The `parameters` string can be used by a plugin to get additional required information, e.g., the URL to the issue tracker, login information, etc. The plugin should verify the `parameters` string and show an error dialog if the string is not valid. The `hParentWnd` parameter should be used for any dialog the plugin shows as the parent window. The plugin must return TRUE if the validation of the `parameters` string is successful. If the plugin returns FALSE, the settings dialog won't allow the user to add the plugin to a working copy path.

```
HRESULT GetLinkText (
  // Parent window for any (error) UI that needs to be displayed.
  [in] HWND hParentWnd,

  // The parameter string, just in case you need to talk to your
  // web service (e.g.) to find out what the correct text is.
  [in] BSTR parameters,

  // What text do you want to display?
  // Use the current thread locale.
  [out, retval] BSTR *linkText
);
```

The plugin can provide a string here which is used in the TortoiseSVN commit dialog for the button which invokes the plugin, e.g., "Choose issue" or "Select ticket". Make sure the string is not too long,

otherwise it might not fit into the button. If the method returns an error (e.g., `E_NOTIMPL`), a default text is used for the button.

```
HRESULT GetCommitMessage (
  // Parent window for your provider's UI.
  [in] HWND hParentWnd,

  // Parameters for your provider.
  [in] BSTR parameters,
  [in] BSTR commonRoot,
  [in] SAFEARRAY(BSTR) pathList,

  // The text already present in the commit message.
  // Your provider should include this text in the new message,
  // where appropriate.
  [in] BSTR originalMessage,

  // The new text for the commit message.
  // This replaces the original message.
  [out, retval] BSTR *newMessage
);
```

This is the main method of the plugin. This method is called from the TortoiseSVN commit dialog when the user clicks on the plugin button. The `parameters` string is the string the user has to enter in the settings dialog when he configures the plugin. Usually a plugin would use this to find the URL of the issue tracker and/or login information or more. The `commonRoot` string contains the parent path of all items selected to bring up the commit dialog. Note that this is *not* the root path of all items which the user has selected in the commit dialog. The `pathList` parameter contains an array of paths (as strings) which the user has selected for the commit. The `originalMessage` parameter contains the text entered in the log message box in the commit dialog. If the user has not yet entered any text, this string will be empty. The `newMessage` return string is copied into the log message edit box in the commit dialog, replacing whatever is already there. If a plugin does not modify the `originalMessage` string, it must return the same string again here, otherwise any text the user has entered will be lost.

6.2. The IBugtraqProvider2 interface

In TortoiseSVN 1.6 a new interface was added which provides more functionality for plugins. This IBugtraqProvider2 interface inherits from IBugtraqProvider.

```
HRESULT GetCommitMessage2 (
  // Parent window for your provider's UI.
  [in] HWND hParentWnd,

  // Parameters for your provider.
  [in] BSTR parameters,
  // The common URL of the commit
  [in] BSTR commonURL,
  [in] BSTR commonRoot,
  [in] SAFEARRAY(BSTR) pathList,

  // The text already present in the commit message.
  // Your provider should include this text in the new message,
  // where appropriate.
  [in] BSTR originalMessage,

  // You can assign custom revision properties to a commit
  // by setting the next two params.
```

```
  // note: Both safearrays must be of the same length.
  //       For every property name there must be a property value!

  // The content of the bugID field (if shown)
  [in] BSTR bugID,

  // Modified content of the bugID field
  [out] BSTR * bugIDOut,

  // The list of revision property names.
  [out] SAFEARRAY(BSTR) * revPropNames,

  // The list of revision property values.
  [out] SAFEARRAY(BSTR) * revPropValues,

  // The new text for the commit message.
  // This replaces the original message
  [out, retval] BSTR * newMessage
);
```

This method is called from the TortoiseSVN commit dialog when the user clicks on the plugin button. This method is called instead of `GetCommitMessage()`. Please refer to the documentation for `GetCommitMessage` for the parameters that are also used there. The parameter `commonURL` is the parent URL of all items selected to bring up the commit dialog. This is basically the URL of the `commonRoot` path. The parameter `bugID` contains the content of the bug-ID field (if it is shown, configured with the property `bugtraq:message`). The return parameter `bugIDOut` is used to fill the bug-ID field when the method returns. The `revPropNames` and `revPropValues` return parameters can contain name/value pairs for revision properties that the commit should set. A plugin must make sure that both arrays have the same size on return! Each property name in `revPropNames` must also have a corresponding value in `revPropValues`. If no revision properties are to be set, the plugin must return empty arrays.

```
HRESULT CheckCommit (
  [in] HWND hParentWnd,
  [in] BSTR parameters,
  [in] BSTR commonURL,
  [in] BSTR commonRoot,
  [in] SAFEARRAY(BSTR) pathList,
  [in] BSTR commitMessage,
  [out, retval] BSTR * errorMessage
);
```

This method is called right before the commit dialog is closed and the commit begins. A plugin can use this method to validate the selected files/folders for the commit and/or the commit message entered by the user. The parameters are the same as for `GetCommitMessage2()`, with the difference that `commonURL` is now the common URL of all *checked* items, and `commonRoot` the root path of all checked items. The return parameter `errorMessage` must either contain an error message which TortoiseSVN shows to the user or be empty for the commit to start. If an error message is returned, TortoiseSVN shows the error string in a dialog and keeps the commit dialog open so the user can correct whatever is wrong. A plugin should therefore return an error string which informs the user *what* is wrong and how to correct it.

```
HRESULT  OnCommitFinished (
  // Parent window for any (error) UI that needs to be displayed.
  [in] HWND hParentWnd,

  // The common root of all paths that got committed.
```

```
  [in] BSTR commonRoot,

  // All the paths that got committed.
  [in] SAFEARRAY(BSTR) pathList,


  // The text already present in the commit message.
  [in] BSTR logMessage,

  // The revision of the commit.
  [in] ULONG revision,


  // An error to show to the user if this function
  // returns something else than S_OK
  [out, retval] BSTR * error
);
```

This method is called after a successful commit. A plugin can use this method to e.g., close the selected issue or add information about the commit to the issue. The parameters are the same as for `GetCommitMessage2`.

```
HRESULT HasOptions(
  // Whether the provider provides options
  [out, retval] VARIANT_BOOL *ret
);
```

This method is called from the settings dialog where the user can configure the plugins. If a plugin provides its own configuration dialog with `ShowOptionsDialog`, it must return TRUE here, otherwise it must return FALSE.

```
HRESULT ShowOptionsDialog(
  // Parent window for the options dialog
  [in] HWND hParentWnd,

  // Parameters for your provider.
  [in] BSTR parameters,

  // The parameters string
  [out, retval] BSTR * newparameters
);
```

This method is called from the settings dialog when the user clicks on the "Options" button that is shown if `HasOptions` returns TRUE. A plugin can show an options dialog to make it easier for the user to configure the plugin. The `parameters` string contains the plugin parameters string that is already set/entered. The `newparameters` return parameter must contain the parameters string which the plugin constructed from the info it gathered in its options dialog. That `paramameters` string is passed to all other IBugtraqProvider and IBugtraqProvider2 methods.

Apêndice A. Frequently Asked Questions (FAQ)

Because TortoiseSVN is being developed all the time it is sometimes hard to keep the documentation completely up to date. We maintain an *online FAQ* [http://tortoisesvn.tigris.org/faq.html] which contains a selection of the questions we are asked the most on the TortoiseSVN mailing lists <dev@tortoisesvn.tigris.org> and <users@tortoisesvn.tigris.org>.

We also maintain a project *Issue Tracker* [http://issues.tortoisesvn.net] which tells you about some of the things we have on our To-Do list, and bugs which have already been fixed. If you think you have found a bug, or want to request a new feature, check here first to see if someone else got there before you.

If you have a question which is not answered anywhere else, the best place to ask it is on one of the mailing lists. <users@tortoisesvn.tigris.org> is the one to use if you have questions about using TortoiseSVN. If you want to help out with the development of TortoiseSVN, then you should take part in discussions on <dev@tortoisesvn.tigris.org>.

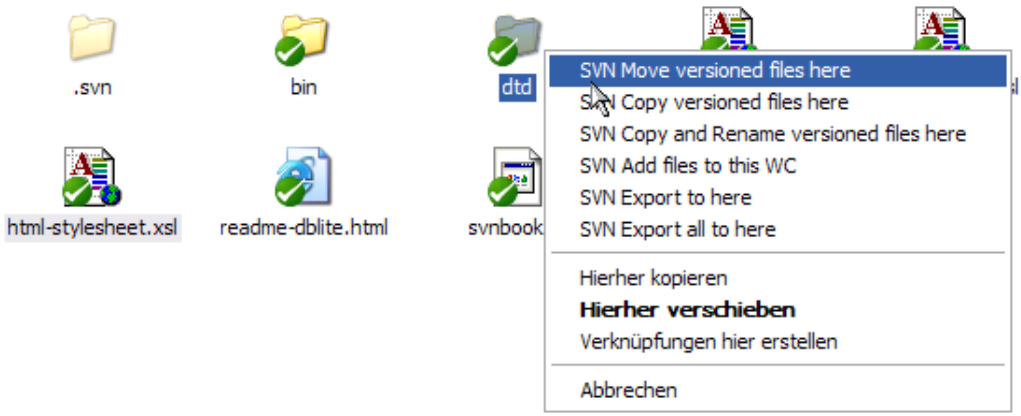
Apêndice B. Como eu faço...

This appendix contains solutions to problems/questions you might have when using TortoiseSVN.

B.1. Move/copy a lot of files at once

Moving/Copying single files can be done by using TortoiseSVN → Rename.... But if you want to move/copy a lot of files, this way is just too slow and too much work.

The recommended way is by right-dragging the files to the new location. Simply right-click on the files you want to move/copy without releasing the mouse button. Then drag the files to the new location and release the mouse button. A context menu will appear where you can either choose Context Menu → SVN Copy versioned files here. or Context Menu → SVN Move versioned files here.



B.2. Force users to enter a log message

There are two ways to prevent users from committing with an empty log message. One is specific to TortoiseSVN, the other works for all Subversion clients, but requires access to the server directly.

B.2.1. Hook-script on the server

If you have direct access to the repository server, you can install a pre-commit hook script which rejects all commits with an empty or too short log message.

In the repository folder on the server, there's a sub-folder `hooks` which contains some example hook scripts you can use. The file `pre-commit.tmpl` contains a sample script which will reject commits if no log message is supplied, or the message is too short. The file also contains comments on how to install/use this script. Just follow the instructions in that file.

This method is the recommended way if your users also use other Subversion clients than TortoiseSVN. The drawback is that the commit is rejected by the server and therefore users will get an error message. The client can't know before the commit that it will be rejected. If you want to make TortoiseSVN have the OK button disabled until the log message is long enough then please use the method described below.

B.2.2. Project properties

TortoiseSVN uses properties to control some of its features. One of those properties is the `tsvn:logminsize` property.

If you set that property on a folder, then TortoiseSVN will disable the OK button in all commit dialogs until the user has entered a log message with at least the length specified in the property.

For detailed information on those project properties, please refer to Seção 4.17, “Configurações do Projeto”

B.3. Update selected files from the repository

Normally you update your working copy using TortoiseSVN → Update. But if you only want to pick up some new files that a colleague has added without merging in any changes to other files at the same time, you need a different approach.

Use TortoiseSVN → Check for Modifications. and click on Check repository to see what has changed in the repository. Select the files you want to update locally, then use the context menu to update just those files.

B.4. Roll back (Undo) revisions in the repository

B.4.1. Use the revision log dialog

The easiest way to revert the changes from a single revision, or from a range of revisions, is to use the revision log dialog. This is also the method to use of you want to discard recent changes and make an earlier revision the new HEAD.

1. Select the file or folder in which you need to revert the changes. If you want to revert all changes, this should be the top level folder.
2. Select TortoiseSVN → Show Log to display a list of revisions. You may need to use Show All or Next 100 to show the revision(s) you are interested in.
3. Select the revision you wish to revert. If you want to undo a range of revisions, select the first one and hold the **Shift** key while selecting the last one. Note that for multiple revisions, the range must be unbroken with no gaps. Right click on the selected revision(s), then select Context Menu → Revert changes from this revision.
4. Or if you want to make an earlier revision the new HEAD revision, right click on the selected revision, then select Context Menu → Revert to this revision. This will discard *all* changes after the selected revision.

You have reverted the changes within your working copy. Check the results, then commit the changes.

B.4.2. Use the merge dialog

To undo a larger range of revisions, you can use the Merge dialog. The previous method uses merging behind the scenes; this method uses it explicitly.

1. In your working copy select TortoiseSVN → Merge.
2. In the From: field enter the full folder URL of the branch or tag containing the changes you want to revert in your working copy. This should come up as the default URL.
3. In the From Revision field enter the revision number that you are currently at. If you are sure there is no-one else making changes, you can use the HEAD revision.
4. make sure the Use "From:" URL checkbox is checked.

5. In the To Revision field enter the revision number that you want to revert to, namely the one *before* the first revision to be reverted.

6. Click OK to complete the merge.

You have reverted the changes within your working copy. Check the results, then commit the changes.

B.4.3. Use `svndumpfilter`

Since TortoiseSVN never loses data, your “rolled back” revisions still exist as intermediate revisions in the repository. Only the HEAD revision was changed to a previous state. If you want to make revisions disappear completely from your repository, erasing all trace that they ever existed, you have to use more extreme measures. Unless there is a really good reason to do this, it is *not recommended*. One possible reason would be that someone committed a confidential document to a public repository.

The only way to remove data from the repository is to use the Subversion command line tool `svnadmin`. You can find a description of how this works in the *Repository Maintenance* [http://svnbook.red-bean.com/en/1.5/svn.reposadmin.maint.html].

B.5. Compare two revisions of a file or folder

If you want to compare two revisions in an item's history, for example revisions 100 and 200 of the same file, just use TortoiseSVN → Show Log to list the revision history for that file. Pick the two revisions you want to compare then use Context Menu → Compare Revisions.

If you want to compare the same item in two different trees, for example the trunk and a branch, you can use the repository browser to open up both trees, select the file in both places, then use Context Menu → Compare Revisions.

If you want to compare two trees to see what has changed, for example the trunk and a tagged release, you can use TortoiseSVN → Revision Graph Select the two nodes to compare, then use Context Menu → Compare HEAD Revisions. This will show a list of changed files, and you can then select individual files to view the changes in detail. You can also export a tree structure containing all the changed files, or simply a list of all changed files. Read Seção 4.10.3, “Comparando Diretórios” for more information. Alternatively use Context Menu → Unified Diff of HEAD Revisions to see a summary of all differences, with minimal context.

B.6. Include a common sub-project

Sometimes you will want to include another project within your working copy, perhaps some library code. You don't want to make a duplicate of this code in your repository because then you would lose connection with the original (and maintained) code. Or maybe you have several projects which share core code. There are at least 3 ways of dealing with this.

B.6.1. Use svn:externals

Set the `svn:externals` property for a folder in your project. This property consists of one or more lines; each line has the name of a sub-folder which you want to use as the checkout folder for common code, and the repository URL that you want to be checked out there. For full details refer to Seção 4.18, “Itens Externos”.

Commit the new folder. Now when you update, Subversion will pull a copy of that project from its repository into your working copy. The sub-folders will be created automatically if required. Each time you update your main working copy, you will also receive the latest version of all external projects.

If the external project is in the same repository, any changes you make there there will be included in the commit list when you commit your main project.

Se o projeto externo estiver num repositório diferente, quaisquer alterações que fizer serão notificadas quando submeter o projeto principal mas terá que submeter essas alterações separadamente.

Of the three methods described, this is the only one which needs no setup on the client side. Once externals are specified in the folder properties, all clients will get populated folders when they update.

B.6.2. Use a nested working copy

Create a new folder within your project to contain the common code, but do not add it to Subversion

Select TortoiseSVN → Checkout for the new folder and checkout a copy of the common code into it. You now have a separate working copy nested within your main working copy.

The two working copies are independent. When you commit changes to the parent, changes to the nested WC are ignored. Likewise when you update the parent, the nested WC is not updated.

B.6.3. Use a relative location

If you use the same common core code in several projects, and you do not want to keep multiple working copies of it for every project that uses it, you can just check it out to a separate location which is related to all the other projects which use it. For example:

```
C:\Projects\Proj1
C:\Projects\Proj2
C:\Projects\Proj3
C:\Projects\Common
```

and refer to the common code using a relative path, eg. `..\..\Common\DSPcore`.

If your projects are scattered in unrelated locations you can use a variant of this, which is to put the common code in one location and use drive letter substitution to map that location to something you can hard code in your projects, eg. Checkout the common code to `D:\Documents\Framework` or `C:\Documents and Settings\{login}\My Documents\framework` then use

```
SUBST X: "D:\Documents\framework"
```

to create the drive mapping used in your source code. Your code can then use absolute locations.

```
#include "X:\superio\superio.h"
```

This method will only work in an all-PC environment, and you will need to document the required drive mappings so your team know where these mysterious files are. This method is strictly for use in closed development environments, and not recommended for general use.

B.7. Create a shortcut to a repository

If you frequently need to open the repository browser at a particular location, you can create a desktop shortcut using the automation interface to TortoiseProc. Just create a new shortcut and set the target to:

```
TortoiseProc.exe /command:repobrowser /path:"url/to/repository"
```

Of course you need to include the real repository URL.

B.8. Ignore files which are already versioned

If you accidentally added some files which should have been ignored, how do you get them out of version control without losing them? Maybe you have your own IDE configuration file which is not part of the project, but which took you a long time to set up just the way you like it.

If you have not yet committed the add, then all you have to do is use TortoiseSVN → Revert... to undo the add. You should then add the file(s) to the ignore list so they don't get added again later by mistake.

If the files are already in the repository, you have to do a little more work.

1. Hold the **Shift** key to get the extended context menu and use TortoiseSVN → Delete (keep local) to mark the file/folder for deletion from the repository without losing the local copy.
2. TortoiseSVN → Commit the parent folder.
3. Add the file/folder to the ignore list so you don't get into the same trouble again.

B.9. Unversion a working copy

If you have a working copy which you want to convert back to a plain folder tree without the `.svn` directories, you can simply export it to itself. Read Seção 4.26.1, “Removing a working copy from version control” to find out how.

B.10. Remove a working copy

If you have a working copy which you no longer need, how do you get rid of it cleanly? Easy - just delete it in Windows Explorer! Working copies are private local entities, and they are self-contained.

Apêndice C. Useful Tips For Administrators

This appendix contains solutions to problems/questions you might have when you are responsible for deploying TortoiseSVN to multiple client computers.

C.1. Deploy TortoiseSVN via group policies

The TortoiseSVN installer comes as an MSI file, which means you should have no problems adding that MSI file to the group policies of your domain controller.

A good walk-through on how to do that can be found in the knowledge base article 314934 from Microsoft: *http://support.microsoft.com/?kbid=314934*.

Versions 1.3.0 and later of TortoiseSVN must be installed under *Computer Configuration* and not under *User Configuration*. This is because those versions need the new CRT and MFC DLLs, which can only be deployed *per computer* and not *per user*. If you really must install TortoiseSVN on a per user basis, then you must first install the MFC and CRT package version 8 from Microsoft on each computer you want to install TortoiseSVN as per user.

C.2. Redirect the upgrade check

TortoiseSVN checks if there's a new version available every few days. If there is a newer version available, a dialog shows up informing the user about that.

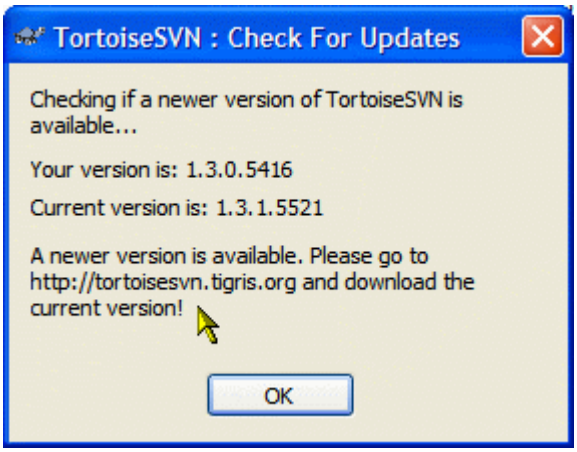


Figura C.1. A janela de atualização

If you're responsible for a lot of users in your domain, you might want your users to use only versions you have approved and not have them install always the latest version. You probably don't want that upgrade dialog to show up so your users don't go and upgrade immediately.

Versions 1.4.0 and later of TortoiseSVN allow you to redirect that upgrade check to your intranet server. You can set the registry key `HKCU\Software\TortoiseSVN\UpdateCheckURL` (string value) to an URL pointing to a text file in your intranet. That text file must have the following format:

```
1.4.1.6000
A new version of TortoiseSVN is available for you to download!
http://192.168.2.1/downloads/TortoiseSVN-1.4.1.6000-svn-1.4.0.msi
```

The first line in that file is the version string. You must make sure that it matches the exact version string of the TortoiseSVN installation package. The second line is a custom text, shown in the upgrade dialog. You can write there whatever you want. Just note that the space in the upgrade dialog is limited. Too long messages will get truncated! The third line is the URL to the new installation package. This URL is opened when the user clicks on the custom message label in the upgrade dialog. You can also just point the user to a web page instead of the MSI file directly. The URL is opened with the default web browser, so if you specify a web page, that page is opened and shown to the user. If you specify the MSI package, the browser will ask the user to save the MSI file locally.

C.3. Setting the SVN_ASP_DOT_NET_HACK environment variable

As of version 1.4.0 and later, the TortoiseSVN installer doesn't provide the user with the option to set the SVN_ASP_DOT_NET_HACK environment variable anymore, since that caused many problems and confusions with users which always install *everything* no matter if they know what it is for.

But that option is only hidden for the user. You still can force the TortoiseSVN installer to set that environment variable by setting the ASPDOTNETHACK property to TRUE. For example, you can start the installer like this:

```
msiexec /i TortoiseSVN-1.4.0.msi ASPDOTNETHACK=TRUE
```

C.4. Desabilitar opções do menu de contexto

As of version 1.5.0 and later, TortoiseSVN allows you to disable (actually, hide) context menu entries. Since this is a feature which should not be used lightly but only if there is a compelling reason, there is no GUI for this and it has to be done directly in the registry. This can be used to disable certain commands for users who should not use them. But please note that only the context menu entries in the *explorer* are hidden, and the commands are still available through other means, e.g. the command line or even other dialogs in TortoiseSVN itself!

The registry keys which hold the information on which context menus to show are HKEY_CURRENT_USER\Software\TortoiseSVN\ContextMenuEntriesMaskLow and HKEY_CURRENT_USER\Software\TortoiseSVN\ContextMenuEntriesMaskHigh.

Each of these registry entries is a DWORD value, with each bit corresponding to a specific menu entry. A set bit means the corresponding menu entry is deactivated.

Valor	Opção do Menu
0x0000000000000001	Obter
0x0000000000000002	Atualizar
0x0000000000000004	Submeter
0x0000000000000008	Adicionar
0x0000000000000010	Reverter
0x0000000000000020	Limpar
0x0000000000000040	Resolver
0x0000000000000080	Switch
0x0000000000000100	Importar
0x0000000000000200	Exportar
0x0000000000000400	Criar Repositório aqui
0x0000000000000800	Ramificar/Rotular...

Valor	Opção do Menu
0x0000000000001000	Combinar
0x0000000000002000	Apagar
0x0000000000004000	Renomear
0x0000000000008000	Atualizar para Revisão
0x0000000000010000	Diff
0x0000000000020000	Log
0x0000000000040000	Conflitos
0x0000000000080000	Reposicionar
0x0000000000100000	Verificar alterações
0x0000000000200000	Ignorar
0x0000000000400000	Navegador do Repositório
0x0000000000800000	Autoria
0x0000000001000000	Criar Correção
0x0000000002000000	Aplicar correção
0x0000000004000000	Gráfico de revisões
0x0000000008000000	Bloquear
0x0000000010000000	Remover bloqueio
0x0000000020000000	Propriedades
0x0000000040000000	Comparar com URL
0x0000000080000000	Excluir não versionados
0x2000000000000000	Configurações
0x4000000000000000	Ajuda
0x8000000000000000	Sobre

Tabela C.1. Menu entries and their values

Example: to disable the “Relocate” the “Delete unversioned items” and the “Settings” menu entries, add the values assigned to the entries like this:

```
  0x0000000000080000
+ 0x0000000080000000
+ 0x2000000000000000
= 0x2000000080080000
```

The lower `DWORD` value (`0x80080000`) must then be stored in `HKEY_CURRENT_USER\Software\TortoiseSVN\ContextMenuEntriesMaskLow`, the higher `DWORD` value (`0x20000000`) in `HKEY_CURRENT_USER\Software\TortoiseSVN\ContextMenuEntriesMaskHigh`.

To enable the menu entries again, simply delete the two registry keys.

Apêndice D. Automatizando o TortoiseSVN

Since all commands for TortoiseSVN are controlled through command line parameters, you can automate it with batch scripts or start specific commands and dialogs from other programs (e.g. your favourite text editor).

Importante

Remember that TortoiseSVN is a GUI client, and this automation guide shows you how to make the TortoiseSVN dialogs appear to collect user input. If you want to write a script which requires no input, you should use the official Subversion command line client instead.

D.1. Comandos do TortoiseSVN

The TortoiseSVN GUI program is called `TortoiseProc.exe`. All commands are specified with the parameter `/command:abcd` where `abcd` is the required command name. Most of these commands need at least one path argument, which is given with `/path:"some\path"`. In the following table the command refers to the `/command:abcd` parameter and the path refers to the `/path:"some\path"` parameter.

Since some of the commands can take a list of target paths (e.g. committing several specific files) the `/path` parameter can take several paths, separated by a `*` character.

TortoiseSVN uses temporary files to pass multiple arguments between the shell extension and the main program. From TortoiseSVN 1.5.0 on and later, `/notempfile` parameter is obsolete and there is no need to add it anymore.

The progress dialog which is used for commits, updates and many more commands usually stays open after the command has finished until the user presses the OK button. This can be changed by checking the corresponding option in the settings dialog. But using that setting will close the progress dialog, no matter if you start the command from your batch file or from the TortoiseSVN context menu.

To specify a different location of the configuration file, use the parameter `/configdir:"path\to\config\directory"`. This will override the default path, including any registry setting.

To close the progress dialog at the end of a command automatically without using the permanent setting you can pass the `/closeonend` parameter.

- `/closeonend:0` don't close the dialog automatically
- `/closeonend:1` auto close if no errors
- `/closeonend:2` auto close if no errors and conflicts
- `/closeonend:3` auto close if no errors, conflicts and merges
- `/closeonend:4` auto close if no errors, conflicts and merges for local operations

The table below lists all the commands which can be accessed using the TortoiseProc.exe command line. As described above, these should be used in the form `/command:abcd`. In the table, the `/command` prefix is omitted to save space.

Comando	Description
:about	Shows the about dialog. This is also shown if no command is given.

Comando	Description
:log	Opens the log dialog. The `/path` specifies the file or folder for which the log should be shown. Three additional options can be set: `/startrev:xxx`, `/endrev:xxx` and `/strict`
:checkout	Opens the checkout dialog. The `/path` specifies the target directory and the `/url` specifies the URL to checkout from.
:import	Opens the import dialog. The `/path` specifies the directory with the data to import.
:update	Updates the working copy in `/path` to HEAD. If the option `/rev` is given then a dialog is shown to ask the user to which revision the update should go. To avoid the dialog specify a revision number `/rev:1234`. Other options are `/nonrecursive` and `/ignoreexternals`.
:commit	Opens the commit dialog. The `/path` specifies the target directory or the list of files to commit. You can also specify the `/logmsg` switch to pass a predefined log message to the commit dialog. Or, if you don't want to pass the log message on the command line, use `/logmsgfile:caminho`, where `caminho` points to a file containing the log message. To pre-fill the bug ID box (in case you've set up integration with bug trackers properly), you can use the `/bugid:"o código do erro aqui"` to do that.
:add	Adds the files in `/path` to version control.
:revert	Reverts local modifications of a working copy. The `/path` tells which items to revert.
:cleanup	Cleans up interrupted or aborted operations and unlocks the working copy in `/path`.
:resolve	Marks a conflicted file specified in `/path` as resolved. If `/noquestion` is given, then resolving is done without asking the user first if it really should be done.
:repocreate	Cria um repositório em `/path`
:switch	Opens the switch dialog. The `/path` specifies the target directory.
:export	Exports the working copy in `/path` to another directory. If the `/path` points to an unversioned directory, a dialog will ask for an URL to export to the directory in `/path`.
:merge	Opens the merge dialog. The `/path` specifies the target directory. For merging a revision range, the following options are available: `/fromurl:URL`, `/revrange:texto`. For merging two repository trees, the following options are available: `/fromurl:URL`, `/tourl:URL`, `/fromrev:xxx` and `/torev:xxx`. These pre-fill the relevant fields in the merge dialog.
:mergeall	Opens the merge all dialog. The `/path` specifies the target directory.
:copy	Brings up the branch/tag dialog. The `/path` is the working copy to branch/tag from. And the `/url` is the target URL. You can also specify the `/logmsg` switch to pass a predefined log message to the branch/tag dialog. Or, if you don't want to pass the log message on the command line, use `/logmsgfile:caminho`, where `caminho` points to a file containing the log message.
:settings	Abre a janela de configurações.
:remove	Removes the file(s) in `/path` from version control.
:rename	Renames the file in `/path`. The new name for the file is asked with a dialog. To avoid the question about renaming similar files in one step, pass `/noquestion`.

Comando	Description
:diff	Starts the external diff program specified in the TortoiseSVN settings. The `/path` specifies the first file. If the option `/path2` is set, then the diff program is started with those two files. If `/path2` is omitted, then the diff is done between the file in `/path` and its BASE. To explicitly set the revision numbers use `/startrev:xxx` and `/endrev:xxx`. If `/blame` is set and `/path2` is not set, then the diff is done by first blaming the files with the given revisions.
:showcompare	Depending on the URLs and revisions to compare, this either shows a unified diff (if the option `unified` is set), a dialog with a list of files that have changed or if the URLs point to files starts the diff viewer for those two files. The options `url1`, `url2`, `revision1` and `revision2` must be specified. The options `pegrevision`, `ignoreancestry`, `blame` and `unified` are optional.
:conflicteditor	Starts the conflict editor specified in the TortoiseSVN settings with the correct files for the conflicted file in `/path`.
:relocate	Opens the relocate dialog. The `/path` specifies the working copy path to relocate.
:help	Abre o arquivo de ajuda.
:repostatus	Opens the check-for-modifications dialog. The `/path` specifies the working copy directory.
:repobrowser	Starts the repository browser dialog, pointing to the URL of the working copy given in `/path` or `/path` points directly to an URL. An additional option `/rev:xxx` can be used to specify the revision which the repository browser should show. If the `/rev:xxx` is omitted, it defaults to HEAD. If `/path` points to an URL, the `/projectpropertiespath:caminho/para/ct` specifies the path from where to read and use the project properties.
:ignore	Adds all targets in `/path` to the ignore list, i.e. adds the `svn:ignore` property to those files.
:blame	Opens the blame dialog for the file specified in `/path`. If the options `/startrev` and `/endrev` are set, then the dialog asking for the blame range is not shown but the revision values of those options are used instead. If the option `/line:nnn` is set, TortoiseBlame will open with the specified line number showing. The options `/ignoreeol`, `/ignorespaces` and `/ignoreallspaces` are also supported.
:cat	Saves a file from an URL or working copy path given in `/path` to the location given in `/savepath:caminho`. The revision is given in `/revision:xxx`. This can be used to get a file with a specific revision.
:createpatch	Creates a patch file for the path given in `/path`.
:revisiongraph	Shows the revision graph for the path given in `/path`.
:lock	Locks a file or all files in a directory given in `/path`. The 'lock' dialog is shown so the user can enter a comment for the lock.
:unlock	Unlocks a file or all files in a directory given in `/path`.
:rebuildiconcache	Rebuilds the windows icon cache. Only use this in case the windows icons are corrupted. A side effect of this (which can't be avoided) is that the

Comando	Description
	icons on the desktop get rearranged. To suppress the message box, pass `/noquestion`.
:properties	Shows the properties dialog for the path given in `/path`.

Tabela D.1. List of available commands and options

Examples (which should be entered on one line):

```
TortoiseProc.exe /command:commit
                 /path:"c:\svn_wc\file1.txt*c:\svn_wc\file2.txt"
                 /logmsg:"test log message" /closeonend:0

TortoiseProc.exe /command:update /path:"c:\svn_wc\" /closeonend:0

TortoiseProc.exe /command:log /path:"c:\svn_wc\file1.txt"
                 /startrev:50 /endrev:60 /closeonend:0
```

D.2. Comandos do TortoiseIDiff

The image diff tool has a few command line options which you can use to control how the tool is started. The program is called `TortoiseIDiff.exe`.

The table below lists all the options which can be passed to the image diff tool on the command line.

Opção	Description
:left	Path to the file shown on the left.
:lefttitle	A title string. This string is used in the image view title instead of the full path to the image file.
:right	Path to the file shown on the right.
:righttitle	A title string. This string is used in the image view title instead of the full path to the image file.
:overlay	If specified, the image diff tool switches to the overlay mode (alpha blend).
:fit	If specified, the image diff tool fits both images together.
:showinfo	Shows the image info box.

Tabela D.2. Lista de opções disponíveis

Example (which should be entered on one line):

```
TortoiseIDiff.exe /left:"c:\images\img1.jpg" /lefttitle:"image 1"
                  /right:"c:\images\img2.jpg" /righttitle:"image 2"
                  /fit /overlay
```

Apêndice E. Command Line Interface Cross Reference

Sometimes this manual refers you to the main Subversion documentation, which describes Subversion in terms of the Command Line Interface (CLI). To help you understand what TortoiseSVN is doing behind the scenes, we have compiled a list showing the equivalent CLI commands for each of TortoiseSVN's GUI operations.

> **Nota**
>
> Even though there are CLI equivalents to what TortoiseSVN does, remember that TortoiseSVN does *not* call the CLI but uses the Subversion library directly.

If you think you have found a bug in TortoiseSVN, we may ask you to try to reproduce it using the CLI, so that we can distinguish TortoiseSVN issues from Subversion issues. This reference tells you which command to try.

E.1. Conventions and Basic Rules

In the descriptions which follow, the URL for a repository location is shown simply as `URL`, and an example might be `http://tortoisesvn.googlecode.com/svn/trunk`. The working copy path is shown simply as `PATH`, and an example might be `C:\TortoiseSVN\trunk`.

> **Importante**
>
> Because TortoiseSVN is a Windows Shell Extension, it is not able to use the notion of a current working directory. All working copy paths must be given using the absolute path, not a relative path.

Certain items are optional, and these are often controlled by checkboxes or radio buttons in TortoiseSVN. These options are shown in [square brackets] in the command line definitions.

E.2. Comandos do TortoiseSVN

E.2.1. Obter

```
svn checkout [-N] [--ignore-externals] [-r rev] URL PATH
```

If Only checkout the top folder is checked, use the `-N` switch.

If Omit externals is checked, use the `--ignore-externals` switch.

If you are checking out a specific revision, specify that after the URL using `-r` switch.

E.2.2. Atualizar

```
svn info URL_of_WC
svn update [-r rev] PATH
```

Updating multiple items is currently not an atomic operation in Subversion. So TortoiseSVN first finds the HEAD revision of the repository, and then updates all items to that particular revision number to avoid creating a mixed revision working copy.

If only one item is selected for updating or the selected items are not all from the same repository, TortoiseSVN just updates to HEAD.

No command line options are used here. Update to revision also implements the update command, but offers more options.

E.2.3. Atualizar para Revisão

```
svn info URL_of_WC
svn update [-r rev] [-N] [--ignore-externals] PATH
```

If Only update the top folder is checked, use the `-N` switch.

If Omit externals is checked, use the `--ignore-externals` switch.

E.2.4. Submeter

In TortoiseSVN, the commit dialog uses several Subversion commands. The first stage is a status check which determines the items in your working copy which can potentially be committed. You can review the list, diff files against BASE and select the items you want to be included in the commit.

```
svn status -v CAMINHO
```

If Show unversioned files is checked, TortoiseSVN will also show all unversioned files and folders in the working copy hierarchy, taking account of the ignore rules. This particular feature has no direct equivalent in Subversion, as the `svn status` command does not descend into unversioned folders.

If you check any unversioned files and folders, those items will first be added to your working copy.

```
svn add CAMINHO...
```

When you click on OK, the Subversion commit takes place. If you have left all the file selection checkboxes in their default state, TortoiseSVN uses a single recursive commit of the working copy. If you deselect some files, then a non-recursive commit (`-N`) must be used, and every path must be specified individually on the commit command line.

```
svn commit -m "LogMessage" [-N] [--no-unlock] PATH...
```

`LogMessage` here represents the contents of the log message edit box. This can be empty.

If Keep locks is checked, use the `--no-unlock` switch.

E.2.5. Diff

```
svn diff CAMINHO
```

If you use Diff from the main context menu, you are diffing a modified file against its BASE revision. The output from the CLI command above also does this and produces output in unified-diff format. However, this is not what TortoiseSVN is using. TortoiseSVN uses TortoiseMerge (or a diff program of your choosing) to display differences visually between full-text files, so there is no direct CLI equivalent.

You can also diff any 2 files using TortoiseSVN, whether or not they are version controlled. TortoiseSVN just feeds the two files into the chosen diff program and lets it work out where the differences lie.

E.2.6. Log

```
svn log -v -r 0:N --limit 100 [--stop-on-copy] PATH
  or
svn log -v -r M:N [--stop-on-copy] PATH
```

By default, TortoiseSVN tries to fetch 100 log messages using the --limit method. If the settings instruct it to use old APIs, then the second form is used to fetch the log messages for 100 repository revisions.

If Stop on copy/rename is checked, use the `--stop-on-copy` switch.

E.2.7. Procurar por Modificações

```
svn status -v CAMINHO
  or
svn status -u -v CAMINHO
```

The initial status check looks only at your working copy. If you click on Check repository then the repository is also checked to see which files would be changed by an update, which requires the `-u` switch.

If Show unversioned files is checked, TortoiseSVN will also show all unversioned files and folders in the working copy hierarchy, taking account of the ignore rules. This particular feature has no direct equivalent in Subversion, as the `svn status` command does not descend into unversioned folders.

E.2.8. Gráfico de Revisões

The revision graph is a feature of TortoiseSVN only. There's no equivalent in the command line client.

What TortoiseSVN does is an

```
svn info URL_of_WC
svn log -v URL
```

where URL is the repository *root* and then analyzes the data returned.

E.2.9. Navegador de

```
svn info URL_of_WC
svn list [-r rev] -v URL
```

You can use `svn info` to determine the repository root, which is the top level shown in the repository browser. You cannot navigate Up above this level. Also, this command returns all the locking information shown in the repository browser.

The `svn list` call will list the contents of a directory, given a URL and revision.

E.2.10. Conflitos

This command has no CLI equivalent. It invokes TortoiseMerge or an external 3-way diff/merge tool to look at the files involved in the conflict and sort out which lines to use.

E.2.11. Resolvido

```
svn resolved CAMINHO
```

E.2.12. Renomear

```
svn rename CAMINHO_ATUAL NOVO_CAMINHO
```

E.2.13. Apagar

```
svn delete CAMINHO
```

E.2.14. Reverter

```
svn status -v CAMINHO
```

The first stage is a status check which determines the items in your working copy which can potentially be reverted. You can review the list, diff files against BASE and select the items you want to be included in the revert.

When you click on OK, the Subversion revert takes place. If you have left all the file selection checkboxes in their default state, TortoiseSVN uses a single recursive (`-R`) revert of the working copy. If you deselect some files, then every path must be specified individually on the revert command line.

```
svn revert [-R] CAMINHO...
```

E.2.15. Limpar

```
svn cleanup CAMINHO
```

E.2.16. Obter bloqueio

```
svn status -v CAMINHO
```

The first stage is a status check which determines the files in your working copy which can potentially be locked. You can select the items you want to be locked.

```
svn lock -m "LockMessage" [--force] PATH...
```

`LockMessage` here represents the contents of the lock message edit box. This can be empty.

If Steal the locks is checked, use the `--force` switch.

E.2.17. Liberar bloqueio

```
svn unlock CAMINHO
```

E.2.18. Ramificar/Rotular...

```
svn copy -m "LogMessage" URL URL
  or
svn copy -m "LogMessage" URL@rev URL@rev
  or
svn copy -m "LogMessage" PATH URL
```

The Branch/Tag dialog performs a copy to the repository. There are 3 radio button options:

- Última revisão no repositório

- especC which correspond to the 3 command line variants above.

`LogMessage` here represents the contents of the log message edit box. This can be empty.

E.2.19. Switch

```
svn info URL_of_WC
svn switch [-r rev] URL PATH
```

E.2.20. Combinar

```
svn merge [--dry-run] --force From_URL@revN To_URL@revM PATH
```

The Test Merge performs the same merge with the `--dry-run` switch.

```
svn diff From_URL@revN To_URL@revM
```

The Unified diff shows the diff operation which will be used to do the merge.

E.2.21. Exportar

```
svn export [-r rev] [--ignore-externals] URL Export_PATH
```

This form is used when accessed from an unversioned folder, and the folder is used as the destination.

Exporting a working copy to a different location is done without using the Subversion library, so there's no matching command line equivalent.

What TortoiseSVN does is to copy all files to the new location while showing you the progress of the operation. Unversioned files/folders can optionally be exported too.

In both cases, if Omit externals is checked, use the `--ignore-externals` switch.

E.2.22. Reposicionar

```
svn switch --relocate From_URL To_URL
```

E.2.23. Criar repositó aqui

```
svnadmin create --fs-type fsfs PATH
```

E.2.24. Adicionar

```
svn add CAMINHO...
```

If you selected a folder, TortoiseSVN first scans it recursively for items which can be added.

E.2.25. Importar

```
svn import -m LogMessage PATH URL
```

`LogMessage` here represents the contents of the log message edit box. This can be empty.

E.2.26. Autoria

```
svn blame -r N:M -v CAMINHO
svn log -r N:M CAMINHO
```

If you use TortoiseBlame to view the blame info, the file log is also required to show log messages in a tooltip. If you view blame as a text file, this information is not required.

E.2.27. Adicionar à lista de ignorados

```
svn propget svn:ignore PATH > tempfile
{edit new ignore item into tempfile}
svn propset svn:ignore -F tempfile PATH
```

Because the `svn:ignore` property is often a multi-line value, it is shown here as being changed via a text file rather than directly on the command line.

E.2.28. Criar Correção

```
svn diff PATH > patch-file
```

TortoiseSVN creates a patch file in unified diff format by comparing the working copy with its BASE version.

E.2.29. Aplicar correção

Applying patches is a tricky business unless the patch and working copy are at the same revision. Luckily for you, you can use TortoiseMerge, which has no direct equivalent in Subversion.

Apêndice F. Detalhes da Implementação

This appendix contains a more detailed discussion of the implementation of some of TortoiseSVN's features.

F.1. Sobreposição dos Ícones

Every file and folder has a Subversion status value as reported by the Subversion library. In the command line client, these are represented by single letter codes, but in TortoiseSVN they are shown graphically using the icon overlays. Because the number of overlays is very limited, each overlay may represent one of several status values.

The *Conflicted* overlay is used to represent the `conflicted` state, where an update or switch results in conflicts between local changes and changes downloaded from the repository. It is also used to indicate the `obstructed` state, which can occur when an operation is unable to complete.

The *Modified* overlay represents the `modified` state, where you have made local modifications, the `merged` state, where changes from the repository have been merged with local changes, and the `replaced` state, where a file has been deleted and replaced by another different file with the same name.

The *Deleted* overlay represents the `deleted` state, where an item is scheduled for deletion, or the `missing` state, where an item is not present. Naturally an item which is missing cannot have an overlay itself, but the parent folder can be marked if one of its child items is missing.

The *Added* overlay is simply used to represent the `added` status when an item has been added to version control.

The *In Subversion* overlay is used to represent an item which is in the `normal` state, or a versioned item whose state is not yet known. Because TortoiseSVN uses a background caching process to gather status, it may take a few seconds before the overlay updates.

The *Needs Lock* overlay is used to indicate when a file has the `svn:needs-lock` property set. For working copies which were created using Subversion 1.4.0 and later, the `svn:needs-lock` status is cached locally by Subversion and this is used to determine when to show this overlay. For working copies which are in pre-1.4.x format, TortoiseSVN shows this overlay when the file has read-only status. Note that Subversion automatically upgrades working copies when you update them, although the caching of the `svn:needs-lock` property may not happen until the file itself is updated.

The *Locked* overlay is used when the local working copy holds a lock for that file.

The *Ignored* overlay is used to represent an item which is in the `ignored` state, either due to a global ignore pattern, or the `svn:ignore` property of the parent folder. This overlay is optional.

The *Unversioned* overlay is used to represent an item which is in the `unversioned` state. This is an item in a versioned folder, but which is not under version control itself. This overlay is optional.

If an item has subversion status `none` (the item is not within a working copy) then no overlay is shown. If you have chosen to disable the *Ignored* and *Unversioned* overlays then no overlay will be shown for those files either.

An item can only have one Subversion status value. For example a file could be locally modified and it could be marked for deletion at the same time. Subversion returns a single status value - in this case `deleted`. Those priorities are defined within Subversion itself.

When TortoiseSVN displays the status recursively (the default setting), each folder displays an overlay reflecting its own status and the status of all its children. In order to display a single *summary* overlay, we use the priority order shown above to determine which overlay to use, with the *Conflicted* overlay taking highest priority.

In fact, you may find that not all of these icons are used on your system. This is because the number of overlays allowed by Windows is limited to 15. Windows uses 4 of those, and the remaining 11 can be used by other applications. If there are not enough overlay slots available, TortoiseSVN tries to be a “Good Citizen (TM)” and limits its use of overlays to give other apps a chance.

- *Normal*, *Modified* and *Conflicted* are always loaded and visible.
- *Deleted* is loaded if possible, but falls back to *Modified* if there are not enough slots.
- *Read-Only* is loaded if possible, but falls back to *Normal* if there are not enough slots.
- *Locked* is only loaded if there are fewer than 13 overlays already loaded. It falls back to *Normal* if there are not enough slots.
- *Added* is only loaded if there are fewer than 14 overlays already loaded. It falls back to *Modified* if there are not enough slots.

Apêndice G. Securing Svnserve using SSH

This section provides a step-by-step guide to setting up Subversion and TortoiseSVN to use the `svn+ssh` protocol. If you already use authenticated SSH connections to login to your server, then you are already there and you can find more detail in the Subversion book. If you are not using SSH but would like to do so to protect your Subversion installation, this guide gives a simple method which does not involve creating a separate SSH user account on the server for every subversion user.

In this implementation we create a single SSH user account for all subversion users, and use different authentication keys to differentiate between the real Subversion users.

In this appendix we assume that you already have the subversion tools installed, and that you have created a repository as detailed elsewhere in this manual. Note that you should *not* start svnserve as a service or daemon when used with SSH.

Much of the information here comes from a tutorial provided by Marc Logemann, which can be found at *www.logemann.org* [http://www.logemann.org/2007/03/13/subversion-tortoisesvn-ssh-howto/] Additional information on setting up a Windows server was provided by Thorsten Müller. Thanks guys!

G.1. Setting Up a Linux Server

You need to have SSH enabled on the server, and here we assume that you will be using OpenSSH. On most distributions this will already be installed. To find out, type:

```
ps xa | grep sshd
```

and look for ssh jobs.

One point to note is that if you build Subversion from source and do not provide any argument to `./configure`, Subversion creates a `bin` directory under `/usr/local` and places its binaries there. If you want to use tunneling mode with SSH, you have to be aware that the user logging in via SSH needs to execute the svnserve program and some other binaries. For this reason, either place `/usr/local/bin` into the `PATH` variable or create symbolic links of your binaries to the `/usr/sbin` directory, or to any other directory which is commonly in the `PATH`.

To check that everything is OK, login in as the target user with SSH and type:

```
which svnserve
```

This command should tell you if svnserve is reachable.

Create a new user which we will use to access the svn repository:

```
useradd -m svnuser
```

Be sure to give this user full access rights to the repository.

G.2. Configurando um Servidor Windows

Install Cygwin SSH daemon as described here: *http://pigtail.net/LRP/printsrv/cygwin-sshd.html*

Create a new Windows user account `svnuser` which we will use to access the repository. Be sure to give this user full access rights to the repository.

If there is no password file yet then create one from the Cygwin console using:

```
mkpasswd -l > /etc/passwd
```

G.3. SSH Client Tools for use with TortoiseSVN

Grab the tools we need for using SSH on the windows client from this site: *http://www.chiark.greenend.org.uk/~sgtatham/putty/* Just go to the download section and get `Putty`, `Plink`, `Pageant` and `Puttygen`.

G.4. Criando Certificados OpenSSH

The next step is to create a key pair for authentication. There are two possible ways to create keys. The first is to create the keys with PuTTYgen on the client, upload the public key to your server and use the private key with PuTTY. The other is to create the key pair with the OpenSSH tool ssh-keygen, download the private key to your client and convert the private key to a PuTTY-style private key.

G.4.1. Create Keys using ssh-keygen

Login to the server as `root` or `svnuser` and type:

```
ssh-keygen -b 1024 -t dsa -N passphrase -f keyfile
```

substituting a real pass-phrase (which only you know) and key file. We just created a SSH2 DSA key with 1024 bit key-phrase. If you type

```
ls -l keyfile*
```

you will see two files, `keyfile` and `keyfile.pub`. As you might guess, the `.pub` file is the public key file, the other is the private one.

Append the public key to those in the `.ssh` folder within the `svnuser` home directory:

```
cat keyfile.pub >> /home/svnuser/.ssh/authorized_keys
```

In order to use the private key we generated, we have to convert it to a putty format. This is because the private key file format is not specified by a standards body. After you download the private key file to your client PC, start PuTTYgen and use Conversions → Import key. Browse to your file `keyfile` which you got from the server the passphrase you used when creating the key. Finally click on Save private key and save the file as `keyfile.PPK`.

G.4.2. Create Keys using PuTTYgen

Use PuTTYgen to generate a public-key/private-key pair and save it. Copy the public key to the server and append it to those in the `.ssh` folder within the `svnuser` home directory:

```
cat keyfile.pub >> /home/svnuser/.ssh/authorized_keys
```

G.5. Teste usando PuTTY

To test the connection we will use PuTTY. Start the program and on the Session tab set the hostname to the name or IP address of your server, the protocol to SSH and save the session as `SvnConnection` or whatever name you prefer. On the SSH tab set the preferred SSH protocol version to 2 and from Auth set the full path to the `.PPK` private key file you converted earlier. Go back to the Sessions tab and hit the Save button. You will now see `SvnConnection` in the list of saved sessions.

Click on Open and you should see a telnet style login prompt. Use `svnuser` as the user name and if all is well you should connect directly without being prompted for a password.

You may need to edit `/etc/sshd_config` on the server. Edit lines as follows and restart the SSH service afterwards.

```
PubkeyAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no
```

G.6. Testando SSH com TortoiseSVN

So far we have only tested that you can login using SSH. Now we need to make sure that the SSH connection can actually run svnserve. On the server modify `/home/svnuser/.ssh/authorized_keys` as follows to allow many subversion authors to use the same system account, `svnuser`. Note that every subversion author uses the same login but a different authentication key, thus you have to add one line for every author.

Note: This is all on one very long line.

```
command="svnserve -t -r <ReposRootPath> --tunnel-user=<author>",
            no-port-forwarding,no-agent-forwarding,no-X11-forwarding,
            no-pty ssh-rsa <PublicKey> <Comment>
```

There are several values that you need to set according to your setup.

`<ReposRootPath>` should be replaced with the path to the directory containing your repositories. This avoids the need to specify full server paths within URLs. Note that you must use forward slashes even on a Windows server, e.g. `c:/svn/reposroot`. In the examples below we assume that you have a repository folder within the repository root called `repos`.

`<author>` should be replaced with the svn author that you want to be stored on commit. This also allows svnserve to use its own access rights within `svnserve.conf`.

`<PublicKey>` should be replaced with the public key that you generated earlier.

`<Comment>` can be any comment you like, but it is useful for mapping an svn author name to the person's real name.

Right click on any folder in Windows Explorer and select TortoiseSVN → Repo-Browser. You will be prompted to enter a URL, so enter one in this form:

```
svn+ssh://svnuser@SvnConnection/repos
```

What does this URL mean? The Schema name is `svn+ssh` which tells TortoiseSVN how to handle the requests to the server. After the double slash, you specify the user to connect to the server, in our case `svnuser`. After the `@` we supply our PuTTY session name. This session name contains all details like where to find the private key and the server's IP or DNS. Lastly we have to provide the path to the repository, relative to the repository root on the server, as specified in the `authorized_keys` file.

Click on OK and you should be able to browse the repository content. If so you now have a running SSH tunnel in conjunction with TortoiseSVN.

Note that by default TortoiseSVN uses its own version of Plink to connect. This avoids a console window popping up for every authentication attempt, but it also means that there is nowhere for error messages to appear. If you receive the error “Unable to write to standard output”, you can try specifying Plink as

the client in TortoiseSVN's network settings. This will allow you to see the real error message generated by Plink.

G.7. Variantes da Configuração SSH

One way to simplify the URL in TortoiseSVN is to set the user inside the PuTTY session. For this you have to load your already defined session `SvnConnection` in PuTTY and in the Connection tab set Auto login user to the user name, e.g. `svnuser`. Save your PuTTY session as before and try the following URL inside TortoiseSVN:

```
svn+ssh://SvnConnection/repos
```

This time we only provide the PuTTY session `SvnConnection` to the SSH client TortoiseSVN uses (TortoisePlink.exe). This client will check the session for all necessary details.

At the time of writing PuTTY does not check all saved configurations, so if you have multiple configurations with the same server name, it will pick the first one which matches. Also, if you edit the default configuration and save it, the auto login user name is *not* saved.

Many people like to use Pageant for storing all their keys. Because a PuTTY session is capable of storing a key, you don't always need Pageant. But imagine you want to store keys for several different servers; in that case you would have to edit the PuTTY session over and over again, depending on the server you are trying to connect with. In this situation Pageant makes perfect sense, because when PuTTY, Plink, TortoisePlink or any other PuTTY-based tool is trying to connect to an SSH server, it checks all private keys that Pageant holds to initiate the connection.

For this task, simply run Pageant and add the private key. It should be the same private key you defined in the PuTTY session above. If you use Pageant for private key storage, you can delete the reference to the private key file in your saved PuTTY session. You can add more keys for other servers, or other users of course.

If you don't want to repeat this procedure after every reboot of your client, you should place Pageant in the auto-start group of your Windows installation. You can append the keys with complete paths as command line arguments to Pageant.exe

The last way to connect to an SSH server is simply by using this URL inside TortoiseSVN:

```
svn+ssh://svnuser@100.101.102.103/repos
svn+ssh://svnuser@mydomain.com/repos
```

As you can see, we don't use a saved PuTTY session but an IP address (or domain name) as the connection target. We also supply the user, but you might ask how the private key file will be found. Because TortoisePlink.exe is just a modified version of the standard Plink tool from the PuTTY suite, TortoiseSVN will also try all the keys stored in Pageant.

If you use this last method, be sure you do not have a default username set in PuTTY. We have had reports of a bug in PuTTY causing connections to close in this case. To remove the default user, simply clear `HKEY_CURRENT_USER\Software\SimonTatham\Putty\Sessions\Default%20Settings\HostName`

Glossary

Adicionar	A Subversion command that is used to add a file or directory to your working copy. The new items are added to the repository when you commit.
Apagar	When you delete a versioned item (and commit the change) the item no longer exists in the repository after the committed revision. But of course it still exists in earlier repository revisions, so you can still access it. If necessary, you can copy a deleted item and “resurrect” it complete with history.
Atualizar	This Subversion command pulls down the latest changes from the repository into your working copy, merging any changes made by others with local changes in the working copy.
Autoria	This command is for text files only, and it annotates every line to show the repository revision in which it was last changed, and the author who made that change. Our GUI implementation is called TortoiseBlame and it also shows the commit date/time and the log message when you hover the mouse of the revision number.
BDB	Berkeley DB. A well tested database backend for repositories, that cannot be used on network shares. Default for pre 1.2 repositories.
Bloquear	When you take out a lock on a versioned item, you mark it in the repository as non-committable, except from the working copy where the lock was taken out.
Combinar	The process by which changes from the repository are added to your working copy without disrupting any changes you have already made locally. Sometimes these changes cannot be reconciled automatically and the working copy is said to be in conflict. Merging happens automatically when you update your working copy. You can also merge specific changes from another branch using TortoiseSVN's Merge command.
Conflito	When changes from the repository are merged with local changes, sometimes those changes occur on the same lines. In this case Subversion cannot automatically decide which version to use and the file is said to be in conflict. You have to edit the file manually and resolve the conflict before you can commit any further changes.
Cópia de Trabalho	This is your local “sandbox”, the area where you work on the versioned files, and it normally resides on your local hard disk. You create a working copy by doing a “Checkout” from a repository, and you feed your changes back into the repository using “Commit”.
Copiar	In a Subversion repository you can create a copy of a single file or an entire tree. These are implemented as “cheap copies” which act a bit like a link to the original in that they take up almost no space. Making a copy preserves the history of the item in the copy, so you can trace changes made before the copy was made.
Correção	If a working copy has changes to text files only, it is possible to use Subversion's Diff command to generate a single file summary of those changes in Unified Diff format. A file of this type is often referred to as a “Patch”, and it can be emailed to someone else (or

	to a mailing list) and applied to another working copy. Someone without commit access can make changes and submit a patch file for an authorized committer to apply. Or if you are unsure about a change you can submit a patch for others to review.
Diff	Shorthand for “Show Differences”. Very useful when you want to see exactly what changes have been made.
Exportar	This command produces a copy of a versioned folder, just like a working copy, but without the local `.svn` folders.
FSFS	A proprietary Subversion filesystem backend for repositories. Can be used on network shares. Default for 1.2 and newer repositories.
GPO	Objeto de política de grupo
Histórico	Show the revision history of a file or folder. Also known as “Log”.
Importar	Subversion command to import an entire folder hierarchy into the repository in a single revision.
Limpar	To quote from the Subversion book: “ Recursively clean up the working copy, removing locks and resuming unfinished operations. If you ever get a *working copy locked* error, run this command to remove stale locks and get your working copy into a usable state again. ” Note that in this context *lock* refers to local filesystem locking, not repository locking.
Log	Show the revision history of a file or folder. Also known as “History”.
Obter	A Subversion command which creates a local working copy in an empty directory by downloading versioned files from the repository.
Propriedade	In addition to versioning your directories and files, Subversion allows you to add versioned metadata - referred to as “properties” to each of your versioned directories and files. Each property has a name and a value, rather like a registry key. Subversion has some special properties which it uses internally, such as `svn:eol-style`. TortoiseSVN has some too, such as `tsvn:logminsize`. You can add your own properties with any name and value you choose.
Propriedade da Revisão (revprop)	Just as files can have properties, so can each revision in the repository. Some special revprops are added automatically when the revision is created, namely: `svn:date svn:author svn:log` which represent the commit date/time, the committer and the log message respectively. These properties can be edited, but they are not versioned, so any change is permanent and cannot be undone.
Ramificação	A term frequently used in revision control systems to describe what happens when development forks at a particular point and follows 2 separate paths. You can create a branch off the main development line so as to develop a new feature without rendering the main line unstable. Or you can branch a stable release to which you make only bug fixes, while new developments take place on the unstable trunk. In Subversion a branch is implemented as a “cheap copy”.
Reposicionar	If your repository moves, perhaps because you have moved it to a different directory on your server, or the server domain name

	has changed, you need to “relocate” your working copy so that its repository URLs point to the new location. Note: you should only use this command if your working copy is referring to the same location in the same repository, but the repository itself has moved. In any other circumstance you probably need the “Switch” command instead.
Repositório	A repository is a central place where data is stored and maintained. A repository can be a place where multiple databases or files are located for distribution over a network, or a repository can be a location that is directly accessible to the user without having to travel across a network.
Resolver	When files in a working copy are left in a conflicted state following a merge, those conflicts must be sorted out by a human using an editor (or perhaps TortoiseMerge). This process is referred to as “Resolving Conflicts”. When this is complete you can mark the conflicted files as being resolved, which allows them to be committed.
Reverter	Subversion keeps a local “pristine” copy of each file as it was when you last updated your working copy. If you have made changes and decide you want to undo them, you can use the “revert” command to go back to the pristine copy.
Revisão	Every time you commit a set of changes, you create one new “revision” in the repository. Each revision represents the state of the repository tree at a certain point in its history. If you want to go back in time you can examine the repository as it was at revision N. In another sense, a revision can refer to the set of changes that were made when that revision was created.
Revisão BASE	The current base revision of a file or folder in your *working copy*. This is the revision the file or folder was in, when the last checkout, update or commit was run. The BASE revision is normally not equal to the HEAD revision.
Submeter	This Subversion command is used to pass the changes in your local working copy back into the repository, creating a new repository revision.
SVN	A frequently-used abbreviation for Subversion. The name of the Subversion custom protocol used by the “svnserve” repository server.
Switch	Just as “Update-to-revision” changes the time window of a working copy to look at a different point in history, so “Switch” changes the space window of a working copy so that it points to a different part of the repository. It is particularly useful when working on trunk and branches where only a few files differ. You can switch your working copy between the two and only the changed files will be transferred.
Última Revisão	The latest revision of a file or folder in the *repository*.

Índice Remissivo

L

M

N

O

P

R

S

T

U

V

W